# YouProve: Authenticity and Fidelity in Mobile Sensing

Peter Gilbert
Duke University
Durham, NC 27708
gilbert@cs.duke.edu

Jaeyeon Jung
Microsoft Research
Redmond, WA 98052
jjung@microsoft.com

Kyungmin Lee
Duke University
Durham, NC 27708
kyungmin.lee@duke.edu

Henry Qin
Duke University
Durham, NC 27708
henry.qin@duke.edu

Daniel Sharkey
Duke University
Durham, NC 27708
daniel.sharkey@duke.edu

Anmol Sheth
Technicolor Research
Palo Alto, CA 94301
anmol.sheth@technicolor.com

Landon P. Cox
Duke University
Durham, NC 27708
lpcox@cs.duke.edu

## Abstract

As more services have come to rely on sensor data such as audio and photos collected by mobile phone users, verifying the authenticity of this data has become critical for service correctness. At the same time, clients require the flexibility to tradeoff the fidelity of the data they contribute for resource efficiency or privacy. This paper describes YouProve, a partnership between a mobile device's trusted hardware and software that allows untrusted client applications to directly control the fidelity of data they upload and services to verify that the meaning of source data is preserved. The key to our approach is trusted analysis of derived data, which generates statements comparing the content of a derived data item to its source. Experiments with a prototype implementation for Android demonstrate that YouProve is feasible. Our photo analyzer is over 99% accurate at identifying regions changed only through meaning-preserving modifications such as cropping, compression, and scaling. Our audio analyzer is similarly accurate at detecting which sub-clips of a source audio clip are present in a derived version, even in the face of compression, normalization, splicing, and other modifications. Finally, performance and power costs are reasonable, with analyzers having little noticeable effect on interactive applications and CPU-intensive analysis completing asynchronously in under 70 seconds for 5-minute audio clips and under 30 seconds for 5-megapixel photos.

## Categories and Subject Descriptors

C.3 [**Special-purpose and Application-based Systems**]: Real-time and embedded systems

## General Terms

Design, Security


*SenSys'11*, November 1–4, 2011, Seattle, WA, USA.


## Keywords

Participatory Sensing, Privacy, Trusted Computing

## 1 Introduction

Mobile phones are fast becoming the eyes and ears of the Internet by embedding digital communication, computation, and sensing within the activities of daily life. The next generation of Internet platforms promises to support services like citizen journalism, mobile social networking [13], environmental monitoring [24], and traffic monitoring [17] by pairing the ubiquitous sensing provided by mobile phones with the large-scale data collection and dissemination capabilities of the cloud.

*Data authenticity* is crucial for service correctness. Mobile social services have already been gamed by participants claiming to be in places they were not [18], and citizen-journalism services have been fooled by falsified images [20, 34]. Correctness is especially important for services such as Al Jazeera's Sharek and CNN's iReport. Deploying trusted reporters and photographers into events such as those recently experienced in Iran, Haiti, Tunisia, Egypt, and Libya is difficult. Due to logistical obstacles, government bans, and reprisals against journalists, anonymous local citizens with camera phones were instrumental in documenting these situations. Thus, given the increasingly large role crowd-sourced content plays in world affairs and the dire consequences that dissemination of falsified media could have, verifying the authenticity of this data is paramount.

One proposed solution is to equip phones with trustworthy sensors capable of signing their readings and to require clients to return unmodified signed data to a service [10]. Unfortunately, requiring clients to send unmodified data is impractical. Mobile clients require the flexibility to trade-off *data fidelity* for efficient resource usage and greater privacy. This is particularly true for media such as audio and photos. For example, a client may wish to upload a photo with reduced resolution or under lossy compression to improve energy-efficiency and performance [6], or a client may wish to blur faces in a photo to conceal someone's identity [26]. Resolving the tension between data authenticity and data fidelity is a key obstacle to realizing the vision of phone-based distributed sensing.

Trusted hardware such as a Trusted Platform Module (TPM) can serve as the foundation of a solution. A partnership between a device's trustworthy hardware and its system software can produce digitally-signed statements about a data item's "chain of custody" to a remote service. However, even given this outline of a solution, several questions remain: What form should the partnership between device hardware and software take? What statements should a client present to a service? On what bases should a service trust a client's statements? What are the energy and performance implications of generating those statements on a resource-constrained mobile client?

In this paper, we address these questions by presenting the design and implementation of *YouProve*, a framework for verifying how data is captured and modified on a sensor-equipped mobile phone. We believe that the ability to verify that a derived data item preserves the meaning of an original sensor reading is an important step for evaluating data authenticity in domains such as citizen journalism. The key to our approach is *type-specific analysis* of derived sensor data. Type-specific analysis can be implemented by using well-known audio-analysis and computer-vision libraries to compare the *content* of a source item (e.g., an original audio clip or photo) to the content of a derived version of the item. The goal of type-specific analysis is to allow client applications to apply fidelity-reducing modifications to data and to give services a basis for trusting that those modifications preserved the meaning of the source.

To meet this goal, YouProve logs sensor data as it is requested by an application, and uses TaintDroid [11] to tag and track data as it flows through the application. If the application generates an output that is derived from a logged source, YouProve invokes the appropriate analyzer to compare the output to its source. The result of type-specific analysis is a *fidelity certificate*, which summarizes the software configuration of a device as well as how closely the content of a derived data item matches its source. By providing trustworthy statements about the degree to which a derived item's content matches its source, YouProve allows a service to verify that the meaning of the source item was preserved without requiring (1) clients to send the source, or (2) services to trust the applications that generated the derived item.

We have implemented a YouProve prototype for Android on a Nexus One and evaluated the performance and accuracy of audio and photo analyzers. Our prototype photo analyzer is over 99% accurate at identifying regions changed only through meaning-preserving modifications such as cropping, compression, and scaling. Our prototype audio analyzer is similarly accurate at detecting which sub-clips of a source audio clip are present in a derived version, even in the face of compression, normalization, splicing, and other modifications. Finally, the performance and power costs of YouProve are reasonable, with logging having little noticeable effect on interactive applications and CPU-intensive analysis completing asynchronously in under 70 seconds for 5-minute audio clips and under 30 seconds for 5-megapixel photos.

# 2 Design Considerations

Mobile sensing services (also called participatory sensing [8]) consist of servers that collect, aggregate, and disseminate geo-tagged sensor data such as audio and images from volunteer mobile clients. The case for fidelity-aware mobile clients is well established [6, 19, 12, 15, 27], while the case for verifying the authenticity of sensing data has been made more recently [10, 14, 21, 30, 31]. *YouProve* is a trustworthy sensing platform built on Android that allows a client to control the fidelity of data it submits and sensing services to verify that the meaning of source data is preserved across any modifications. In this section, we provide background information on key aspects of YouProve's design, including descriptions of (1) the relationship between data authenticity and fidelity, (2) Trusted Platform Modules (TPMs), and (3) our underlying design principles.

## 2.1 Authenticity, fidelity, and trust

Two crucial concerns for a system that allows data fidelity to be traded off for energy, privacy, or other considerations are (1) what code modifies a source data item, and (2) what code consumes the derived item. An important part of verifying the authenticity of a derived item is the process of certifying that its content preserves the meaning of its source. We further define the term "meaning" in the context of audio and photo data in Section 5.

Fidelity has traditionally been studied in the context of mobile clients retrieving data from servers over a wireless network [12, 19, 27]. In these settings, a small set of servers are trusted to maintain canonical copies of all source data and generate reduced-fidelity versions at the request of a client. Clients typically trust only a small set of servers based on the reputations of the servers' administrators. As long as a server can prove to a client that it is a member of the trusted set, the client considers the server's data to be authentic.

In a mobile sensing service, clients use sensors such as cameras, microphones, and GPS receivers to generate source data items and may produce a derived item by reducing the source's fidelity. Servers receive derived data and interpret its content to implement a service's logic. However, unlike in a traditional system like a distributed file system, a mobile sensing service cannot always rely on reputations to verify data authenticity. Data may be provided by clients without a prior history, by those who wish to remain anonymous, or by clients whose reputations are inaccurate due to Sybil-style gaming [9]. Other potential bases of trust are also problematic. Verifying authenticity by relying on a majority vote among related items is vulnerable to Sybil attacks, in which an attacker exercises disproportionate influence by creating a large number of identities. Relying on co-located trusted "witnesses" limits authenticity guarantees to data from locations with trustworthy infrastructure [21, 30].

Several groups have sought to decouple client reputations from data authenticity using trusted hardware such as a Trusted Platform Module (TPM) [10, 14, 25, 31, 37] or ARM TrustZone [29]. TPMs are included in most PCs sold today, and a specification for a Mobile Trusted Module (MTM) for mobile phones has been released [5]. Similarly, most shipping ARM processors support TrustZone, which is a hardware-isolated, secure-execution environment that could be leveraged by phone manufacturers to implement functionality similar to a TPM's [4]. We take the presence of trusted hardware on mobile clients as a given, and we have designed

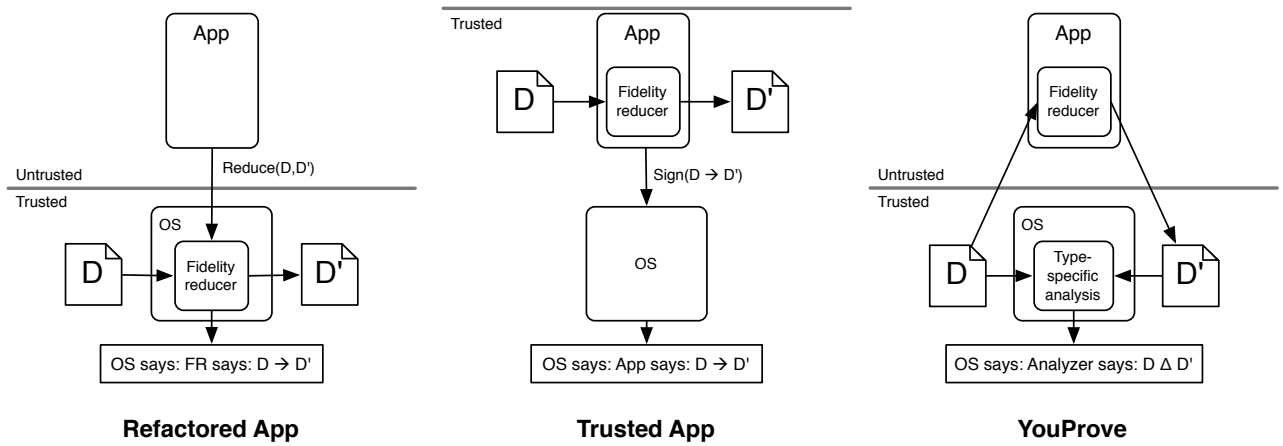


**Figure 1. Possible locations of fidelity-reduction code. Modules below each gray line are trusted by the remote service.**

YouProve as a set of software services running on top of such hardware.

## 2.2 TPM background

TPMs provide a root of trust on each YouProve client. A TPM can be used to provide a verifiable boot sequence, in which each piece of code that runs during boot is *measured* by the cryptographic hash of its content prior to being executed. Each measurement is *extended* into a Platform Configuration Register (PCR) by the TPM such that the value of a PCR is loaded with a hash of its current value concatenated with the new measurement. Each TPM includes an array of PCRs that can be updated only through the extend operation and reset only by rebooting the device.

A TPM can *attest* to the state of its PCRs by generating a *quote* that is signed with a private key. Each TPM contains a unique public-private key pair called an Endorsement Key (EK) that is installed at manufacture time. Each EK uniquely identifies an individual device. To perform anonymous quoting, a TPM can generate new public-private key pairs called Attestation Identity Keys (AIKs). In order for services to trust an AIK, a trusted third-party privacy certificate authority (privacy CA) must generate a certificate for the public half of the AIK. We assume that trusted privacy CAs will only certify a small number of AIKs for each EK to limit the scope of Sybil attacks.

A service can verify that a device's system software is trusted by checking that the PCR values reported in a quote match known values for a trusted configuration. A client's software configuration is trusted by a service if it ensures a trustworthy chain of custody for sensor data. As many have noted [25, 36, 37], verifying that the software platform of a mobile phone matches a trusted configuation is a promising technique because phone manufacturers release a relatively small number of read-only firmware updates that encapsulate the entire trusted codebase (TCB) of a device. Third-party applications are typically excluded from the TCB via an isolated execution environment. Furthermore, mobile devices do not give users root access by default, limiting opportunities for modifying the TCB configuation. This results in a high degree of homogeneity among TCB configurations on mobile devices. In contrast, traditional PC systems are more amenable to customization, resulting in numerous possible TCB configurations that must be evaluated.

While a user can gain root access on her phone by flashing a customized firmware with relative ease, such changes would be exposed to a remote verifier by a TPM's measurement of the boot sequence. Furthermore, users who root their phones are in the minority, allowing a service to reason about the trustworthiness of the vast majority of devices by establishing trust in a manageable number of configurations.

## 2.3 Design principles

The trusted chain of custody for sensor data must minimally include a trustworthy bootloader, OS kernel or hypervisor, and device drivers. There are a number of tradeoffs to consider in choosing the bases of trust for the rest of the chain. In particular, the critical challenge for YouProve is handling the code that performs fidelity reductions on source data. With this challenge in mind, we designed YouProve using the following principles:

### Build on deployed systems.

We could have taken a clean-slate approach to YouProve by developing a new operating system or by using an experimental OS for trusted hardware like Nexus [32]. For example, alternative architectures using virtual machines provide a smaller trusted computing base than YouProve by removing components inessential to handling sensor data [14, 16, 31]. Instead we designed YouProve as a new set of trusted services for Android. Building on top of Android is simple, creates a lower barrier to deployment, and allows us to take advantage of existing Android tools. The disadvantage of our approach is that YouProve must be secured within Android's security model. We describe the process of securing YouProve in greater detail in Section 4.

### Allow applications to directly modify data.

Verifying a data item's authenticity involves proving that it was derived from source data in a trustworthy way. One way to enable verification is to refactor applications by requiring trusted code to reduce the fidelity of source data on applications' behalf. Fidelity-reducing code would be included in the kernel or run as a trusted server in user space,

and would use the trusted hardware to generate statements describing the reductions it performed. This approach is shown as Refactored App in Figure 1. The appeal of this approach is that it simplifies verification by limiting the number fidelity-reducing codebases that a service needs to trust.

APIs for controlling some forms of data fidelity at *capture time* already exist. For example, Android apps can adjust the fidelity of the location readings it receives via the `location.Criteria` class, the resolution and quality of the photos they receive via the `Camera.Parameters` class, and the sampling rate of audio via the `MediaRecorder` class. However, extending these existing APIs to support (1) a broader range of fidelity-reducing operations (e.g., cropping or blurring subregions of an image) and (2) data modifications during *post-processing* would require significant modifications to thousands of apps.

For example, there are nearly 6,000 apps under the "Photography" category of Apple's App Store, and many are media editors that operate on data captured in the past. Camera+, iMovie, and Garageband are several high profile editors. Similarly, Adobe Photoshop Express for Android has been installed over one million times. Furthermore, standalone media editors are not the only apps that perform fidelity reduction on data after it has been captured. Images taken using an iPhone 4 have a resolution of 2592x1936 pixels, but the Facebook API documentation strongly recommends that third-party apps resize images to a maximum of 2048 pixels along the longest edge before uploading [28]. Similarly, Instagram, a popular photo-sharing service which recently surpassed 2 million users [23], requires users to scale and crop images using the app to fit within a 612x612 pixel square before uploading.

As a result, rather than force existing apps to be refactored, YouProve allows unmodified third-party apps to continue to directly perform fidelity reduction at any point in a data item's lifetime.

**Trust analysis rather than synthesis.**

Another option is to allow apps to perform fidelity reduction themselves, but to provide a system API for generating signed statements about the app and its execution. This approach is exemplified by CertiPics, a trustworthy image-editing application developed for Nexus [32]. As long as a service trusts a program to correctly modify source data and to correctly describe what it did, then the service can verify the authenticity of the program's output. This approach is shown as Trusted App in Figure 1.

The primary disadvantage of this approach is that either (1) a user must restrict herself to using the small number of apps that her services trust, or (2) a service must establish trust in each of the thousands of apps a user might wish to use. Forcing clients to use a small number of apps deemed trustworthy by her services undermines the surge of development activity that has made consumer mobile devices popular and useful. On the other hand, forcing services to reason about the trustworthiness of the thousands of apps that directly manage data fidelity is impractical. Mobile apps are generally closed source (making them difficult to inspect), and verifying the trustworthiness of tens of thousands of mobile developers is infeasible.

In the terminology of the Nexus Authentication Logic, both refactoring apps and certifying statements generated by trusted apps offer *synthetic bases* of trust. In both approaches, trusted code transforms source data and generates a signed statement about its output. This is similar to a trusted compiler generating a signed statement about the type safety of executable code that it outputs. However, as we have observed, synthetic bases of trust force developers to make significant modifications to existing apps, constrain which apps a user can use, or impose an impractical trust-management burden on services.

As a result, YouProve eschews synthetic bases of trust in favor of *analytic bases* of trust. An analytic basis for trust requires verifiers to trust code to analyze inputs as opposed to trusting code to synthesize an output. Trusted analyzers can be included in a device's firmware and allow untrusted apps to handle fidelity reduction. At a high level, an analyzer generates a report comparing the content of an application's output to the content of the original source. YouProve can then embed the analyzer's report and a measurement of the device's software configuration in a signed *fidelity certificate*. Services use fidelity certificates to reason about both the trustworthiness of a device and whether a derived data item preserves the meaning of its source.

Relying on trusted analysis rather than trusted synthesis allows YouProve to (1) simplify verification by placing all trusted code within the device firmware, (2) preserve the autonomy of apps to directly perform fidelity reduction, and (3) maximize users' choice of applications. This approach is shown as YouProve in Figure 1.

# 3 Trust and Threat Model

In this section, we discuss assumptions made by YouProve about a device's hardware and software configuration.

## 3.1 Trust assumptions

As stated in Section 2.2, the root of trust for each YouProve client is a TPM. If the private half of a TPM's EK or the private half of an AIK becomes compromised through an attack against the TPM hardware, then an attacker can generate arbitrary TPM quotes. However, as long as the privacy CAs that certify AIKs are not compromised, then an attacker will only be able to generate arbitrary quotes using the limited number of AIKs certified by the privacy CAs. Thus, as long as privacy CAs remain trustworthy, a single attacker cannot successfully masquerade as a large number of devices unless it compromises a large number of TPMs.

Building YouProve on top of Android instead of an experimental operating system was an explicit design choice, and a consequence of this choice is that YouProve inherits Android's security model. Thus, YouProve relies on Android's existing mechanisms to thwart attacks against the platform. The essential components of Android's security model are a Linux kernel, user-space daemons called services running under privileged UIDs, and an IPC framework called Binder. Each Android app is signed by its developer and runs as a Linux process with its own unique, unprivileged UID. Apps access protected resources such as the camera service through library code that makes IPC calls. Platform services limit interactions with untrusted applica-

tion code by applying UID-based access-control policies on Binder communication.

YouProve assumes that modified firmware allowing users to execute code as root can be detected by inspecting a device's TPM quotes. Stock Android firmware typically does not give users root access, although some users will reflash their device to gain root access. As long as the bootloader remains uncompromised, any changes to the user-modifiable firmware, which includes the trusted software platform, will be apparent to a remote service via the TPM quote embedded in a fidelity certificate.

Eliminating all software vulnerabilities is beyond the scope of this paper—attacks against specific vulnerabilities in the implementation of a secure platform such as Android are long-standing and serious problems that are unlikely to disappear anytime soon. If an attacker manages to gain root access through a runtime exploit without modifying a device's trusted firmware, then the attacker can generate false analyses of sensor data. Nonetheless, as we will describe in Section 4, our design applies the principle of least privilege to isolate the effects of attacks against specific system components such as the camera and audio services and the log of source sensor data.

Other side-channel attacks that take advantage of physical access are beyond the scope of this work. YouProve stores encryption keys in memory and is therefore susceptible to "cold boot" attacks, in which an attacker retrieves residual data values from RAM after a hard reboot. An attacker could also inject false sensor readings while avoiding detection by physically interposing on the bus used for inter-device communication inside the phone. Specific hardware support would be needed to prevent these attacks.

Finally, "analog" attacks such as staged photos, photos of photos, or forged GPS signals are beyond the scope of YouProve. Incorporating data from multiple sensors into authenticity analysis could be helpful in detecting photos of photos or forged GPS signals; for example, it may be possible to identify a forged GPS location by checking if a contemporaneous temperature reading has a compatible value. However, there is only so much that a computer system can do to ensure data authenticity. Approaches like YouProve that consider original sensor readings as the "root" of authenticity cannot ensure that the event captured in a photo or audio recording was not staged.

## 3.2 Privacy implications

Previous work on trustworthy mobile sensing has suggested that a fundamental tradeoff exists between data authenticity and user privacy [14]. However, YouProve seeks to enhance both data authenticity and privacy by allowing a user to provide verifiable proof that a data item is authentic, even after fidelity reductions have been applied locally to remove identifying information. Rather than relying on the identity or reputation of a data contributor as a basis for trust, services can instead verify properties of a device's underlying software and hardware configuration. Thus, YouProve allows a device owner to attest to the authenticity of sensor data without revealing her identity.

On the other hand, YouProve's design also introduces new potential privacy risks for users. When a user publishes a fidelity certificate, she exposes detailed information about the hardware and software configuration of her device. However, it is important to note that the decision to upload a fidelity certificate is an explicit choice that remains under the user's control. We see YouProve as an opt-in service for users who wish to prove that data they generate is authentic.

While YouProve supports anonymous submissions, a design choice mentioned in Section 2.3 affects the extent to which a YouProve user can remain anonymous: reuse of an AIK when generating multiple certificates. Because TPM quotes signed with the same AIK can be attributed to the same physical device, anonymity may be compromised when a YouProve client reuses an AIK for multiple submissions. As previously mentioned, a privacy CA will certify only a small number of AIKs for a given device to limit the scope of Sybil attacks, in which a single device masquerades as multiple devices submitting data. Thus, a tradeoff exists between preventing Sybil attacks and enabling individual users to generate many unlinkable certificates. As a compromise, YouProve allows several AIKs to be certified for a device and leaves to the user the responsibility of managing multiple identities associated with different AIKs. The choice of how many AIKs to certify for a device should be based on the potential damage to service quality due to Sybil attacks—we expect that the degree to which Sybil attacks can be tolerated will vary for different mobile sensing domains. The task of assisting users in managing multiple identities to prevent deanonymization is left for future work.

# 4 YouProve

YouProve consists of four trusted software components responsible for performing the following tasks:

- *Logging* sensor data returned by the Android platform in response to requests from apps,
- *Tracking* data derived from sensor readings as it is manipulated by untrusted third-party apps,
- *Analyzing* the content of a derived data item and its source reading, and
- *Attesting* to the results of content analysis and the integrity of the software platform.

The rest of this section describes the design of each of these components.

## 4.1 Design overview

When an Android app wants to access sensor data it submits a request via a platform API (e.g., `Camera.takePicture`). YouProve interposes on such requests, assigning each request a unique identifier and inserting the response data along with its identifier into a *secure log*. YouProve must then track this data as it is modified by untrusted apps until a final data item is uploaded to a service provider. Standard meta-data tags are not sufficient for this purpose because sensor data may propagate through multiple file formats and representations. For example, consider an audio clip that is received by a third-party app as uncompressed samples in a memory buffer, processed by the app, converted to MP3 format, and then written to disk.

To track sensor data independently of its format, YouProve uses the TaintDroid [11] information-flow moni-

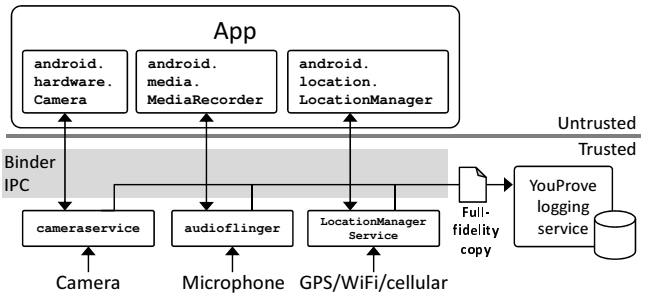


**Figure 2. Logging sensor readings.**

tor to tag the response data with the identifier and propagate the tag to derived data in program memory, files, and IPC. TaintDroid is not used as a security mechanism. Rather, it is used to expose dependencies between source data and applications' outputs. The only consequence of TaintDroid's losing track of a taint tag would be that the authenticity of an application's output could not be verified with a fidelity certificate since the output would not be mappable to a source reading for analysis.

When an app generates an output such as a file write, YouProve inspects the output's taint tag. If the tag can be mapped to a logged data request, YouProve forwards the app's tagged output and its logged input to a *type-specific analyzer*. The analyzer synchronously generates an identifier for the output (e.g., by computing its cryptographic hash). At that point, YouProve can explicitly return the certificate identifier to the app or transparently embed it in the meta-data of the output such as an image's Exif data. The analyzer then asynchronously generates a report comparing the app's input and output. YouProve then embeds the report in a fidelity certificate and posts it to a remote hosting service.

A service that receives a data item can retrieve the item's fidelity certificate from a hosting service using the appropriate identifier, and can decide whether the data is authentic based on the content of the certificate.

This overview raises a number of questions: How does YouProve secure the log of responses and the type-specific analyzers? How do services establish trust in fidelity certificates? What information is included in a fidelity certificate?

## 4.2 Logging sensor readings

YouProve makes trustworthy statements about the content of a derived data item by comparing it to source data captured by a sensor. To support this analysis, it is necessary for the trusted platform to collect a full-fidelity copy of any sensor reading returned to an application and to protect the integrity of the stored copy as long as a user wishes to generate fidelity certificates for data derived from the reading. YouProve's *logging service* provides this functionality.

The logging service assigns a unique ID to each sensor reading and stores the original copy to a file. It also writes a database entry with a timestamp and type-specific metadata, as well as a pointer to the original copy on the filesystem and a digest of its contents. The database is stored on internal flash storage, while the actual data contents are stored on external storage (i.e., an SD card) due to the limited amount of built-in storage available on many smartphones.

The accuracy of the timestamp recorded for an original sensor reading is a critical part of data authenticity for many services. Because YouProve's timestamps are generated using a device's local clock, we must ensure that the clock is synchronized with a trustworthy source and protected from tampering. For this purpose, YouProve synchronizes the device's local clock with a trusted time server at boot-time using authenticated NTP [3].

On a typical smartphone platform, sensor data passes through a number of software layers between a hardware sensor and application code, including device drivers and user-level platform code. Choosing the level at which to log full-fidelity source data has implications for the size of the TCB, the performance and storage overheads of logging, and portability to different hardware devices. To minimize performance and storage overheads and maximize portability, YouProve captures data at a high level in the software stack, immediately before it enters an untrusted app's address space. As a result, full-fidelity copies have the same format, including encoding and possibly compression, as the data received by an app, and all platform code that handles sensor data prior to handoff to the app is included in the TCB. We discuss this tradeoff further in Section 6.

Android provides Java interfaces to apps for accessing camera, microphone, and location data. This Java library code runs in an app's address space and communicates via IPC with a system service designated to handle the specific type of sensor data. Android only allows the system services `cameraservice` and `audioflinger` to communicate with the camera and microphone device drivers, respectively. The `LocationManagerService` handles GPS and network-based location data. We instrumented these three system services to report sensor readings to the YouProve logging service via IPC before returning data to a requesting app. Figure 2 depicts the interaction between Android platform services and the YouProve logging service.

Ensuring the integrity of data recorded by the logging service is critical. If an attacker can impersonate one of the Android services entrusted to handle sensor data (i.e., `cameraservice`, `audioflinger`, or `LocationManagerService`) and submit inauthentic data to the logging service, fidelity reports will not be trustworthy. Trustworthiness will also be undermined if sensor readings or metadata in the log database can be modified by untrusted code without detection. As a result, YouProve must verify the identity of the services providing sensor data and verify the integrity of sensor data and metadata when it is read from persistent storage.

To authenticate requests to insert sensor data into the log, YouProve relies on Android's UID-based privilege-separation model and the process-identity information provided by Android's Binder IPC subsystem. Binder tells each endpoint of an IPC connection the UID of the other communicating process. A special UID "media" is assigned to the `mediaserver` process that hosts `cameraservice` and `audioflinger`. `LocationManagerService` runs under the UID "system", which is shared by a number of trusted services. Android executes only trusted system code under the "system" UID, and no process may run under the "media" UID after the `mediaserver` launches during Android's boot sequence. YouProve leverages the restrictions placed on

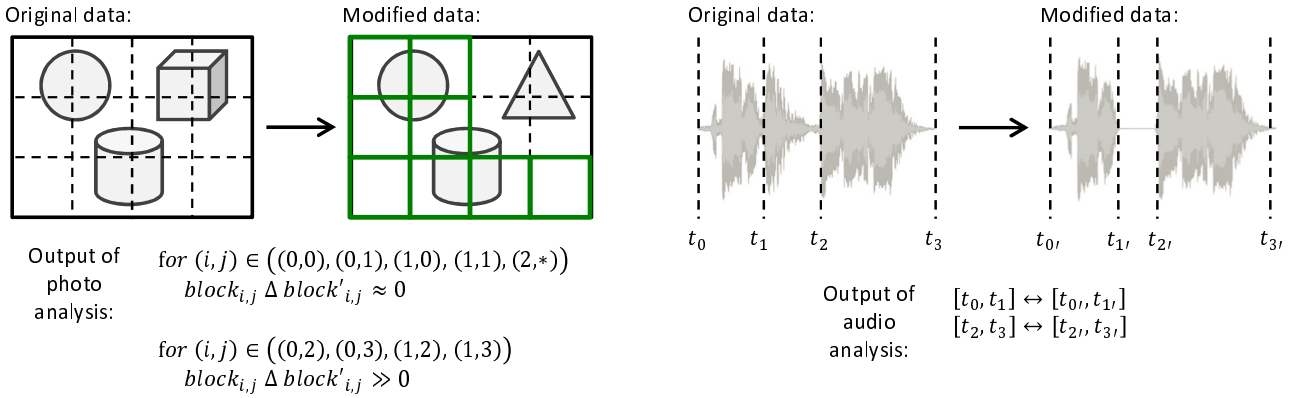


**Figure 3. Overview of photo and audio content analysis.**

these UIDs by the kernel to authenticate requests to log data. The logging service only accepts requests to log photo and audio data from processes with the "media" UID (i.e., the `mediaserver`) and requests to log location data from processes with the "system" UID (i.e., the `LocationManagerService` and other privileged services).

In addition to authenticating requests to the logging service, YouProve must also protect the integrity of the log database and the full-fidelity copies stored on disk. Full-fidelity copies are kept on external storage and may be vulnerable to external modification. YouProve ensures that modifications to these copies will be detected by verifying the SHA-1 digest of the data against the value stored in the log. To protect the integrity of the log database, YouProve signs each log entry with the private half of a key pair generated at install-time and bound to a trusted software configuration using a TPM's *sealed storage* functionality.

Sealed storage allows the system to submit arbitrary data to the TPM to be "sealed" to the current values of selected PCRs. Upon such a request, the TPM returns an encrypted blob that can later be "unsealed" only when the same PCRs are in the same state. This ensures that the private key used to sign log entries will be accessible to device software only after the trusted platform has booted into a known, trusted state. Note that YouProve does not attempt to prevent denial of service attacks wherein a user deletes full-fidelity copies or log entries—these attacks simply result in the user being unable to attest to fidelity-reduced data.

## 4.3 Tracking derived data

YouProve's type-specific analyzers operate on two data items: a full-fidelity source item and a derived version of the source. To enable comparisons YouProve relies on the TaintDroid [11] information-flow monitoring framework as a lightweight means for tracking data dependencies throughout the Android system. Before the platform returns sensor data to a user app, it attaches a taint tag encoding the 32-bit unique ID assigned to the sensor reading—this ID serves as the primary key for the entry in the log database. If tainted data is appended to a file or IPC message already marked with a different ID, the file or message is marked with a newly allocated ID and the mapping to the two previous IDs is recorded in the log. In this way, YouProve can properly

```
<cert dev_id="device pseudonym" cert_id="unique per device">
  <report>
    <content_digest>SHA1(content)</content_digest>
    <timestamp>from original sensor reading</timestamp>
    <analysis>type-specific content analysis results</analysis>
  </report>
  <report_digest>SHA1(report)</report_digest>
  <platform>
    <pcr0>
      <boot>SHA1(boot partition)</boot>
      <system>SHA1(system partition)</system>
    </pcr0>
    <aik_pub>AIK_pub</aik_pub>
    <tpm_quote>sig{PCR_0, report_digest}_AIK_priv</tpm_quote>
  </platform>
</cert>
```

**Figure 4. Format of a fidelity certificate.**

track all dependencies for high-level data items such as geo-tagged images. Due to space constraints, we have left out a longer discussion of handling data items whose taint tags map to multiple sensor readings. Our YouProve prototype currently supports geo-tagged camera and microphone data.

## 4.4 Analyzing content

YouProve's type-specific analyzers report the differences between an original sensor reading and a derived data item. Analyzers implement a simple, common interface: they take as input two files containing sensor data, and they output a human-readable report identifying portions of the modified data item that preserve "meaning" from the original sensor reading. Additional information may be provided about the differences between corresponding regions in the source and derived items when their contents do not match.

For both photo and audio data, YouProve's approach is to divide a derived data item into smaller regions and then attempt to match the content of each region to that of a corresponding region in the source sensor reading. Photos are divided by rectangular grid, while audio analysis considers time segments. An overview YouProve's approach and the basic format of the output of analysis is shown in Figure 3. Our prototype analyzers for photo and audio content are described in detail in Section 5.

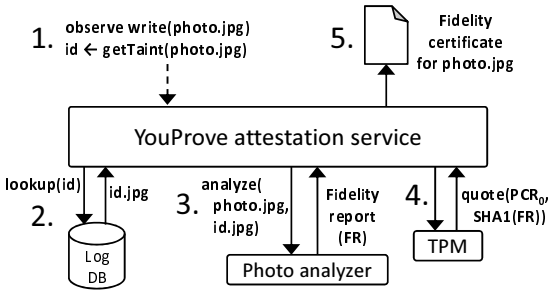


**Figure 5. Steps for attestation.**

## 4.5 Attesting to analysis and platform

To enable data consumers to reason about the trustworthiness of a data item, YouProve's *attestation service* generates fidelity certificates that report the results of type-specific analysis and a timestamp for the original reading. Fidelity certificates also contain information about the device's software platform, allowing remote verifiers to decide whether or not to trust reports generated by the device. The format for fidelity certificates is shown in Figure 4. The two basic parts are (1) a *report* that describes a data item and is bound to the data by a content digest, and (2) a description of the platform software configuration, including a TPM quote that attests to the state of the software platform and binds the platform-specific part of the certificate to the content-analysis part.

Fidelity certificates are posted by a YouProve client to a remote *certificate hosting service*—a simple key-value store that makes submitted certificates available through well-known public URLs. YouProve generates fidelity certificates in response to explicit requests by the user, or by monitoring the user filesystem for writes to files with supported data types (e.g., jpegs and mp3s). If desired, a link to the URL where the certificate will eventually be available can be embedded as metadata in the media file before uploading to a sensing service. The basic steps performed by YouProve to generate a fidelity certificate are shown in Figure 5.

To verify a certificate, a service first verifies the signatures and hashes. Next it uses the analysis report to evaluate a service-defined policy regarding the authenticity of received data. We imagine that an analyzer's report will most commonly be used to strengthen the case for an item's authenticity. Items with ambiguous reports leave services in their current positions of relying strictly on other means to verify data's authenticity. We discuss the kind of information analyzers embed in their reports in the following section.

# 5 Type-specific Analyzers

In this section, we describe YouProve's analyzers for photo and audio content. The central design challenge was to apply state-of-the-art content analysis techniques without incurring excessive runtime or energy overheads.

## 5.1 Photo content

The goal of YouProve's photo analysis is to identity regions of a derived photo that preserve meaning from a source photo captured by the camera hardware. We consider the meaning of a photo to be preserved if its appearance remains roughly unchanged. Thus, transformations that typically preserve the meaning of photo content include image compression, relatively small adjustments to image parameters such as brightness and contrast, and fixed-aspect ratio scaling. Furthermore, cropping preserves the appearance of any regions which are not removed. We consider manipulations that distort the image or significantly change regions in other ways (e.g., pasting in content from another photo) to alter the meaning of the sub-region of the photo. Ultimately, we seek to allow a viewer to categorize sub-regions of a modified photo based on whether they preserve meaning from a corresponding region in the source photo.

Our analysis utilizes two well-known techniques from computer vision: Speeded-Up Robust Features [7] (SURF) and Sum of Squared Differences (SSD). SURF facilitates matching regions in two images by locating "keypoints" such as corners, and then computing a feature vector called a *descriptor* to represent each keypoint. A "matching" between the descriptors in two images can be found by computing the distance between the vectors. SSD is useful for approximating the visual similarity of two equal-sized blocks of image content. It is a simple metric that computes the difference between the value of a pixel in the first image and the corresponding pixel in the second image, and then sums the square of these differences over all pixels in the block.

### Analysis procedure

Before comparing a derived photo with its source, we scale down both photos as necessary so that each fits inside a 1024x768 pixel bounding box. The maximum resolution for the Nexus One camera is 2592x1944; thus, source photos will be scaled down by a factor of at most 2.5. This is necessary due to the memory requirements of our analysis routines and the limited memory available on our target mobile device (512MB RAM for a Nexus One). As device RAM increases we will not need to scale images as drastically.

In addition, all analysis operates on grayscale versions of the photos, produced by taking a weighted average of the RGB channels. We believe that scaling down source photos to 1024x768 and converting to grayscale should not hinder our ability to recognize preserved regions and localize transformations that alter the source's meaning. If color modifications not apparent in the grayscale version are a concern, we can perform the same analysis on each of the three RGB channels at the expected expense of a 3x increase in runtime.

After resizing the input photos, the analysis proceeds in two phases. In the first phase, the analyzer attempts to find a correspondence between the derived photo and a region in the source photo. Note that in some cases the entire modified photo will map to a sub-region of the source photo due to cropping. To find this mapping, we use SURF to extract descriptors from each photo and then compute the set of matching descriptors. We then find the minimum-sized rectangular region (with sides parallel to the coordinate axes) in the source photo containing the keypoints of all matching features—this is the region that the analyzer considers for further investigation. At this point, we scale down the larger of the matching region and the derived photo to make their sizes equal. If no mapping is found between the content of the modified photo and the source photo, we continue analysis assuming that the entire source maps to the derived photo.

After identifying the matching region of the source photo the analyzer subdivides both the derived photo and the matching region into equal-sized, approximately-square blocks, with eight blocks along the longer dimension, subject to a minimum block size of 32x32 pixels. This heuristic is intended to localize content-altering modifications with sufficient precision, while ensuring a reasonable number of pixels for computing SSD. Once the images have been divided into blocks, we compute the SSD of corresponding blocks.

A caveat of SSD's pixel-by-pixel comparison is that it is highly sensitive to alignments of image regions off by even a single pixel. To account for potentially imprecise alignments found using SURF, for each block we take a sub-image from the center 12% smaller in each dimension and compare it to each equal-sized sub-image in the corresponding block from the other photo. For example, if the block size is 128x128 pixels, we take a 112x112-pixel sub-image from one block and compare it with each of the 256 possible equal-sized sub-images from the other block. We record the minimum of the computed SSD values for each pair of blocks.

To account for different block sizes, we define a block size-independent metric, *per-pixel sum of squared differences* (PPSSD), as the SSD value for a block divided by its area in pixels. In Section 7, we report PPSSD values resulting from various modifications and show that it is possible to define a threshold on PPSSD values which segregates blocks of a photo preserving meaning from the source from those containing content-altering local modifications.

## 5.2 Audio content

Similar to our photo analysis approach, the goal of YouProve's audio analysis is to identify contiguous time segments of audio which were modified only in ways that do not alter the way the audio will be perceived by a listener. Transformations that typically preserve the meaning of audio content include compression, slight changes to volume, and "enhancing" filters such as normalize. Other manipulations such as time distortion and splicing in audio from other sources are considered to alter the meaning of the audio clip.

At a high level, YouProve's audio analyzer extracts sequences of *spectral peak frequencies* from the source and derived audio clips and applies *local sequence alignment* to find matching time segments. The use of spectral peak analysis to compare audio data was inspired by the Shazam audio recognition system [35]. The technique is well-suited for our analysis because spectral peaks are a central feature in human hearing and thus are independent of audio encoding. They are largely maintained across transformations that preserve the way audio will be perceived (e.g., compression). To identify time segments in a modified clip which preserve sequences of spectral peak frequencies from a source clip, we use a modified version of the well-known Smith-Waterman algorithm [33] for local sequence alignment.

We note that the naive approach of simply performing sequence alignment on audio samples is unsuitable for two reasons. First, sample values are completely changed by even simple transformations such as normalization, which adjusts volume and therefore the value of all samples. Second, the sequence alignment algorithm runs in time proportional to the product of the sequence lengths. Because there are generally hundreds to thousands of samples in a 0.1-second segment of audio (the interval over which we compute each spectral peak frequency value), the naive approach would be slower by a factor of $10^4$ to $10^6$ for sequence alignment.

### Analysis procedure

Our analysis begins by verifying that the two input audio clips have the same sampling rate and audio format to ensure an equal number of samples for a given time duration. The `audioflinger` service, which supplies the full-fidelity source clip to YouProve's logging service, outputs raw PCM samples in WAV format. If the modified clip is stored in a compressed format, we first decompress it to raw PCM data. At this point, we proceed with analysis only if the sampling rates of the source and derived clips are equal. In practice, the sampling rate of a modified clip rarely differs from that of the source clip, because the Android audio recording API allows applications to request a particular sampling rate from the `audioflinger` service. YouProve's logging service captures the source clip at this requested sampling rate, and we assume that applications will not change the sampling rate of a clip once it has been captured.

After verifying the aforementioned assumptions, we continue by computing the frequencies of spectral peaks for each audio clip as follows. First, we compute $N$, the number of samples in 0.1 seconds of audio. For each region of $N$ samples in the clip, we perform a windowed Fourier transform to convert the samples into a collection of frequency bins and their amplitudes. We then find the frequency bin with the greatest amplitude and output that frequency. The result is a sequence of frequencies for each clip.

To facilitate finding multiple matching time segments, we modify Smith-Waterman to make the gap penalty infinite and then examine local alignments with scores above a certain threshold (which we set), rather than simply the max scoring (optimal) local alignment. To deal with limited floating-point precision and to allow for some variation due to compression, we consider two frequencies a match for scoring purposes if the frequencies are within 10 Hz of each other. This value was chosen because at a 44.1 kHz sampling rate, adjacent frequency bands roughly differ by 10 Hz.

Each local alignment maps a specific range of audio samples (technically 0.1-second length clusters of samples) in one audio clip to a specific range of samples in the second. We order these alignments by descending score, and add them, in order, to our output set of alignments as long as there is no overlap with pre-existing alignments in either of the sequences. For the final output, we convert ranges of samples to regions of time.

As long as the edits performed to produce a modified clip do not sub-divide the source clip at finer than 0.1-second granularity, we expect to be able to identify preserved segments from the source clip. Moreover, even if edits splice at finer than 0.1-second granularity, this should not cause us to falsely label time segments in the modified clip that do not actually preserve content from the source clip.

# 6 Implementation

In this section, we describe our prototype implementation of YouProve for Nexus One smartphones and Android 2.2.

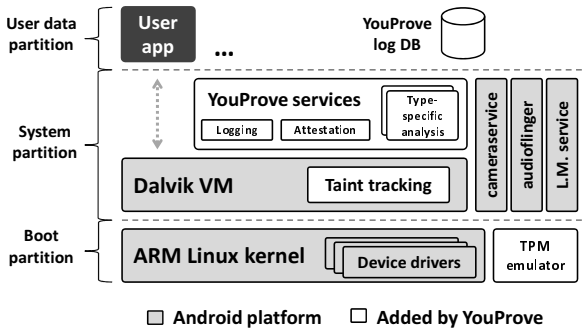


**Figure 6. YouProve prototype architecture.**

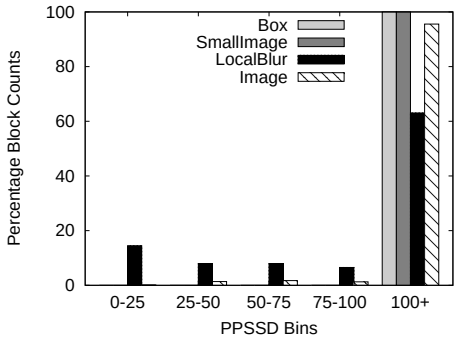


**Figure 7. Block PPSSD for different local modifications.**

YouProve's logging and attestation services are implemented in approximately 2,000 lines of Java code. Like all other commodity smartphones of which we are aware, the Nexus One does not include a TPM chip. Instead, we ported a popular open-source TPM emulator [1] to Android, along with the TrouSerS open-source TCG stack [2]. Our prototype photo and audio analyzers are written in C++ and C, respectively. The audio analyzer uses the open-source LibXtract and FFTW libraries for converting audio samples into frequency bins. We ported these libraries to the Android platform. The photo analyzer uses the SURF implementation from the popular open-source computer-vision library OpenCV. In addition, we use the open-source TaintDroid information-flow tracking framework [11], which we updated from Android 2.1 to 2.2.

As discussed in Section 2, YouProve builds upon the security model offered by the underlying Android platform. An important feature of this model is that any code that runs with elevated privileges must be loaded from the device's read-only firmware. Android partitions the device's internal flash storage into at least three partitions: the "boot" partition contains the Linux kernel and an initial ramdisk, the "system" partition includes all user-level Android platform code, and the "user data" partition stores the user's apps and data. The boot and system partitions comprise the read-only portions of the firmware. Effectively, all trusted code is loaded from these two partitions.

All trusted YouProve code is added to the read-only firmware. The arrangement of YouProve's software components and relevant Android platform components is shown in Figure 6. To enable an Android device to attest to the state of its TCB, YouProve modifies Android's boot procedure to measure the boot and system partitions when they are mounted. Because Android's bootloader does not provide support for measured boot, our prototype actually measures both partitions after the boot partition is loaded by the bootloader and control has passed from the bootloader, to the kernel, and finally to the first user task, `init`. These measurements are extended into one of the TPM's PCRs to support subsequent attestation using TPM quotes.

The TPM quote included in a fidelity certificate attests only to the value of the bits of the firmware booted by the device. The trustworthiness of fidelity certificate contents depends on whether the booted firmware provides sufficient protection for the TCB.

# 7 Evaluation

In evaluating YouProve, we sought to answer two questions: (1) Can YouProve's type-specific analyzers accurately identify modified portions of a data item that preserve the meaning of its source? and (2) What are the performance and power costs of running YouProve?

## 7.1 Analyzer Accuracy

Current mobile sensing services have no way to verify the authenticity of the data they accept. YouProve's goal is to improve on this state by identifying regions of a derived data item that preserve the meaning of its source, while minimizing the number of incorrectly categorized regions.

### *7.1.1 Photo analysis*

As described in Section 5.1, YouProve's photo analysis compares two photos block-by-block and reports a per-pixel SSD (PPSSD) value for each block of the derived photo. The first goal of our evaluation was to determine whether paired-block PPSSD values provide a good metric for identifying blocks that preserve the meaning of their source. We also sought a guideline PPSSD threshold for categorizing blocks.

The basis for our test dataset was a diverse collection of sixty-nine photos taken on a college campus using a Nexus One. Subject matter included individual students, crowds, buildings, offices, landscapes, walls with flyers, and bookshelves full of books. The photos varied in level of detail, level of focus, and quality of lighting. All photos were taken at the default resolution of 1944x2592 pixels. We then used the ImageMagick image-editing tool to apply two classes of modifications to our photos.

*Global modifications* included cropping, scaling, and JPEG compression. We consider these modifications to preserve the meaning of the source. Cropping test cases were created by cropping out either the top, bottom, left, or right half of an image, leaving a rectangular half-image. Scaling maintained aspect ratio while reducing each dimension to either 75%, 50%, or 25% of its original size. Compression produced JPEGs at 75%, 50%, and 25% quality.

*Local modifications* included overlaying a black box, pasting in a small photo, pasting in a large photo, or blurring a region. We consider these modifications to alter the meaning of the source. Our goal was to determine whether our photo analyzer could identify blocks containing local modifications, possibly even if global modifications had also been applied. To generate a test case containing a local modifica-

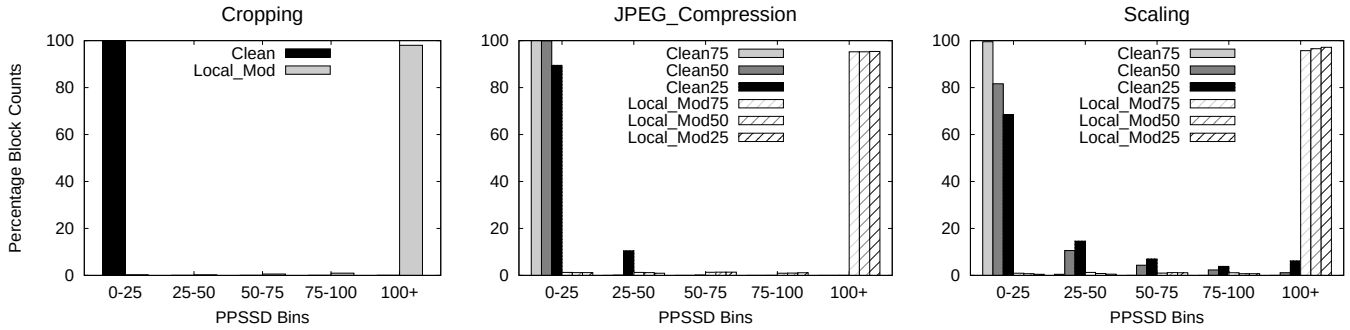


**Figure 8. Results of photo analysis accuracy experiments.**

tion, we applied one of the four types to a random, fixed location of the source. The black box, small photo, and blur regions were scaled with the dimensions of the photo to cover approximately 3% of the photo's area. The large photo covered approximately 30% of the area. Blocks that were untouched by a local modification were considered *clean*, even if they were transformed by a global modification.

As a basic test of the ability of YouProve's photo analysis to identify local modifications, we included in our first experiment only local modifications and no global modifications. We ran photo analysis on these photos and measured the PPSSD of locally-modified blocks and clean blocks.

This experiment showed a clear separation between the PPSSD values of locally modified blocks and clean blocks: all clean blocks exhibited PPSSD values of less than 25, indicating a high degree of similarity to the source. However, slightly more than 3% of locally modified blocks also had PPSSD values of less than 50. To understand the cause of these low PPSSD values, we separated the PPSSD distribution of each local modification, as shown in Figure 7. Note that the y-axis of Figure 7 shows the percentage of blocks that were modified in a particular way, not the percentage of all locally-modified blocks.

The histogram shows that blocks containing blurred regions often exhibit low PPSSD values: over 15% of blurred blocks exhibited a PPSSD value of less than 25 and only 60% of blurred blocks exhibited a PPSSD value of greater than 100. The reason for these numbers is that blurring a region of nearly solid color produces very little change, and we did not bias our choice of region to blur based on its level of detail. Manual inspection of blurred blocks with low PPSSD values indicated that many arose from blurring already blurry regions, or blurring solid colors such as white office walls and the blue sky. In other words, the appearance of the blurred region was not significantly changed in these cases.

Next, we evaluated the accuracy of YouProve's photo analysis in the presence of global modifications. The analyzer's ability to handle cropping depends on the accuracy of our SURF-based approach for mapping a derived photo to a region of the source. Thus, we applied local modifications to a set of cropped photos and then compared the PPSSD values for clean blocks and locally-modified blocks within a cropped region. Figure 8 shows that all of the clean blocks within our cropped regions registered a PPSSD of less than 25. Furthermore, less than .5% of locally-modified blocks

| **Global modification** | **Clean, PPSSD>50** | **L-Modified, PPSSD≤50** |
|---|---|---|
| None | 0.0% | 3.10% |
| Cropping | 0.0% | 0.43% |
| JPEG quality: 75% | 0.0% | 2.50% |
| JPEG quality: 50% | 0.0% | 2.37% |
| JPEG quality: 25% | 0.09% | 2.11% |
| Scaling, 75% | 0.0% | 2.17% |
| Scaling, 50% | 7.79% | 1.52% |
| Scaling, 25% | 16.94% | 0.99% |

**Table 1. Miscategorized blocks, PPSSD threshold = 50.**

exhibited PPSSDs of less than 50. This demonstrates that SURF is highly accurate in mapping a cropped photo to the corresponding region in the source.

It is interesting to note that the percentage of locally-modified blocks in cropped images with PPSSDs of less than 50 decreased by a factor of more than 7 compared to the case in which there were no global modifications. The reason for this is that cropping re-allocates blocks to the smaller area so that there are more blocks covering the same area of the image. For example, a face that was covered by one block before cropping might be covered by four blocks after cropping. In general, this will lead to a smaller proportion of "barely-changed" blocks, i.e., blocks with almost all preserved content and a small piece of locally-modified content. Because we only considered blocks that were completely untouched by local modifications to be clean, fewer blocks with only minor changes reduced the number of locally modified blocks with low PPSSD values.

Finally, we evaluated our photo analyzer's robustness to compression and scaling. To better understand how these transformations affect the PPSSD distribution of clean blocks, we investigated several degrees of scaling and compression. From Figure 7, we observe that the degree of compression has almost no impact on the PPSSD distributions. However, the PPSSD distribution for clean, scaled blocks shifts to the right as the degree of scaling increases. Luckily, the photo analyzer can include the scaling factor exhibited by a derived photo in its report so that services can increase their PPSSD threshold for more scaled-down images.

In light of our results, we believe that a threshold of 50 PPSSD is reasonable for identifying photo blocks that pre-

| | No additional compression | | | | | | | | Compress+decompress before applying modification | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | **Music** | | **Comedy** | | **Speech** | | **Lecture** | | **Music** | | **Comedy** | | **Speech** | | **Lecture** | |
| **Modification** | *correct* | *false* | *correct* | *false* | *correct* | *false* | *correct* | *false* | *correct* | *false* | *correct* | *false* | *correct* | *false* | *correct* | *false* |
| None | 1.0 | 0.0 | 1.0 | 0.0 | 0.999 | 0.0 | 1.0 | 0.0 | 1.0 | 0.0 | 1.0 | 0.0 | 0.999 | 0.0 | 1.0 | 0.0 |
| MP3, 128 kbit/s | 0.999 | 0.0 | 0.956 | 0.0 | 0.996 | 0.0 | 0.949 | 0.0 | 1.0 | 0.0 | 1.0 | 0.0 | 0.77 | 0.0 | 1.0 | 0.0 |
| MP3, 64 kbit/s | 0.999 | 0.0 | 0.887 | 0.0 | 0.998 | 0.0 | 0.949 | 0.0 | 1.0 | 0.0 | 1.0 | 0.0 | 0.77 | 0.0 | 1.0 | 0.0 |
| Dither | 1.0 | 0.0 | 1.0 | 0.0 | 0.999 | 0.0 | 1.0 | 0.0 | 0.999 | 0.0 | 0.956 | 0.0 | 0.996 | 0.0 | 0.949 | 0.0 |
| Double pad | 1.0 | 0.0 | 1.0 | 0.0 | 0.833 | 0.0 | 1.0 | 0.0 | 0.999 | 0.0 | 0.822 | 0.0 | 0.984 | 0.0 | 0.939 | 0.0 |
| Normalize | 1.0 | 0.0 | 1.0 | 0.0 | 0.999 | 0.0 | 1.0 | 0.0 | 0.999 | 0.0 | 0.956 | 0.0 | 0.996 | 0.0 | 0.949 | 0.0 |
| Replace w/ noise | - | 0.066 | - | 0.0 | - | 0.0 | - | 0.0 | - | 0.065 | - | 0.0 | - | 0.0 | - | 0.0 |
| Lower pitch | - | 0.073 | - | 0.0 | - | 0.0 | - | 0.017 | - | 0.057 | - | 0.0 | - | 0.0 | - | 0.02 |
| Raise pitch | - | 0.054 | - | 0.0 | - | 0.0 | - | 0.0 | - | 0.053 | - | 0.0 | - | 0.0 | - | 0.0 |
| Remove middle | 1.0 | 0.0 | 1.0 | 0.0 | 0.999 | 0.0 | 1.0 | 0.0 | 0.999 | 0.0 | 0.809 | 0.0 | 0.993 | 0.0 | 0.919 | 0.0 |
| Remove multiple | 1.0 | 0.0 | 1.0 | 0.0 | 0.988 | 0.0 | 1.0 | 0.0 | 0.999 | 0.0 | 0.753 | 0.0 | 0.992 | 0.0 | 0.9 | 0.0 |
| Segment splice | 1.0 | 0.014 | 1.0 | 0.0 | 0.998 | 0.0 | 1.0 | 0.0 | 0.999 | 0.013 | 0.762 | 0.0 | 0.984 | 0.0 | 0.903 | 0.0 |
| Crop | 1.0 | 0.0 | 1.0 | 0.0 | 0.998 | 0.0 | 1.0 | 0.0 | 0.98 | 0.0 | 0.58 | 0.0 | 0.784 | 0.0 | 0.982 | 0.0 |

*correct*: proportion of preserved regions correctly identified
*false*: proportion of modified regions incorrectly identified as preserved

**Table 2. Results of audio analysis accuracy experiments.**

serve the meaning of their source for photos that are not scaled down by more than 50%. The rates of miscategorization for a threshold of 50 PPSSD for all experiments are summarized in Table 1. For photos scaled down to 25%, a higher threshold of 100 PPSSD greatly improves our overall accuracy. Using this threshold, only 6.16% of clean blocks are miscategorized as modified, while 2.83% of locally modified blocks are miscategorized as clean.

We further observe that that the PPSSD accuracy loss we suffer in scaling test cases does not appear to afflict us during the scaling phase of our analysis, as demonstrated by the low SSD values of the clean blocks in the other experiments. We attribute this to the fact that our photo analyzer scales both images with the same algorithm using the OpenCV library, while ImageMagick uses a different scaling algorithm.

### *7.1.2 Audio analysis*

As described in Section 5.2, our audio analyzer compares two audio files, and reports time regions in one file that it determines to be derived from the other.

We aim to evaluate its accuracy at identifying regions derived through content-preserving modifications such as lossy compression, normalization, and dithering, while ignoring regions modified through content-altering effects such as pitch changes and spliced-in audio from other sources.

The test cases used to evaluate audio analysis were derived from 5-minute clips of an excerpt from Vivaldi's *The Four Seasons*, a stand-up comedy show, an excerpt from the iconic "I Have A Dream" speech, and an undergraduate lecture. Starting with these four clips, we used the Sound eXchange (SoX) tool to generate derivations from the following transformations: extracting and removing subclips, splicing in other audio, inserting silences, applying lossy (MP3) compression, dithering, normalizing, and pitch altering.

For each test case, we examined the output of our analysis and compared it with ground truth to compute a proportion of actually-preserved regions that were correctly identified. We refer to these as *correct*. We also computed the proportion of modified regions that we incorrectly identified as preserved, which we label as *false*. The results are summarized in the left half of Table 2. A '-' is displayed in place of a correct percentage in those tests where no region of the audio clip was actually preserved from the original.

For variations of subclip splicing, our approach correctly identified all preserved regions of audio in all cases, demonstrating that it is as robust as naive matching on binary-encoded raw PCM samples. For dithering and normalization, our approach also achieved close to perfect results.

For MP3 compression, our analysis is fairly robust for those regions of audio where volume is not so high that many samples are clipped during transformation. In both the comedy show and the speech recordings, there were samples in which the volume was sufficiently high that clipping could not be avoided during compression and decompression. On these tests our success rate was lower, but our failure rate did not increase. However, for the music and the lecture clips which did not have such irregularities, our analysis proved very robust to compression.

We expect that a common use case for audio editing will be for an application to both edit and compress a file, so we also evaluated our approach against a combination of compression and the other transformations. We created test cases by compressing the original audio clips to MP3 at 128 kbps, decompressing back to WAV, and then applying the other transformations. Analysis results are summarized in the right half of Table 2. As previously discussed, we are primarily concerned with keeping the false rate low. We note that for almost all tests, the false rate is close to zero while maintaining a high success rate.

Moreover, for the entire set of results, when we manually examined the regions which were falsely recognized as preserved, we observed that they are almost all moments of silence. We expect the data consumer to be able to recognize this when she listens to the actual derived audio.

| Data item | Latency in sec (stddev) |
|---|---|
| JPEG, 1296x972 | 28.0 (0.12) |
| JPEG, 2592x1944 | 28.9 (0.34) |
| MP3, 30 sec | 20.2 (0.12) |
| MP3, 60 sec | 23.9 (0.59) |
| MP3, 5 min | 64.1 (2.25) |

**Table 3. Latency of generating fidelity certificates.**

## 7.2 Performance evaluation

Our primary goal in evaluating YouProve's performance was to measure the latency and power overheads of generating fidelity certificates. We expect the major contributor to be the computationally-expensive content analysis routines. We used our Nexus One YouProve prototype to produce fidelity certificates for a variety of data items, while measuring power using a Monsoon Power Monitor. Latencies are reported in Table 3. For photos ranging in size from 1296x972 to 2592x1944, it took just under 30 seconds to generate a fidelity certificate. For audio clips, latency ranged from about 20 seconds for a 30-second modified clip to about 64 seconds for a 5-minute clip. In all cases, the modified clip was encoded as MP3 and was compared against a 5-minute original clip. All reported data is an average over ten trials.

The average power consumption while generating fidelity certificates for both photo and audio content was relatively constant, between 1000 mW and 1100 mW for all trials. For comparison, we measured the power consumption of common tasks such as playing music ($\sim$600 mW) and recording video ($\sim$1800 mW). All measurements were performed with the screen dimmed but not powered off.

The other question we sought to answer is whether any overheads imposed by YouProve negatively affect the user experience of using a mobile phone to gather sensor data. Specifically, does YouProve's logging increase the perceived latency of snapping a photo or recording an audio clip?

We found YouProve's impact on user experience to be minimal, presumably due to the asynchronous interfaces provided by Android's sensor APIs. Anecdotally, we did not perceive an increase to the delay of restoring the viewfinder after snapping a photo with the Camera app. While Android does not package a standard audio recording app, we did not notice any slowdown for the handful of recording apps we tested. The time to boot, or the delay from pressing the power button until the user interface becomes available, increased from 26 seconds with Nexus One stock firmware to 90 seconds for the YouProve prototype, mostly due to the SHA-1 digest computed over the read-only firmware.

We feel that these small and infrequent overheads are well worth YouProve's added authenticity guarantees.

# 8 Related work

Fidelity has traditionally been studied in the context of mobile clients retrieving data from servers over a wireless network [12, 19, 27]. In these settings, a small set of trusted servers maintain canonical copies of all source data and can generate reduced-fidelity versions at the request of a client. In a mobile sensing service, the roles of clients and servers are reversed: clients use sensors to generate original data and may reduce its fidelity locally before sending it to a server which operates on the data to implement the service logic.

Several groups have identified data authenticity as a critical problem in mobile sensing. Dua et al. [10] proposed pairing a mobile device with a trustworthy sensing peripheral that can attest to its software configuration and sign its sensor readings. This approach only applies to raw sensor readings and cannot ensure the authenticity of locally modified data. Other groups have proposed deploying trustworthy signing infrastructure that can accept sensor readings from nearby devices and provide signed timestamps and location coordinates [21, 30]. However, this approach can only prove that a data item existed at a particular time and place.

Two recent position papers made the case for trustworthy sensing on mobile devices [14, 31]. Saroiu et al. [31] describe two architectures for signing sensor readings from a device. One is similar to Dua's approach[10] and embeds signing hardware in the sensors themselves so that services do not need to make any trust assumptions about a device's software. The other proposed approach utilizes a TPM and virtual machines (VMs) to minimize the TCB by including sensor drivers in the hypervisor and placing all other functionality in untrusted VMs. Not-a-bot [16] used a similar architecture to minimize the TCB and certify when outgoing network messages were temporally correlated with keyboard activity. Both approaches provide a smaller attack surface than YouProve, but neither provides authenticity guarantees for modified data, which is our primary goal.

Gilbert et al. [14] also proposed an architecture that utilizes VMs and a TPM. In this approach an application that is trusted to modify sensor data is encapsulated within a VM along with any relevant device drivers. As we discussed in Section 2.3, the main disadvantage of this approach is that it limits choice by forcing users to modify their sensor data with applications that a service deems trustworthy.

There have been several recent projects aimed at designing new trustworthy software architectures leveraging a TPM [22, 25, 32]. Nauman et al. [25] describe how a TPM can be leveraged to attest to the individual java classes that Android's Dalvik virtual machine loads. Nexus [32] is a micro-kernel OS for TPM-enabled machines that provides a general framework for generating and reasoning about certifiable statements. YouProve could be implemented on top of Nexus. Flicker [22] can attest to having executed a program, including its inputs and outputs, even if the machine's BIOS and OS have become compromised. It does this by relying on a small TCB of hundreds of lines of code and the late launch feature of a TPM. It may be possible to leverage architectures such as Flicker to provide stronger integrity guarantees for the sensor log and fidelity certificates.

CertiPics [32] is a trusted image editor for Nexus structured as a pipeline of small, stand-alone programs implementing a single operation such as cropping or resizing. To capture how an image was modified, each program invokes the Nexus kernel to generate a signed statement describing the hash of the program and a semantically-rich description of the operation it performed. As mentioned in Section 2.3, this approach either limits users' choice or places an unmanageable trust-management burden on services.

## 9 Conclusion

This paper has presented the design and implementation of YouProve, a system that enables mobile sensing services to verify that contributed data has not been manipulated in a way that alters its original meaning while allowing clients to use untrusted editing applications to directly control the fidelity of data they upload. Verifying that contributed data preserves the meaning of original sensor readings is a key requirement for ensuring data authenticity in domains such as citizen journalism. The key to YouProve's approach is providing analytic bases of trust for remote services. YouProve relies on trusted, content type-specific analyzers running on the mobile device to generate reports summarizing differences between a derived data item and an original sensor reading. Results of experiments with a prototype are promising. Logging source data does not noticeably affect application responsiveness, and our content analyzers are accurate and complete their tasks in under 70 seconds for 5-minute audio clips and under 30 seconds for 5-megapixel photos.

## Acknowledgements

We would like to thank our shepherd, Sivan Toledo, and the anonymous reviewers for their helpful comments. We would also like to thank Stefan Saroiu, Gün Sirer, Fred Schneider, and Ben Zhao for their insightful feedback on early drafts of this paper. Finally, our work was supported by Intel Research and the National Science Foundation under NSF awards CNS-0747283, CNS-0916649, and CNS-1018547.

## 10 References

[1] TPM Emulator. `http://tpm-emulator.berlios.de/`.

[2] TrouSerS. `http://trousers.sourceforge.net/`.

[3] Network Time Protocol (Version 3) Specification, Implementation and Analysis. IETF, March 1992.

[4] ARM Security Technology Building a Secure System using TrustZone Technology. ARM Technical Paper, 2009.

[5] Trusted Computing Group (TCG) Mobile Trusted Module Specification 1.0, version 7.02. TCG Published, April 2010.

[6] X. Bao, T. Narayan, A. A. Sani, W. Richter, R. R. Choudhury, L. Zhong, and M. Satyanarayanan. The Case for Context-Aware Compression. In *HotMobile*, 2011.

[7] H. Bay, A. Ess, T. Tuytelaars, and L. Van Gool. Speeded-Up Robust Features (SURF). *Comput. Vis. Image Underst.*, 110:346–359, June 2008.

[8] J. Burke, D. Estrin, M. Hansen, A. Parker, N. Rmanathan, S. Reddy, and M. B. Srivastava. Participatory Sensing. In *World-Sensor-Web*, 2006.

[9] J. R. Douceur. The Sybil Attack. In *IPTPS*, 2001.

[10] A. Dua, N. Bulusu, W. chang Feng, and W. Hu. Towards Trustworthy Participatory Sensing. In *HotSec*, 2009.

[11] W. Enck, P. Gilbert, B.-G. Chun, L. P. Cox, J. Jung, P. McDaniel, and A. N. Sheth. TaintDroid: An Information-Flow Tracking System for Realtime Privacy Monitoring on Smartphones. In *OSDI*, 2010.

[12] A. Fox, S. D. Gribble, E. A. Brewer, and E. Amir. Adapting to Network and Client Variability via On-Demand Dynamic Distillation. In *ASPLOS*, 1996.

[13] S. Gaonkar, J. Li, R. R. Choudhury, L. Cox, and A. Schmidt. Micro-Blog: Sharing and Querying Content Through Mobile Phones and Social Participation. In *MobiSys*, 2008.

[14] P. Gilbert, L. P. Cox, J. Jung, and D. Wetherall. Toward Trustworthy Mobile Sensing. In *HotMobile*, 2010.

[15] M. Gruteser and D. Grunwald. Anonymous Usage of Location-Based Services Through Spatial and Temporal Cloaking. In *MobiSys*, 2003.

[16] R. Gummadi, H. Balakrishnan, P. Maniatis, and S. Ratnasamy. Not-a-Bot: Improving Service Availability in the Face of Botnet Attacks. In *NSDI*, 2009.

[17] B. Hoh, M. Gruteser, R. Herring, J. Ban, D. Work, J.-C. Herrera, A. M. Bayen, M. Annavaram, and Q. Jacobson. Virtual Trip Lines for Distributed Privacy-Preserving Traffic Monitoring. In *MobiSys*, 2008.

[18] J. Kincaid. Foursquare Starts To Enforce The Rules, Cracks Down On Fake Check-Ins. TechCrunch, April 2010.

[19] E. D. Lara, D. S. Wallach, and W. Zwaenepoel. Puppeteer: Component-based Adaptation for Mobile Computing. In *USITS*, 2001.

[20] W. Laube. Sickening tsunami of faked photos. The Sydney Morning Herald, March 2011.

[21] V. Lenders, E. Koukoumidis, P. Zhang, and M. Martonosi. Location-based Trust for Mobile User-generated Content: Applications, Challenges and Implementations. In *HotMobile*, 2008.

[22] J. M. McCune, B. J. Parno, A. Perrig, M. K. Reiter, and H. Isozaki. Flicker: An Execution Infrastructure for TCB Minimization. In *EuroSys*, 2008.

[23] R. J. Moore. Instagram Now Adding 130,000 Users Per Week: An Analysis. TechCrunch, March 2011.

[24] M. Mun, S. Reddy, K. Shilton, N. Yau, J. Burke, D. Estrin, M. Hansen, E. Howard, R. West, and P. Boda. PEIR, the Personal Environmental Impact Report, as a Platform for Participatory Sensing Systems Research. In *MobiSys*, 2009.

[25] M. Nauman, S. Khan, X. Zhang, and J.-P. Seifert. Beyond Kernel-level Integrity Measurement: Enabling Remote Attestation for the Android Platform. In *TRUST*, 2010.

[26] E. M. Newton, L. Sweeney, and B. Malin. Preserving Privacy by De-Identifying Face Images. *IEEE Transactions on Knowledge and Data Engineering*, 17, February 2005.

[27] B. D. Noble, M. Satyanarayanan, D. Narayanan, J. E. Tilton, J. Flinn, and K. R. Walker. Agile Application-Aware Adaptation for Mobility. In *SOSP*, 1997.

[28] S. Odio. More Beautiful Photos. Official Facebook Blog, September 2010.

[29] N. Santos, H. Raj, S. Saroiu, and A. Wolman. Trusted Language Runtime (TLR): Enabling Trusted Applications on Smartphones. In *HotMobile*, 2011.

[30] S. Saroiu and A. Wolman. Enabling New Mobile Applications with Location Proofs. In *HotMobile*, 2009.

[31] S. Saroiu and A. Wolman. I am a Sensor, and I Approve This Message. In *HotMobile*, 2010.

[32] F. B. Schneider, K. Walsh, and E. G. Sirer. Nexus Authorization Logic (NAL): Design Rationale and Applications. *ACM Transactions on Information and System Security*, 14(1), May 2011.

[33] T. F. Smith and M. S. Waterman. Identification of common molecular subsequences. *Journal of molecular biology*, 147(1):195–197, Mar. 1981.

[34] S. J. Snyder. Gotham Tornado: Amazing Photo of Twister Passing Statue of Liberty. Time Newsfeed, September 2010.

[35] A. Wang. An Industrial Strength Audio Search Algorithm. In *ISMIR*, 2003.

[36] J. Winter. Trusted Computing Building Blocks for Embedded Linux-based ARM TrustZone Platforms. In *STC Workshop*, Oct. 2008.

[37] X. Zhang, O. Aciiçmez, and J.-P. Seifert. Building Efficient Integrity Measurement and Attestation for Mobile Phone Platforms. In *MobiSec*, June 2009.