Brandeis Mobile CRT

App Policy Specification for Privacy-Enhanced Android
Draft, October 2 2017

Introduction

One of the goals of the PEARLS[1] project is to make it possible for app developers to specify policies as to what sensitive data their smartphone apps will use and why, along with any constraints about how that data will be used. There are four major uses cases for these **app policies**:

- These policies can be used to check an app's behaviors when uploaded to an app store, using a combination of static, dynamic, or crowd analysis.
- These policies can be used to check and/or enforce an app's behaviors on one's smartphone, at install time, at run time, or at configure time.
- These policies can be used to fill out user interface templates that can explain to people what behaviors an app will have with respect to personal data.
- These policies can be used to check and/or enforce how data is used on other computers once it has egressed from one's smartphone

Note that there are also several different kinds of policies within PEARLS, described below. Figure 1 shows how these different policies interact with one another.

[1] PEARLS includes Privacy-Enhanced (PE) Android, which will implement these capabilities.

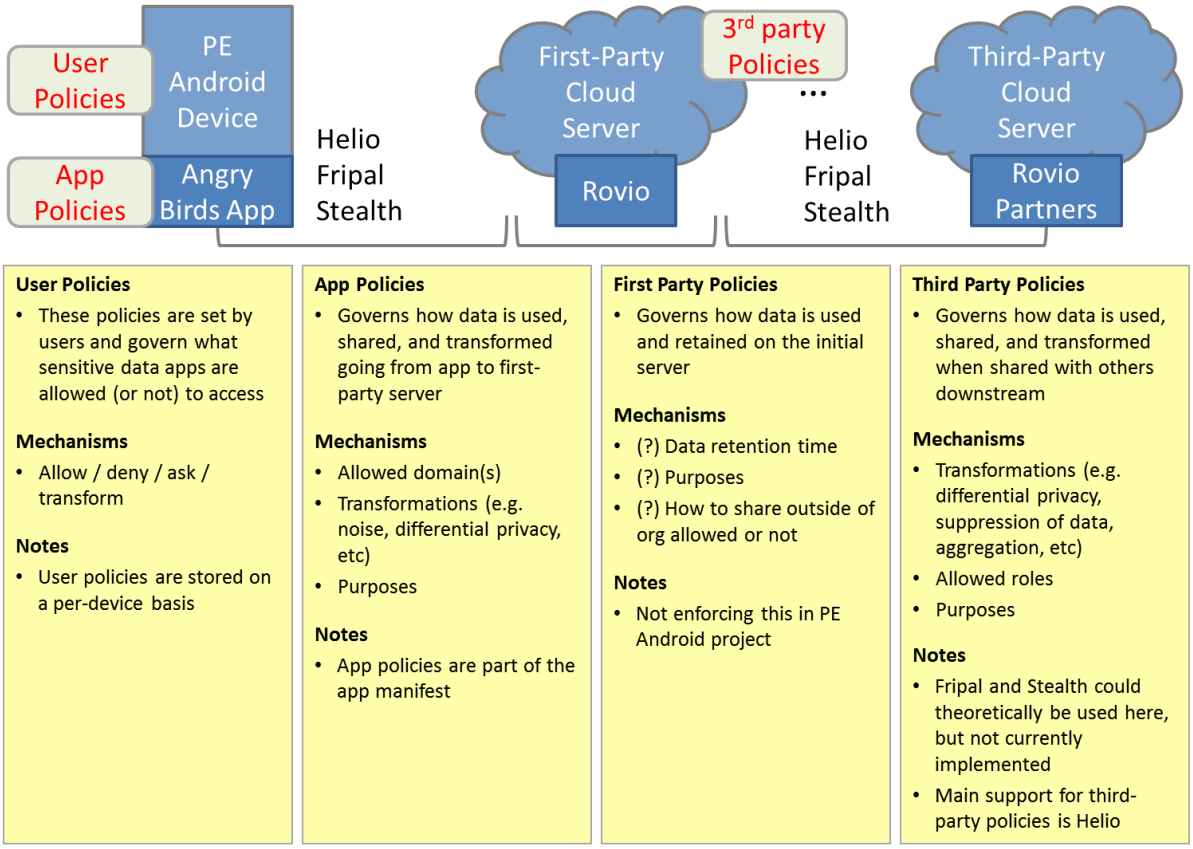

Figure 1. This figure shows the four kinds of policies and their relation to one another.

The four kinds of policies include:

- **User policies** are specified by end-users and reside on one's smartphone. For example, a user policy might allow or deny an app or a set of apps access to one's location data or contact list.
- **App policies** are specified by developers and are expected to be associated with a specific app via the app's manifest file. For example, an app policy might say that an app will use location data to infer speed and will only be used for insurance purposes.
- **First-party policies** are ones that govern how the data will be used on the first party receiving the data from the app (e.g., the app developers).
- **Third-party policies** are ones that the app / service uses when sharing data with third parties. For example, a third-party policy might govern what data fields can and cannot be shared with business partners that lie outside of the organization's boundaries.

Note that there are several caveats and limitations about our current work on policies, which we discuss at the end of this document.

End-to-End vs Local Protection of Data

Note that there are two major use cases we are exploring for protecting sensitive data. The first use case is constraints that can be enforced on the device itself (PE Android device to first-party cloud server). The second use case is the data going from first-party to third-party servers.

With respect to full end-to-end protection of data from PE Android devices to first-party to third-party servers, our current thinking is that while this is possible (in particular through TA1 Helio), it would be difficult to make it work in practice for Phase 2 for several reasons. However, we will examine this issue in more depth in Phase 3. The issues are:

- First, there is a strong mismatch in use cases. These app policies focus on *data collection* behaviors of apps, whereas Helio focuses on *data processing*. Helio does have support for transforming data before it leaves a secure enclave, but much of the other functionality that it offers would not be used.
- Second, there is a mismatch in data models. These app policies focus on a few tuples from a single user, whereas Helio was designed for batch processing of lots of tuples from multiple users. For example, Amazon Echo sends the entire audio to their cloud servers for processing, and then returns a response. Request-response types of interactions are very different from batch processing.
- Third, Helio's approach to policies currently exposes the underlying data model, eliminating any possibility for abstraction or indirection. In the database scenario that Helio was intended for, this makes sense. In an app scenario, this makes less sense b/c it also restricts many kinds of abstraction and indirection, and may make evolving an app and adding new functionality to an app harder. It's like two-tier vs three-tier architectures.
- Fourth, specifying how data is processed in the backend would severely limit the agility of organizations, as the source of data (in this case, smartphone apps) would have to be modified before data pipelines can be updated. For example, suppose Uber wanted to analyze existing data for fraud detection, and came up with new heuristics to use. This would require updating the app every time there was a new heuristic or data processing pipeline, which does not scale well. Also, given the lag in updating apps, this could lead to a lot of inconsistency in policies.
- Fifth, having full end-to-end policies would also expose who an organization's partners are, what kinds of data they are using, and how they are using it. While this would be good for privacy, most companies would reject this as it may expose too many secrets about an organization's activities, e.g. how they do anti-fraud detection, new kinds of customer analytics, and so on.

Again, instead of full end-to-end protection of data, the approach we are taking is to split it into two halves, from PE Android device to first-party cloud service (and backed by an app policy), and backend processing from first-party to third-party servers (Helio-enabled computers, backed by third-party policies).

All the TA1s (Helio, Stealth, Fripal) can be used for both data collection and for third-party processing. In Stealth and Fripal's cases, the processing is simpler, which is that data will be modified before egressing from the PE Android device to first-party cloud server, or from first-party cloud server to third-party. For Helio, the protection is also split. In the former case (PE Android to first-party cloud server), not all of Helio functionality is used. However, the encryption part of Helio will still be used, so that first-parties cannot use the data unless it is part of the Helio ecosystem and so any associated policies can be enforced.

Design Goals for App Policies

There are several competing goals for app policies. Ideally, the app policies should be:

- Easy for developers to understand and specify
- Flexible enough to cover a wide range of useful scenarios
- Easy for end-users to understand, at install time, runtime, and/or configure time
- Easy to implement in PE Android (relatively speaking)
- Easy to check, enforce, and/or audit a specified app policy against an app's actual privacy-related behaviors, by app stores, by smartphones, and by remote computers

.

Example App Policy

Here is an example policy based on a working session from the entire Mobile CRT in January 2017. Note that this is a draft and subject to change. Also, note that this is not necessarily the final syntax, the example below is formatted to make it easy to read.

The scenario here is an app by Liberty Mutual wanting to use the smartphone's sensor data to determine one's current driving speed for insurance purposes, where good drivers get discounts and no negative consequences for speeding.

```
Uses        ID.WiFiMACAddress -> "U_AppID"    # comments go here
            SENSOR.location, SENSOR.accelerometer -> "U_Speed"
With        "U_AppID" : ID.AppSpecificID
            "U_Speed" : "How fast you are traveling"
Where       "U_Speed" -> fuzz(uniform, 5)     # the U_ is naming convention
            destination -> *.libertymutual.com
For         UrbanAnalytics: "Offer dynamic pricing based on driving speed"
In          LibertyMutualMonitorActivity
Comments    We will only use your speed data for insurance purposes.
```

Clause	Description
Uses	A list of sensitive data that the app uses. Each data element consists of what raw data the app uses and the name of the key used in network traffic (assumes HTTP and key-value pairs in payload)
With	More information about the keys. For identifiers, it describes the properties of that identifier (see Table 2). For all other keys, it is a string of descriptive text.
Where	Any constraints on the data before it leaves the device, e.g. transformations
For	The purpose of use (based on an ontology) as well as a human readable description. See Appendix A for initial version of ontology.
In	The Android Activity or Service that this policy is associated with. Note, the app name is implicit since the policy is expected to be in the app's manifest file.
Comments	Any additional comments about how the data will be used

Table 1. Summary of the clauses in PE Android app policies.

An app can have multiple policies, with each policy describing what sensitive data is used, how, and why. A policy can be broken into several clauses, e.g., Uses, With, Where, etc. Each clause can have comments, with everything after the # ignored. Comments are meant for the app developers and will not be shown to end-users in any form. To simplify things, for now we are not allowing **Except** clauses for exceptions to a policy.

Uses Clause

The **Uses** field specifies a list of what sensitive data might be used by the app. Each data element of the list describes the raw data being used and a variable name for that data (an arbitrary string). This variable name might be the raw data itself or an inference using the raw data.

In the example above, the raw data are `ID.WiFiMACAddress, SENSOR.location,` and `SENSOR.accelerometer`, with the latter two being used to infer data element `U_Speed` on the smartphone. Note that `U_Speed` is an arbitrary string and has no semantics, but ideally should be a useful human-understandable name. It is also expected that this string will be the same name as any keys in key-value pairs used in HTTP payloads when an app communicates with a remote server, thus making it easier to analyze network traffic and tracking Personally Identifiable Information (PII) flows end to end. It is also expected that all of the possible fields in an HTTP payload will be listed in the corresponding policy. The `U_` in `U_Speed` also has no special semantics, it's just a naming convention that we are using here in this document to make it easy to spot HTTP payload variables.

We have organized the possible raw data into four categories: ID, PERSONAL, PHONE, and SENSOR. See Table 2 for more details about these categories as well as specific instances of these categories.

Category	Description	Instances
ID	Identifiers used to track the phone and/or user	WiFiMACAddress, BluetoothMACAddress, IMEI, AndroidID, AndroidAdvertisingID, AppInstallTime, UUID, Username, EmailAddress, Random
PERSONAL	Sensitive data related to the user of the phone	Contacts, Calendar, Call Log, SMS Log, Storage, Web history, Photos, Check-ins, Apps installed, Accounts, Tasks
PHONE	Sensitive data related to the phone itself. These tend to be discrete or string values.	Battery, DeviceModel, ScreenSize, Network, CallState, ScreenLight, DisplayStatus, PluggedIn, Notifications, Processes
SENSOR	Sensitive data related to sensed data. These	Camera, Location, Microphone, Accelerometer, Gyroscope, Magnetometer, Proximity

	tend to be continuous values.	

Table 2. Categories and instances of sensitive data that apps can acquire.

Note that the app policy specifies what identifiers are used to track the phone and/or user. Later on, in the With clause, the app policy specifies what properties that identifier has.

Note that we don't require the policy to specify how the raw data is processed and transformed. Instead, we only require the raw input and the name of the data element before it leaves the smartphone. We considered having this kind of pipeline be part of the policy, but felt that it would add a lot of complexity.

Here are examples of Uses clauses:

- `Uses ID.AppInstallTime, ID.Random -> "U_ID"`
  `(uses these two ID values to generate variable U_ID)`
- `Uses SENSOR.location -> "U_Loc"`
  `(uses location data to generate variable U_Loc)`
- `Uses PERSONAL.Contacts -> "U_NumContacts"`
  `(uses contact list to generate variable U_NumContacts)`
- `Uses PERSONAL.Contacts, PERSONAL.CallLog -> "U_ClosenessCalls"`
  `(uses these contact list and call log to generate U_ClosenessCalls)`

With Clause

The **With** clause describes each list element and is meant as a comment rather than being machine checkable. The With clause describes either properties of the data element if it is an identifier, or is an arbitrary text string that can convey to users (and auditors) what the data element is. These are primarily meant to be comments that can be fed directly into user interfaces. The minor exception case here is for identifiers, where the With clause describes the type of data, since there are only three possible choices. These three options can then be used to generate text for user interfaces. Currently, there aren't other types used, though this issue should be revisited in the future. Examples of future types might include location, time, phone number, social security number, etc.

For identifiers, there are three possible choices of properties: `ID.AppSpecific`, `ID.PhoneSpecific`, and `ID.Advertising`. Table 3 offers some more details on these IDs as well as some of their tradeoffs.

An `AppSpecific` ID is one scoped only for that app or collection of apps. The `AppSpecific` ID might be automatically generated by the app (e.g., a UUID), or might require user input (e.g., a username and password). `AppSpecific` IDs can typically be reset by re-installing the app or by logging out of the app or account.

A `PhoneSpecific` ID is one that directly uses identifiers in the hardware of the smartphone. Note that an app might use hardware identifiers to generate an `AppSpecific` ID. If the hardware identifiers cannot be readily extracted, we would consider this an `AppSpecific` ID. In contrast, if they can be easily reverse engineered or extracted, we would consider this a `PhoneSpecific`

ID. Note that most `PhoneSpecific` IDs can be reset via the Factory Data Reset (FDR) functionality, though some are also persistent even across FDRs. Generally, `PhoneSpecific` IDs should be avoided due to privacy concerns. A hope here is that requiring developers to specify this might discourage them to use these kinds of identifiers.

An `Advertising` ID is one that uses an advertising identifier, either Google's or Apple's Advertising ID. These `Advertising` IDs are similar to web cookies and can be reset any time.

Category	**Example unique identifiers**	**Scope**	**Resettability**	**Example use cases**
AppSpecific ID	Google Instance ID, globally unique ID (GUID), user-generated account, email address	App level	Install reset	Generating signed-out user analytics; tracking signed-out user preferences
PhoneSpecific ID	Android ID, Wi-Fi/Bluetooth MAC address, International Mobile Equipment Identity (IMEI)	Device level	FDR-reset / FDR-persistent	Managing telephony and carrier functionality; detecting high value stolen credentials
Advertising ID	Google Advertising ID	Device level	User resettable	Building user profiles for interest-based ads

Table 3. This table shows different kinds of identifiers for Android devices and their properties.

Note that in the running example, the policy states:

```
Uses        ID.WiFiMACAddress -> "U_AppID"
With        "U_AppID" : ID.AppSpecificID
```

At first glance, this may seem to be incorrect, in that `ID.WiFiMACAddress` (which is a hardware-specific identifier) is being used for `U_AppID`, but the declared property of `U_AppID` is `ID.AppSpecificID` rather than `ID.PhoneSpecificID`. In this specific case, the app is using the WiFi MAC address in some way to create the identifier but not in a way that makes the MAC address easily available. Again, the app policy only specifies inputs and outputs, but not the specifics of how the identifier is generated. This issue of how the identifier is generated is something we may want to revisit in the future.

Here are some more examples of the With clause:

- `With "U_ID" : ID.AppSpecificID`
  `(States that U_ID is an identifier and the kind of identifier)`
- `With "U_Loc" : "Your current GPS location"`
  `(Gives a human-readable text description of U_Loc)`
- `With "U_NumContacts" : "Total number of contacts in contact list"`
  `(Gives a human-readable text description of U_NumContacts)`

- `With “U_ClosenessCalls” : “Estimate of closeness based on calls”`
  `(Gives a human-readable text description of U_ClosenessCalls)`

Where Clause

The **Where** field describes constraints on the elements before they leave the device. In the running example, it specifies that `U_Speed` will be fuzzed before leaving the device. For the first version of app policies, we are aiming to support a small but useful set of constraints, as specified in Table 4.

Constraint	Description	Examples
Transformation	How is the data modified before leaving the device?	Fuzz(), FunctionSecretSharing(), DifferentialPrivacy(), Blur(), etc
Destination	What domain(s) will the data be sent to?	*.libertymutual.com

Table 4. Example Where constraints for app policies.

For purposes of version 1, we are expecting at most one kind of transformation constraint before leaving the device.

For destination, we are expecting a list of domains that the specified data might be sent to.

Note that the Where field is positioned to be under the Uses field to make it easier for developers to see how raw data is used, transformed, and constrained. For example, the policy above can be read as:

```
SENSOR.location, SENSOR.accelerometer -> Speed -> Fuzz()
```

This issue of Where constraints is something we may want to revisit in future iterations. In particular, it might be useful to make Destination required, and also to specify TLS / SSL or not.

Here are some more examples of Where clauses:

- `Where “U_Loc” -> Location.asCity()`
  `(States U_Loc will be transformed to city granularity before egress)`
- `Where “U_Location” : DifferentialPrivacy(epsilon=0.1, delta=0.00001,...)`
  `(States that U_Location will be transformed before egress)`
- `Where Destination -> *.millenialmedia.com`
  `(States that the destination for data will be millenialmedia.com)`
- `Where Destination -> localhost`
  `(States that the destination for data will be localhost, ie data will only be used locally on the device)`

For Clause

The **For** field describes the purpose of use. Part of this field is based on an ontology of purposes. See Appendix A for an initial version of this ontology. The other part of this field is a human readable string that offers more details and can also be fed directly into a user interface.

Here are example For clauses:

- `For Health: “Used to sense if you are sleeping or not”`
- `For Advertising: “Used to personalize ads shown to you”`
- `For Geotagging: “Lets us track what words people are looking up where”`

Note that the exact syntax for specifying purposes is still being defined and may change.

In Clause

The **In** field describes the current execution context. For Android, this means what Activity (running in foreground) or Service (running in background) is using this data. This clause lets us separate out different uses of the same data for different purposes. Currently we are assuming one purpose per activity for our initial implementations, though we expect to expand our capabilities in future versions.

Here are example In clauses:

- `In IrateBirdsAdvertisingActivity`
- `In SleepMonitorActivity`
- `In VegetableNinjaLocationUpdateService`

Comments Clause

The **Comments** field lets developers put in other short text about how sensitive data will be used. Content in this field will not normally be shown, but can optionally be shown in PE Android’s privacy user interfaces if the user wants to see more details. To prevent this field from becoming a lengthy Terms and Conditions, we will have a cap on its length (Length Limit TBD).

Here are some example Comments clauses:

- `Comments We will only use collected data for advertising within the Irate Birds app.`
  `(Arbitrary text string describing the app’s usage of sensitive data)`
- `Comments Location data used to direct you to aid stations.`
  `(Arbitrary text string describing the app’s usage of sensitive data)`

More Example App Policies

In this section, we will go through more example app policies.

Irate Birds - A hypothetical app that uses sensitive data internally for advertising. We assume it only gets location when showing ads, and only needs city granularity. Here, we assume that

there are standard data types (in this case, Location), and that there are common or well-known ways of transforming the data to the desired granularity or to have the desired properties.

```
Uses        ID.AppInstallTime, ID.Random -> “U_ID”
            SENSOR.location -> “U_Loc”
With        “U_ID” : ID.AppSpecificID
            “U_Loc” : “Your current GPS location”
Where       “U_Loc” -> Location.asCity()
            Destination -> *.millenialmedia.com
For         Advertising: “Used to personalize ads shown to you”
In          IrateBirdsAdvertisingActivity
Comments    We will only use collected data for advertising within
            the Irate Birds app.
```

Vegetable Ninja - A hypothetical app that uses sensitive data internally for advertising. We assume it periodically gets location in the background. We thought about adding what library it was from, but that seems harder for developers to specify reliably, and also has potential issues with obfuscated code. Of course, this also means that we assume that the names of the Activities and Services aren’t obfuscated.

```
Uses        ID.AppInstallTime, ID.Random -> “U_ID”
            SENSOR.location -> “U_Loc”
With        “U_ID” : ID.AppSpecificID
            “U_Loc” : “Your current GPS location”
Where       “U_Loc” -> Location.asCity()
            Destination -> *.millenialmedia.com
For         Advertising: “Used to personalize ads shown to you”
In          VegetableNinjaLocationUpdateService
```

Sleep Monitor - An app that uses microphone to get loudness, used only locally and no network.

```
Uses        SENSOR.microphone -> “Loudness”
With        “Loudness” : “How loud it currently is, sensed every 2 minutes”
Where       Destination -> localhost
For         Health: “Used to sense if you are sleeping or not”
In          SleepMonitorActivity
```

Dictionary.com - Existing app that uses location data to help tag what words you are looking up and to search for nearby words. Note that there are two different Activities with different purposes, leading to two different app policies within the same app.

```
Uses        ID.Random, ID.WiFiMACAddress -> “U_USERID”
            SENSOR.location -> “U_GEO”
```

```
With        “U_USERID” : ID.AppSpecificID
            “U_GEO” : “Your current GPS location”
Where       destination -> api.dictionary.com
For         Geotagging: “Track what words people are looking up where”
In          SearchDictionaryActivity
```

```
Uses        ID.Random, ID.WiFiMACAddress -> “U_USERID”
            SENSOR.location -> “U_GEO”
With        “U_USERID” : ID.AppSpecificID
            “U_GEO” : “Your current GPS location”
Where       destination -> api.dictionary.com
For         Nearby: “Lets us show what words people are looking up nearby”
In          SeeNearbyWordsActivity
```

Social Closeness - Hypothetical research app that estimates the tie strength of people in your contact list, based on SMS messages and call logs. Sends summaries of the data to a CMU server.

```
Uses        ID.username -> “U_ParticipantID”
            PERSONAL.Contacts -> “U_NumContacts”
            PERSONAL.Contacts, PERSONAL.CallLog -> “U_ClosenessCalls”
            PERSONAL.Contacts, PERSONAL.SMSLog -> “U_ClosenessSMS”
With        “U_ParticipantID” : ID.AppSpecificID
            “U_NumContacts” : “Total number of contacts in contact list”
            “U_ClosenessCalls” : “Estimate of closeness based on calls”
            “U_ClosenessSMS” : “Estimate of closeness based on SMS”
Where       destination -> “relationships.cs.cmu.edu”
For         Social: “Modeling how close you feel to others, for research”
In          CMUClosenessMonitorService
Comments    This data collection is approved by our university’s IRB.
```

RapidGather

```
TBD
```

HelpMe

```
Uses        ID.Random -> “U_ID”
            SENSOR.location -> “U_Location”
With        “U_ID” : ID.AppSpecificID
            “U_Location” : “Current location”
Where       “U_Location” : DifferentialPrivacy(epsilon=0.1, delta=0.00001,
```

```
                                                D=...)
            destination -> *protected* -- what do we put here?
For         FirstAid: “Directing you to a first aid station”
In          RapidGatherApp
Comments    Location data used to direct you to aid stations.
```

Issues:

1. How do we protect the data before the point of reporting?
2. How can we compose epsilons from many policies? Especially when the notions of privacy are different - can we even compose these?
3. How do we communicate the policy to the user? Especially when complicated formal definitions like differential privacy are involved.
4. How does the user know that he or she can trust that the policy will be enforced? I.e. how can we give the user a chain of trust automatically - ideally the policy shouldn’t have to list this.

Fall Detection

```
Uses        SENSOR.gyroscope, SENSOR.accelerometer -> “U_ActivityFeatures”
With        “U_ActivityFeatures” : “Did the person fall?”
Where       “U_ActivityFeatures” : DR (DR_Method, No_ReducedDim)
            destination -> "HelpMe.com"
For         FirstAid/Emergency: “Fall Detection”
            Not "Person Identification" (Not sure about this)
In          FallingDetectionActivity:Line101
Comments    Increase accuracy of detecting falls while decrease accuracy of
            PID
```

Caveats and Limitations About Our Policies in this Document

This document focuses on on-device policies and enforcement and thus focuses on app policies. In later iterations we will add specifications for user policies.

First-party policies are currently out of scope.

We do not yet define where third-party policies reside (e.g. should they be part of app policies, or live elsewhere?).

Currently, app policies (and user policies) for PEARLS are linked to Android permissions (e.g., ACCESS_COARSE_LOCATION or READ_CONTACTS). The main strengths of this approach are that there is already a broad understanding of these permissions by the developer community, many tools for analyzing permissions, and existing support within the Android Open Source Project for checking permissions. The main weaknesses of this approach are (a) permissions can be hard for users to understand, (b) permissions might not match what developers want (e.g., I only really need loudness but have to get full microphone access to do so), (c) permissions are not complete in protecting sensitive data. As an example of the last

item, there is no Android permission for credit card or email address, or for any sensitive data that devices connected to one's smartphone might send out.

Currently, we focus on augmenting the existing permission system with app policies rather than defining new permissions, however, by the end of Phase 2, we are considering have a richer set of permissions.

Ongoing Challenges for App Policies for the Mobile CRT

There are still a large number of open issues with this draft app policy specification. First, the *ontology of purposes* needs to be fully specified, have good coverage, and user tested for understandability. We have a version of this ontology in Appendix A, though it still needs more testing to ensure breadth.

The app policies could also use some notion of *time*. This might include data retention time or frequency of requests. We opted not to include data retention time, so that the app policies can be focused primarily on the device itself rather than on remote computers, due to reasons of simplicity and complexity of possible off-device behaviors. We considered frequency of requests (e.g., this app will only request your contact list once, or this app will only request your location at most 3 times per hour). We opted not to include this feature in this version due to complexity of enforcement. That is, PE Android would need to keep track of additional state about requests, and more state tends to lead to more complexity and more bugs. It would also very likely lead to new kinds of bugs for app developers (why isn't my app's location request working?), especially given that requests tend to be event driven rather than time driven.

The app policies also need to *integrate the functionality of the different TA1s*. TA1s that are dataflow-oriented would naturally fit in. For example, Stealth's function secret sharing could be a Where constraint that describes how the data is transformed before it leaves the device. USF/Princeton's image processing would be a bit harder since it does not naturally fit into a dataflow, though this is worth discussing more with their team. Helio is more difficult for reasons explained previously (see the subsection about the Where clause). We think it's better to split the data-collection aspect of apps (ie smartphone+app to cloud) from backend distributed processing (ie Helio).

One particular challenge for integrating TA1s is *how to explain to typical smartphone users* what these different transformations are. For example, can end-users really understand differential privacy, function secret sharing, secure enclaves, etc.? One idea here is to talk about the guarantees rather than the specific methods or techniques used.

When specifying a dataflow transformation, should parameters also be included? For example, let's suppose we had a differential privacy function DifferentialPrivacy that can be applied before any data egresses. Should the policy include the parameters, e.g., DifferentialPrivacy(ε=0.2, δ=0.3)? One advantage is that this info might help end-users understand what's going on. One disadvantage is that it adds complexity. Another disadvantage is that parameters are now in two places, the policy and the actual code, which might lead to inconsistency between the two.

Another open question is, what happens if a single activity has *different modes*? For example, for RapidGather, we are expecting a "normal" mode and an "emergency" mode. To make the

code easier to enforce, developers could structure their app so that there are different Activities associated with normal and emergency modes, though this could lead to a lot of redundancy. In short, it's not clear how well the current policies might work with typical devs for apps like this.

The current draft of app policies also uses *domain names directly as destinations*. An alternative is to specify desired properties of remote computers that can handle or process sensitive data, e.g., "uses a differentially private database". Though it is an open question as to how to ensure and enforce this behavior. One possibility is some kind of remote attestation. Another is to allow cloud services like Google or Amazon or Microsoft to enforce this, e.g., app says it uses a database with property xyz, and Microsoft has APIs to confirm that this is indeed the case. For the first version, we think that sticking with domain names as destinations is sufficient.

There is also the issue of specifying the *syntax* of these policies. The example policy above is more like pseudocode, and there needs to be a clearer specification that meshes well with existing Android standards. For example, a likely scenario is that the policy will be specified using an XML or JSON format.

It's also worth noting that there are some potential future directions that can increase flexibility and utility of these app policies, though we will likely not be able to examine these in the scope of this project. One such idea is dynamic policies based on context, e.g., disallow camera in these secure locations, but allow them in other ordinary locations. Another such idea is organizational policies, that is policies specified by one's organization to govern how sensitive data and functions can or cannot be used.

Next Steps

Two major tasks that will need to be addressed are (a) iterating on this app policy specification so that it can cover a majority of useful app use cases with respect to privacy, and (b) showing that these app policies can be used, checked, and/or enforced in useful ways (as in the four use cases in the introduction). These two challenges will comprise the bulk of the work for Phase 2 with respect to policies.

Appendix A - Purposes

This appendix lists our current set of purposes.

General Purposes

These are purposes that can be applied to any source of personal data.

Health
Emergency
Advertisement
Analytics
Sharing
Timestamp
Game

Identifiers

These are purposes for why an identifier is being used.

Personalization
Anti-fraud
Handling multiple installations
Authentication
Tracking signed-out user behavior
Managing telephony and carrier functionality
Abuse detection: Identifying bots and DDoS attacks(/Safenet API)

Phone Status

Battery / Charging	**Device model / Screen size**	**Network (wifi/lte)**	**Phone State (call state, screen light, display on/off)**	**Notification**	**Foreground / Background tasks**	**App Info**
power management	ui customization	Task Trigger (Context inference)	Task Trigger (Context inference) (e.g. send notification, call recording, video chat interruption)	UI Personalization (lock screen)	Task management	SDK
intelligent power saving	advertising	lower resolution image		Interruption management	Cross app communciation	Package ID
Task Trigger (Context inference)	Language					

	OS information					
	Manufacturer					

Personal Data

Contacts	Calendar	SMS	Storage	Accounts
Backup and Synchronization	Task Trigger (Context inference)	send sms (screenshot, app link, text, automatic reply, geo location, encrypted sms)	access album (photo uploading/editing)	3rd party login
Contact Management	Schedule	organize sms (clustering, delete, re-rank)	photo editing	
Blacklist	Alarm	extract sms content (check notification)	data backup	
Call and SMS		block sms	download	
Contact-based Customization		send sms commands/confirmation	persistent logging	
Email		schedule sms		
Find friends		back up/synchronize sms		
Record		receive msg/messaging		
Fake Calls and SMS				
Reminders				

Sensor Data

These are purposes that can be applied where the source is a sensor.

Camera	Location	Microphone	Accel	Gyroscope	Magneto-meter	Proximity
Flashlight	Search nearby places	Sound detection / recognition			Navigation	Speaker / display activation
Video streaming / Video chat	Location-based customization	Voice communication				
QRCode / Barcode scan	Transportation Information	Voice controls / command				
Document scan (biz card, coupon, check)	Recording	Speech recognition				
Augmented Reality	Navigation	Audio recording				
Social Media	Geosocial Networking	Authentication				

Text recognition / Translation	Geotagging	Data transmission				
Video recording	Location Spoofing	Music				
	Alert and Remind					
	Location-based game					