CleanOS: Limiting Mobile Data Exposure with Idle Eviction

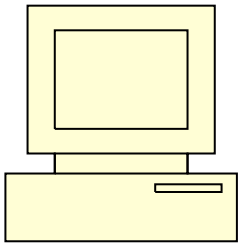
Yang Tang, Phillip Ames, Sravan Bhamidipati,
Ashish Bijlani, Roxana Geambasu, Nikhil Sarda

Columbia University

The Transition to Mobile Devices

- Mobiles are replacing desktops as primary computing platform

Desktops



✓ Linux, OS X, Windows
✓ Emails, Web browsers, Office

Mobile Devices

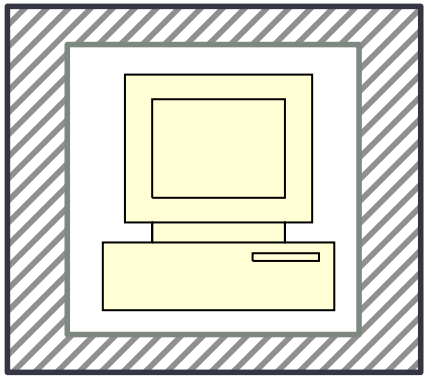


✓ Android, iOS, Windows Phone
✓ Emails, Web browsers, G Docs

\+ New, fun, mobile apps
\+ Increased productivity
\+ Pervasive connectivity

Mobiles Bring New Challenges

Desktops



Physical security, firewalled networks

data

Mobile Devices



Unprotected outdoors, mobiles can be easily stolen, seized, or lost

OSes Not Designed for Challenges

- Despite threats, mobile OSes have not evolved to protect sensitive data
- Like desktop OSes, mobile OSes accumulate enormous sensitive data
  - OS doesn't securely erase deallocated data
  - FS doesn't securely erase deleted files
  - Applications hoard sensitive data for performance or convenience
- This data is placed at risk if device is stolen or lost
  - Thief can dump RAM, flash memory contents
  - Thief can break passwords
  - …

Examples of Exposed Data

App	Password	Contents
Email	✔	✔
Y! Mail	✔	✔
Google Docs		✔
OI Notepad		✔
Dropbox		✔
KeePass	✔	✔
Keeper	✔	✔
Amazon		
Pageonce	✔	✔
Mint		✔
Google+		✔
Facebook		✔
LinkedIn		✔
TOTAL	5/14	13/14

- We dumped memory and SQLite DB for 14 popular Android apps

Default Email App:

- Password and email snippets are kept in cleartext RAM at all times
- Everything is present in cleartext SQLite DB

Examples of Exposed Data

App	Password	Contents
Email	✔	✔
Gmail		✔
Y! Mail	✔	✔
Google Docs		✔
OI Notepad		✔
Dropbox		✔
KeePass	✔	✔
Keeper	✔	✔
Amazon		
Pageonce	✔	✔
Mint		✔
Google+		✔
Facebook		✔
TOTAL	**5/14**	**13/14**

- We dumped memory and SQLite DB for 14 popular Android apps

Overall:

- Captured some sensitive data from 13/14 apps
- All 13/14 apps hoard some sensitive data in cleartext DB/RAM
- 9/14 apps hoard sensitive data in cleartext RAM at all times

Securing Data Is Darn Hard!

- Example protection systems:
  - Encrypted file systems
  - Encrypted RAM
  - Remote wipe-out systems
- Challenges / limitations:
  - Users don't lock their devices (57%) or configure poor passwords
  - Physical attacks are notoriously difficult to protect against
    - E.g., memory dumps, cold boot attacks, breaking trusted-hardware seals can reveal data or decryption keys
- In general, these are imperfect stop-gaps on top OSes that were never designed with physical insecurity in mind

Time for New OS Abstractions

- Mobile OSes should manage sensitive data rigorously, so as to maintain the device "clean" at any point in time in anticipation of device theft/loss
- If device is stolen or lost:
  1. The minimal amount of sensitive data is exposed
  2. User retains post-theft control over unexposed data

CleanOS

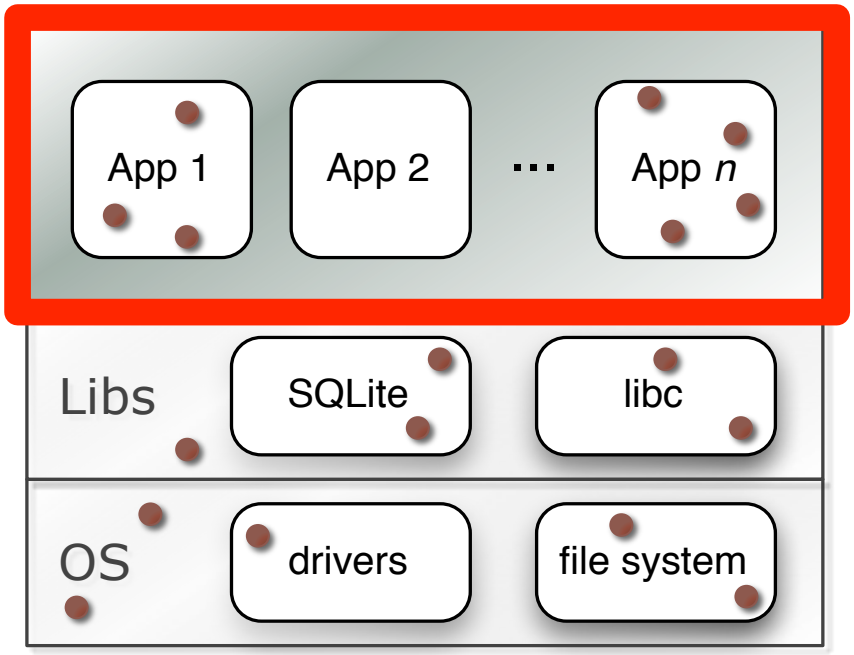
- New Android-based OS that minimizes sensitive data exposure by evicting it to a trusted cloud whenever not under active use

- Implements sensitive data objects (SDOs)
  - Identifies locations of sensitive data in RAM and stable storage
  - Monitors its use by applications
  - "Evicts" sensitive data to the cloud whenever it is not under active use

- The cloud intermediates all accesses to unused SDOs and can offer a lot of useful post-loss functions:
  - Disable SDO access after theft
  - Audit SDO exposure and access
  - Rate-limit SDO accesses

This Talk

- All layers of current OSes require cleanup
- As first step, we focus on cleaning up the application layer
- This talk: limit exposure of sensitive data hoarded by apps



Outline

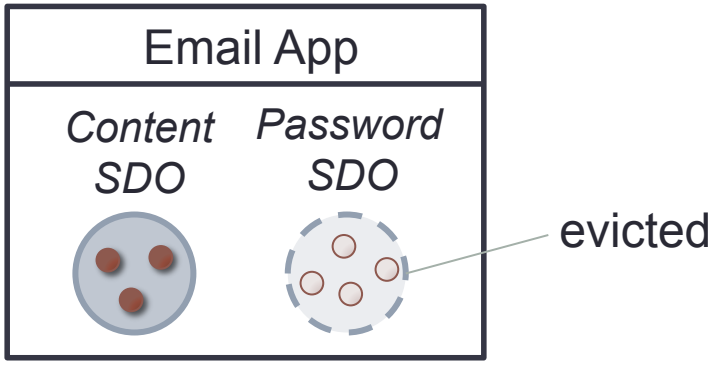
- Motivation
- Architecture and Prototype
- Evaluation
- Conclusions

Mobile OS Insights

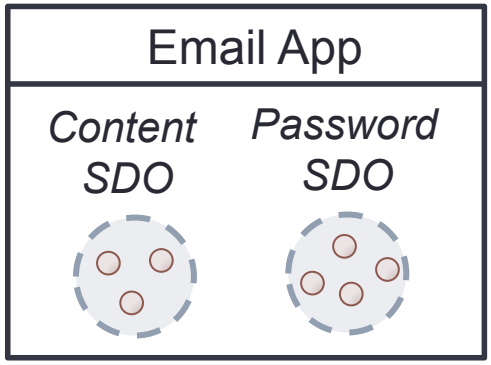
1. Although exposed permanently, much sensitive data is only actually used very rarely
   - Email password is constantly exposed, but only used upon refresh
   - Subjects are constantly exposed, but only used upon inbox listing
   - Contents are often exposed, but only used when user reads email
2. Most mobile apps have a cloud component, which already stores the data
3. Mobiles are (becoming) mostly connected
   - Wi-Fi / cellular coverage is becoming pervasive

CleanOS Basic Functioning

- Applications create SDOs and add sensitive data to them
- CleanOS implements three functions for SDOs:
  1. Tracks data in SDOs using taint tracking
  2. Evicts SDOs to a trusted cloud whenever idle
  3. Decrypts SDO data when it is accessed again

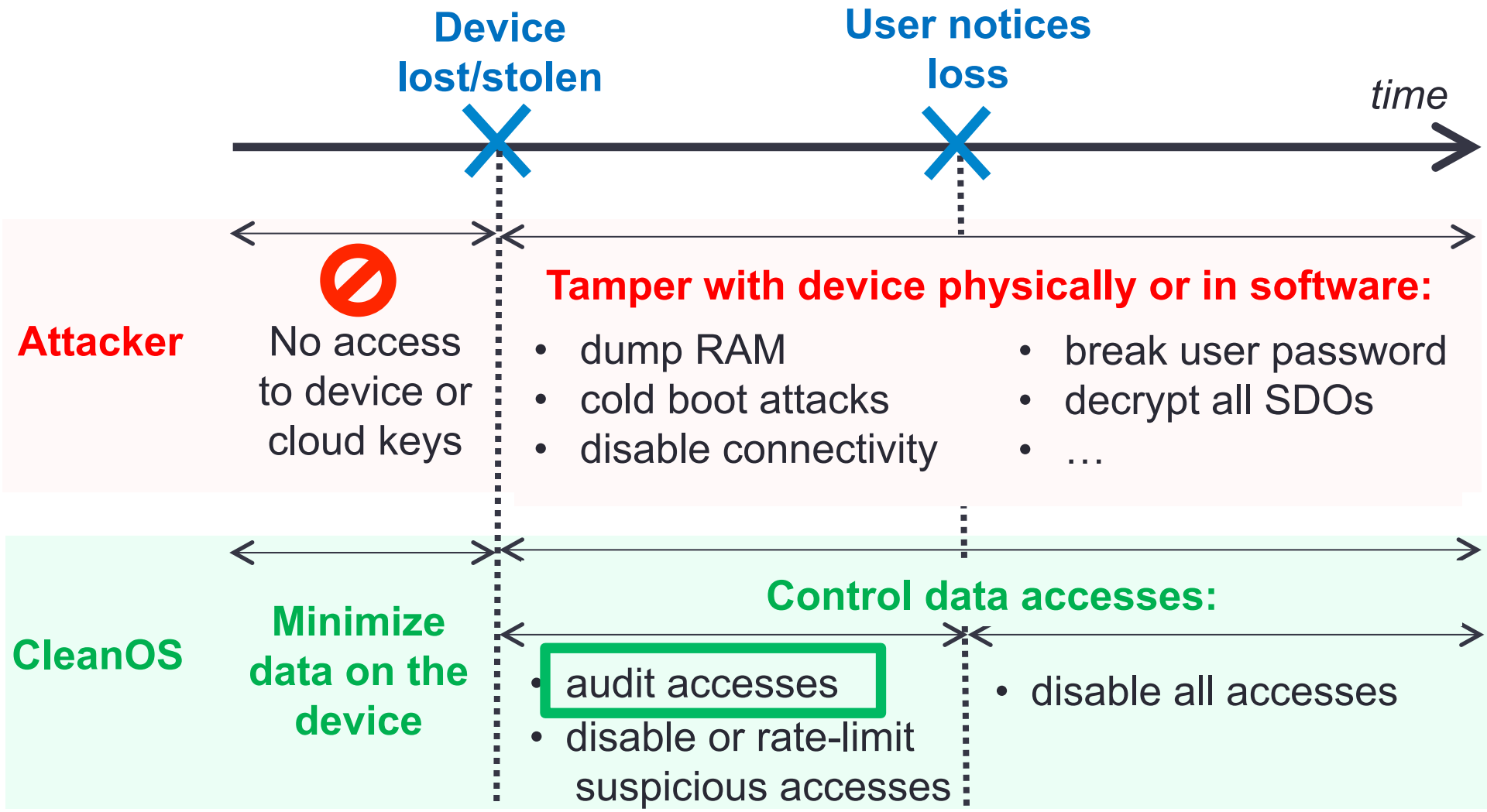


when user reads an email



when the app goes to the background

CleanOS Increases Post-Loss Control



The SDO Abstraction

- An SDO is a logical collection of Java objects that contain sensitive data and that are related somehow
  - Email-password SDO: password string + all objects computed from it
  - Email-contents SDO: all emails in a thread
  - Bank-account SDO: all transactions in an account

human-readable and used for auditing

```
class SDO {
    SDO(String description, ...);     // create SDO
    void add(Object o);               // adds object to SDO
    void remove(Object o);            // removes object from SDO
    …
    private int sdoID;                // unique ID for SDO
}
```

used for identifying SDOs locally and remotely

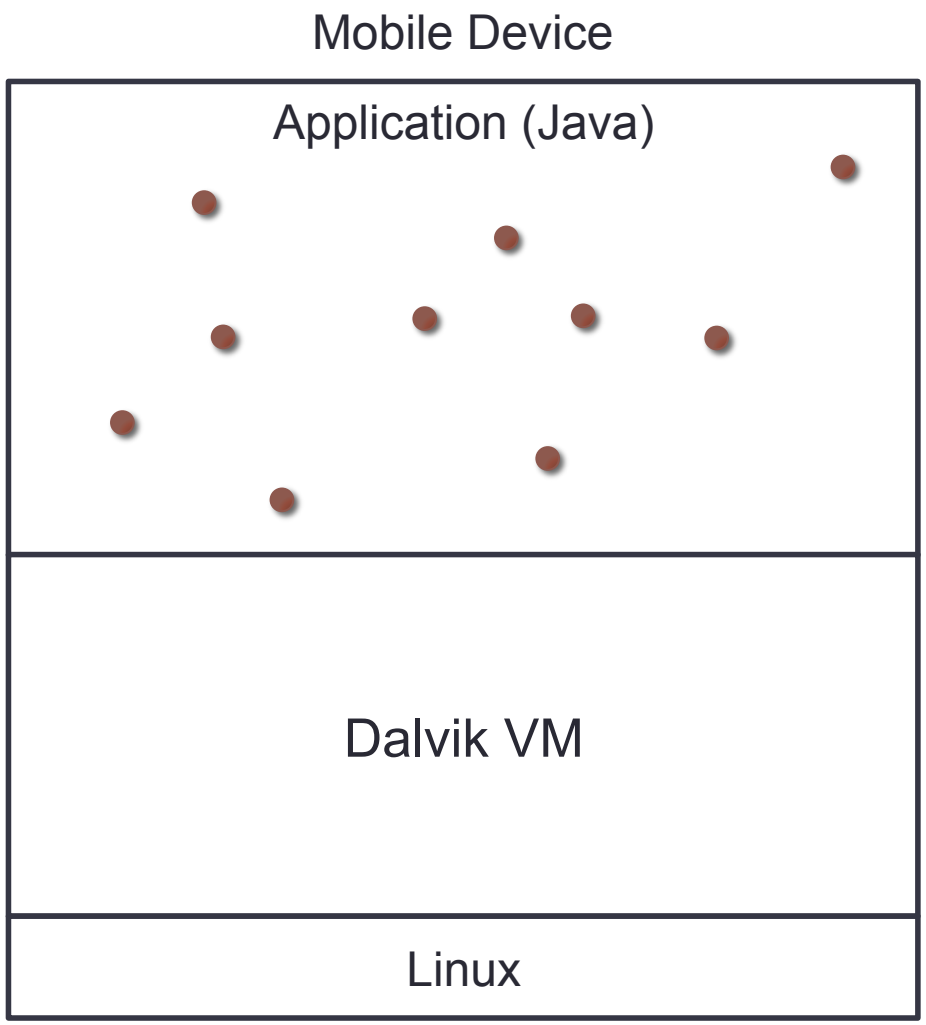
Example: A Clean Email App

- Default Email app hoards passwords and subjects
- Hard for apps to manage sensitive on their own
- With CleanOS, developers simply create SDOs and place their sensitive data in them

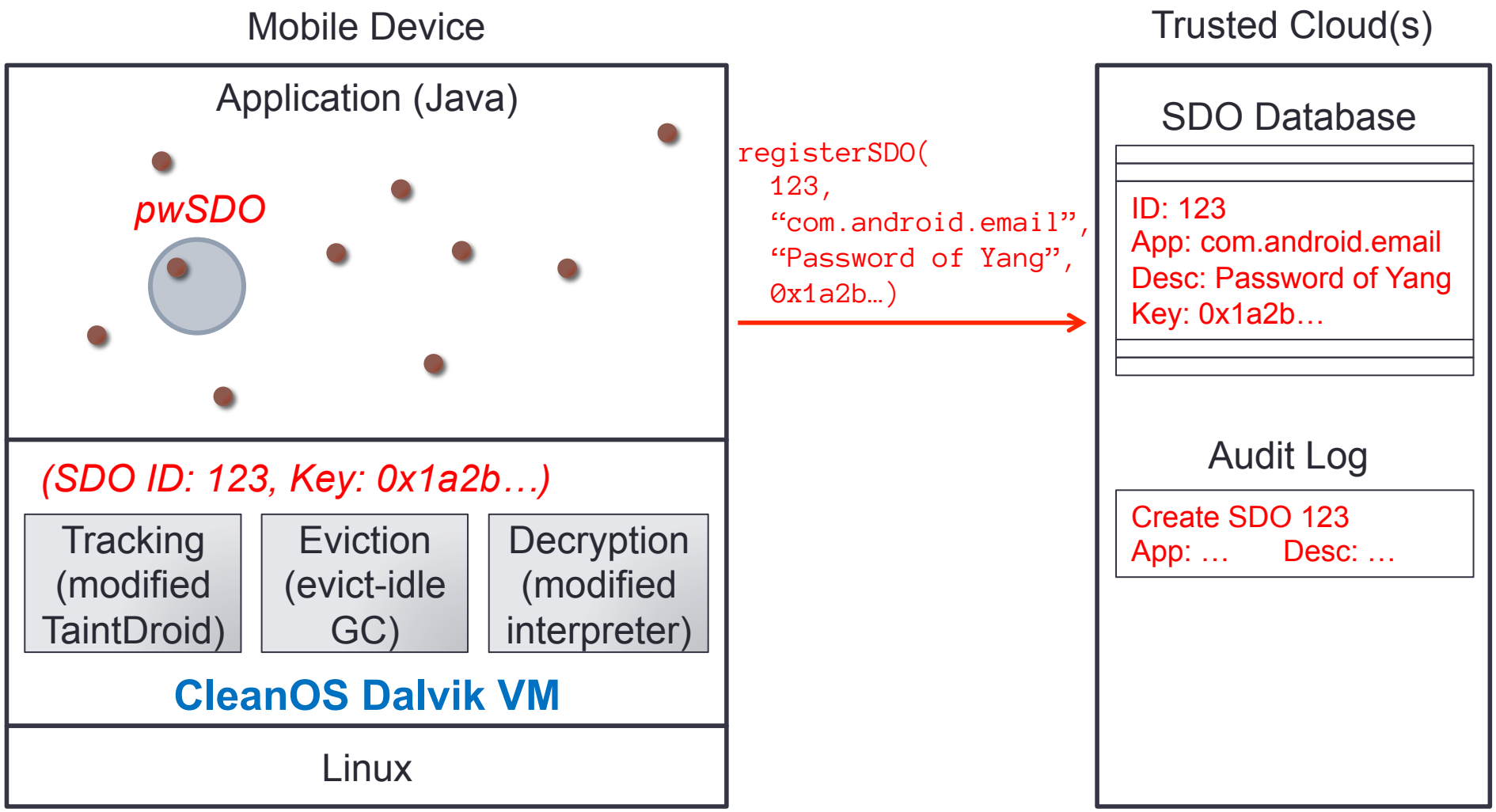
```
SDO pwSDO = new SDO("Password of " + user);
pwSDO.add(password);
...
SDO emailSDO = new SDO("Email: " + mSubject);
emailSDO.add(mSubject);
emailSDO.add(mTextContent);
emailSDO.add(mHtmlContent);
emailSDO.add(mTextReply);
emailSDO.add(mHtmlReply);
```

(code adapted for clarity)

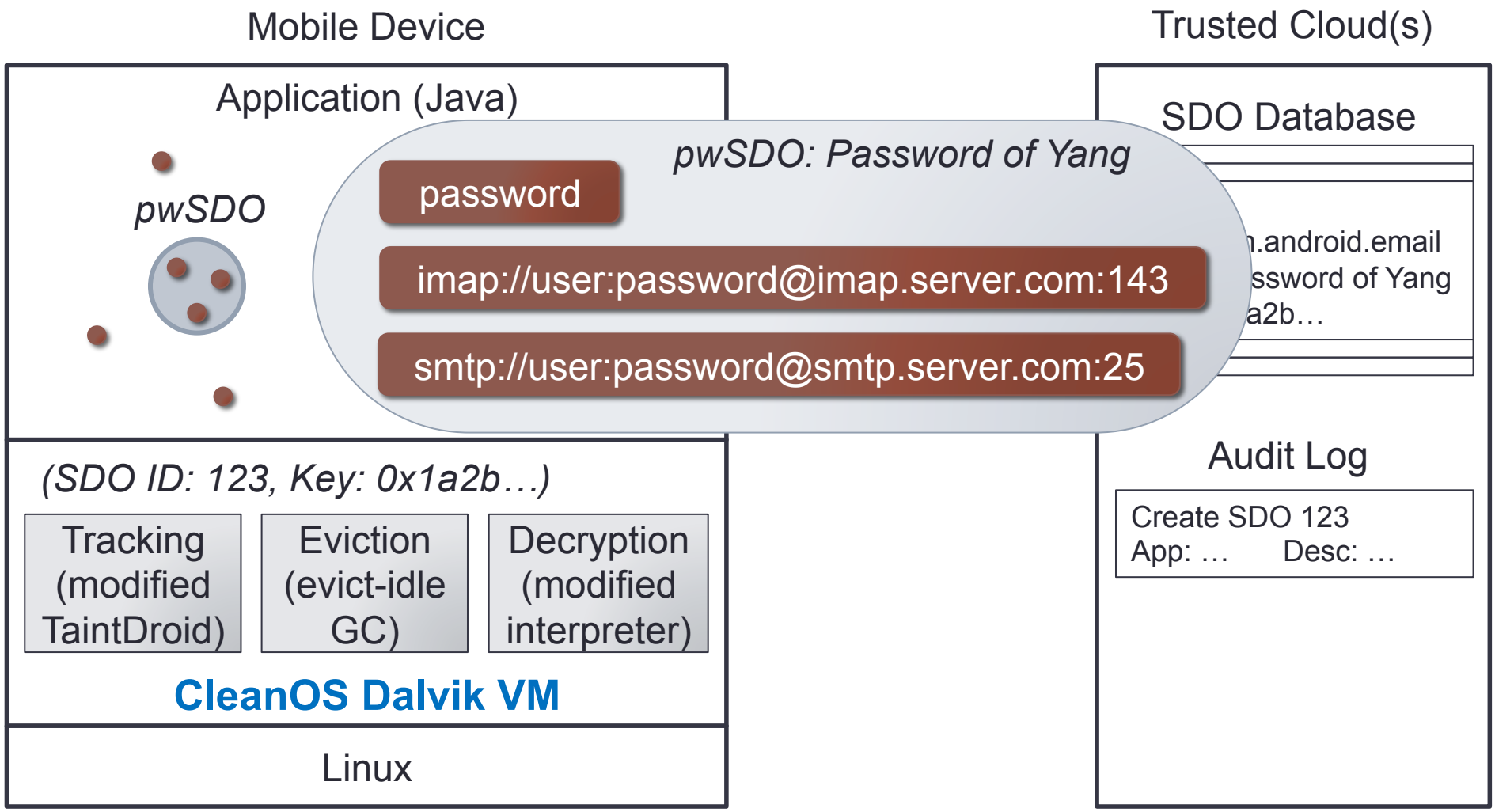
The CleanOS Architecture



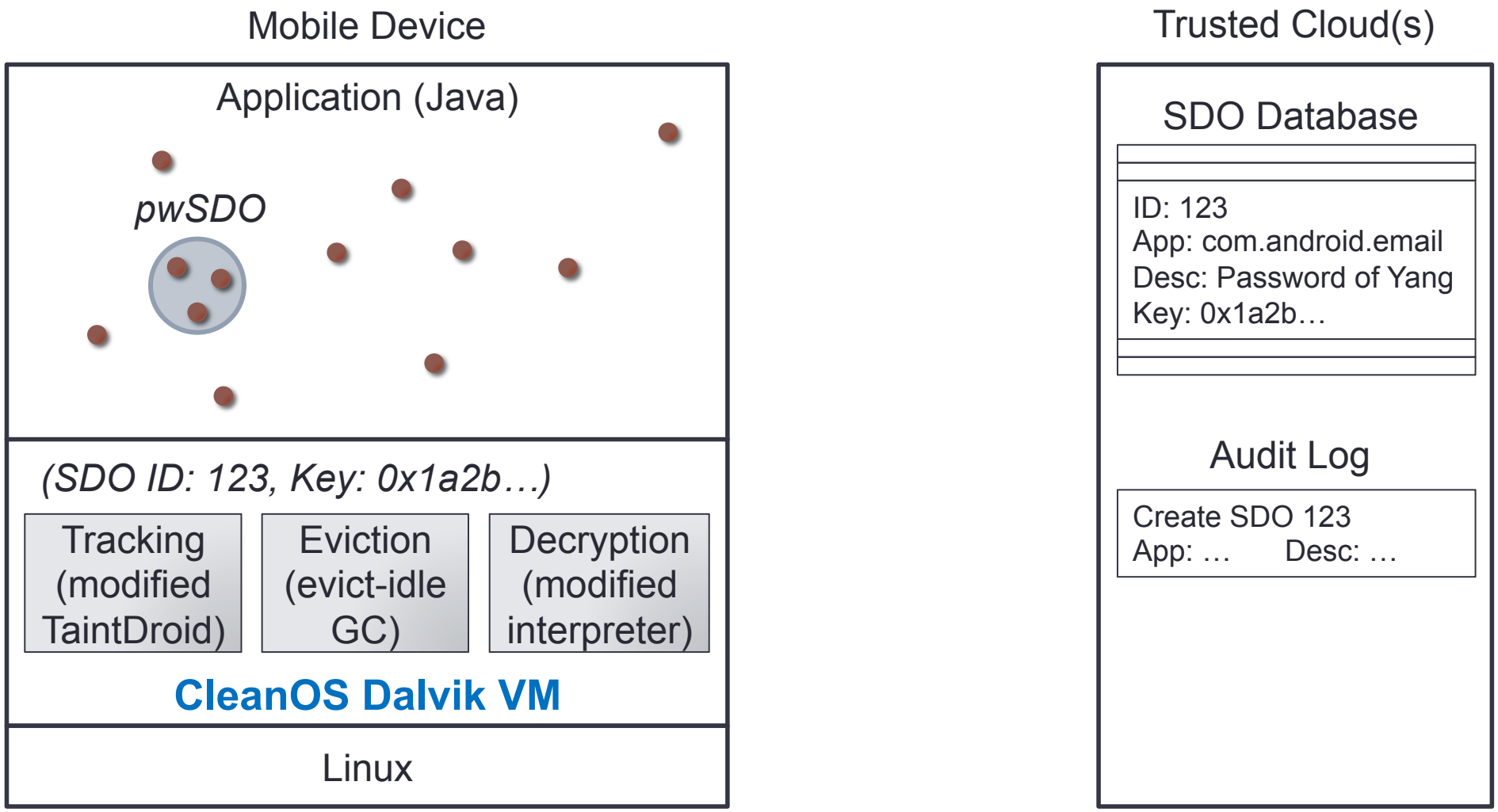
The CleanOS Architecture



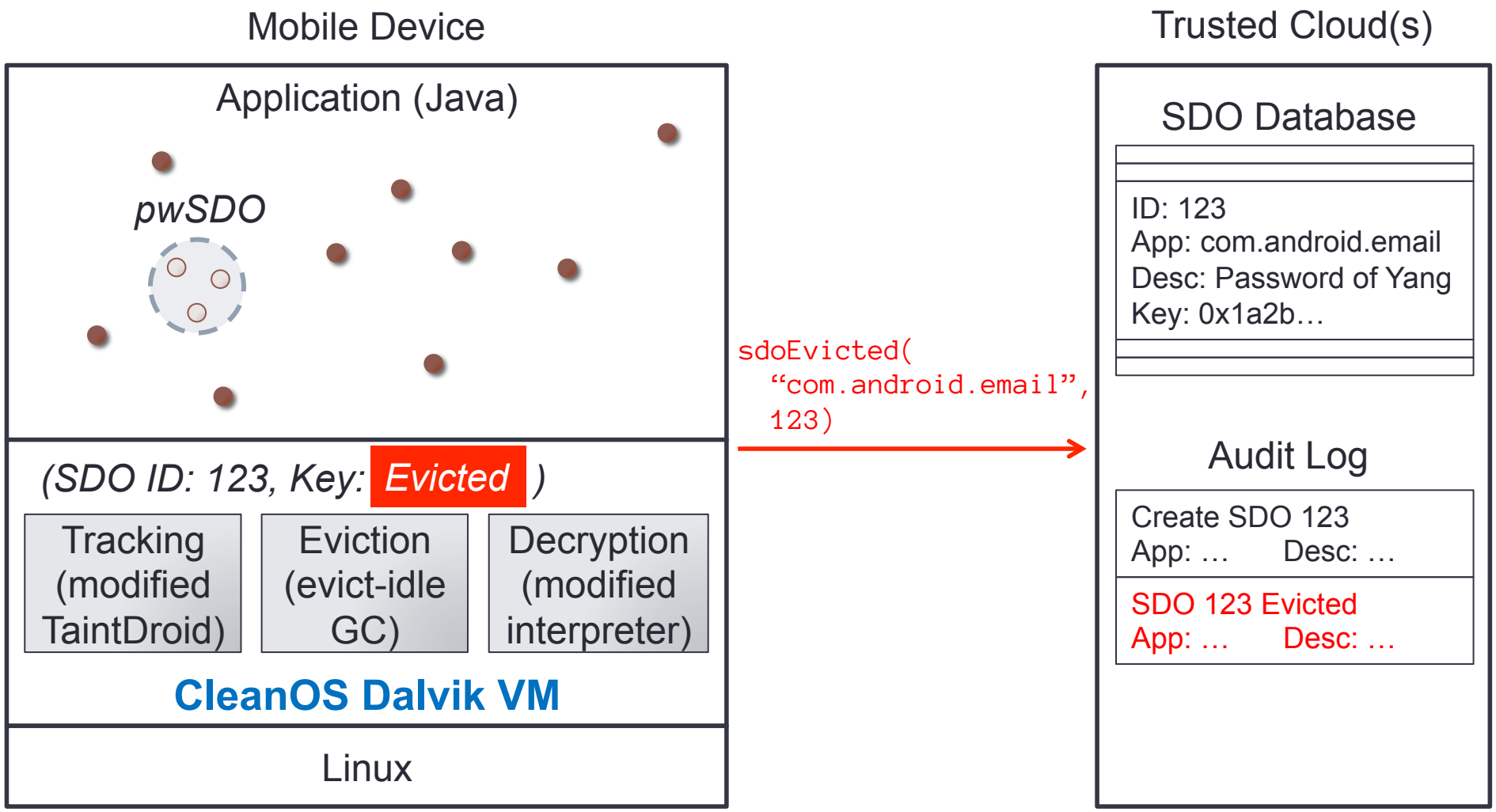
The CleanOS Architecture



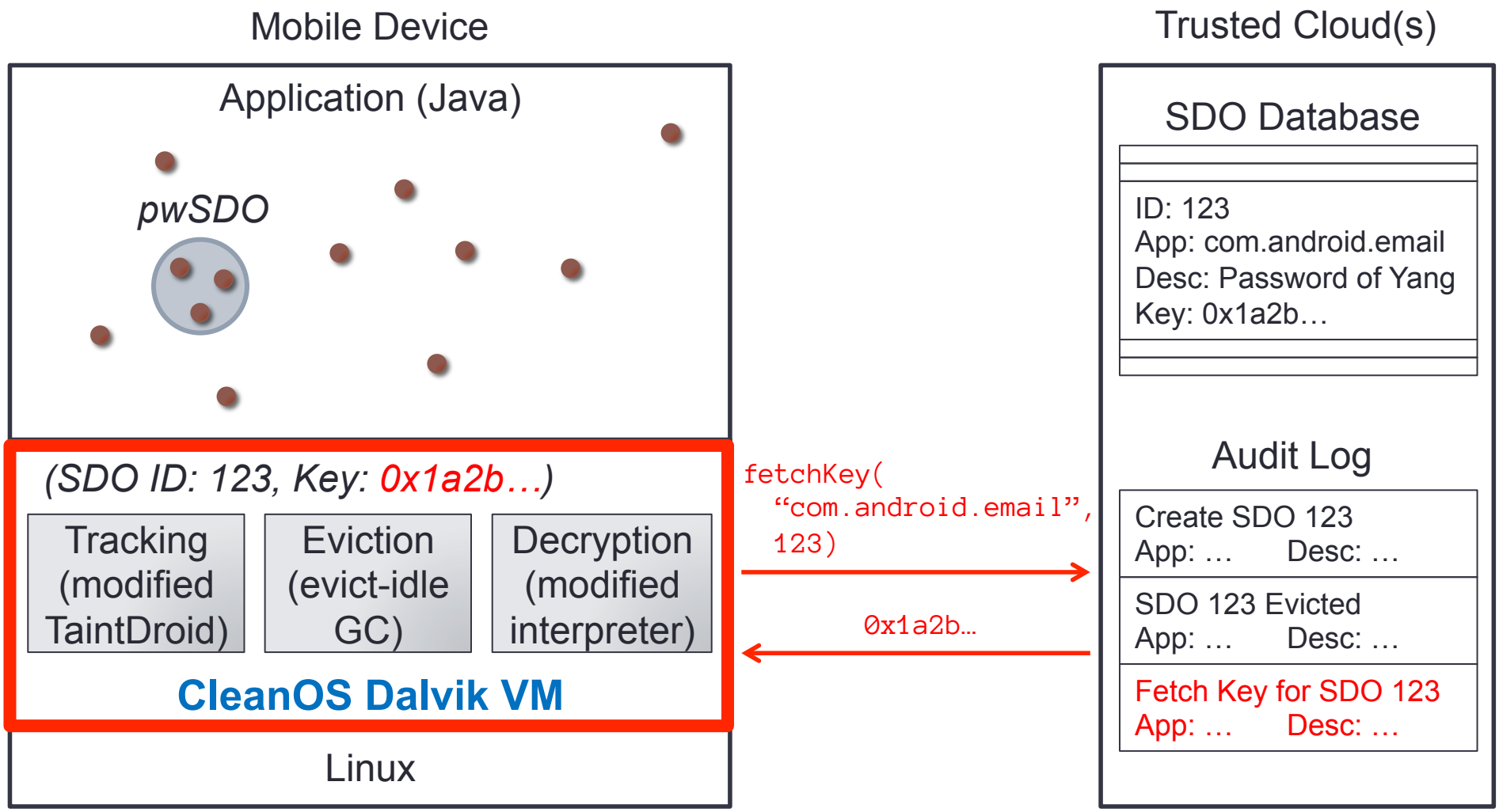
The CleanOS Architecture



The CleanOS Architecture

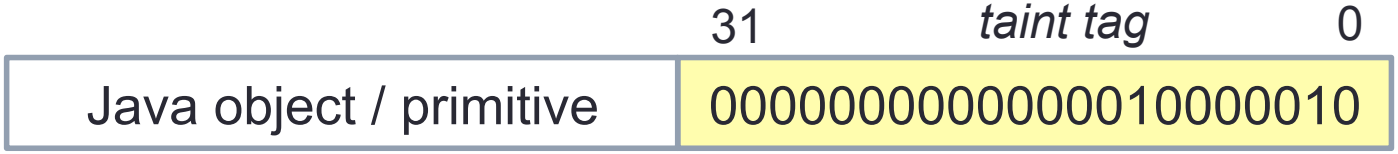


The CleanOS Architecture

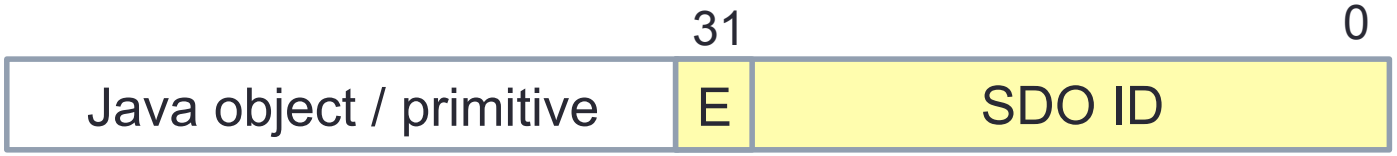


SDO Tracking

- CleanOS uses a modified TaintDroid version to track SDOs
- TaintDroid uses 32-bit shadow taint tags to mark objects/ primitives with one or more taints



- CleanOS imposes a specific structure on taint tags
  - Includes new tainting scheme to propagate one SDO ID (see paper)



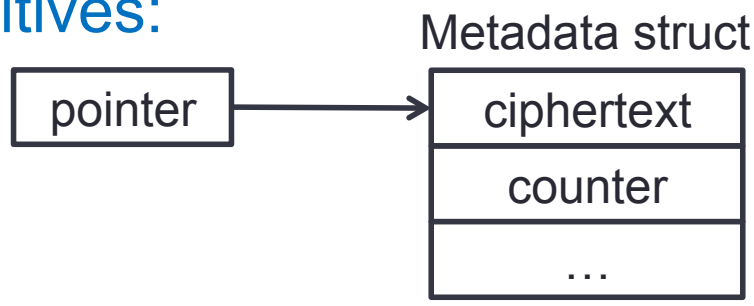
Evict-Idle Garbage Collection

- To evict SDOs, we introduce a new type of Java garbage collector: the evict-idle garbage collector (eiGC)
- A traditional GC: deallocates only objects guaranteed to never be used in the future
- The eiGC: evicts objects that have not been used for a period of time, *even if* they might be used again in the future
- We run the eiGC periodically to evict idle SDOs
  - For any Java object that is tainted with an idle-SDO's ID, the eiGC "evicts" that object, setting its E bit in the tag

Sensitive Data Encryption

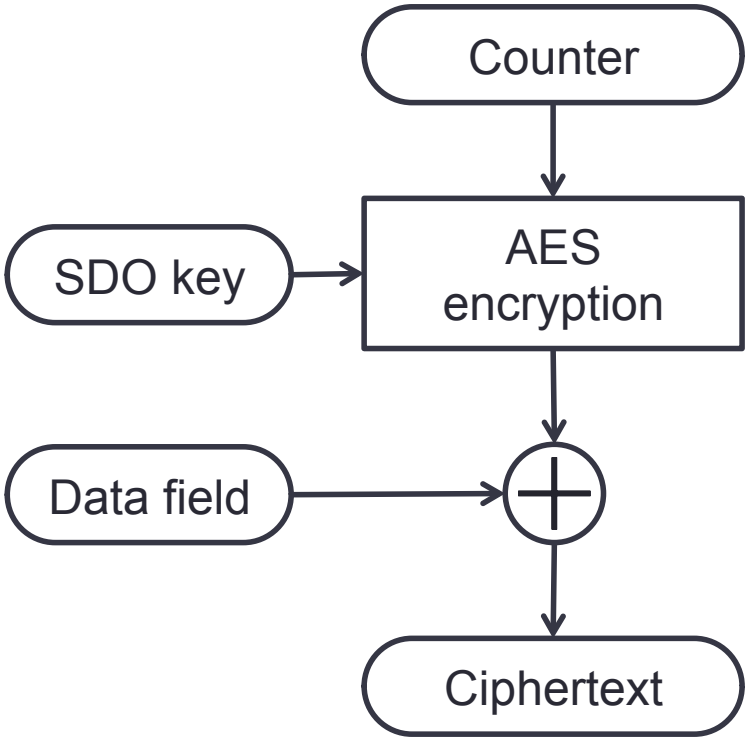
- To “evict” a Java object, eiGC traverses it down to the data-bearing fields (primitives & arrays of primitives) and encrypt those with a key generated from the SDO’s key

Primitives:



Arrays of primitives:

```
struct ArrayObject {
    contents[];  // evict in place
    counter;
    evicted_lock;
    ...;
}
```



CleanOS Defaults

- We hope for, but don't rely upon app modifications
- Hence, we provide a set of solid defaults:
  - "SSL" SDO
  - "User Input" SDO
  - "Password" SDO
- Although coarse-grained, default SDOs limit exposure of sensitive data in unmodified apps

In SSL library (Java SDK):

```
SDO sslSDO = new SDO("ssl");
sslSDO.add(cleartextDataBuffer);
```

In InputMethod (Android SDK):

```
SDO inputSDO = new SDO("user input");
SDO pwSDO = new SDO("password");
if (type == InputType.TEXT_PASSWORD)
    pwSDO.add(text);
else
    inputSDO.add(text);
```

(code adapted for clarity)

Optimizations

CleanOS includes a number of optimizations, which help with both performance and energy consumption

1. Pausing the eiGC
   - After the app becomes idle, we stop running eiGC, until the app is active again
2. Bulk eviction
   - The eiGC evicts all SDOs at once soon after the app becomes idle
3. Bulk prefetching
   - The eiGC remembers which SDO keys were used recently and fetches all those keys at once

Prototype

- Modified Android and TaintDroid in significant ways

Component	New or Changed Features
Dalvik (JVM)	Evict-idle Garbage Collector (eiGC) Eviction-aware bytecode interpretation Secure deallocation of interpreted stacks
Android SDK	SDO API Default SDO heuristics
TaintDroid	Support for millions of taints SQLite vulnerability fix
SQLite	Taint tracking in database
Webkit	Screen-buffer purging
Bionic (libc)	Secure user-space deallocation
Linux Kernel	Secure page deallocation with grsecurity

- Modified two “dirty” apps to use SDOs: Email and KeePass
  - Modifications are trivial: 7 LoC for each application

Post-Loss Services

- Built a CleanOS service on Google App Engine that supports post-loss auditing

device	message	time
cd5493c1befeb9075442862afa046182	fetchKey(9.408 - com.android.email - password)	2012-04-28 17:26:50.590000
cd5493c1befeb9075442862afa046182	registerSDO(com.android.email - Invitation to develop "Clean OS")	2012-04-28 17:27:01.140000
cd5493c1befeb9075442862afa046182	fetchKey(30.709 - com.android.keepass - Entry)	2012-04-28 17:27:48.500000

Screenshot of the CleanOS audit log

- Other services could be supported:
  - Disable keys after user notices theft
  - Monitor key accesses, detect anomalies, and rate-limit/disable access
  - Disable accesses to certain SDOs while outside a trusted network
  - …

Outline

- Motivation and Goals
- Architecture and Prototype
- Evaluation
- Conclusion

Questions

- Does CleanOS limit data exposure?
- Does CleanOS support effective auditing?
- Is CleanOS' performance practical?
- Is CleanOS' energy consumption practical?
- Is CleanOS network consumption practical?

Does CleanOS Limit Data Exposure?

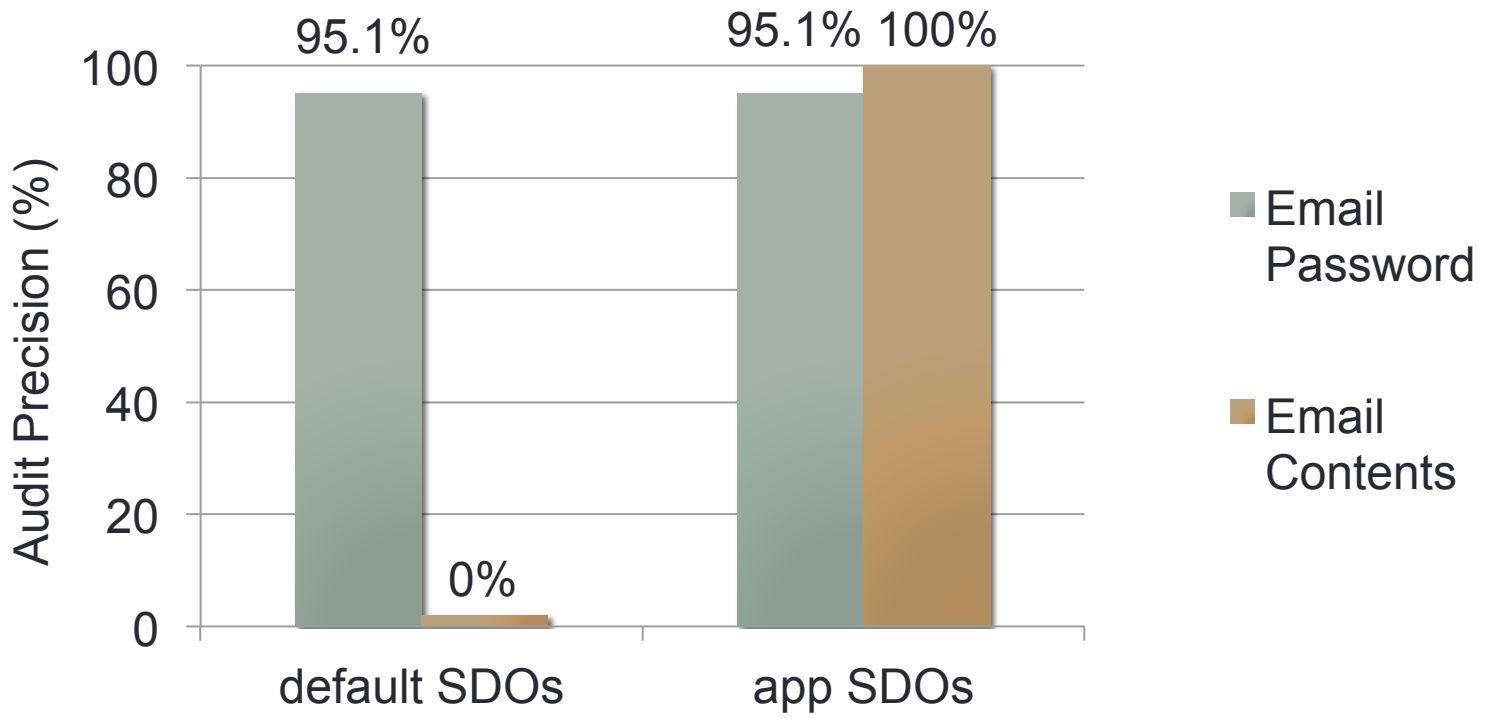
Email App Version	Password SDO	Contents SDO
Email without CleanOS	100%	95.5%
Email with **default SDOs**	6.5%	5.9%
Cleaned Email with **app SDOs**	6.5%	0.3%

Fraction of time in which sensitive data was exposed

- Default SDOs reduce exposure by 89.6-93.5%
- App SDOs further reduce content exposure to 0.3%
- Similar results in Facebook and Mint apps

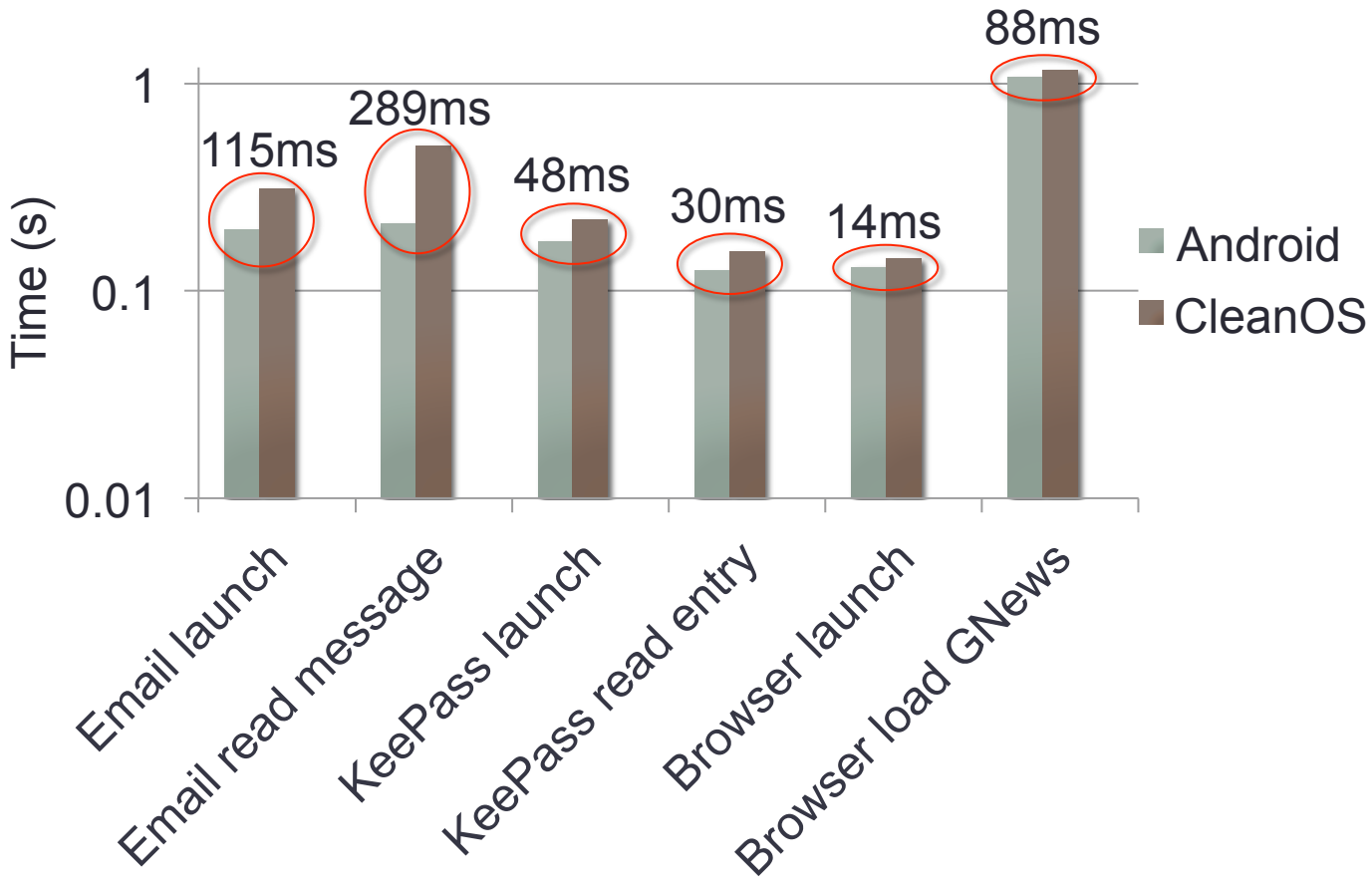
Thus, default SDOs drastically curb sensitive-data exposure

Is Auditing Effective?



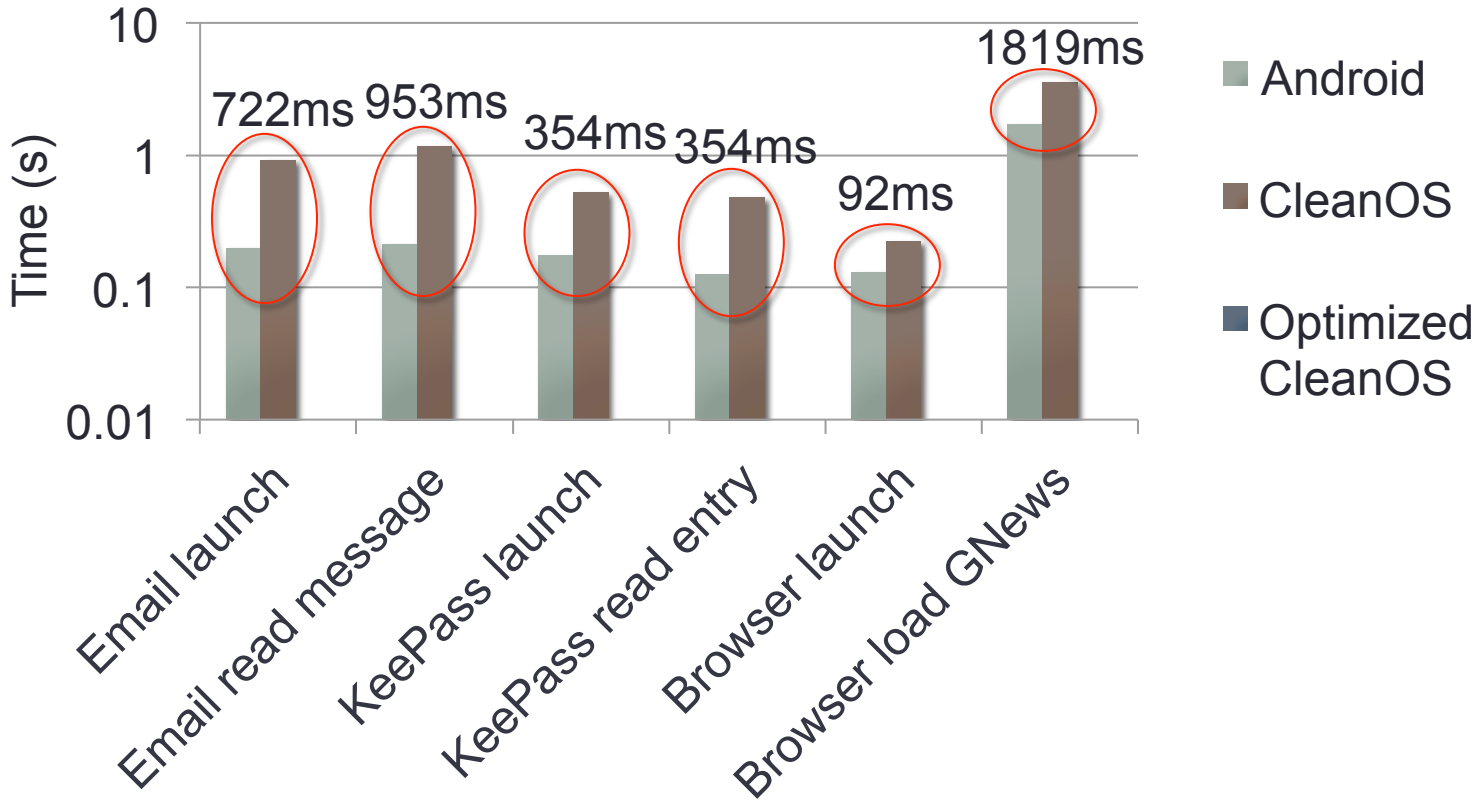
Thus, for best auditing properties, applications should define their own SDOs

Is CleanOS Practical (Wi-Fi)?

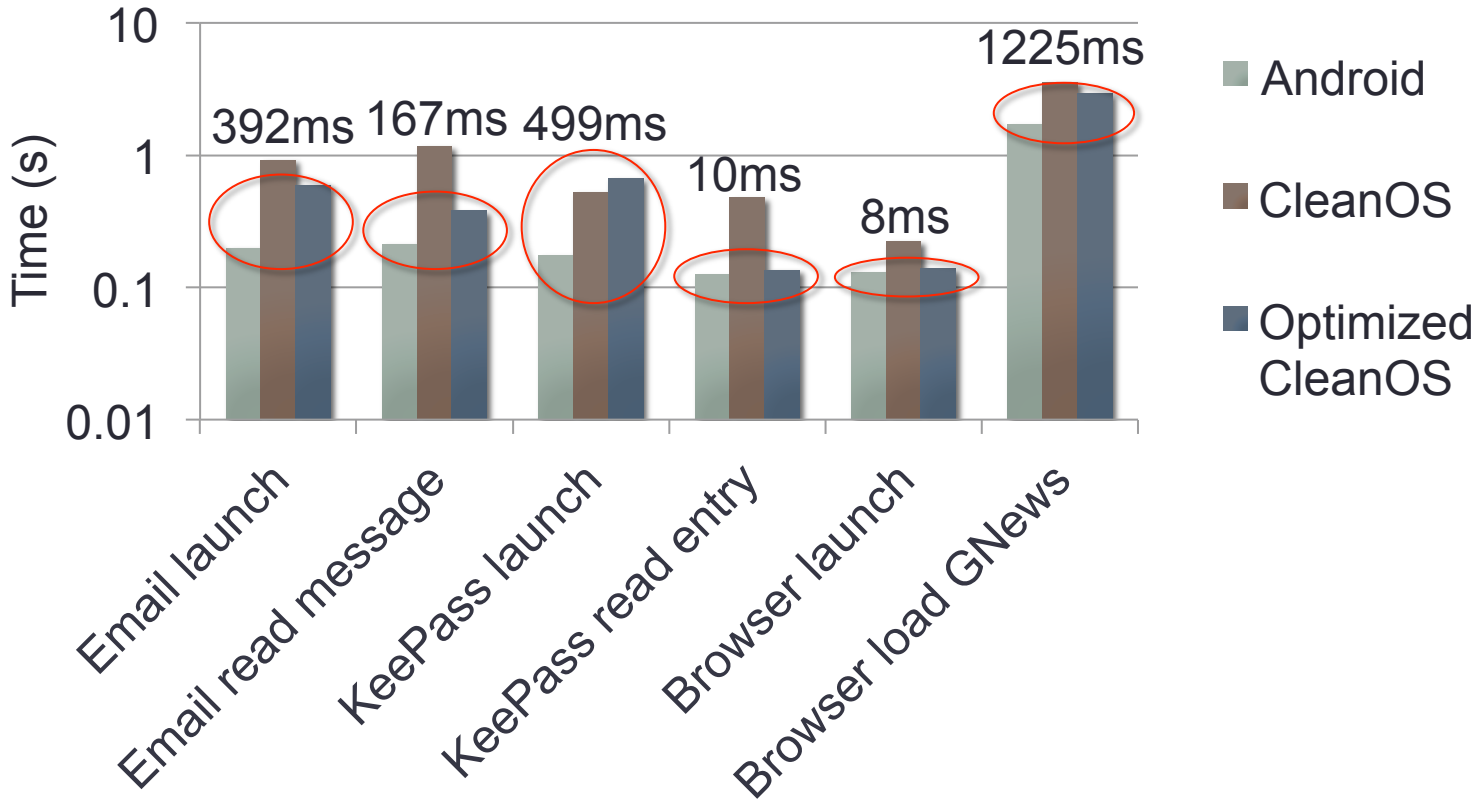


Thus, overheads are largely unnoticeable over Wi-Fi

Is CleanOS Practical (3G)?



Is CleanOS Practical (3G)?



Optimizations improve 3G performance significantly, making CleanOS practical

Conclusions

- Mobile OSes and apps accumulate a lot of sensitive data, exposing it to attacks if device is stolen or lost
- CleanOS is a new Android-based OS design that minimizes amount of sensitive data exposed on device at any time
  - SDO: new OS abstraction for managing sensitive data
  - eiGC: garbage-collector that evicts unused SDO data
- CleanOS brings new view on data security: minimize and audit exposure of sensitive data to attack
  - We believe that this view is applicable more broadly to other domains, such as datacenter and Web data security