QUIRE: Lightweight Provenance for Smart Phone Operating Systems

Dan S. Wallach
Rice University

Joint work with Mike Dietz, Yuliy Pisetsky, Shashi Shekhar, and Anhei Shu

Android's security is awesome in theory... and fails in practice.

Android differs from other OSs

Key Point:
Single User System

User separation → Application separation

- One Linux User ID per App
- File system access control via UID
- Permissions bound to App UID

Application separation in theory...

- Private storage for apps
  - Enforced at filesystem level
- Simple permission system
  - Does the calling UID have permission to access this resource?
- Crash Insulation

… but in practice

Application separation is violated all the time

How things work now

- Third party libraries run in the same context as their host app
  - And with the host's permissions
  - And some third party libraries require the host ask for more permissions
- Problems
  - Credential phishing, leaked application data, violation of least privilege, bugs

How things **should** work

- Third party libraries run in their own context as apps
- Apps offer services over Inter Process Communication (IPC) Channels

Two Main Goals

1. Protect apps from third party libraries
    - ex. Bugs, stolen data

2. Protect third party libraries from apps
    - ex. Click Fraud

Advertisements

A big deal to app developers

Angry Birds is making over $1 million a month in ad revenue

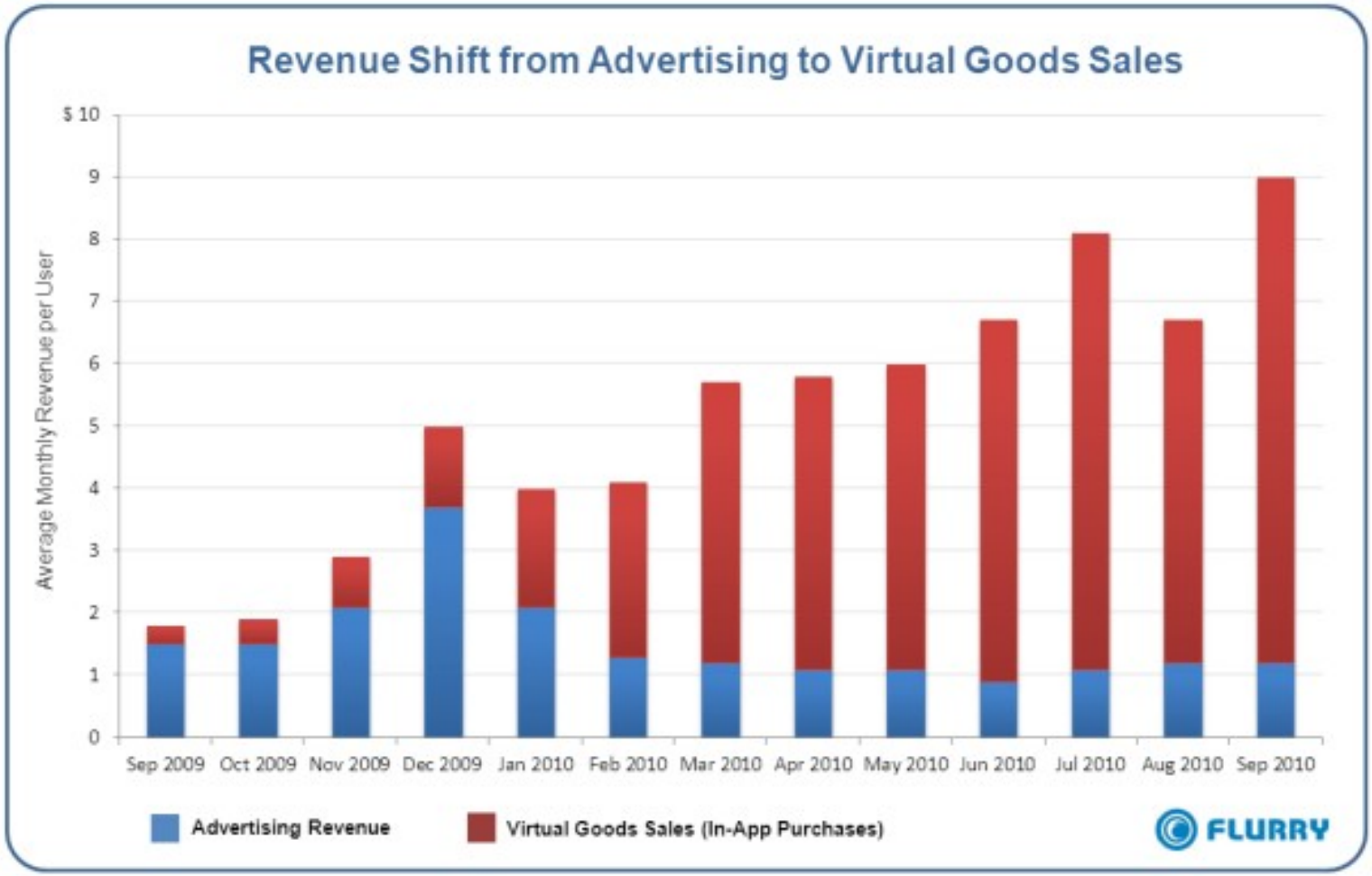
http://googleblog.blogspot.com/2010/12/great-advice-from-industry-experts-on.html

A big deal to the ad providers

Click fraud cost Google
$100 million a year

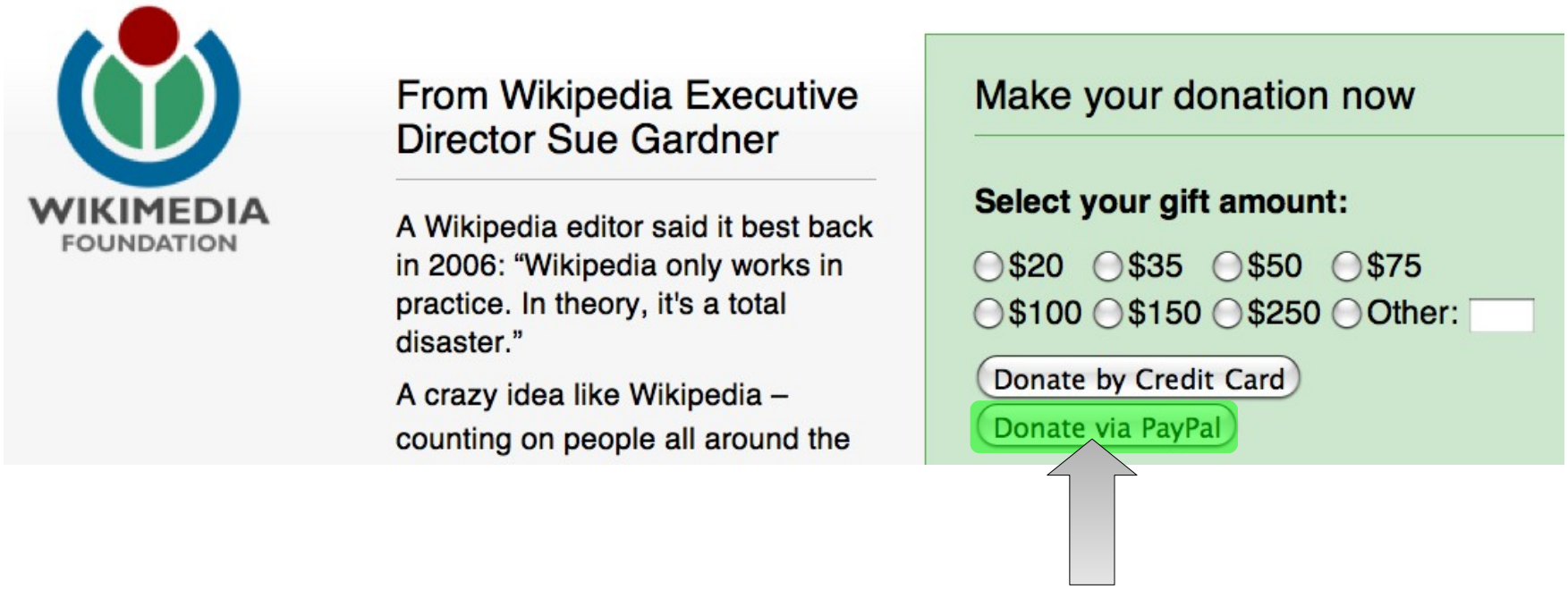
http://adwords.blogspot.com/2007/02/invalid-clicks-googles-overall-numbers.html

The next step: Micro-payments

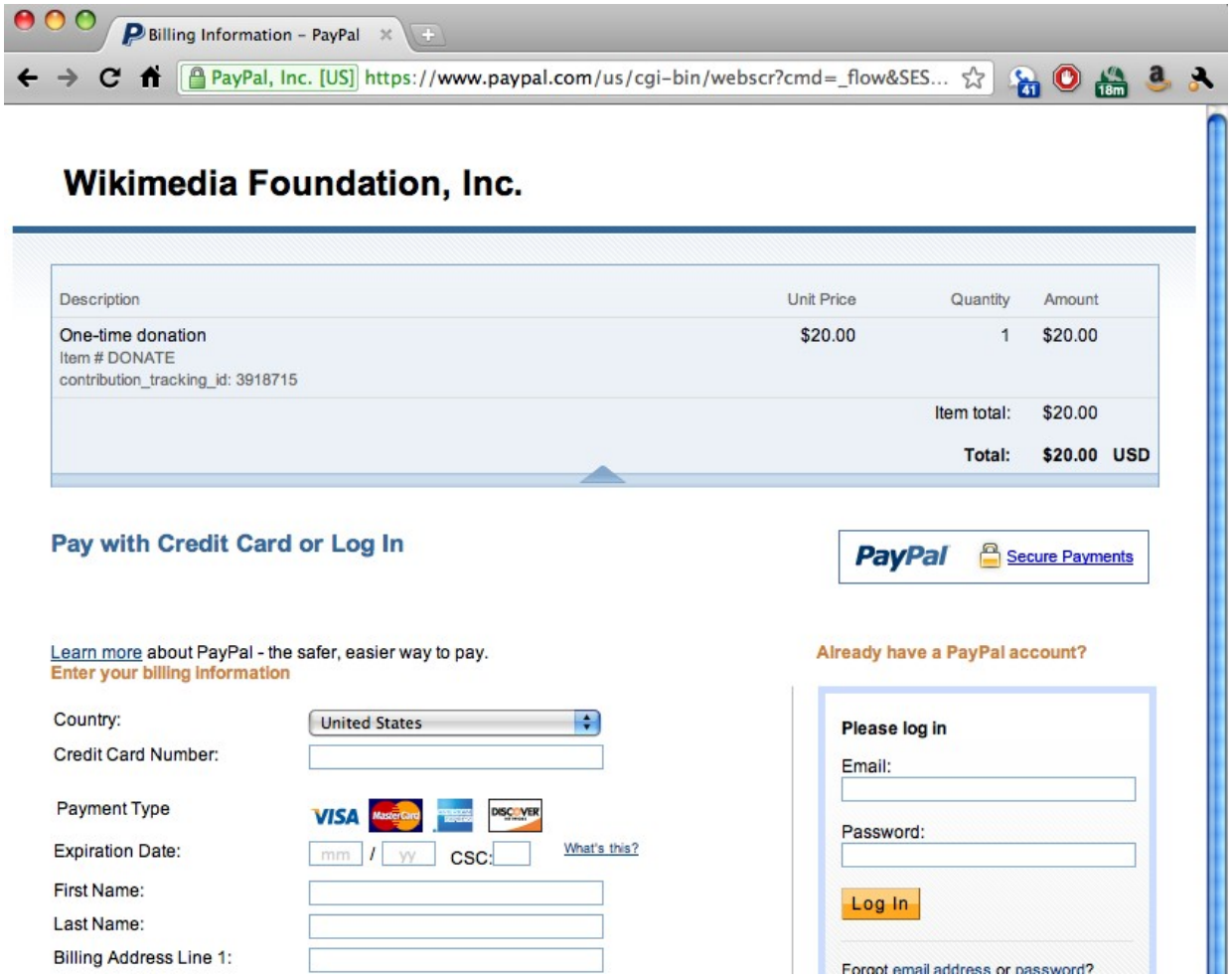


http://blog.flurry.com/bid/48418/Madison-Avenue-and-the-Land-of-Make-Believe

Motivation: Mobile Payment

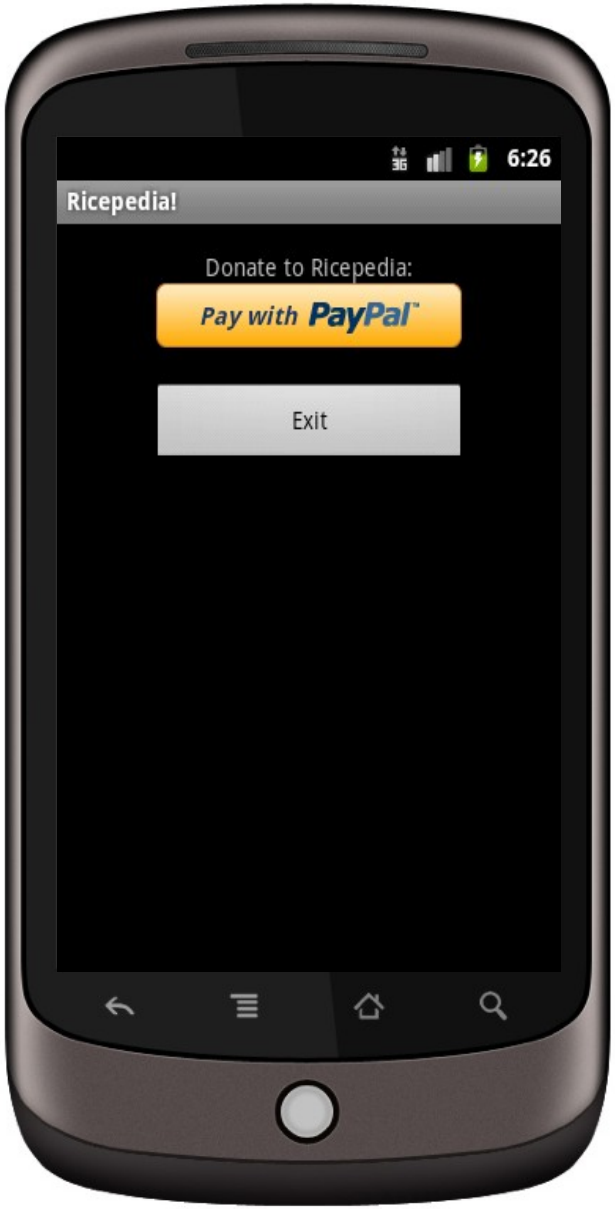


Motivation: Mobile Payment

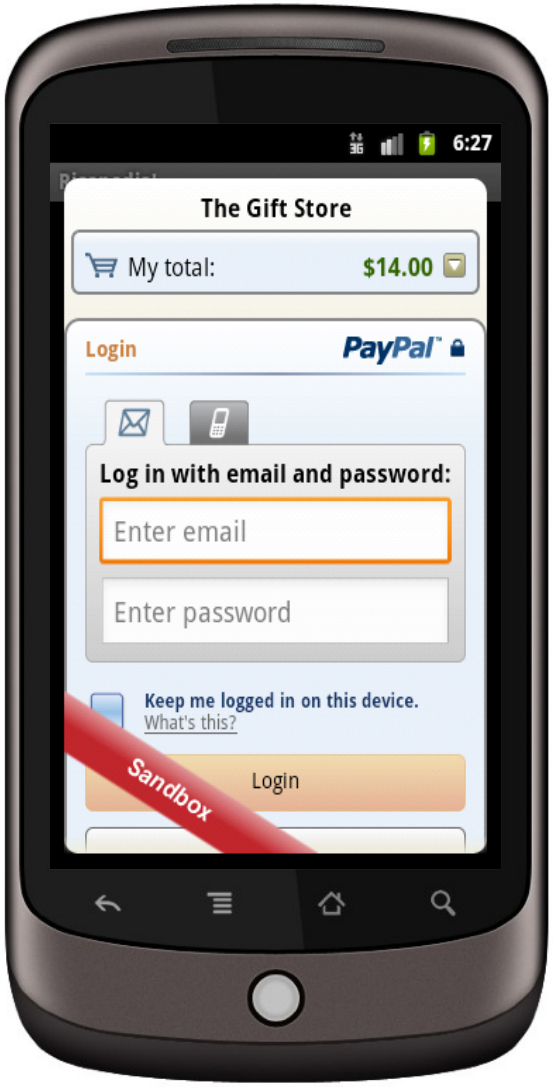


- Visual clues
- URL indicates this is PayPal
- SSL indicators

Motivation: Mobile Payment



Motivation: Mobile Payment



- No visual clues that this is PayPal
- Hosting app could spoof this dialog
- PayPal code running in same context as hosting app
- Hosting app needs internet permission to use this library

Our Approach: QUIRE

Quire n.

2. A collection of leaves of parchment or paper, folded one within the other, in a manuscript or book.

Two methods

- Protect apps from third party libs
  - Enforce application separation
  - Augment IPC with call chain provenance
- Protect third party services from apps
  - Data integrity from OS verified statements
  - Trusted path for user input
  - Attestation of on-phone state to remote endpoints

Trusted OS

- Mutually untrusting apps use OS to authenticate IPC
- Register shared secret between app and OS
- IPC messages signed with shared secret
- Authenticated later by OS service
- Network service creates RPC attestations

Caveat: Rooted phones can't be trusted

Principals

- On-Phone principal
  - UID assigned at install time, unambiguous
- Off-Phone principal
  - Can't use UID, only meaningful on phone
  - Resolve UIDs to signing key or package name
  - Signed by OS

MAC functions

- Built from crypto hash functions (e.g., HMAC-SHA1)
  - Radically faster than digital signatures
- Require a shared secret
  - Can only be verified if you know the secret
- Our approach: each app shares secret with the kernel
  - Requires syscall to verify a MAC

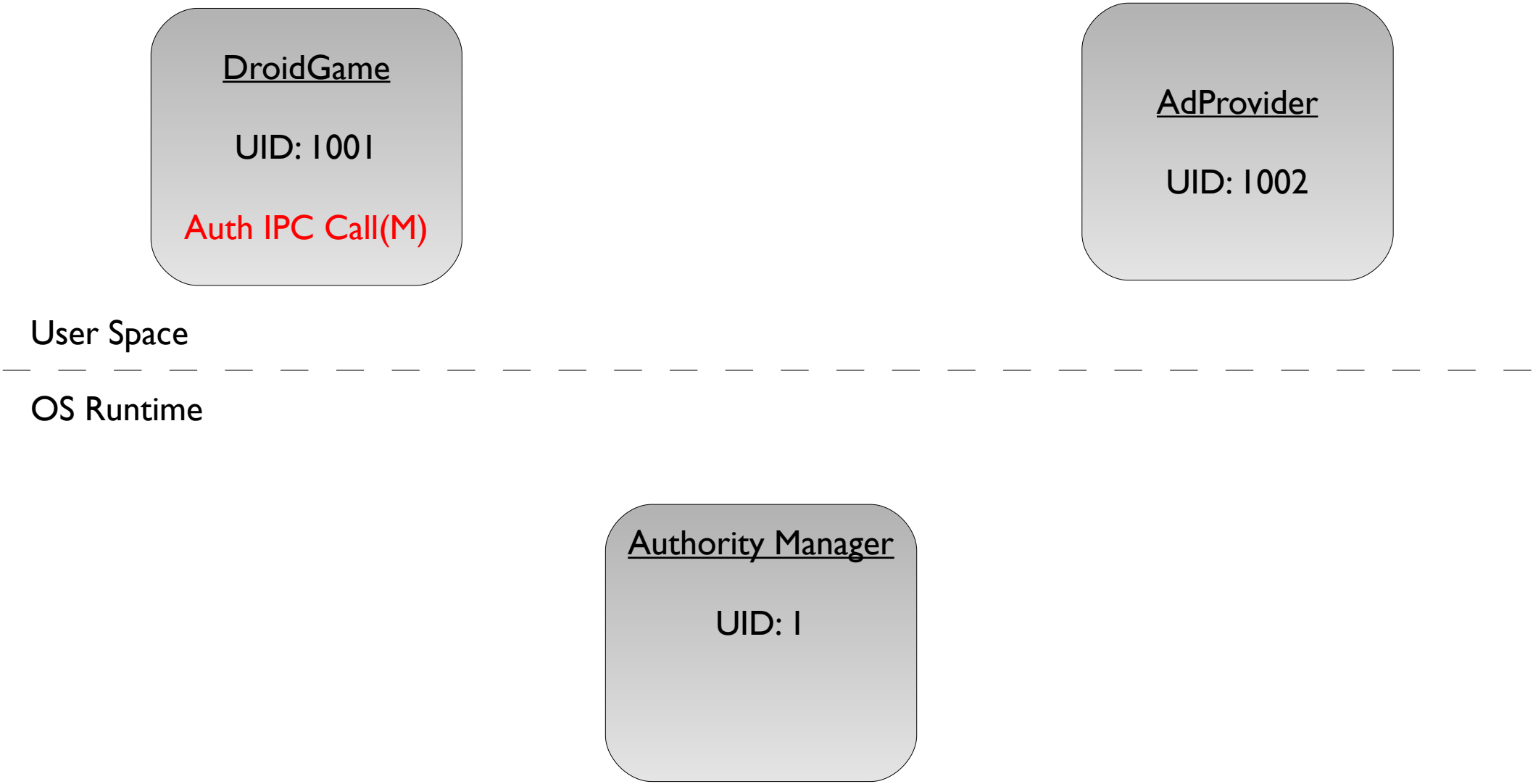
Verifiable Statements

- Fundamental unit of authenticated communication
  - Attached to objects passed over IPC
- Verified with a MAC over the principal's secret shared with the OS
- OS authentication service can verify a statement upon request

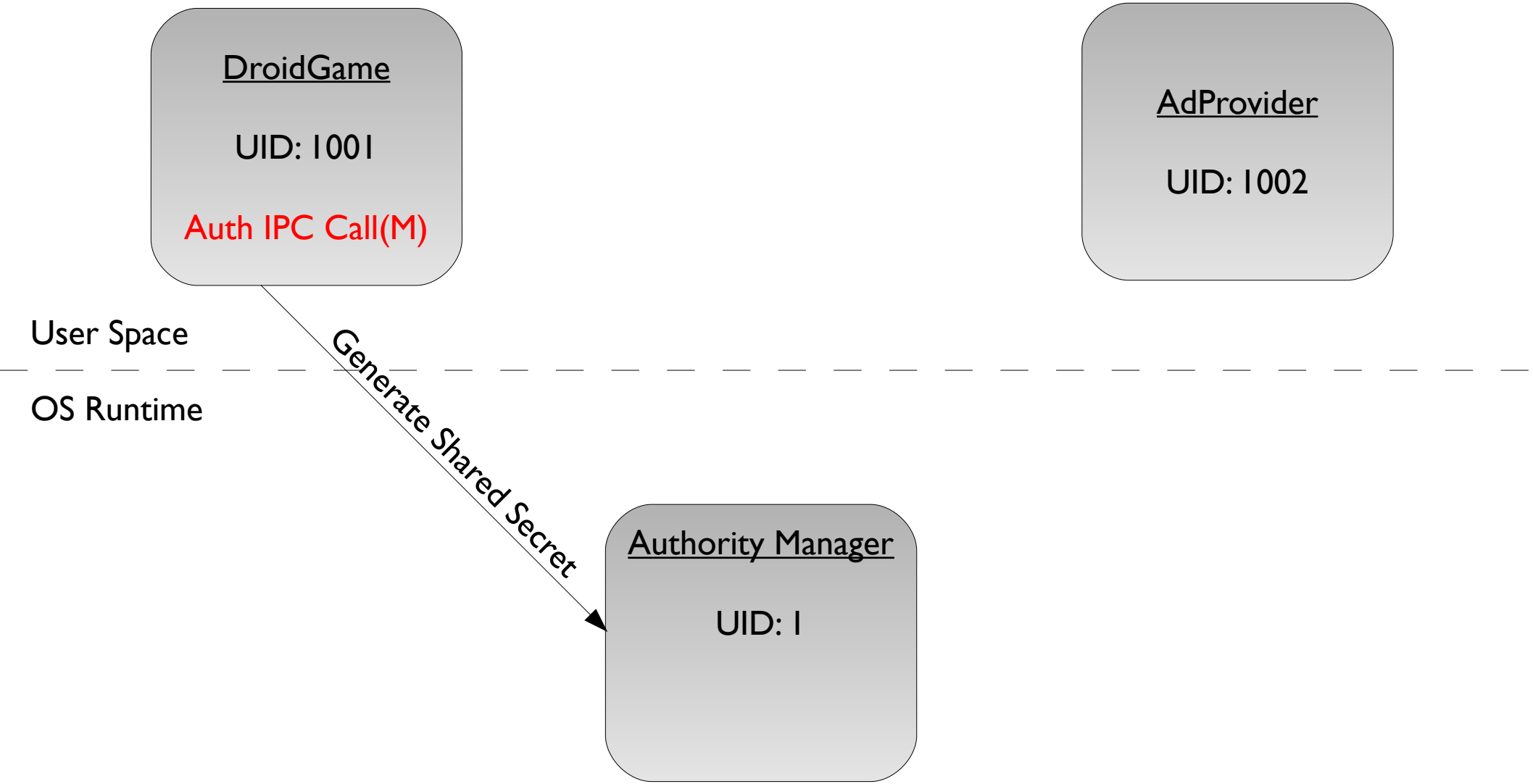
Contrast: Java stack inspection

- Same ABLP logic for the design
  - Quoting chains to reduce privileges
- Annotating Java stack vs. Android IPCs
  - Much lower overhead on Android
  - Supports Android native code transparently
  - Requires recompiling Android apps

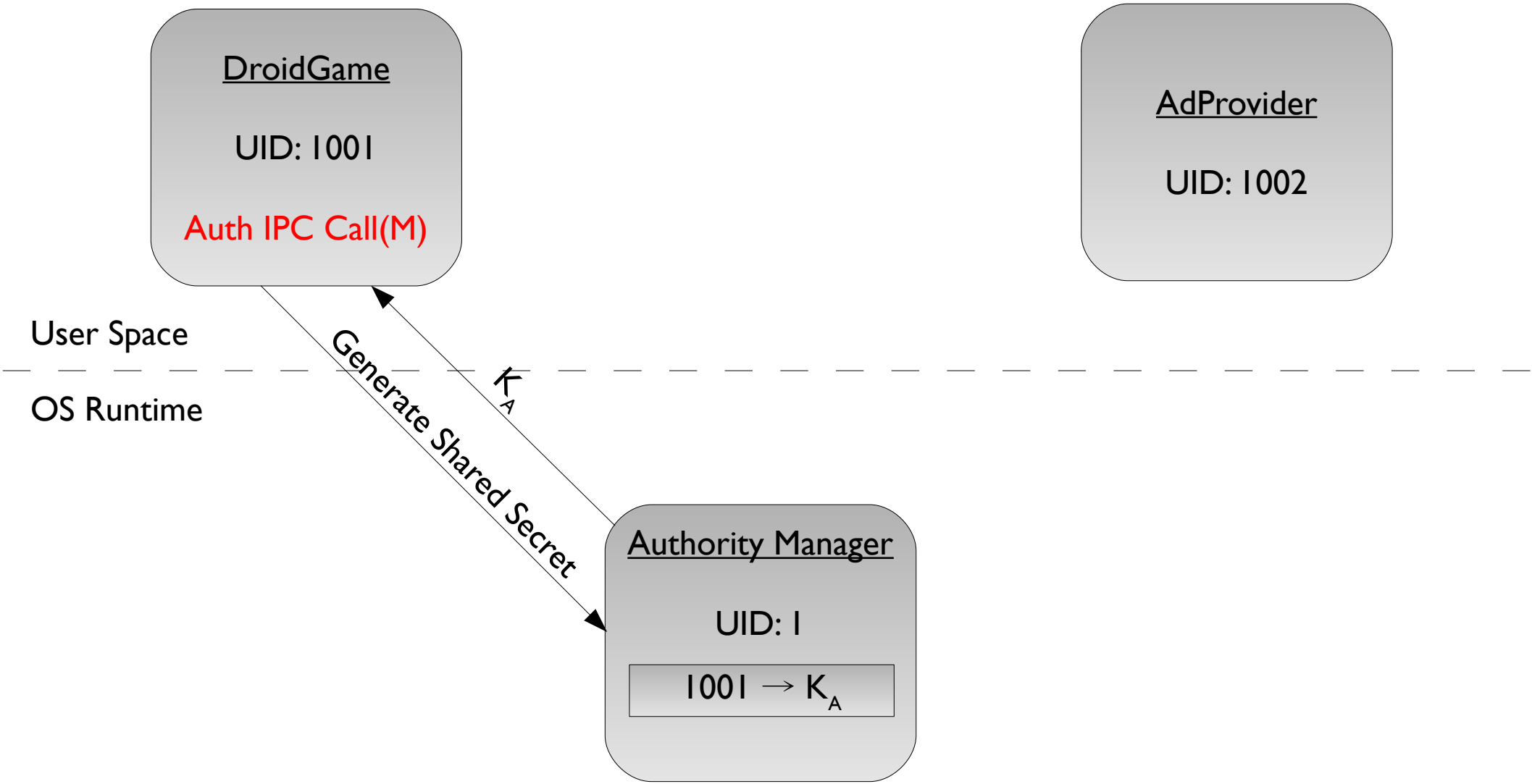
Statement Creation



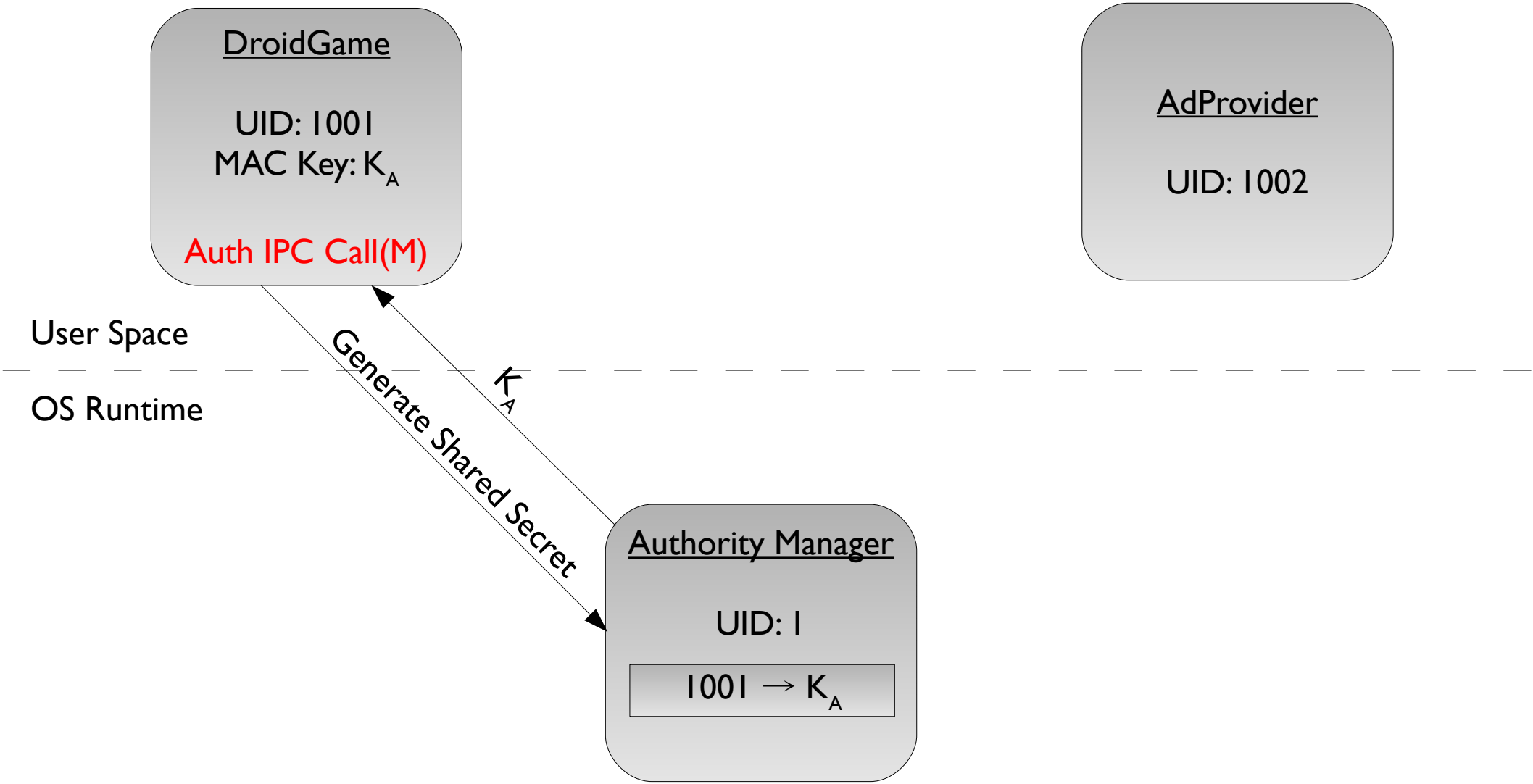
Statement Creation



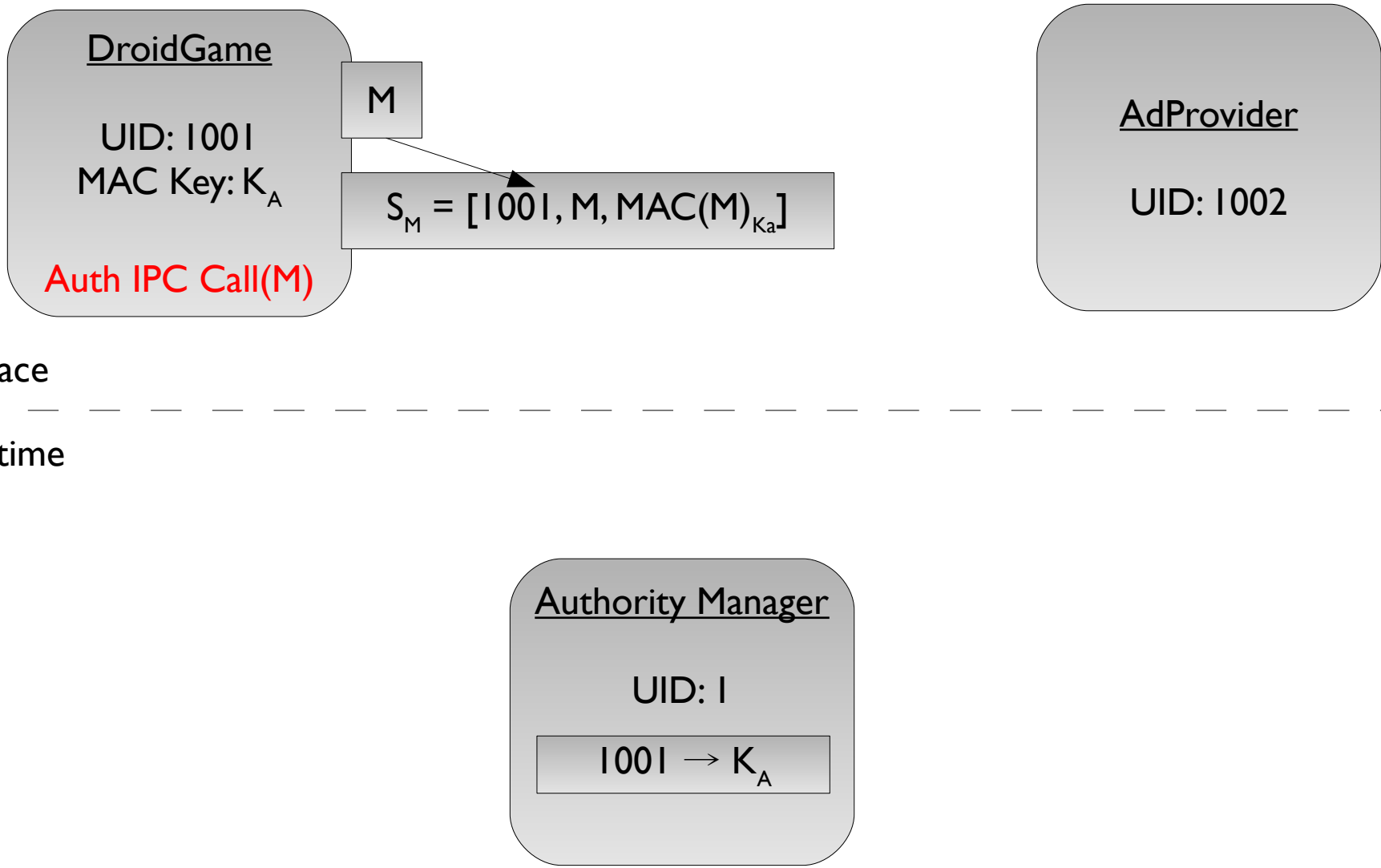
Statement Creation



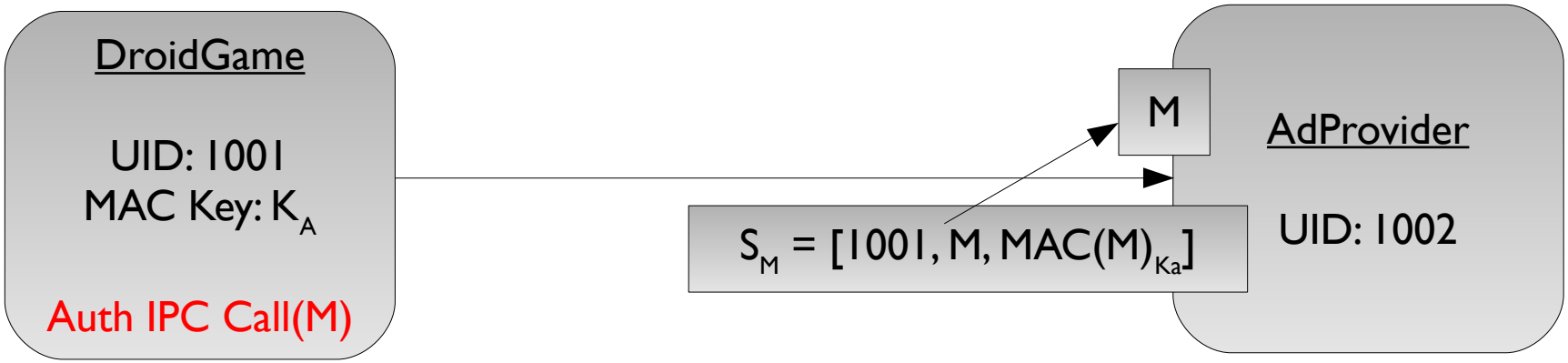
Statement Creation



Statement Creation

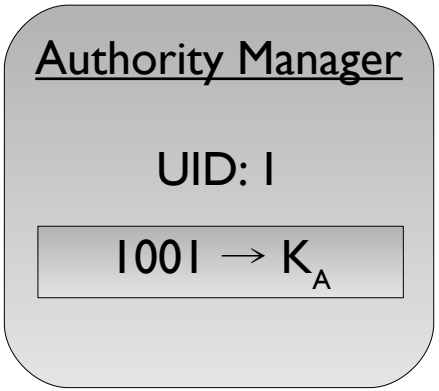


Statement Creation

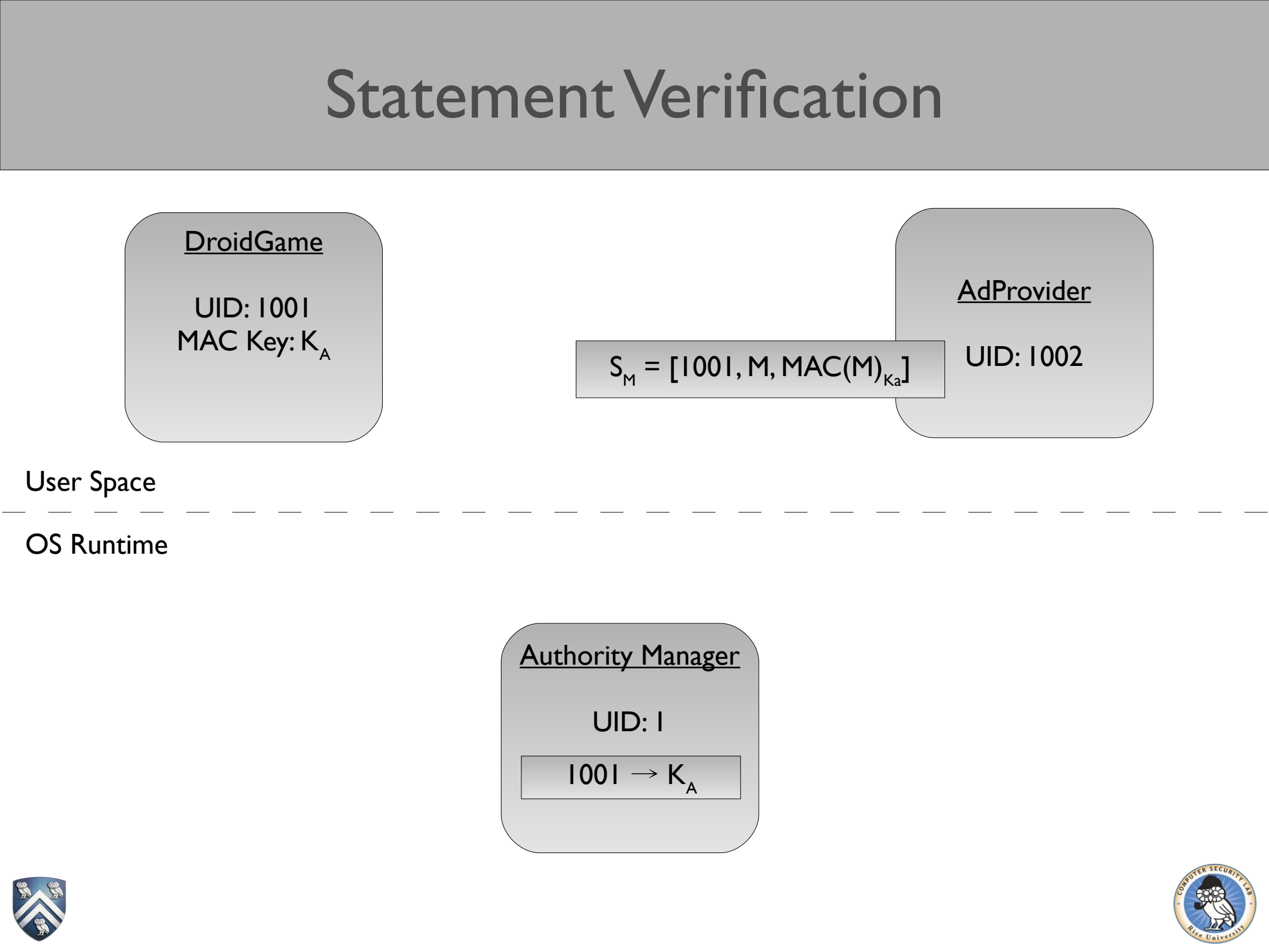


User Space

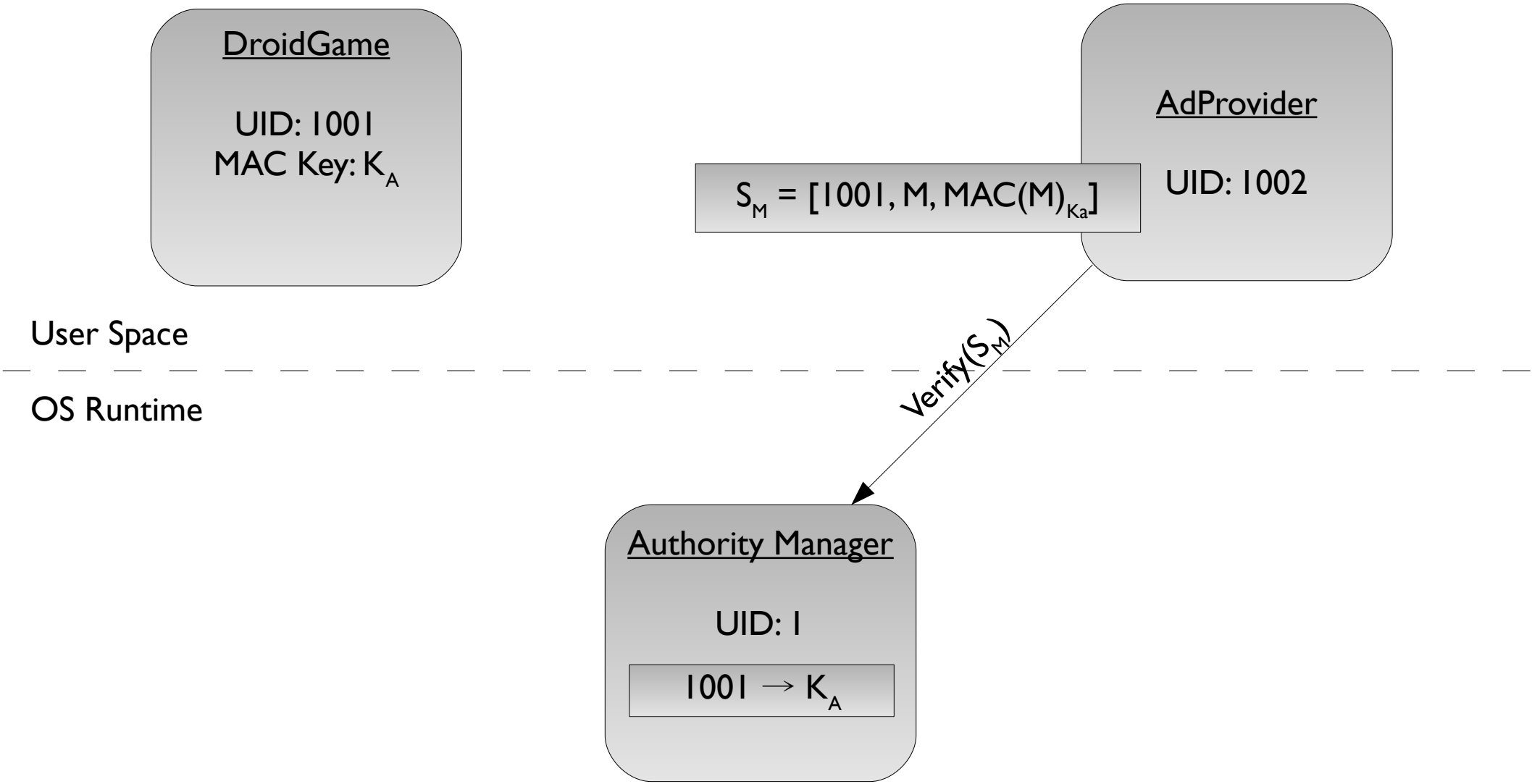
OS Runtime



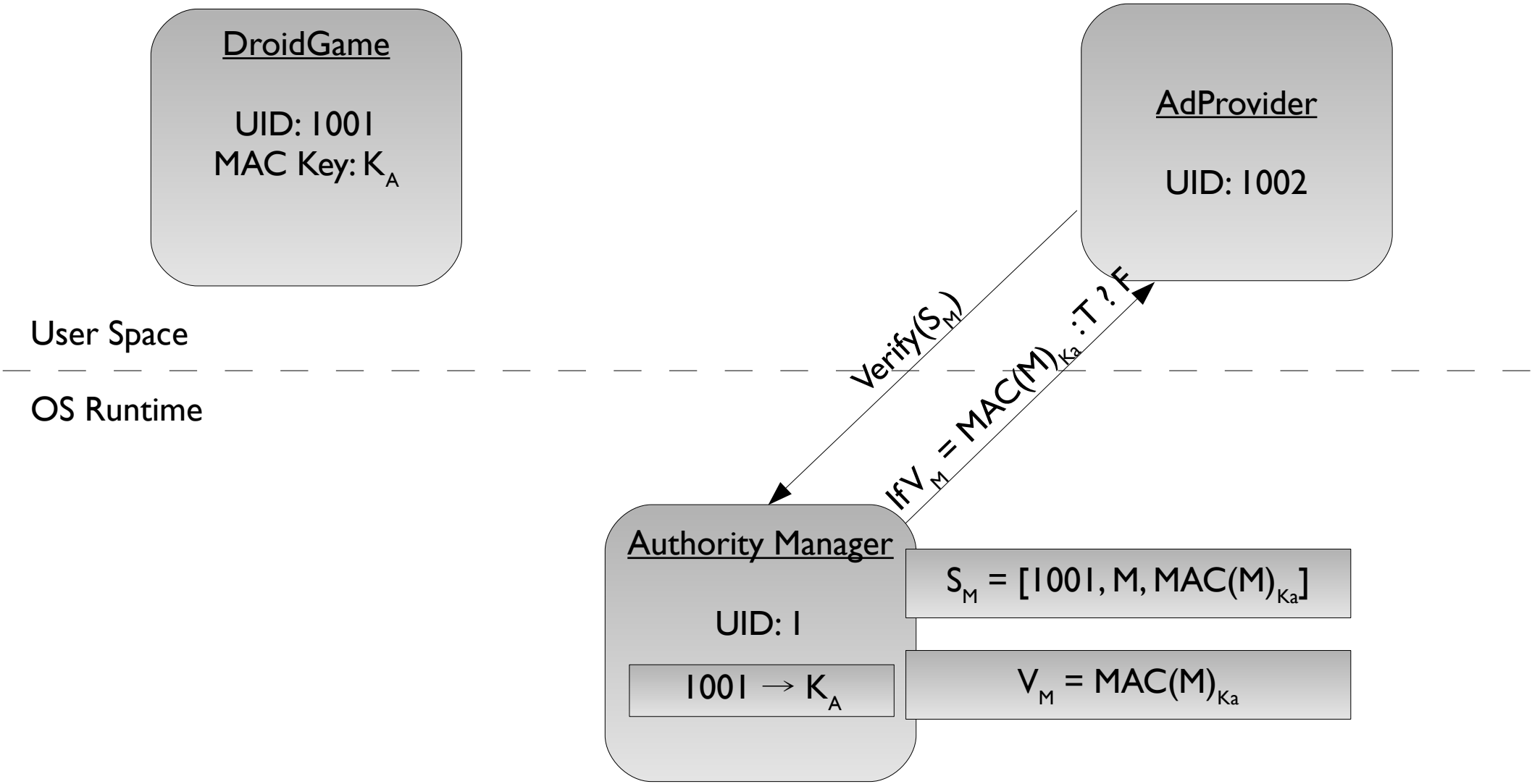
Statement Verification



Statement Verification



Statement Verification



Ad Example - Conclusions

- Sample policy
  - Assume Ad Payout **if** *AdProvider* can verify a click originated from the OS
  - *OS UI Manager* → *an app* → *AdProvider app* → RPC to *AdProvider.com*

Confused Deputy Attacks

- Example
  - Some App C protects the user's fine grained GPS location
  - App B has permission to retrieve the GPS information from C, but App A does not
  - A can escalate its privilege by triggering B to act on A's behalf

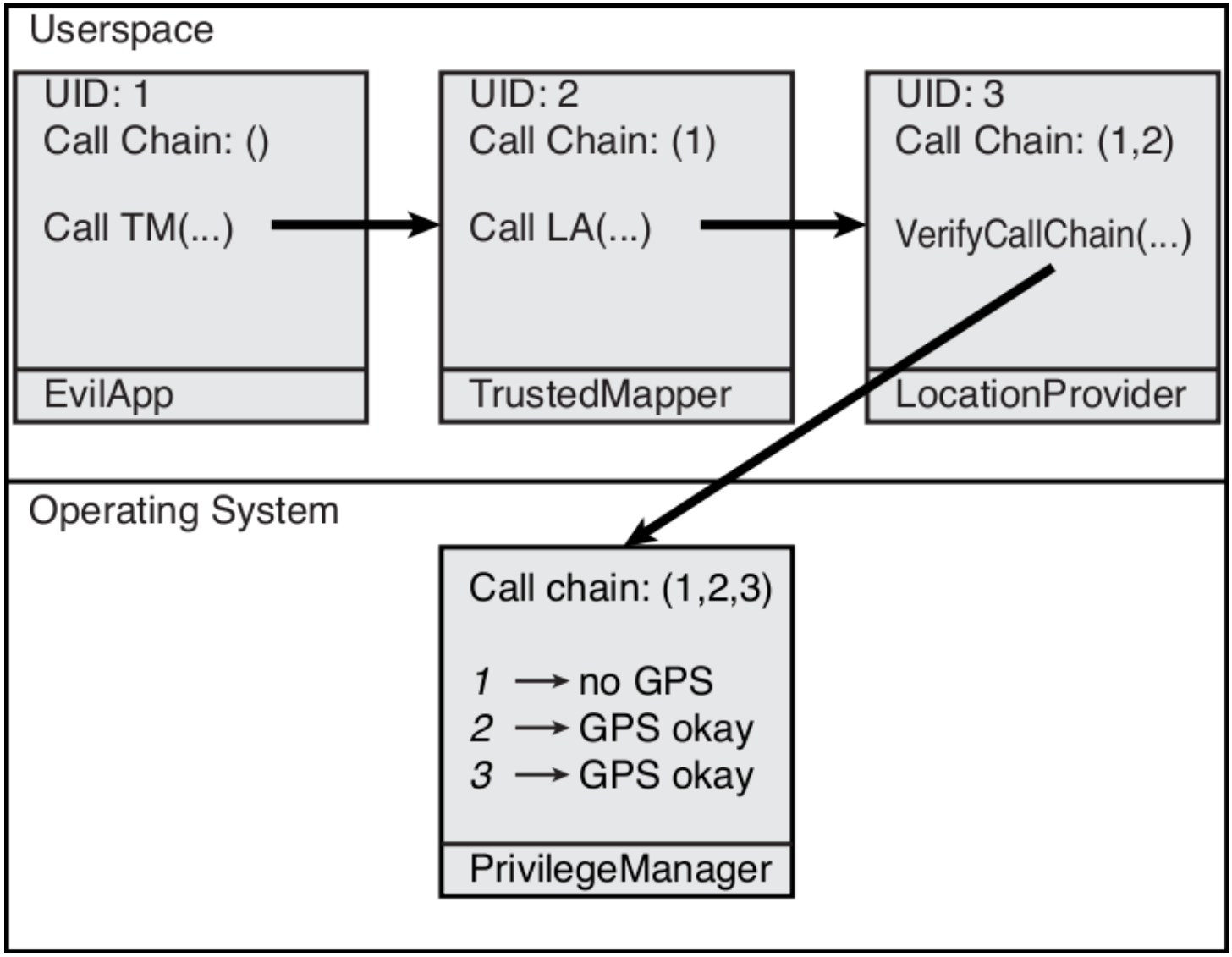
Augmented IPC

- An app can protect itself by quoting the app or call chain that called it
  - "I want access to the GPS, but only because A asked me to get it"
- We use Abadi, Burrows, Lampson, and Plotkin (ABLP) Logic to reason about this

Key Point

- Quoting can only reduce an app's privilege level
  - *B says A says X* means
    - B told you "don't just do it because you trust me, make sure you're okay with A as well."
    - Insight: quoting strictly reduces privileges
- Therefore there's no need to encrypt or MAC the call chain (cheap!)
- MAC-based statements when necessary (also cheap)

Confused Deputy Example



Implementation

- 2300 lines of modifications to the Android 2.3 OS runtime libraries
  - Main Components
    - Authority Manager OS service
    - IPC stub/proxy code generator
    - Network Provider
  - SHA-1 HMAC used for message authentication
    - Implemented as an Intrinsic VM function to avoid Java / JNI slowdown

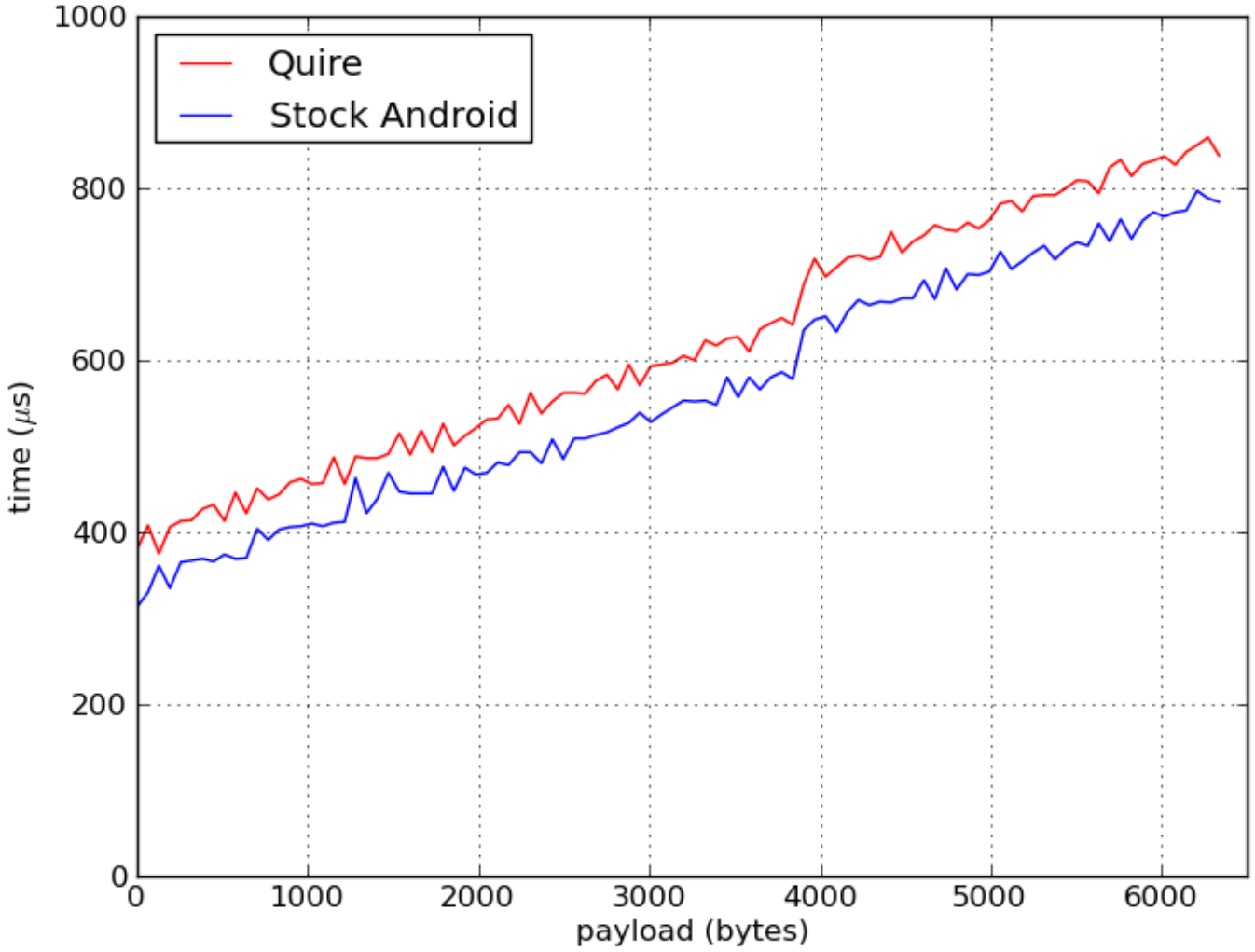
Experimental Methodology

- All experiments performed on Nexus One dev phone running Android 2.3 OS
  - 1 Ghz ARM core
  - 512 MB RAM
  - 512 MB Flash
  - Replacement of CPU intensive live wallpaper with static image to normalize CPU load
- "Stock" Android – 2.3 Gingerbread from 12/23/10
- "Quire" build based on same source with modifications

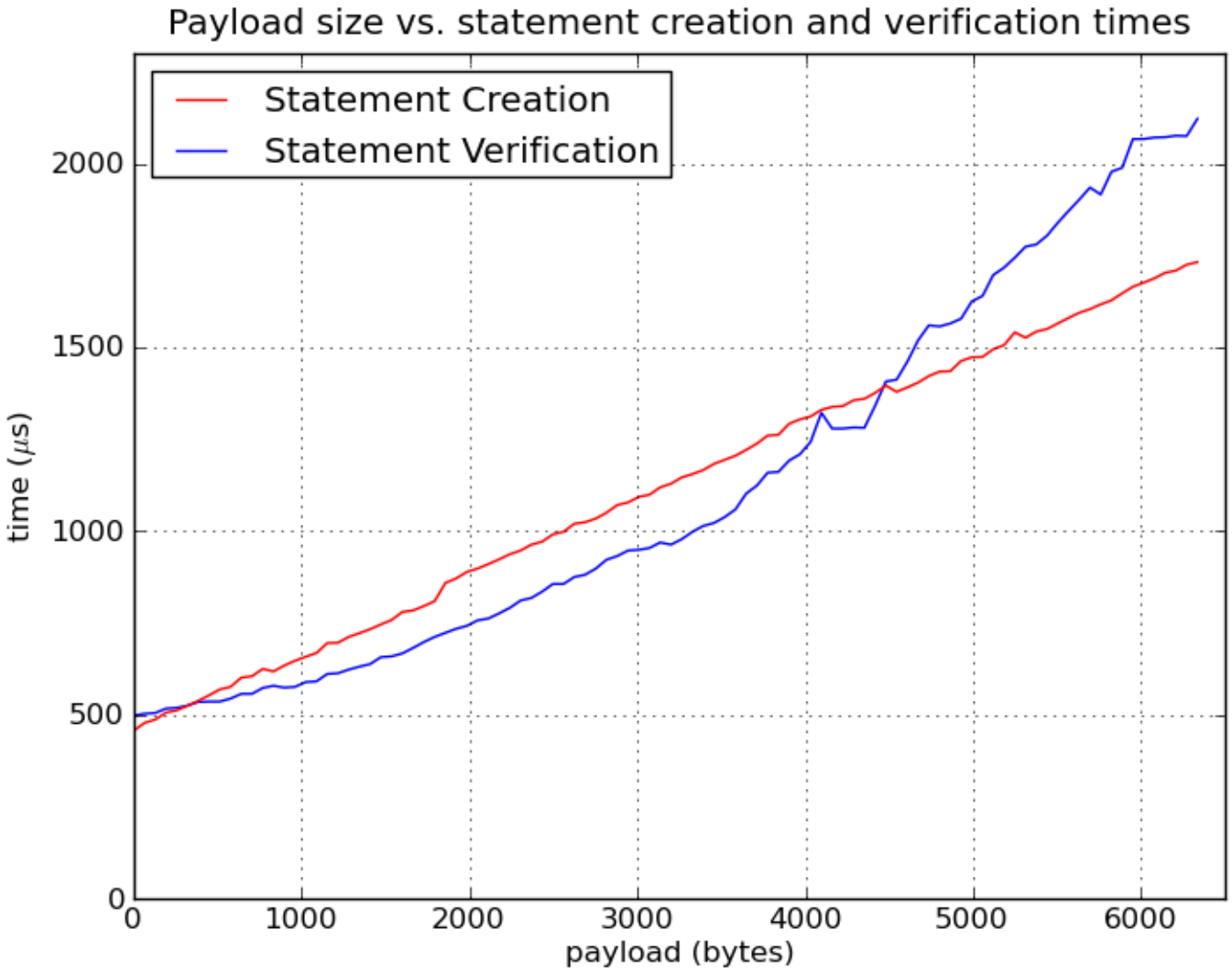
Microbenchmarks

- Purpose
  - Analyze augmented IPC performance as it compares to stock Binder IPC calls
  - Measure verifiable statement creation and verification
- Methodology
  - Average of 10k operations with varying payload sizes and statement depths
  - Issues: Garbage Collection, JIT Compiler, Imprecise timer

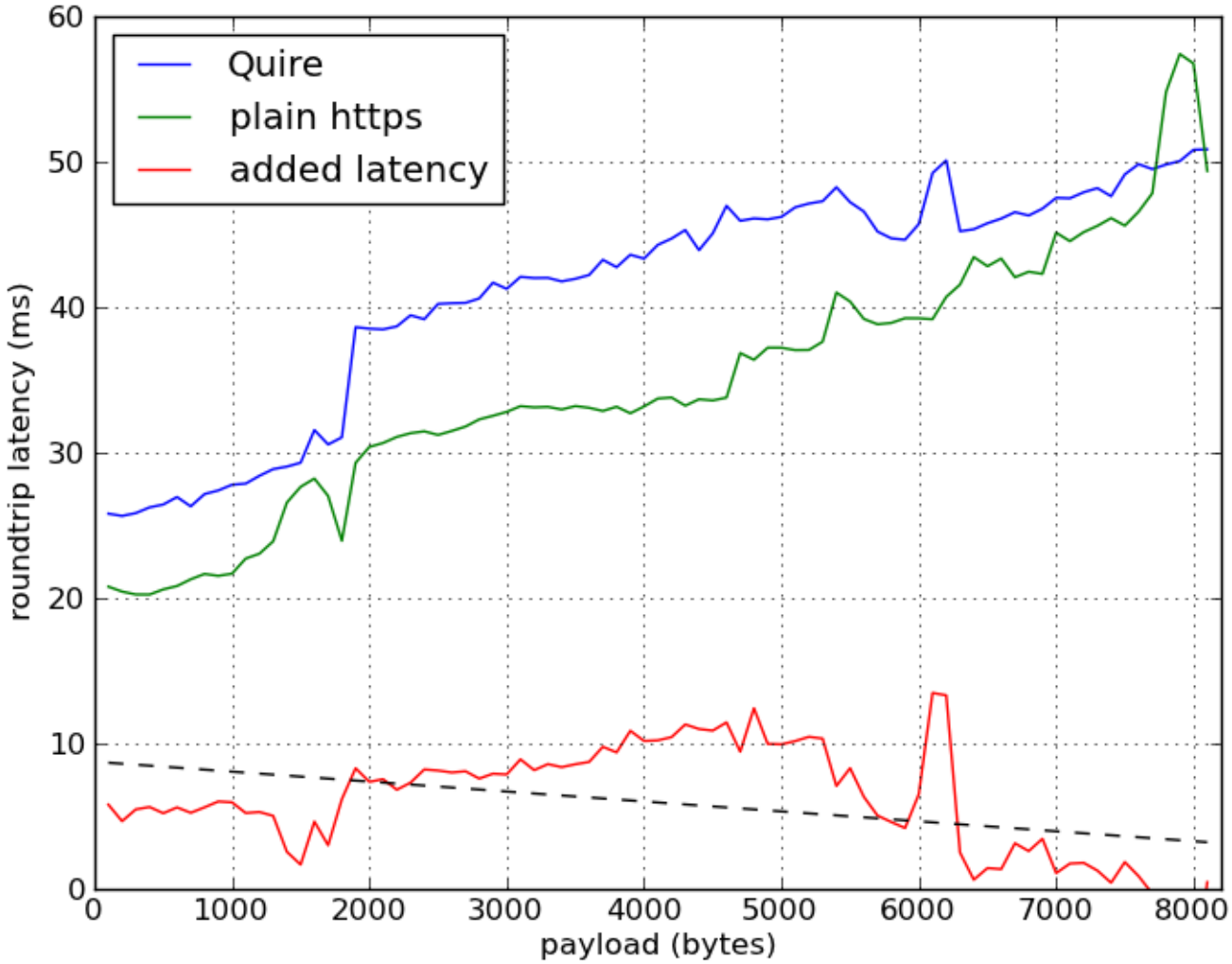
Augmented IPC Performance



Authenticated Statement Performance



Authenticated RPC Performance



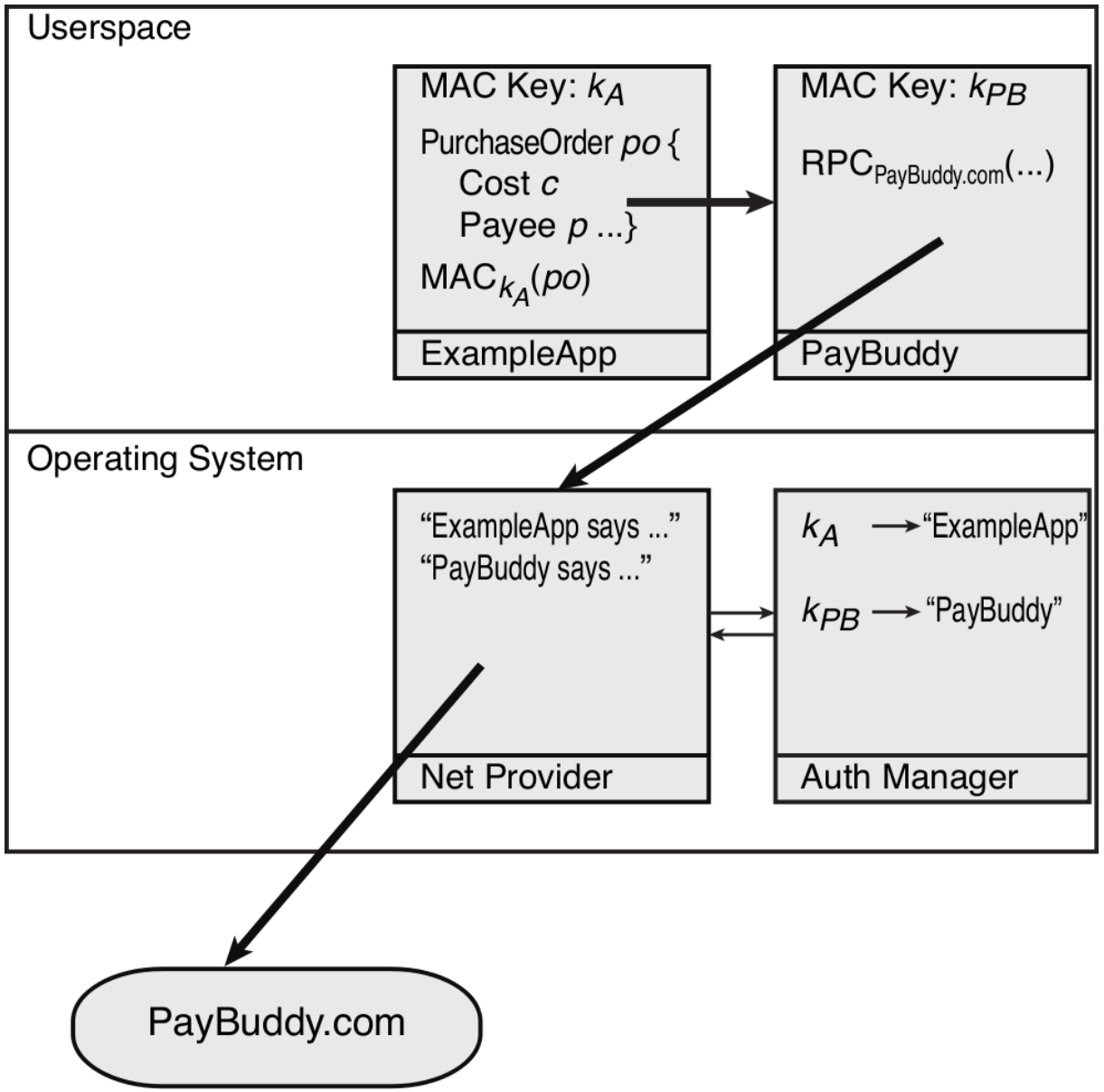
Analysis

- Provenance carrying IPC results in 20% overhead vs. stock Android (no crypto, cheap)
- Verification and creation of crypto statements are also cheap (symmetric crypto)
- Microbenchmarks represent worst case scenario

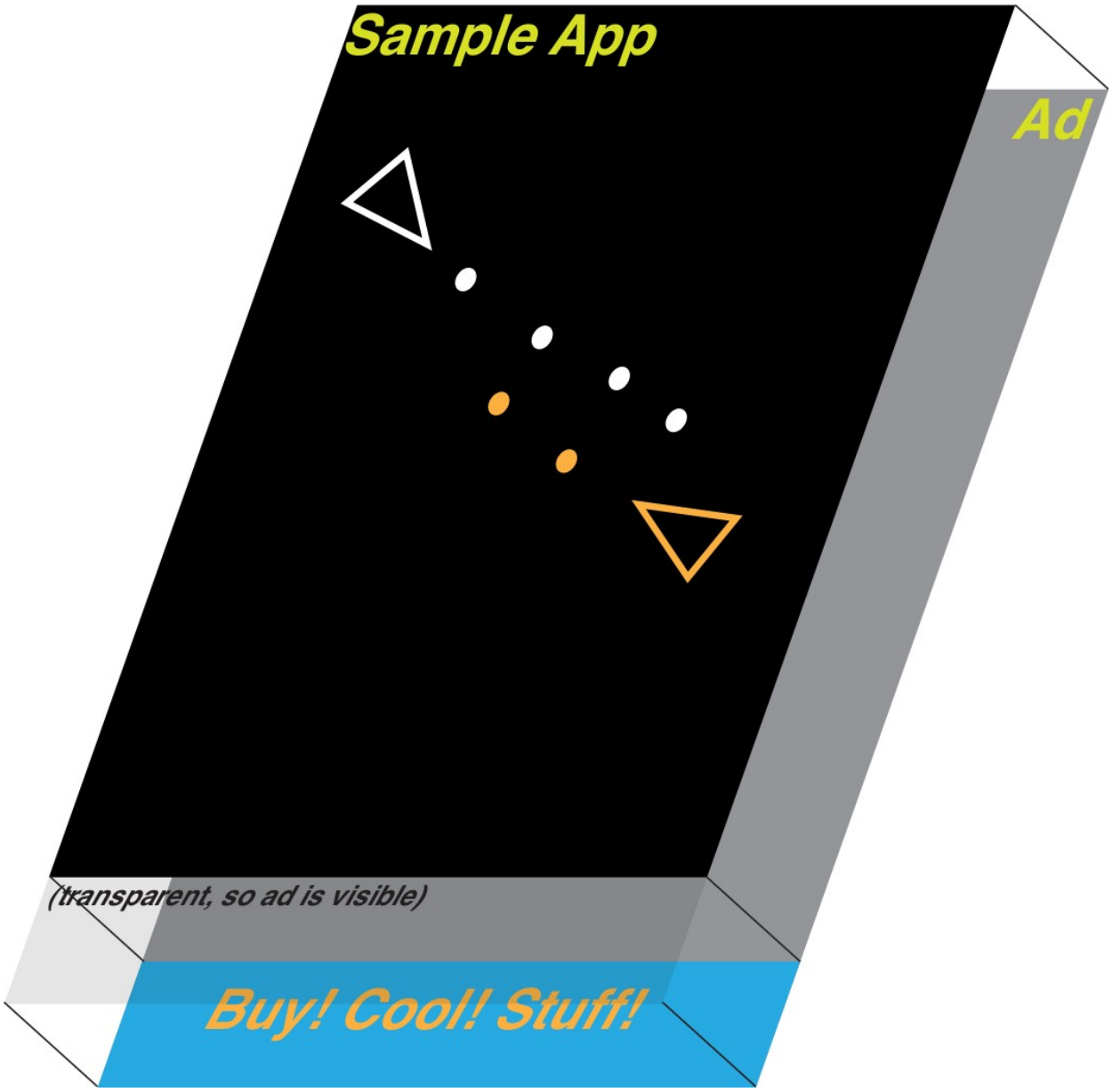
Applications built with Quire

- PayBuddy
  - Secure mobile payment system
  - IPC call chain and data integrity communicated to remote endpoint
- Mobile Ad System
  - Verifiable clicks from OS
  - UI mechanisms to detect pop-overs

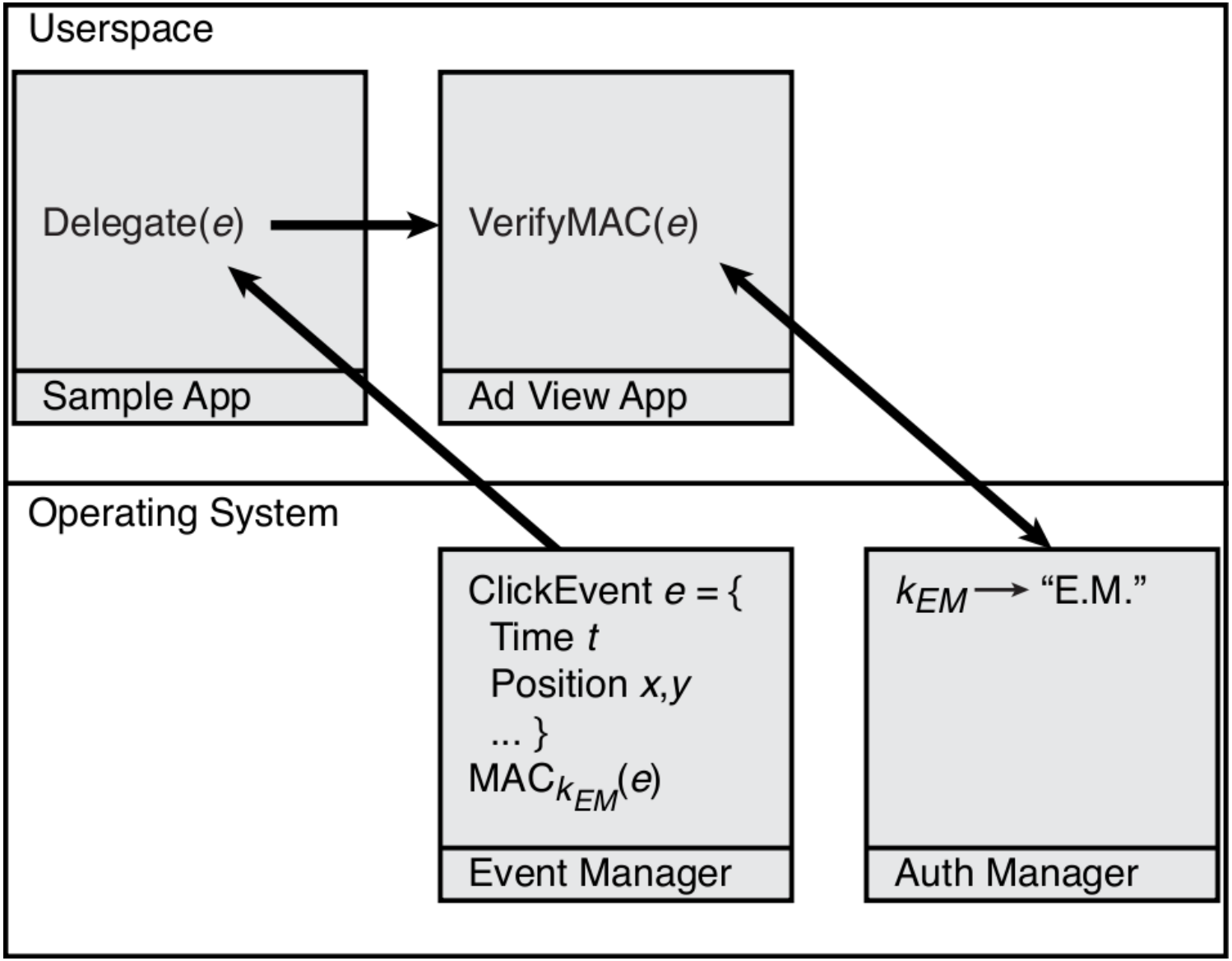
PayBuddy


Userspace
MAC Key: k_A
PurchaseOrder po {
Cost c
Payee p ...}
$MAC_{k_A}(po)$
ExampleApp
MAC Key: k_{PB}
$RPC_{PayBuddy.com}(...)$
PayBuddy
Operating System
"ExampleApp says ..."
"PayBuddy says ..."
Net Provider
k_A → "ExampleApp"
k_{PB} → "PayBuddy"
Auth Manager
PayBuddy.com

Mobile Ad System



Mobile Ad System



Related Approaches

- Define sensitive resources and protect them
  - Taint tracking
  - Stack inspection
- Statically verify third party code
  - Google's Native Client
  - Java bytecode verification
- Process-like isolation
  - Chrome, Gazelle

Taint Tracking: Overview

- Define taint sources
  - Private app data, GPS info, IMEI, user input, etc.
- Define taint sinks
  - Internet, Bluetooth, display...
- Propagate taint from a source to all objects that touch the source, repeat.
- If a tainted object ever hits a taint sink, yell!

Taint Tracking: Issues

- Requires annotation of application code
- Native code can strip taint
- Definition of a taint source?
- False positives

Static Analysis

- Google's Native Client
  - Statically analyzes code to prevent buffer overflow, return-to-libc, and other shellcode attacks against x86 code
  - Only works on x86 (for now), requires special libraries + recompilation of existing code

Process-like Separation

- Android and modern browsers closely related
  - Mutually untrusting principals
  - Third party code
  - Application / Origin segregation
- Same class of problems
  - Click fraud
  - Credential phishing
  - Ad driven services
- Notable: HTTP "Origin" header similar to our statements

Conclusions

Quire:

- Adds provenance to IPC and RPC
- Enables true application separation + apps as services
- Useful to applications and third party services
- Minimizes coding effort

Future directions

- Integrate static analysis? Foundation for web browser security?