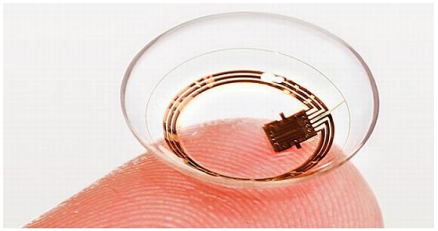
RUTGERS
THE STATE UNIVERSITY
OF NEW JERSEY

Regulating Smart Devices in Restricted Spaces

Daeyoung Kim
Department of Computer Science, Rutgers University

Committee: Prof. Vinod Ganapathy (Chair), Prof. Badri Nath, Prof. Abhishek Bhattacharjee, Dr. Pratyusa Manadhata (Micro Focus)

Devices are everywhere



Devices are increasingly capable

Model	CPU (GHz)	Screen (1000x)	Rear camera	Front camera	Battery (mAh)	Sensors other than Camera/Microphone
iPhone	0.4	153	2MP	-	1,400	3 (light, accelerometer, proximity)
iPhone3	0.6	153	3MP	-	1,150	4 (+= compass)
iPhone4	0.8	614	5MP	0.3MP	1,420	6 (+= gyroscope, infrared)
iPhone5	1.3 (2 cores)	727	8MP	1.2MP	1,560	7 (+=fingerprint)
iPhone6	2.0 (2 cores)	1000	12MP	5.0MP	1,715	8 (+= barometer)
iPhoneX	2.39 (6 cores)	2740	12MP	7MP	2,716	9 (+= face recognition)

How can devices be misused?

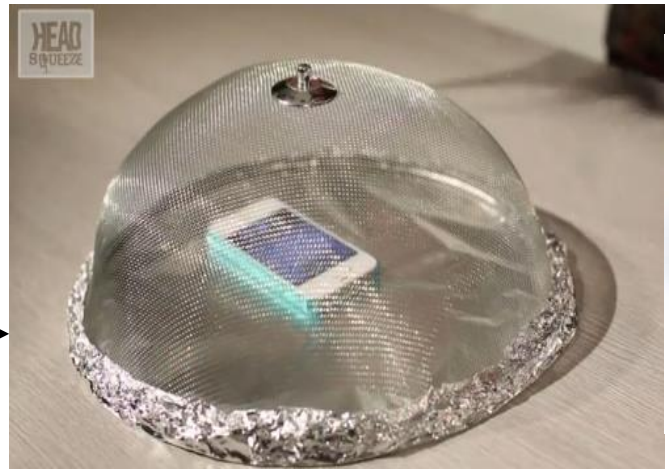
- Malicious end-users can leverage sensors to exfiltrate of infiltrate unauthorized data

- Malicious apps on devices can achieve similar goals even if end-user is benign

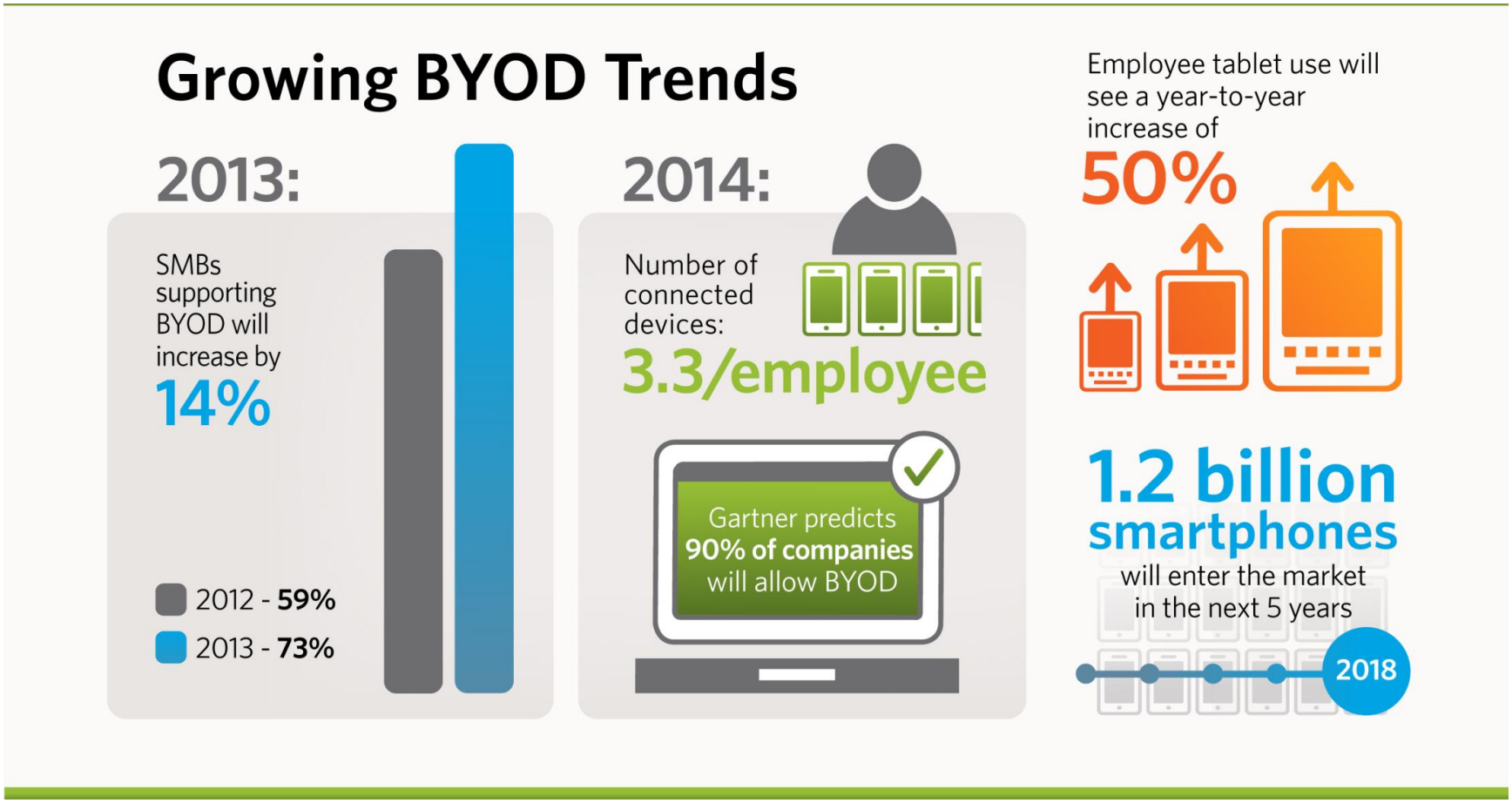
Government or corporate office

- Problem: Sensitive documents and meetings can be ex-filtrated using the camera, microphone, and storage media
- Current solution: Physical security scans, device isolation

Faraday cages →



Challenge: Bring Your Own Device



Classroom and exam setting

- Problem: Personal devices can be used to infiltrate unauthorized information

[NY Times July 2012]

N.Y. / REGION

At Top School, Cheating Voids 70 Pupils' Tests

By AL BAKER JULY 9, 2012

Email

Share

Seventy students were involved in a pattern of smartphone-enabled cheating last month at Stuyvesant High School, New York City officials said Monday, describing an episode that has blemished one of the country's most prestigious public schools.

The Telegraph

calcutta, india

Edition | Wednesday , May 6 , 2015 |

Front Page > Calcutta > Story

Like 13 Tweet G+1 0

Scanners catch JEE cheats

OUR BUREAU

Roving invigilators armed with signal scanners to detect mobile data and call traffic inside examination halls caught five JEE candidates using a smartphone or a smartwatch to cheat on the first day of the test.

[Financial Crypto 2014]

Outsmarting Proctors with Smartwatches:
A Case Study on Wearable Computing Security

Alex Migicovsky, Zakir Durumeric, Jeff Ringenberg, and J. Alex Halderman

Classroom and exam setting

- Current solution: Deterrence via rules and threats. Invigilation to ensure compliance

NO MOBILE PHONES, iPODs, MP3/4 PLAYERS.

NO PRODUCTS WITH AN ELECTRONIC COMMUNICATION/STORAGE DEVICE OR DIGITAL FACILITY.

Possession of unauthorised items is an infringement of the regulations and could result in

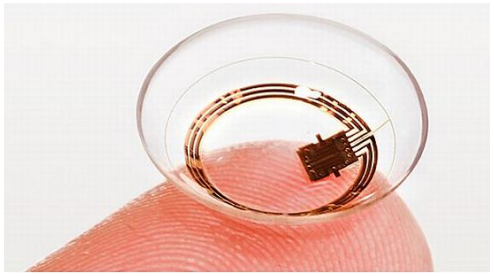
DISQUALIFICATION

from the current examination and the overall qualification. Candidates are advised that mobile phones in particular **must not** be in their possession whether switched on or not.

This poster must be displayed in a prominent place both inside and outside each examination room.

Challenge: Assistive devices

- Students may wish to use devices for legitimate reasons:
  - Smart glass or contacts for vision correction
  - Bluetooth-enabled hearing aids
  - Smart watches to monitor time



Other social settings

- Restaurants, conferences, gym locker rooms, private homes, …
- Problems:
  - Recording private conversations
  - Pictures of individuals taken and posted to social networks without their consent
  - Pictures and videos of otherwise private locations, e.g., private homes

Other social settings

- Current solutions: Informal enforcement
- Challenge: Social isolation

"For the first time ever this place, Feast, in NYC just asked that I remove Google Glass because customers have complained of privacy concerns […] I left"

Malicious apps exploiting sensors

Sensory malware

(sp)iPhone: Decoding Vibrations From Nearby Keyboards Using Mobile Phone Accelerometers

Philip Marquardt*
MIT Lincoln Laboratory
244 Wood Street, Lexington, MA USA
philip.marquardt@ll.mit.edu

Arunabh Verma, Henry Carter and Patrick Traynor
Georgia Institute of Technology
{arunabh.verma@, carterh@, traynor@cc.}gatech.edu

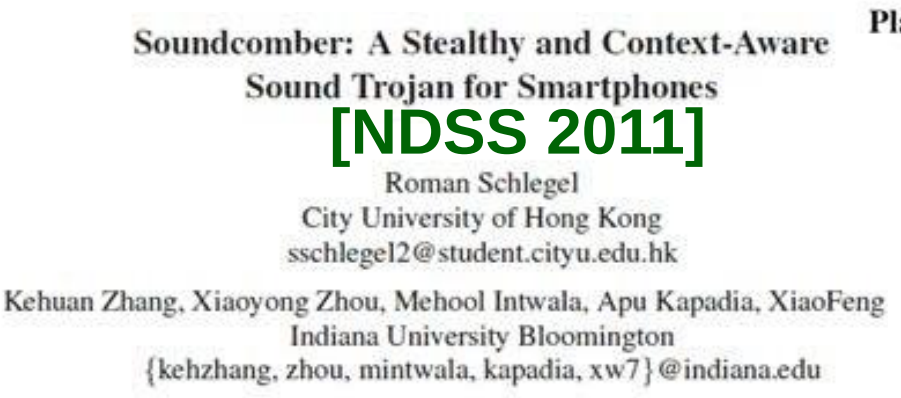
Figure 1: Our experimental placement of a mobile phone running a malicious application attempting to recover text entered using the nearby keyboard.

← Early example of sensory malware **[CCS 2011]**

- Use accelerometer and record keystroke press vibrations
- Up to 80% accuracy in word recovery

Malicious apps exploiting sensors

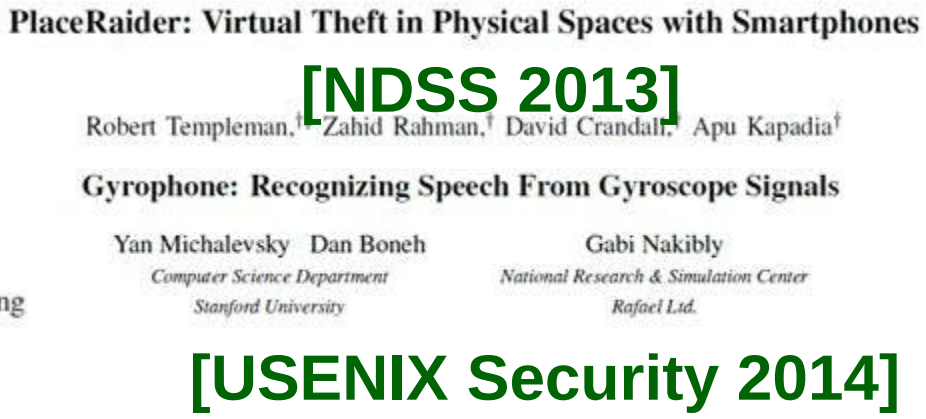
Sensory malware


Soundcomber: A Stealthy and Context-Aware Sound Trojan for Smartphones

[NDSS 2011]

Roman Schlegel
City University of Hong Kong
sschlegel2@student.cityu.edu.hk

Kehuan Zhang, Xiaoyong Zhou, Mehool Intwala, Apu Kapadia, XiaoFeng
Indiana University Bloomington
{kehzhang, zhou, mintwala, kapadia, xw7}@indiana.edu


PlaceRaider: Virtual Theft in Physical Spaces with Smartphones

[NDSS 2013]

Robert Templeman,† Zahid Rahman,† David Crandall,‡ Apu Kapadia†

Gyrophone: Recognizing Speech From Gyroscope Signals

Yan Michalevsky Dan Boneh
Computer Science Department
Stanford University

Gabi Nakibly
National Research & Simulation Center
Rafael Ltd.

[USENIX Security 2014]

- Attacks have now been demonstrated using every imaginable sensor
- Attack accuracy will ***improve*** with each generation of devices and sensors

Claim

Smart devices will become integrated with daily lives → *Ad hoc* solutions, *e.g.*, banning device use, will no longer be acceptable

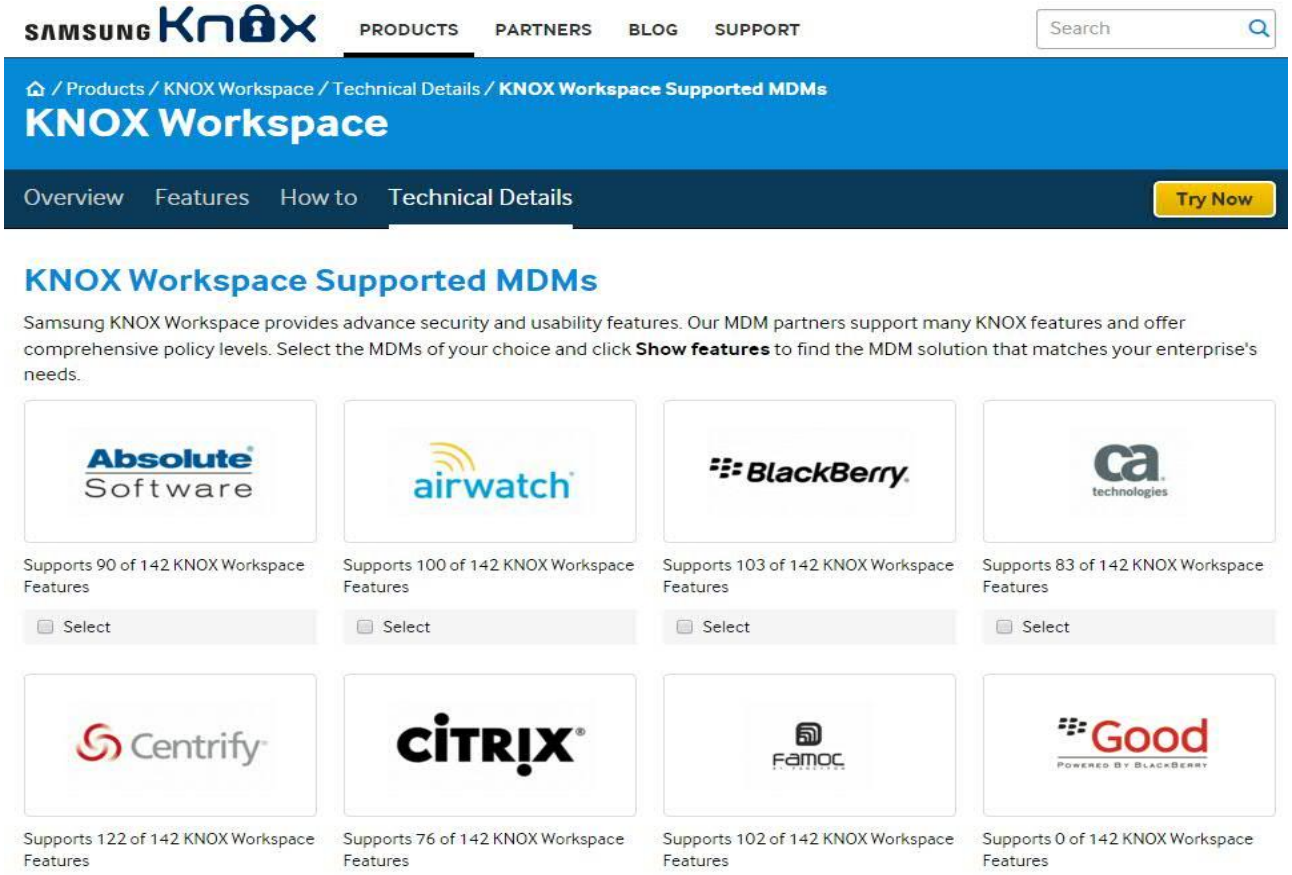
Vision

Need systematic methods to regulate devices and ensure responsible use

Discussion: Only considering **overt** device use. Covert use detection still requires traditional physical security measures.

What solutions exist today?

Mobile device management (MDM) solutions



Mobile device management

- Solution for enterprises that offer *Bring your own device* (BYOD) models
- Employees are given a mobile device outfitted with a secure software stack
- Enterprise policies “pushed” to device when employee changes device persona

Mobile device management

Main shortcoming of current MDM solutions

- Enterprise must trust software stack on guest device to enforce policies correctly
- But guest devices under control of possibly malicious end-users

- Solution for enterprises that offer *Bring your own device* (BYOD) models
- Employees are given a mobile device outfitted with a secure software stack
- Enterprise policies “pushed” to device when employee changes device persona

My thesis

We can leverage ARM TrustZone devices to build methods to regulate smart devices and ensure responsible use in restricted spaces.

Contributions

- Regulating ARM TrustZone Devices in Restricted Spaces **[MobiSys 2016]**
- ForceDroid: Enforcing Policy on Smart Devices in Restricted Spaces

RUTGERS
THE STATE UNIVERSITY
OF NEW JERSEY

Regulating Smart Devices with Remote Memory Operation

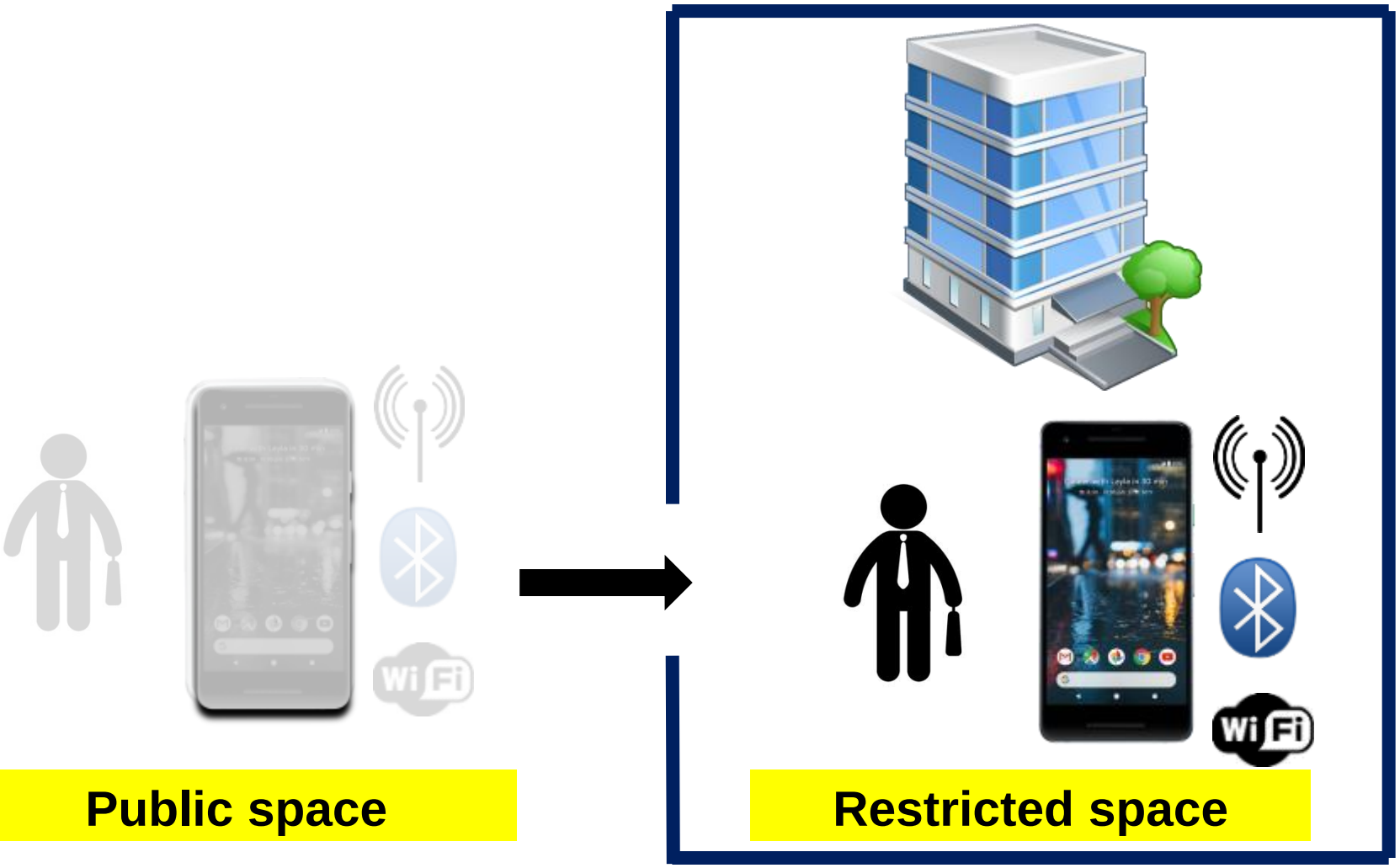
Contributions of our work

- **Restricted space**: Location owned by a **host**, where **guest devices** must follow the host's usage policies
- Enable guest devices to **prove** policy compliance to restricted space hosts
- Use a simple, low-level API that **reduces size of trusted computing base** on guest devices

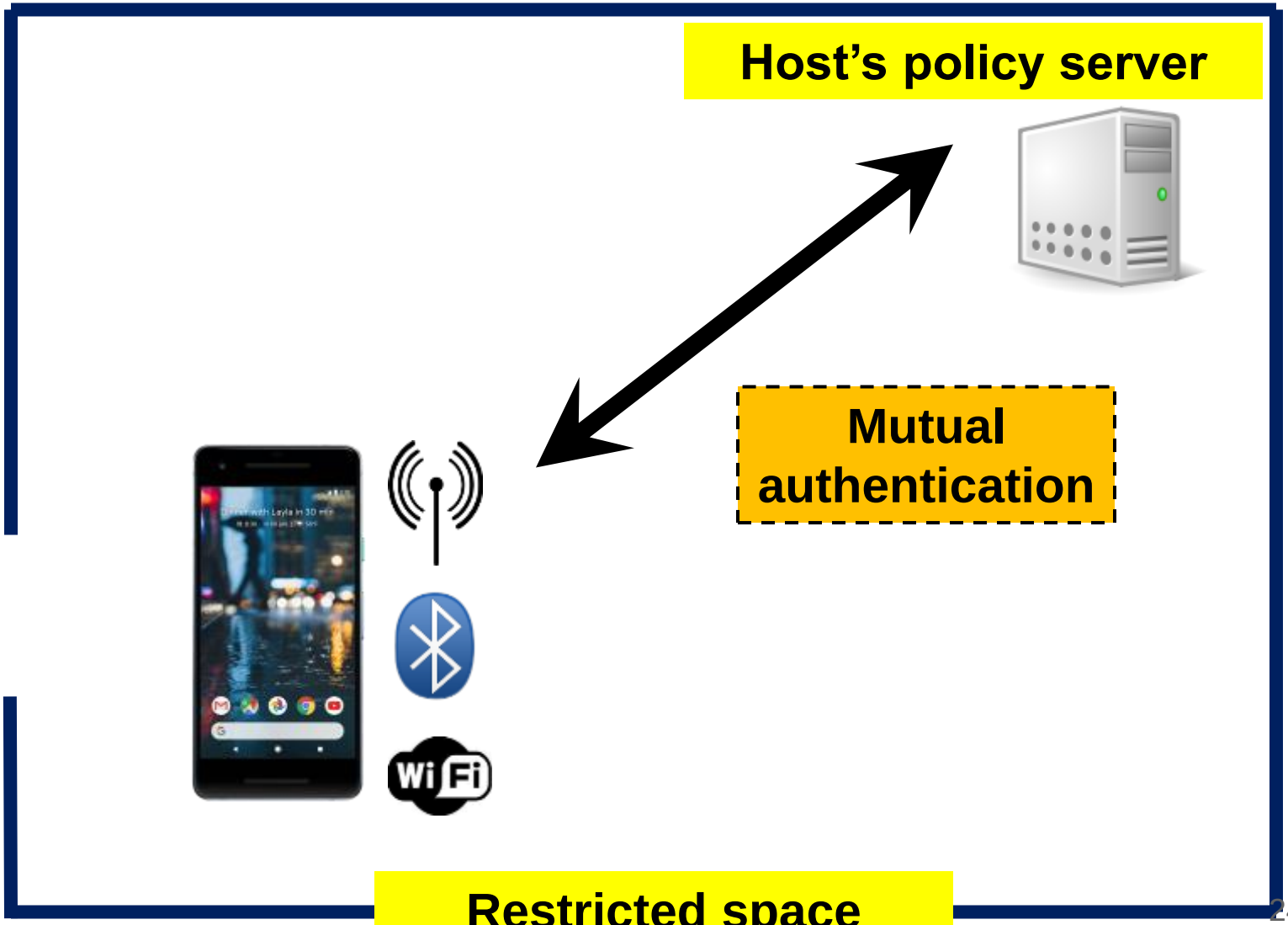
Threat model

- **Hosts and guests are mutually-distrusting**:
  - Hosts do not trust end-user of guest device or its end-user software stack
  - Guests do not trust host's *reconfiguration requests* to ensure policy compliance
- **Trusted hardware on guest devices**:
  - Guest devices equipped with ARM TrustZone
- **Guest devices are used overtly**:
  - Host must still use traditional physical methods to detect covert device use

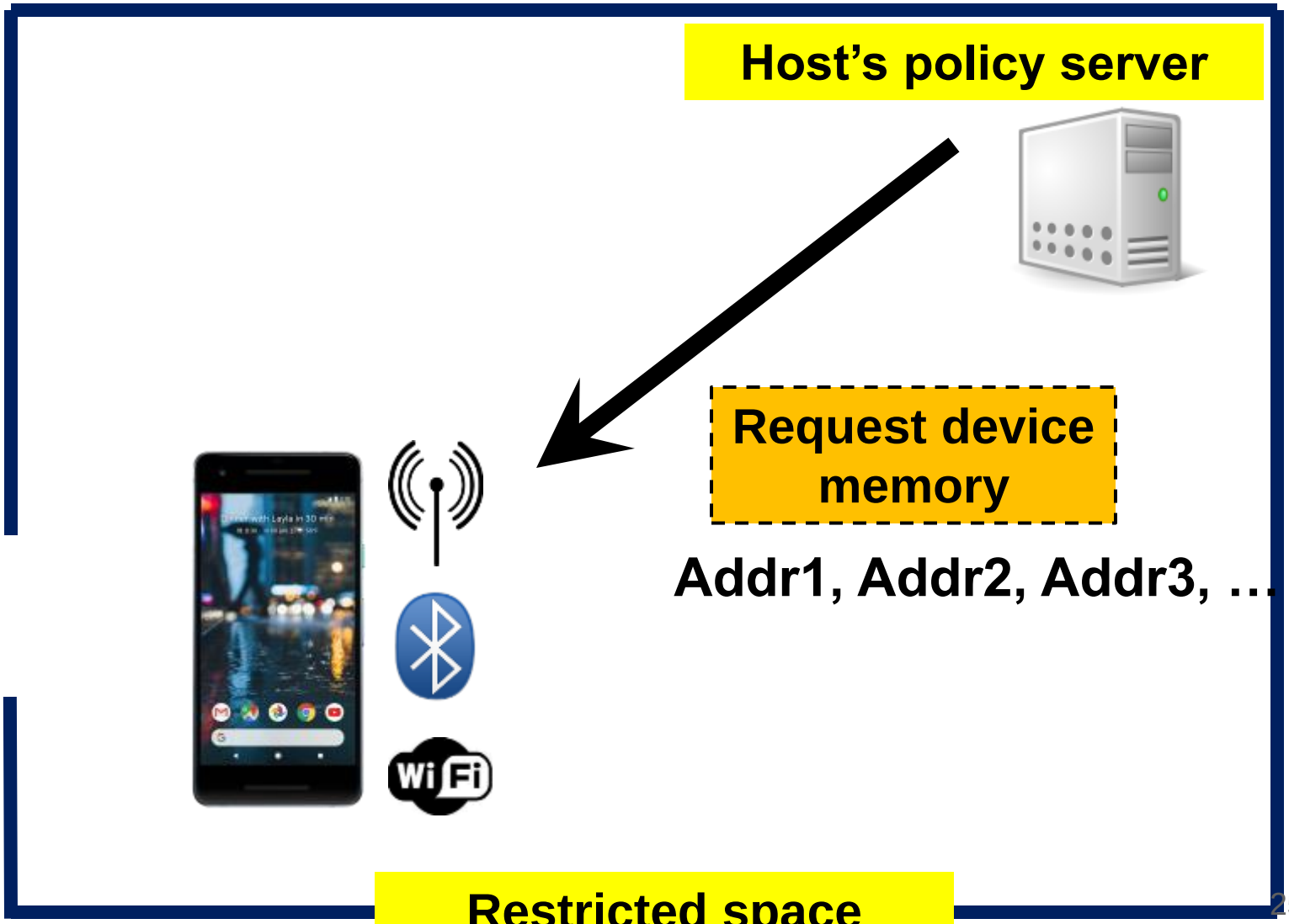
Guest device check-in



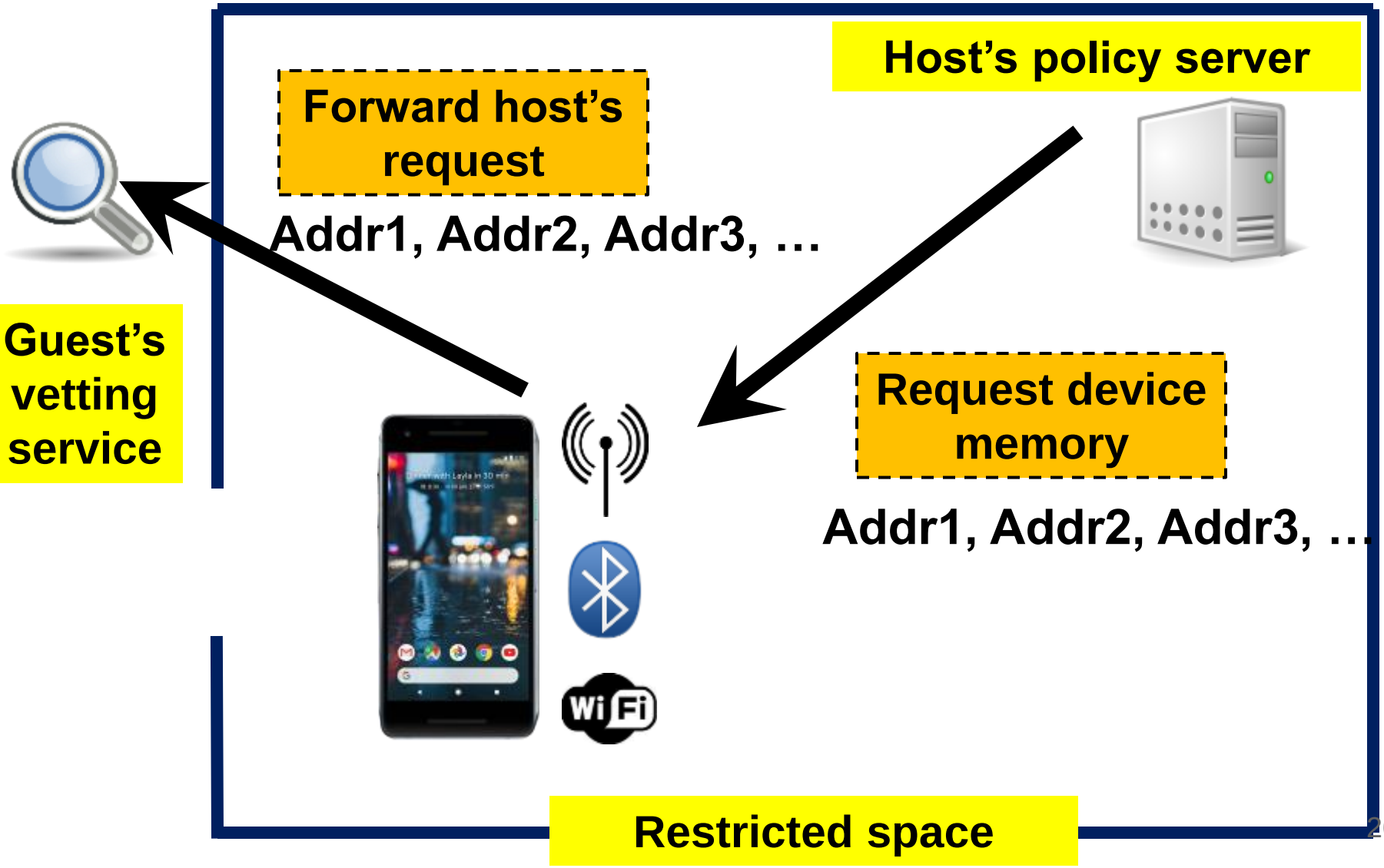
Mutual authentication



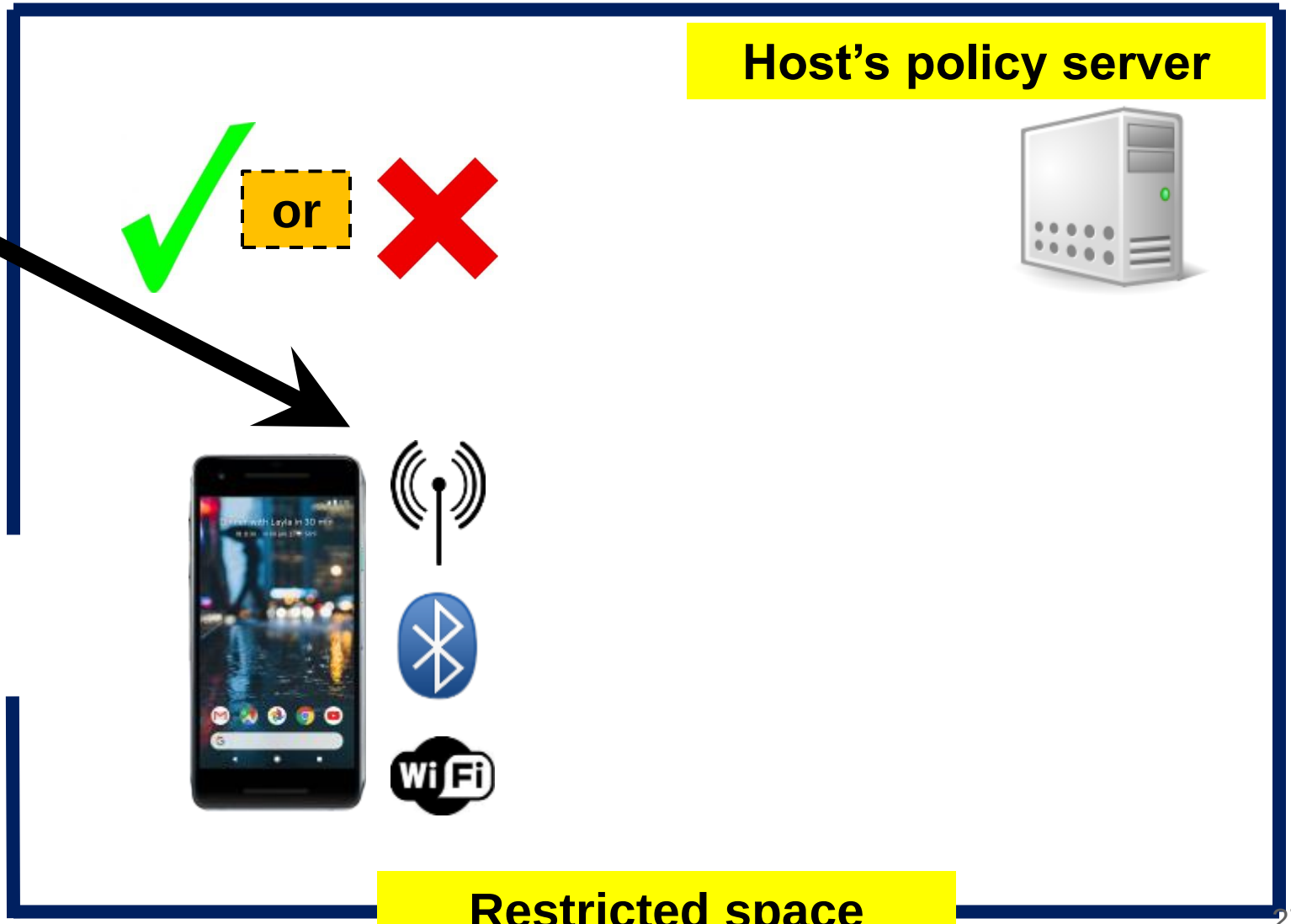
Host requests guest analysis



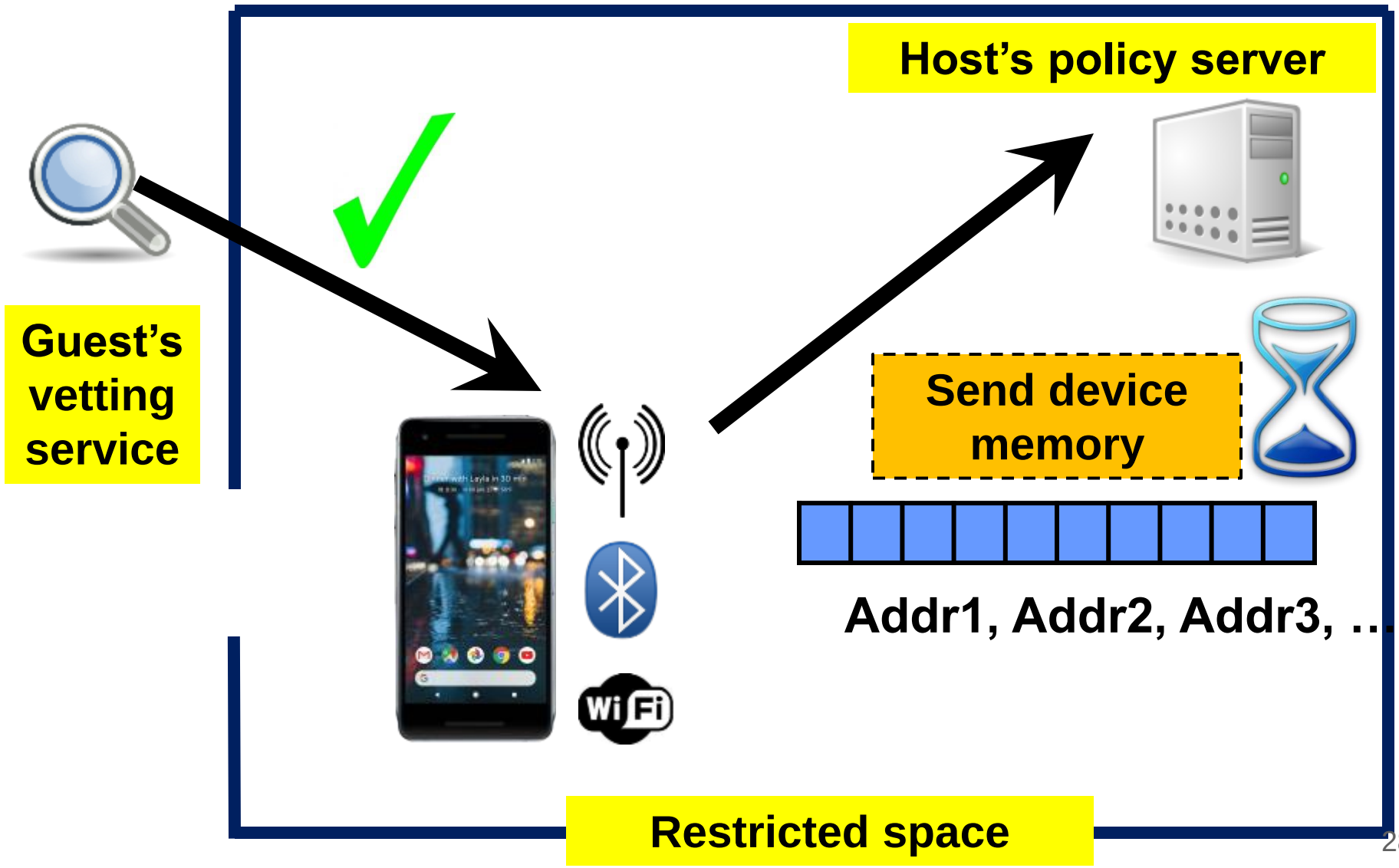
Guest vets host's request



Guest vets host's request



Host analyzes guest device

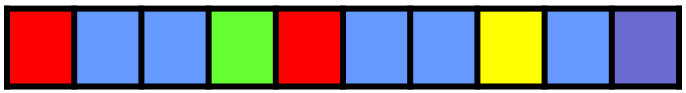


Host pushes policy to guest

Guest's vetting service

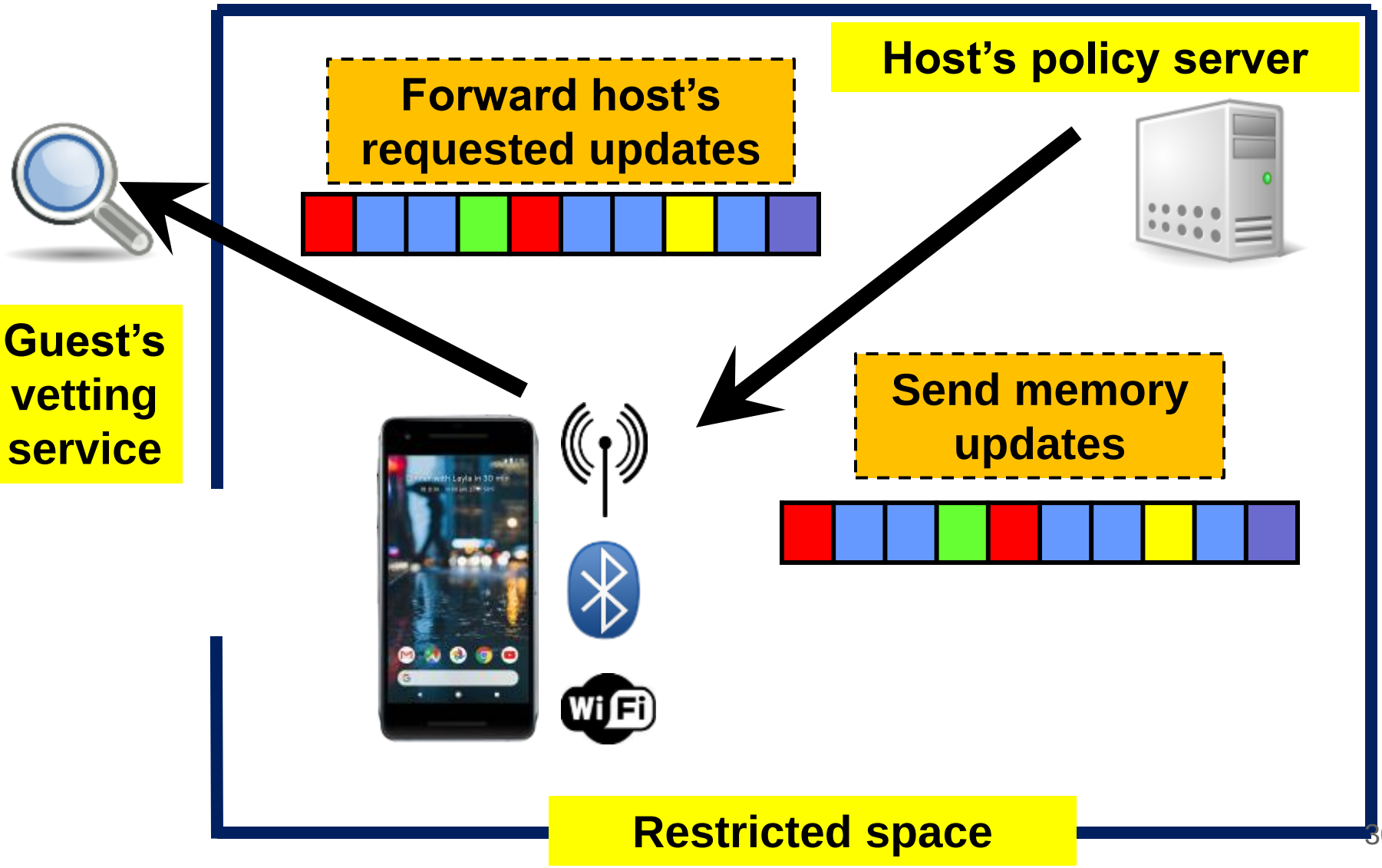
Host's policy server

Send memory updates

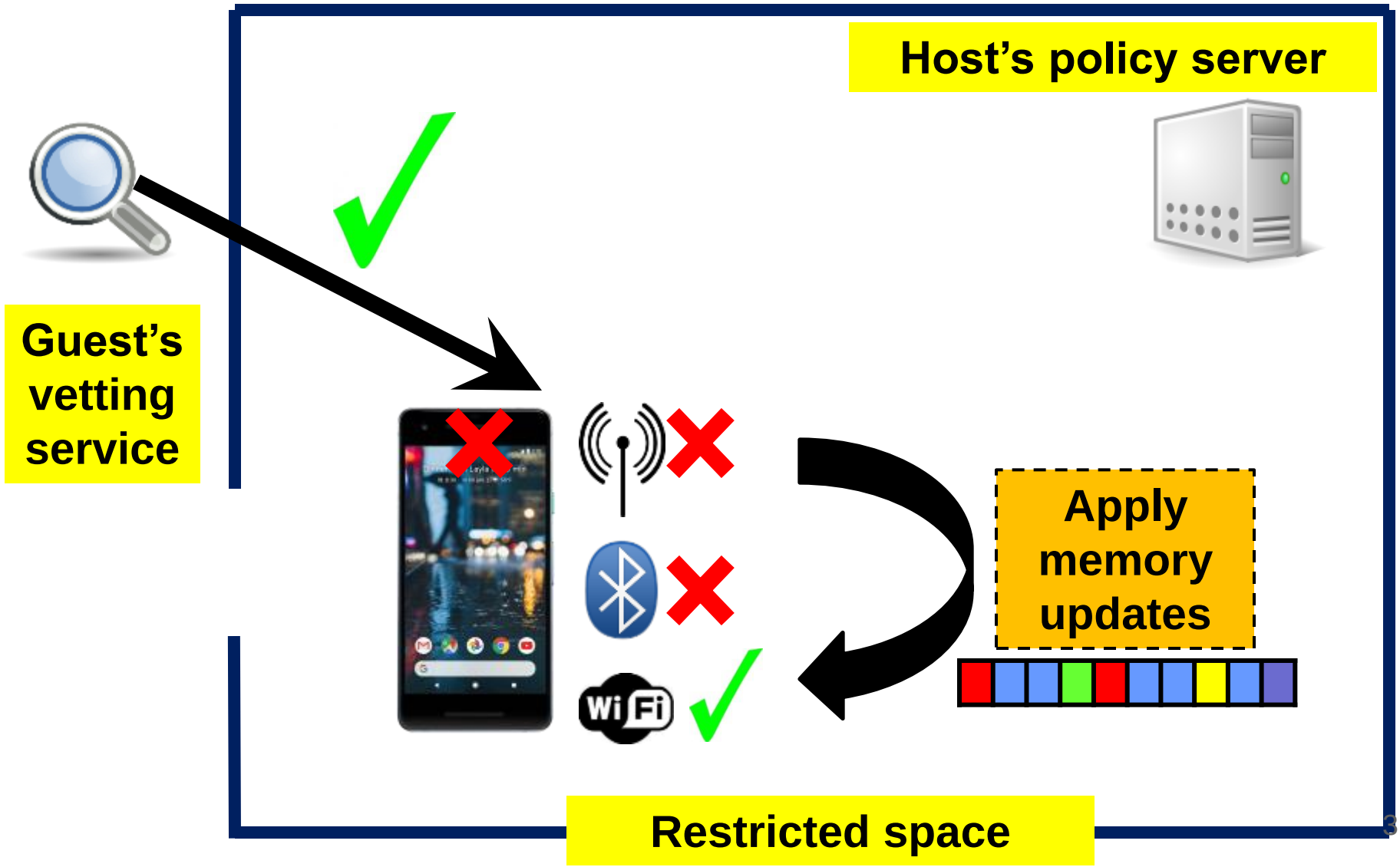


Restricted space

Guest vets host's updates



Guest applies host's updates



Host requests proof

Guest's vetting service

Host's policy server

Request proof of policy compliance

Restricted space

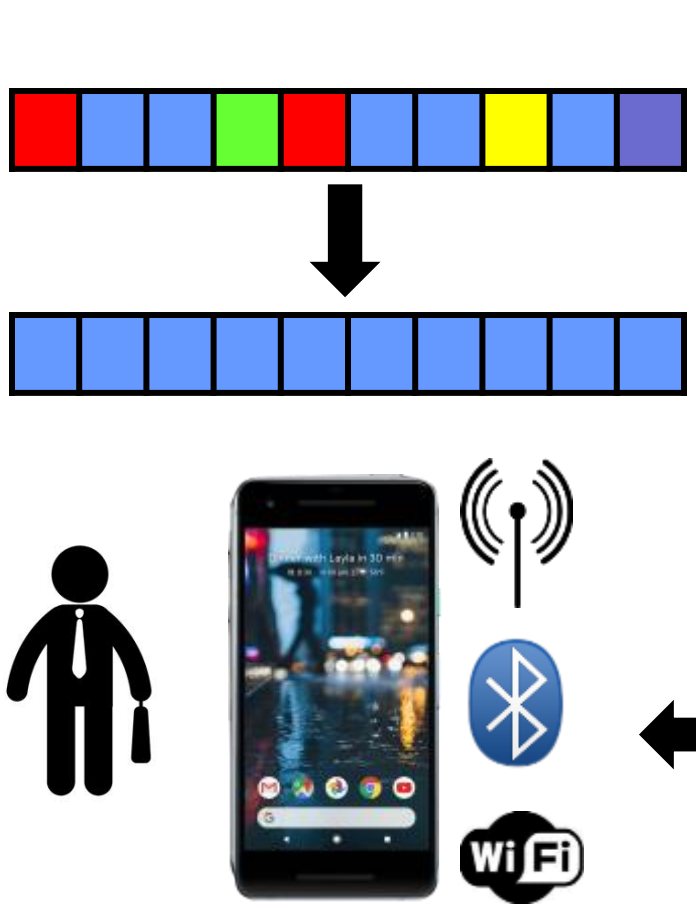
Guest sends proof

Guest's vetting service

Host's policy server

Restricted space

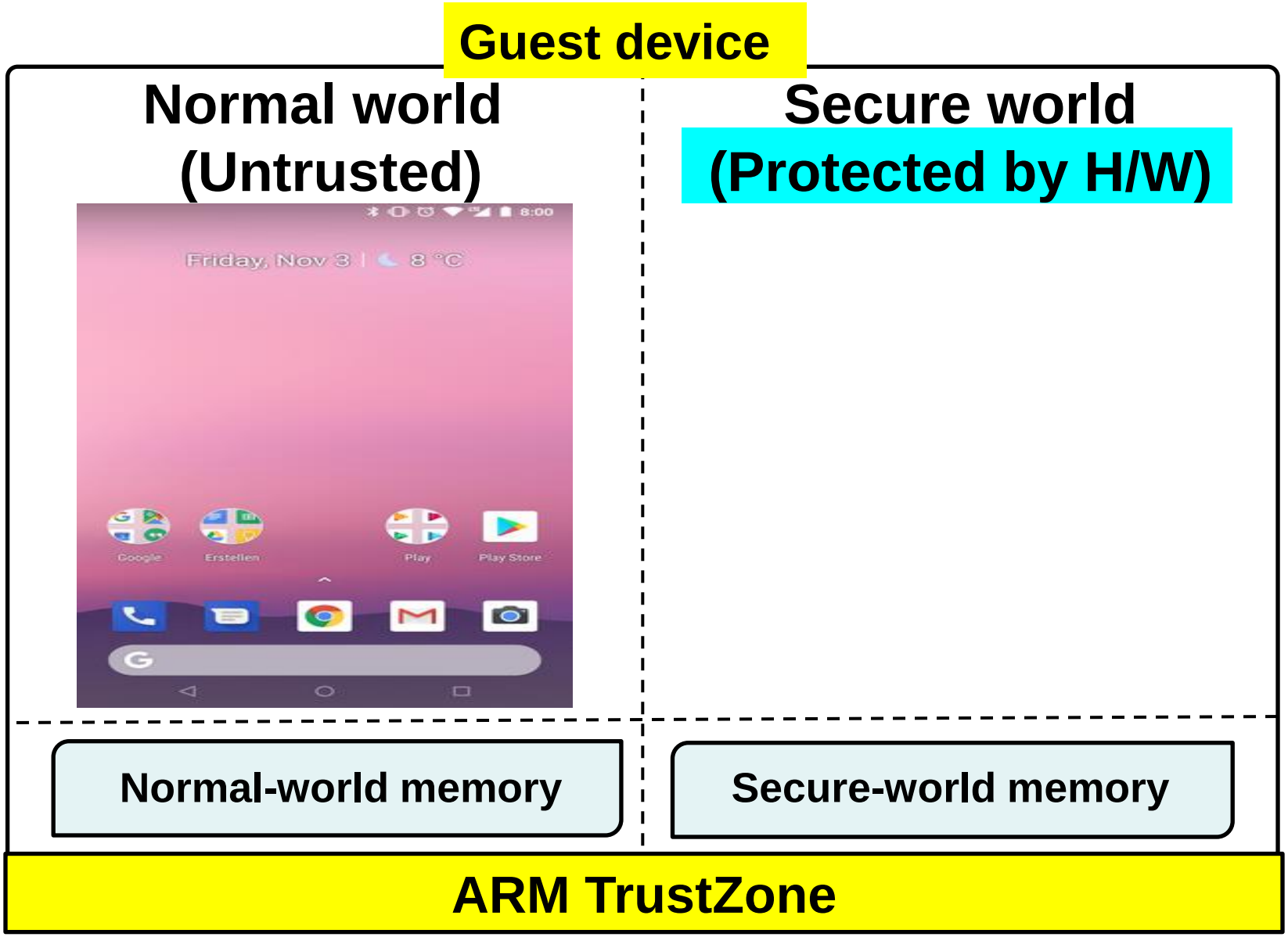
Guest device check-out



Public space

Restricted space

The ARM TrustZone



Secure boot protects secure world

Normal world (Untrusted)

Secure world

Secure boot

Normal-world memory

Secure-world memory

ARM TrustZone

Secure world stores keys

Normal world (Untrusted)

Secure world

Secure boot

PubKeyG, PrivKeyG

Normal-world memory

Secure-world memory

ARM TrustZone

Memory is partitioned

Normal world (Untrusted)

Secure world

Secure boot

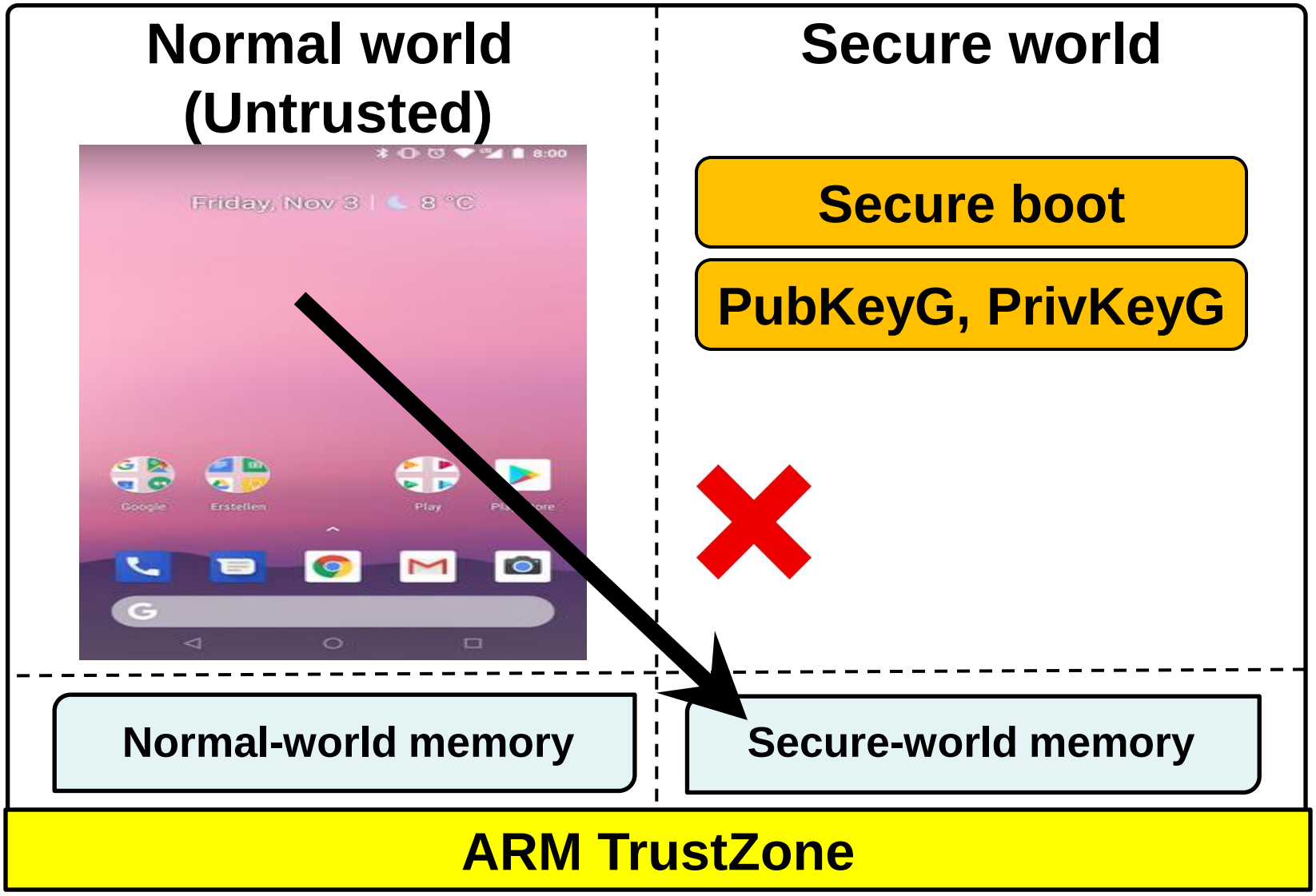
PubKeyG, PrivKeyG

Normal-world memory

Secure-world memory

ARM TrustZone

Memory is partitioned



We enhance the secure world

Normal world (Untrusted)

Secure world (booted securely)

1. **Authentication**
2. **NW analysis**
3. **NW updates**
4. **Verif. tokens**

Normal-world memory

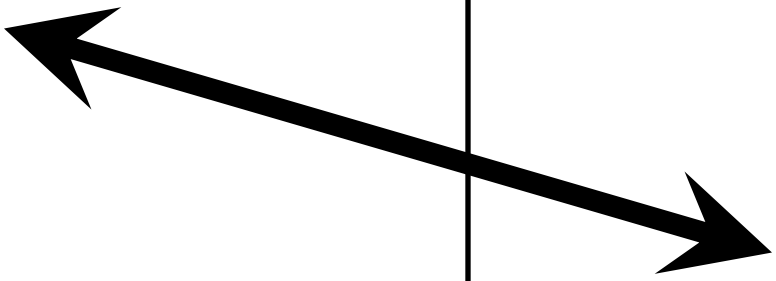
Secure-world memory

ARM TrustZone

Mutual authentication

Host's policy server

k_s



Secure world

k_s

Goal

Establish shared session key k_s between host and guest

Secure-world memory

ARM TrustZone

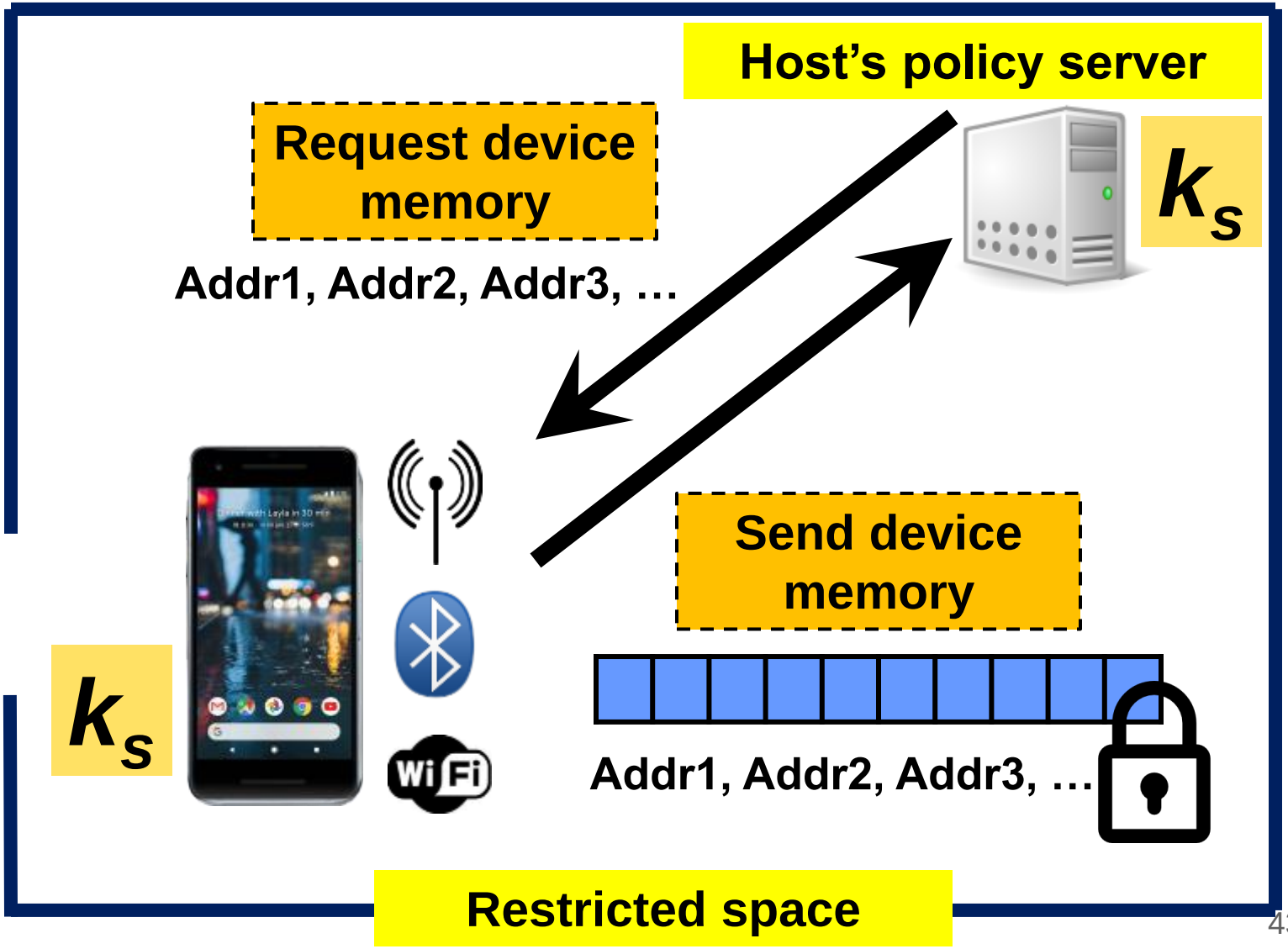
Establishing session key k_s

Simplified TLS/SSL handshake

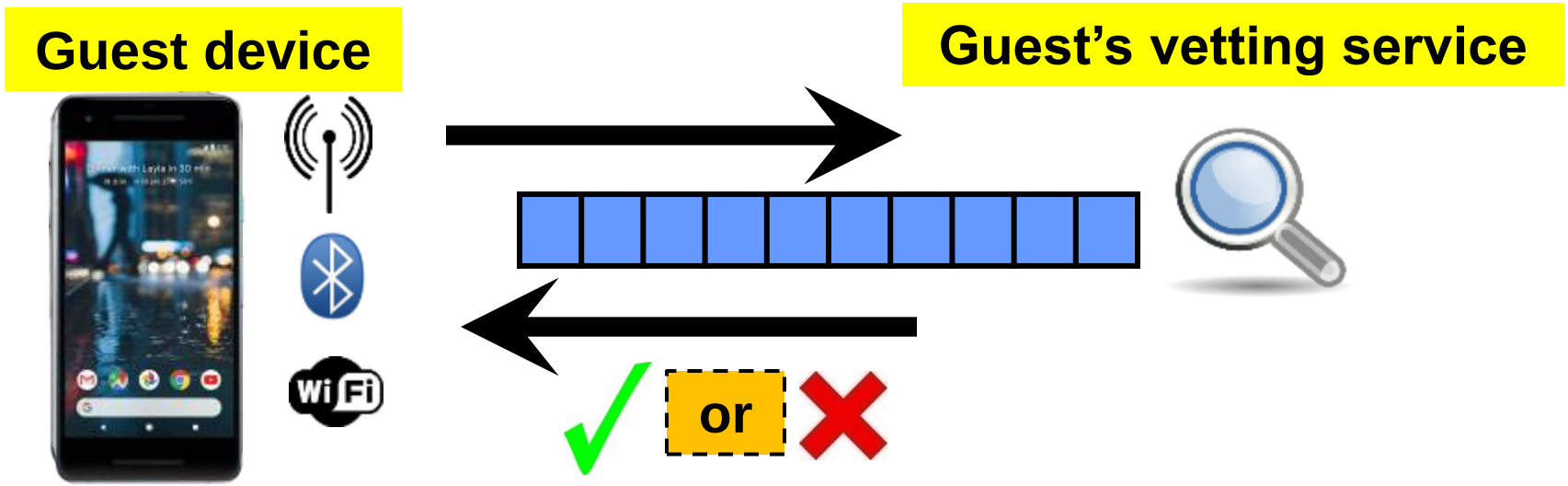
- Host's keypair: **PubKeyH**, **PrivKeyH**
- Guest's keypair: **PubKeyG**, **PrivKeyG**

1. **Guest ←→ Host**: Exchange/verify public keys
2. **Host → Guest**: $Enc_{\mathbf{PubKeyG}}(k_s)$+ $Signature_{\mathbf{PrivKeyH}}$
3. **Guest (secure world)**: Verify host signature, decrypt message and obtain k_s

Guest device analysis

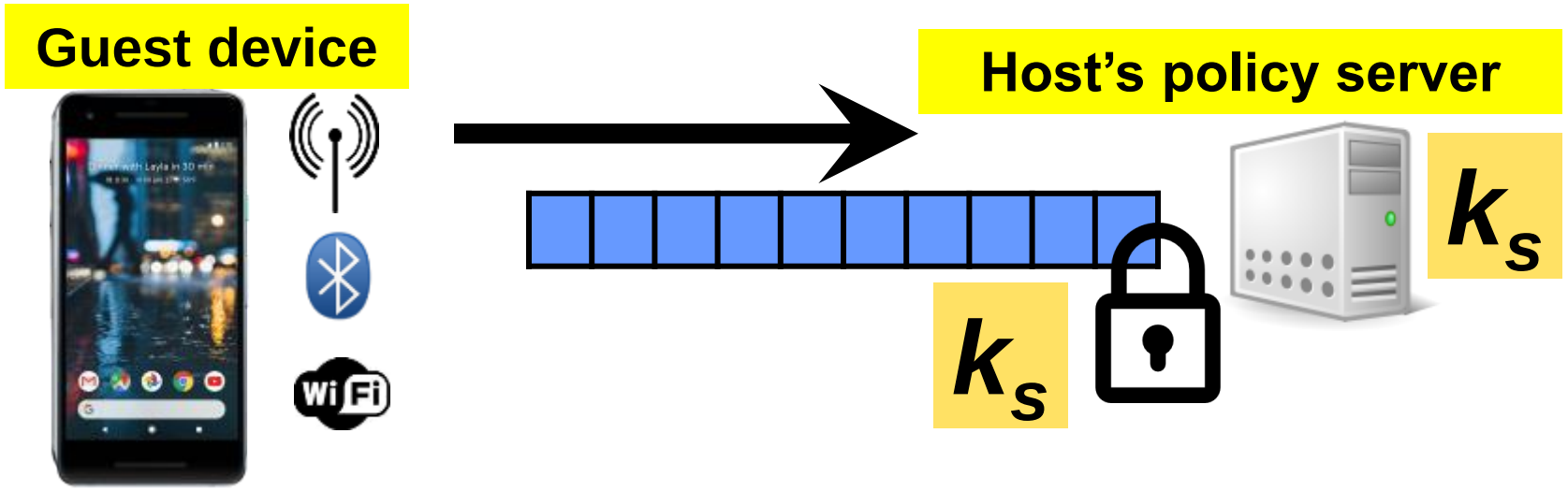


Vetting host's requests



- Vetting server ensures that host's requests do not compromise guest privacy
- **Vetting policy**: Host only allowed to request *guest device's kernel memory*

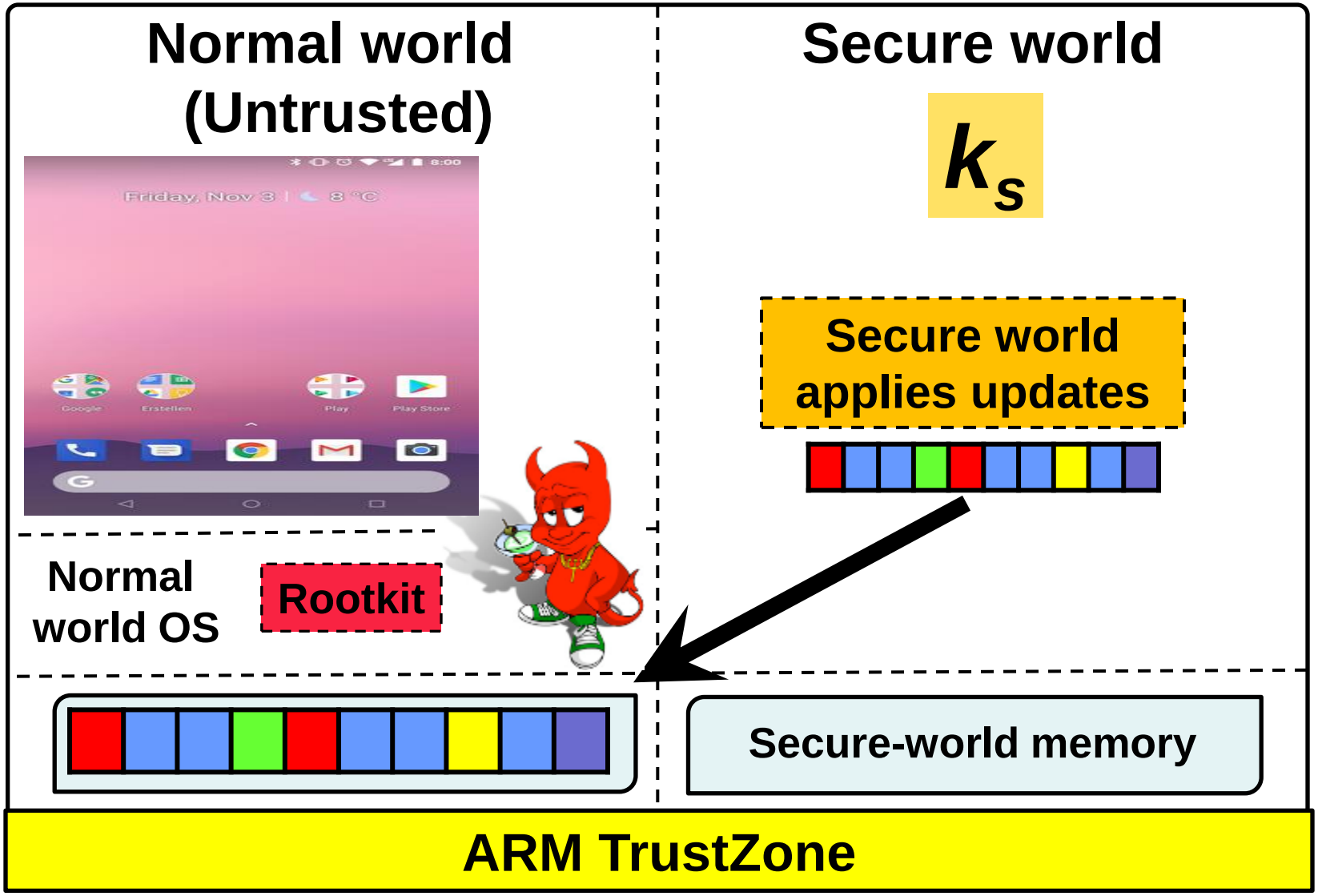
Analysis of NW memory snapshot



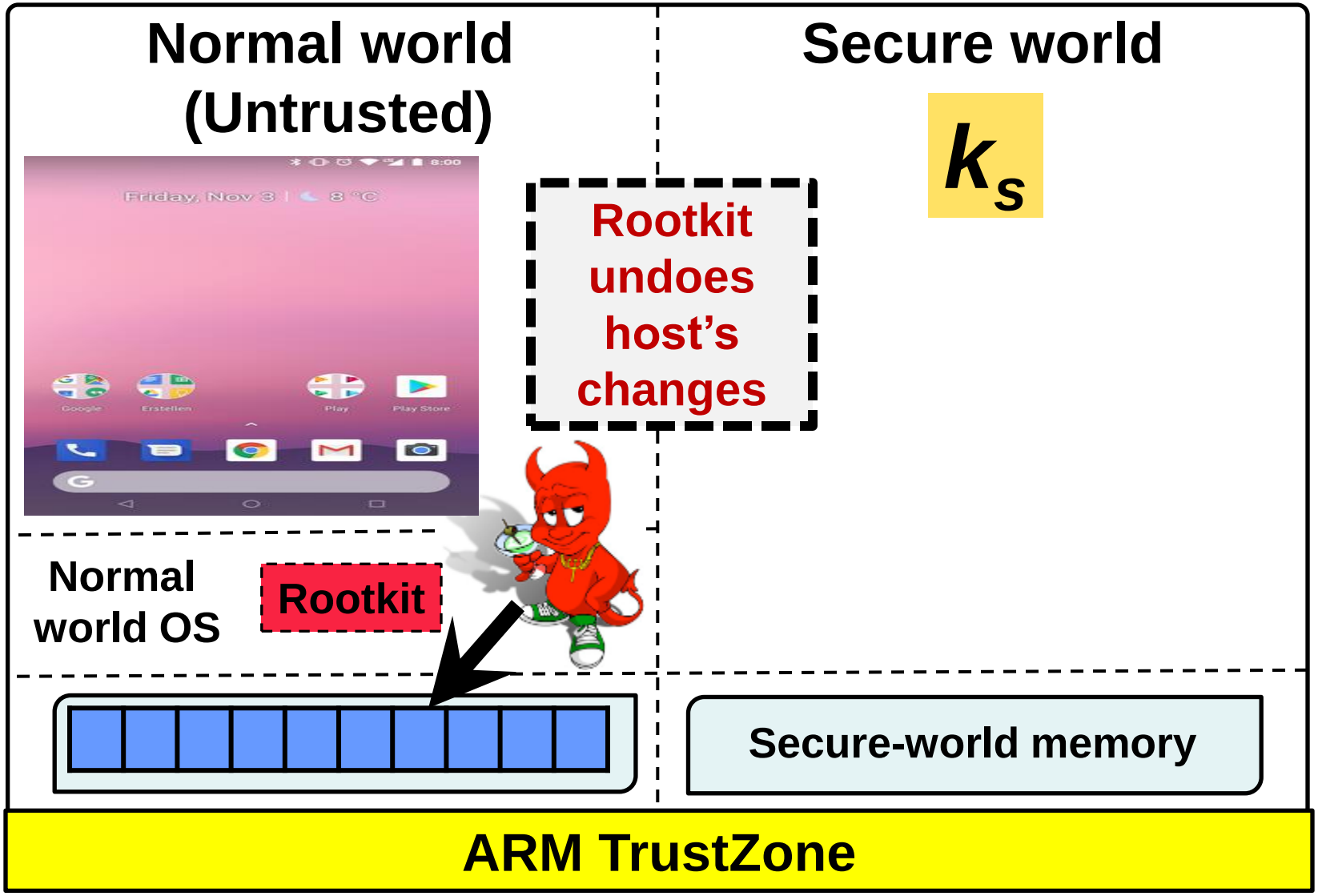
- Infer what peripherals are installed, and where in memory their drivers are installed
- Detect guest device for malware infection, including kernel-level rootkits

[Baliga, Ganapathy, Iftode, ACSAC'08, TDSC'11]

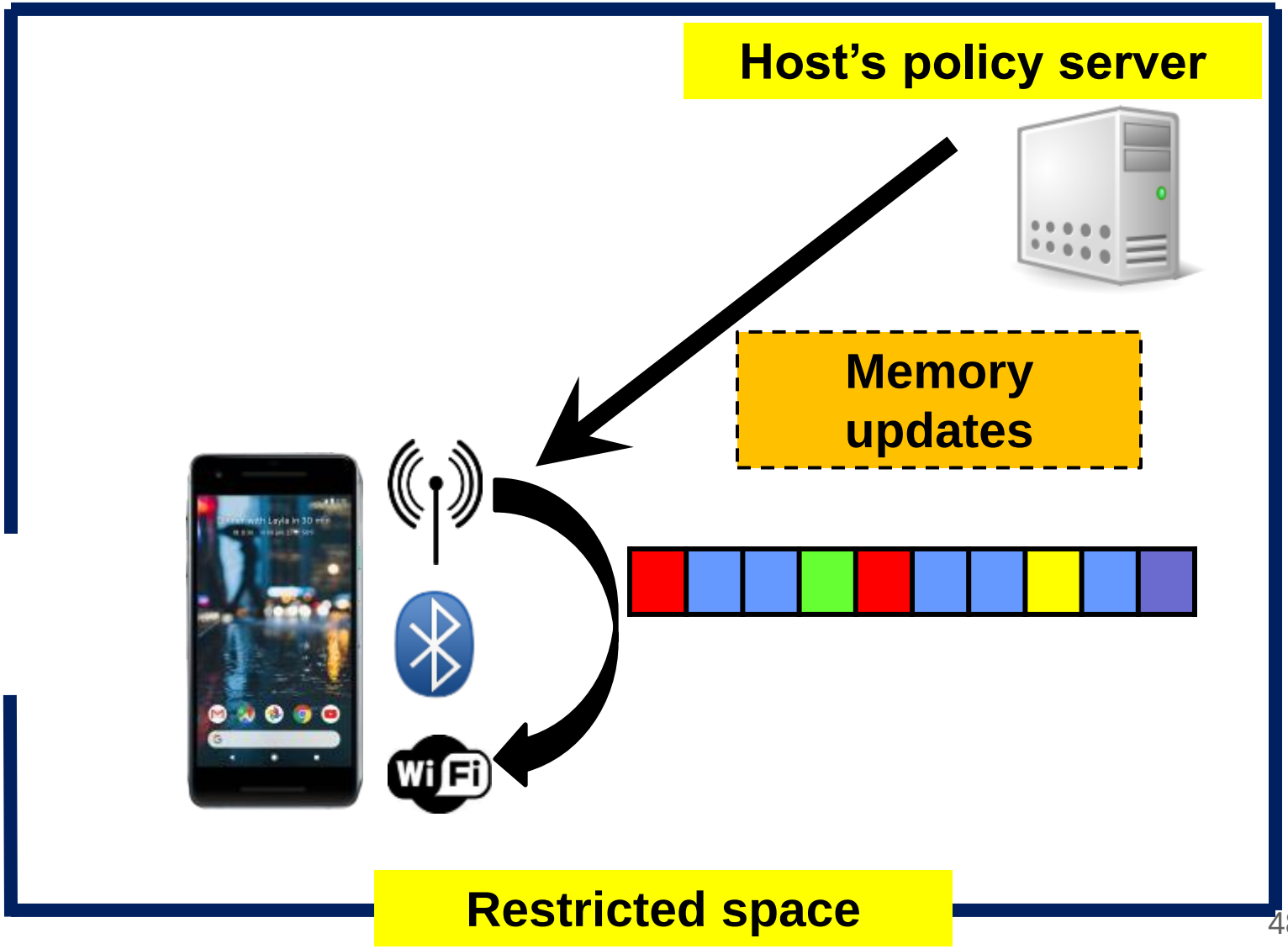
Why look for NW rootkits?



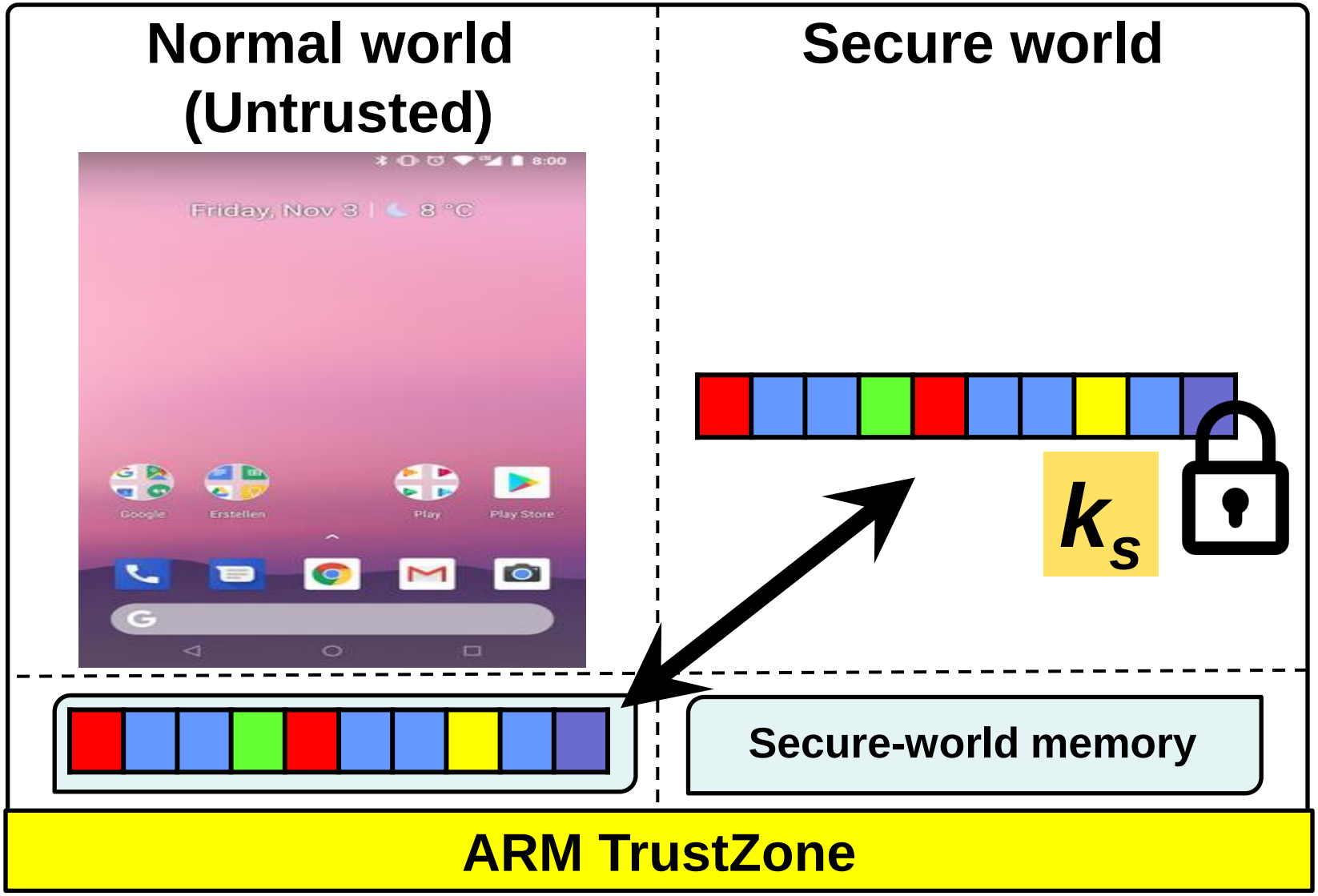
Why look for NW rootkits?



Guest device update

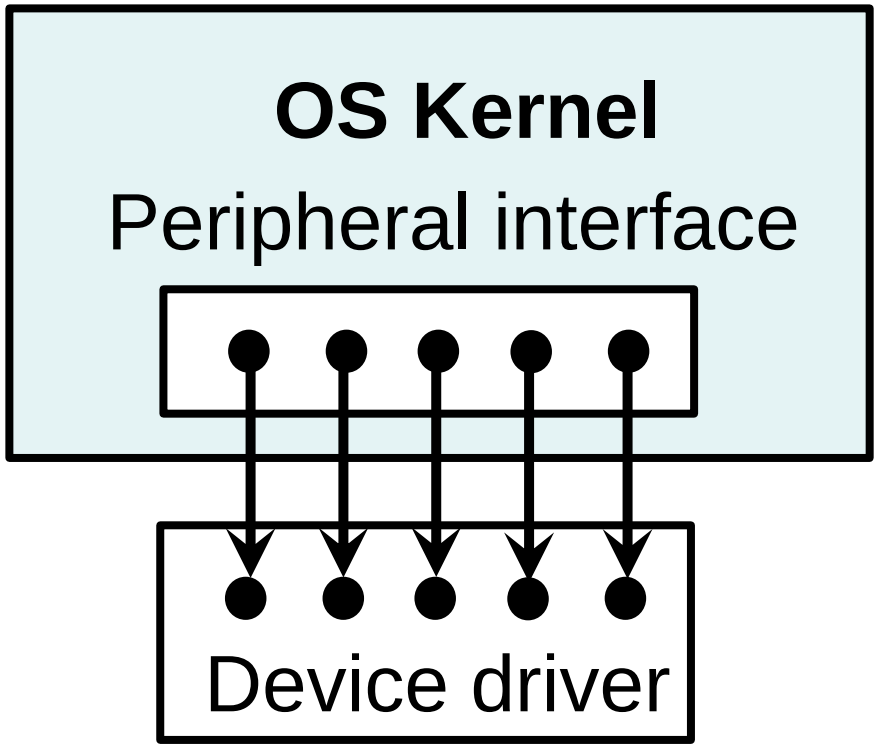


SW updates NW memory



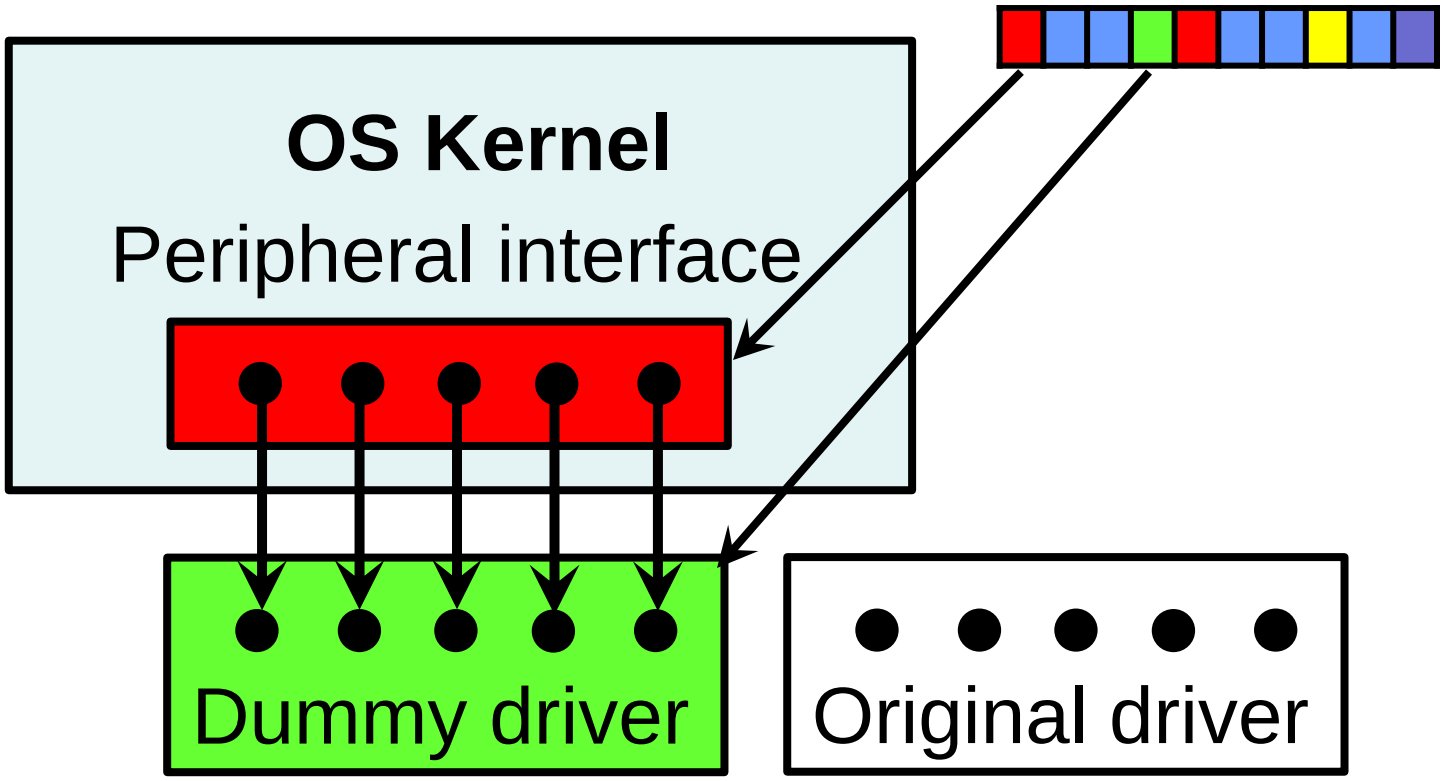
Updating peripheral drivers

- Device drivers in normal world control execution of device peripherals



Updating peripheral drivers

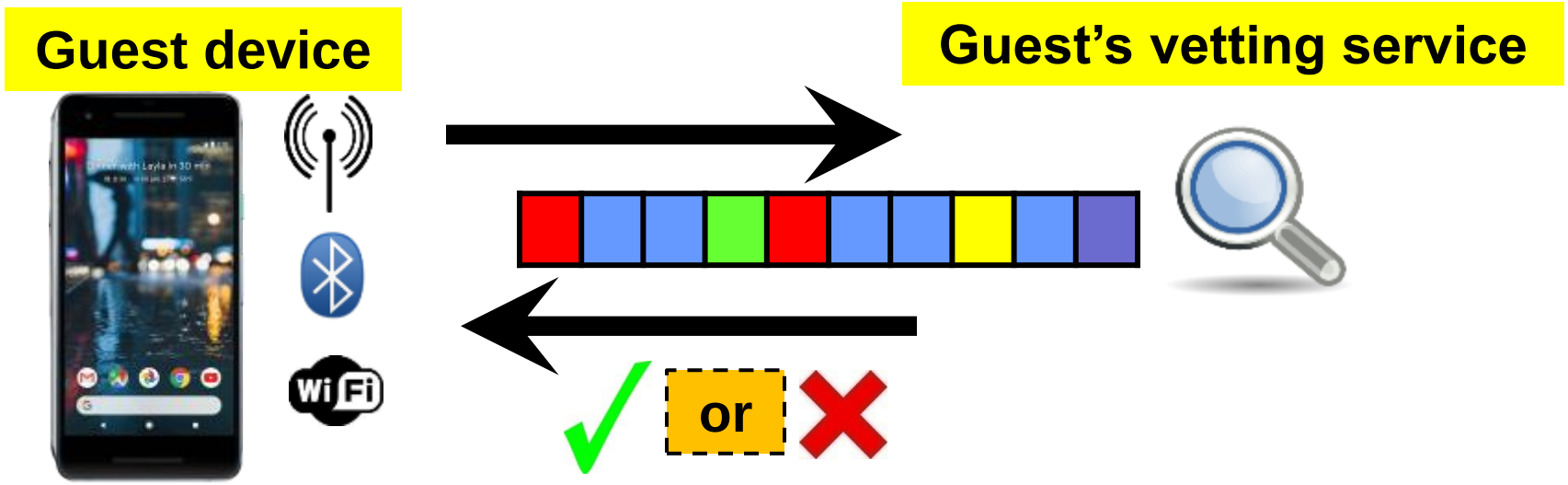
- Introduce dummy driver to control peripheral (e.g., disable it). Update kernel driver hooks.



Are driver updates effective?

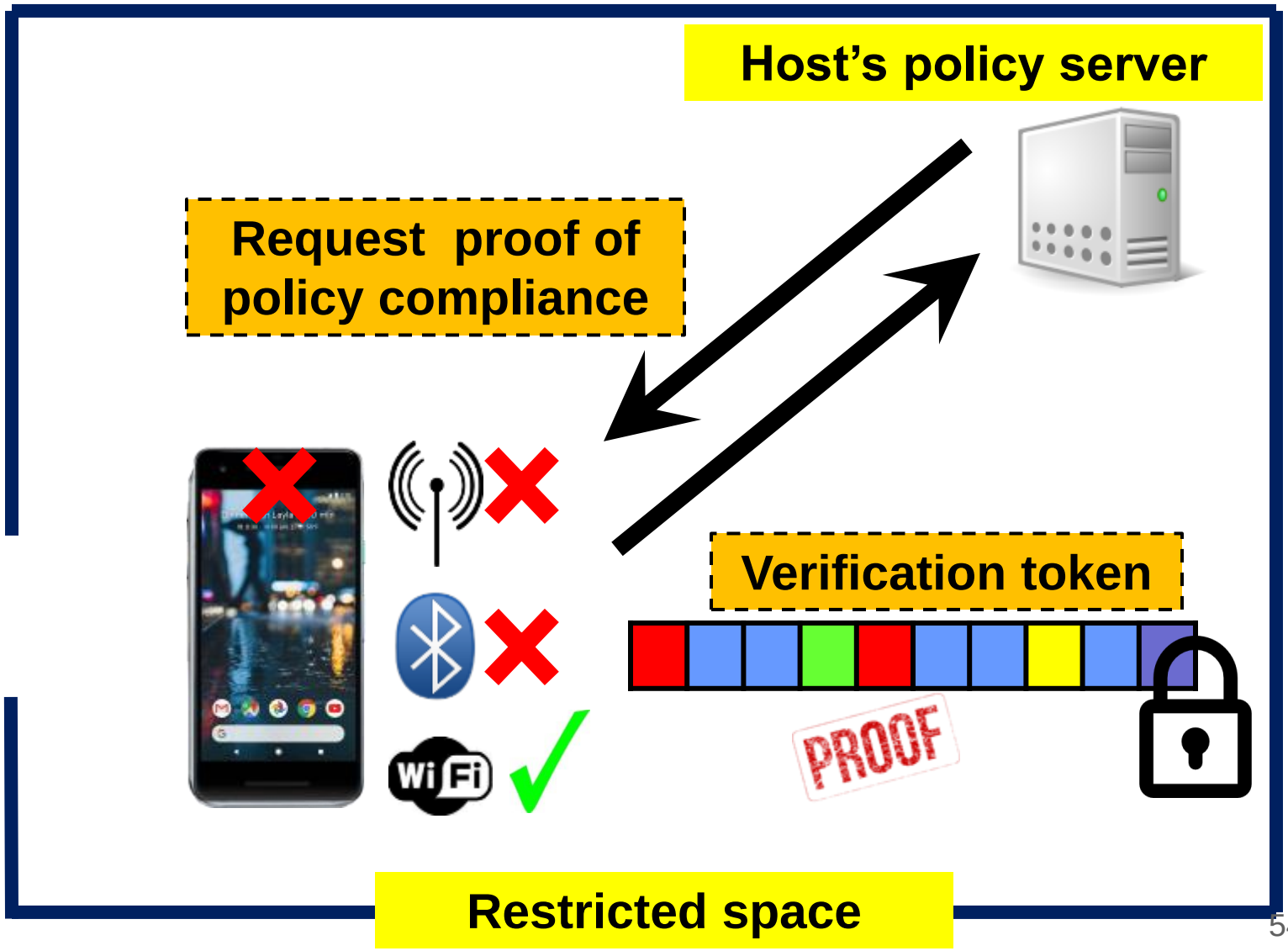
Peripheral considered	Update size (bytes)	Guest device	Peripheral disabled?
USB webcam	302	i.MX53	✓
Camera	212	Nexus phone	✓
WiFi	338	Nexus phone	✓
3G (Data)	252	Nexus phone	✓
3G (Voice)	224	Nexus phone	✓
Microphone	184	Nexus phone	✓
Bluetooth	132	Nexus phone	✓

Vetting host's updates



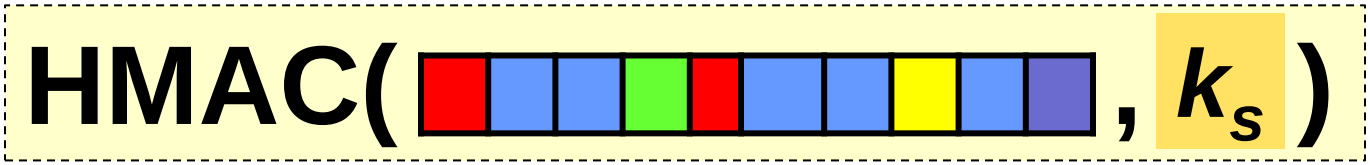
- An untrusted host can introduce new code into guest devices
- **Vetting policy**: Ensure that dummy drivers are a *subset* of the original drivers
  - Via ARM-binary analysis on

Proof of compliance



Verification tokens

- Host requests proof of compliance
- Secure world computes a fresh snapshot of all NW memory locations updated by host
- Verification token:



- Verification token matches if and only if normal world memory still in compliance with the host's usage policy

Summary

- Low-level API allows hosts to analyze and control guests
  - Simplifies design and size of TCB
- Hosts can obtain proofs of guest compliance
  - Relies on ARM TrustZone hardware
- Vetting service balances guest privacy with host's usage policies

RUTGERS
THE STATE UNIVERSITY
OF NEW JERSEY

Regulating Smart Devices with SEAndroid

Shortcomings of our previous work

1. Sharing memory images is too intrusive
2. Policy language is not user-friendly
3. Low deployability & maintenance difficulty

➢ **Solution: Leveraging SEAndroid**

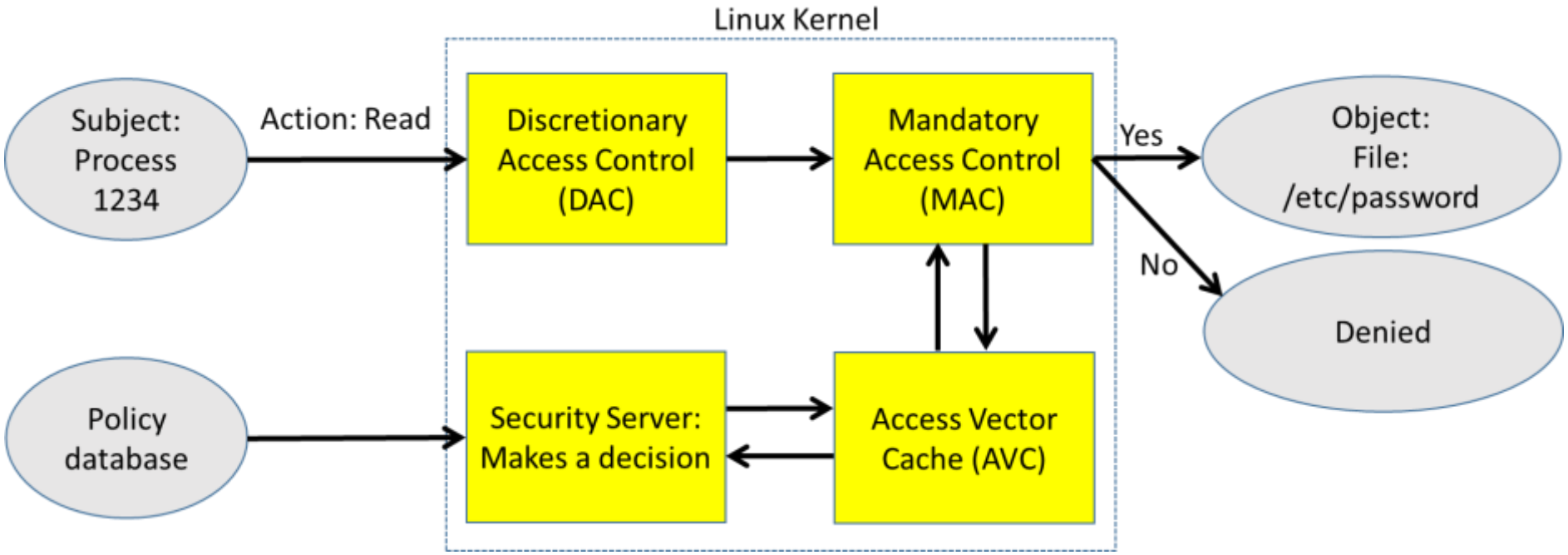
Background

- SEAndroid
  - Provide Mandatory Access Control (MAC)
- NFC
  - Short-range wireless technologies between two devices
- OP-TEE
  - Open-source Portable Trusted Execution Environment for ARM TrustZone-enabled devices

Contributions of our work

- No privacy concerns with pre-defined SEAndroid policies on guest devices
- Fine-grained policy enforcement with SEAndroid
  - Peripherals, apps, file system level control
- Provide secure policy enforcement mechanism with ARM TrustZone
- Easy to use with NFC

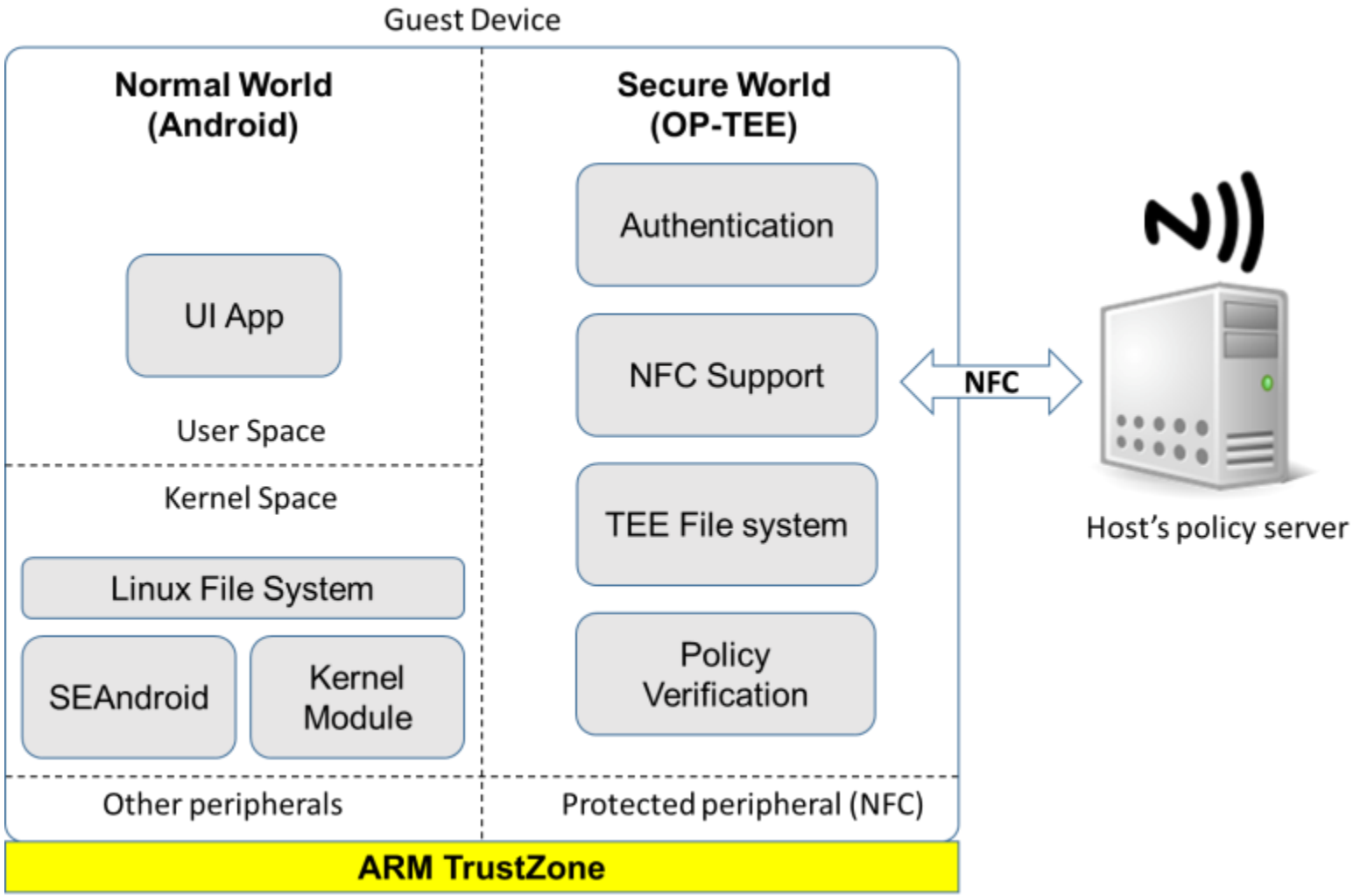
SEAndroid


Linux Kernel
Subject: Process 1234
Action: Read
Discretionary Access Control (DAC)
Mandatory Access Control (MAC)
Yes
Object: File: /etc/password
No
Denied
Policy database
Security Server: Makes a decision
Access Vector Cache (AVC)

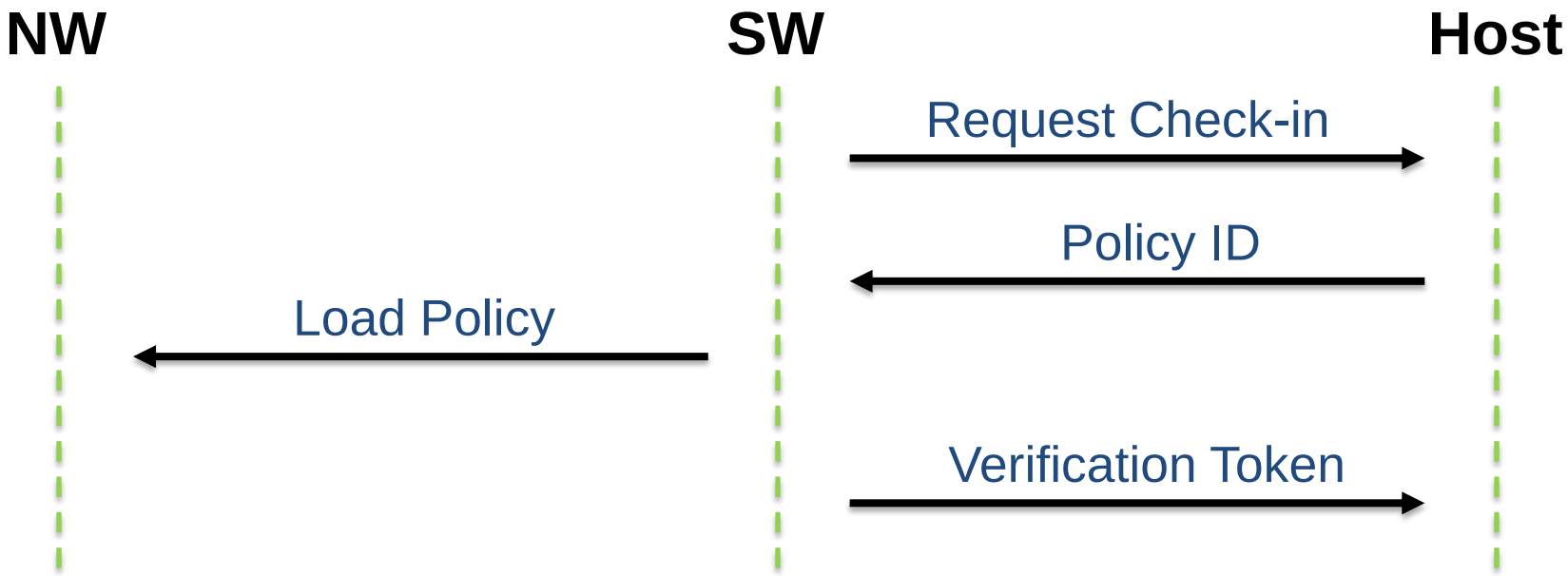
Threat Model

- The normal world can be compromised
- The NFC peripheral operates in the secure world
- Pre-defined SEAndroid policies do not have vulnerabilities
- OP-TEE as a secure OS is not vulnerable

System Architecture



Policy Enforcement Procedure



SEAndroid Policy Example

- SEAndroid Policy form:

```
allow domains types:classes permissions;
```

- Removing following rules disables USB device peripherals

```
allow system_server usb_device:chr_file rw_file_perms;
allow system_server usb_device:dir r_dir_perms;
```

Implementation

- Guest device:
  - Integrated Android and OP-TEE on i.MX6 development board
  - Ported NFC device driver to secure world
- Host server:
  - Developed NFC application on Android device

Summary

- Higher-level abstraction for fine-grained policy enforcement with SEAndroid
- Hosts can obtain proofs of guest compliance
  - Relies on ARM TrustZone hardware
- No privacy concerns

Conclusion

We present systematic methods to regulate devices and ensure responsible use

- Remote memory operation
  - Low-level API allows hosts to analyze and control guests
- SEAndroid approach
  - Higher-level abstraction for fine-grained policy enforcement
- Hosts can obtain proofs of guest compliance
  - Relies on ARM TrustZone hardware

Future directions

- Balance between security and privacy
- Automated kernel image analysis tool
- Automated policy rules generator tool
- Policy enforcement on connected wearable devices

Other Contributions

- "Seeing is believing: Sharing Real-time Traffic Images via Vehicular Clouds," **[IEEE Access 2016]**
- “Detecting Plagiarized Mobile Apps using API Birthmarks,” **[JASE 2015]**
- “DoppelDriver: Counterfactual Actual Travel Times for Alternative Routes,“ **[PERCOM 2015]**
- "Data-Driven Inference of API Mappings," **[PROMOTO 2014]**
- "Tweeting Traffic Image Reports on the Road," **[MobiCase 2014]**

Acknowledgement

- **Advisor:** Vinod Ganapathy
- **Thesis Committee:** Vinod Ganapathy, Badri Nath, Abhishek Bhattacharjee, and Pratyusa Manadhata
- **Co-authors:** Liviu Iftode, Badri Nath, Abhinav Srivastava, Amruta Gokhale, Daehan Kwak, Ruilin Liu, Ferdinand Brasser, Christopher Liebchen, Ahmad-Reza Sadeghi
- **Discolab members**
  - Mohan Dhawan, Shakeel Butt, Lu Han, Liu Yang, Rezwana Karim, Hai Nguyen, Daehan Kwak, Amruta Gokhale, Ruilin Liu, Nader Boushehrinejadmoradi, Wenjie Sha, Hongzhang Liu

RUTGERS
THE STATE UNIVERSITY
OF NEW JERSEY

Thank you!

Daeyoung Kim
daeyoung.kim@cs.rutgers.edu

RUTGERS
THE STATE UNIVERSITY
OF NEW JERSEY

Backup Slides

Check-in Protocol

The host and the guest share a secret key, k_s via the NFC-SEC protocol.

1. **Guest** → **Host**: Requesting a new session
2. **Host** → **Guest**: PolicyID || Nonce_H || HMAC_{k_s}(PolicyID || Nonce_H)
3. **Guest**: Invoke reloading policy
4. **Guest** → **Host**: Nonce_G || HMAC_{k_s}(Policy || Nonce_G)
5. **Host**: Verify HMAC_{k_s}(Policy || Nonce_G)

Operational details

1. How can host trust guest to apply policy?

- **Answer:** Leverage ARM TrustZone

2. Why memory snapshots and updates?

- **Answer:** Powerful low-level API. Reduces TCB

3. How does vetting service ensure safety?

- **Answer:** Simple, conservative program analysis

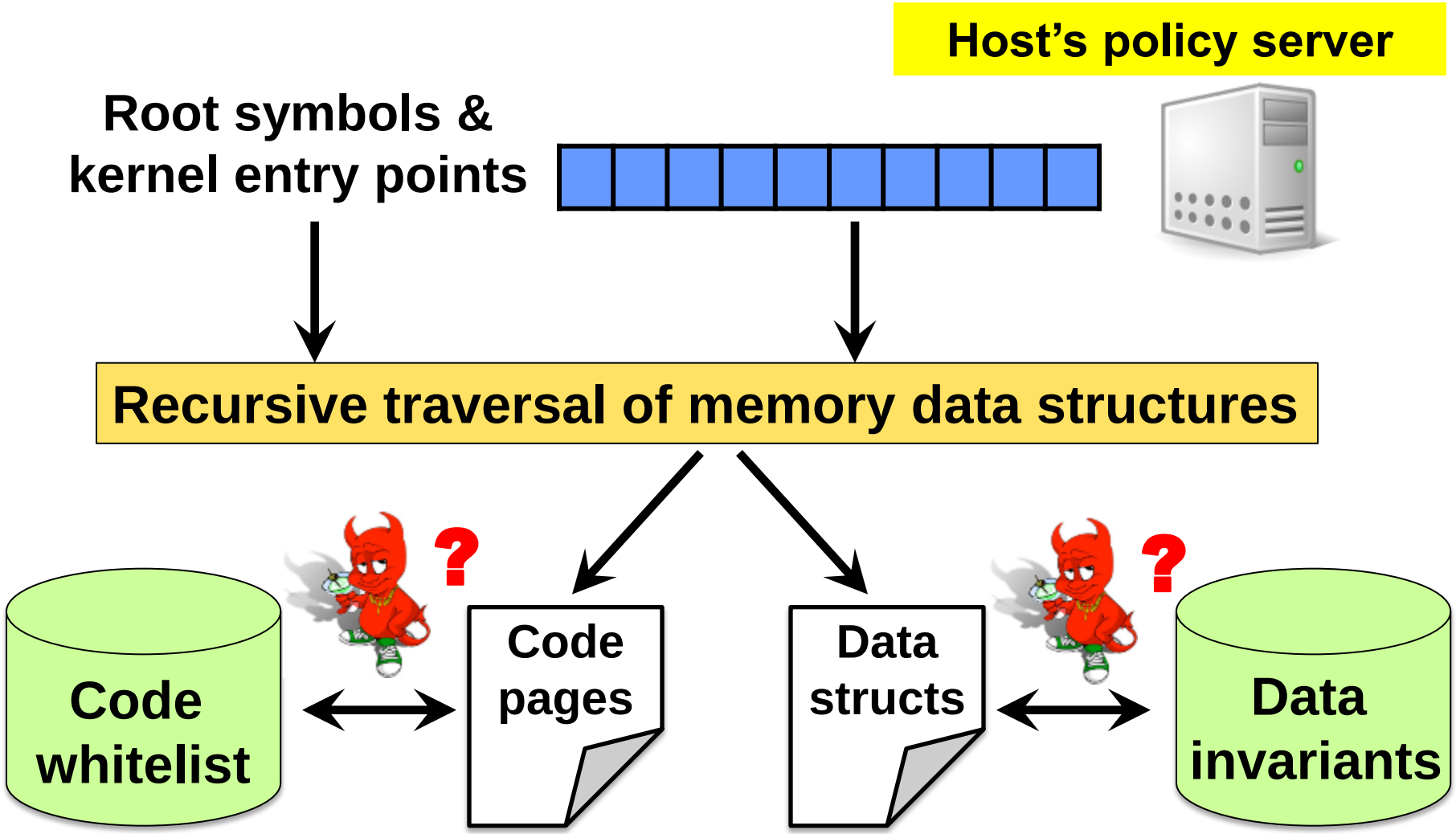
4. Can't guest device simply reboot to undo?

- **Answer:** REM-suspend protocol

Related approaches

- Device virtualization:
  - Heavyweight; probably not for all devices
  - Still requires host to trust hypervisor on guest
- Mobile device management solutions:
  - No proofs to host
  - Device-dependent TCB on guest
- Context-based access control:
  - Same shortcomings as MDM solutions above

Analysis of NW memory snapshot



Are memory updates the right API?

- Powerful, low-level API for device control
- Simplifies design of secure world (TCB) and keeps it device-independent

TCB component	SLOC
Memory manager	1381
Authentication	1285
Memory ops., verification tokens	305
REM-suspend	609
SHA1 + HMAC	861
X509	877
RSA	2307

Do memory updates affect app stability?

Passive updates: Update memory and start the app

USB	*MobileWebCam*	*ZOOM FX*	*Retrica*	*Candy Cam*	*HD Cam Ultra*
	App Error	**Android Error**	**App Error**	**App Error**	**Android Error**
Camera	*Android Cam*	*Camera MX*	*ZOOM FX*	*Droid HD Cam*	*HD Cam Ultra*
	Android Error	**App Error**	**App Error**	**Android Error**	**Android Error**
WiFi	*Spotify*	*Play Store*	*YouTube*	*Chrome*	*Facebook*
	No Connection	**No Connection**	**No Connection**	**No Connection**	**No Connection**
3G (Data)	*Spotify*	*Play Store*	*YouTube*	*Chrome*	*Facebook*
	No Connection	**No Connection**	**No Connection**	**No Connection**	**No Connection**
3G (Voice)	*Default call application*				
	Unable to place call				
Micro-phone	*Audio rec*	*Easy voice rec*	*Smart voice rec*	*Snd/voice rec*	*Smart voice rec*
	App Error	**App Error**	**App Error**	**App Error**	**App Error**

Do memory updates affect app stability?

Active updates: Update memory with “live” app

USB	*MobileWebCam*	*ZOOM FX*	*Retrica*	*Candy Cam*	*HD Cam Ultra*
	App Error	**App Error**	**App Error**	**App Error**	**App Error**
Camera	*Android Cam*	*Camera MX*	*ZOOM FX*	*Droid HD Cam*	*HD Cam Ultra*
	Blank Screen	**App Error**	**Android Error**	**Blank Screen**	**Blank Screen**
WiFi	*Spotify*	*Play Store*	*YouTube*	*Chrome*	*Facebook*
	No Connection	**No Connection**	**No Connection**	**No Connection**	**No Connection**
3G (Data)	*Spotify*	*Play Store*	*YouTube*	*Chrome*	*Facebook*
	No Connection	**No Connection**	**No Connection**	**No Connection**	**No Connection**
3G (Voice)	*Default call application*				
	Unable to place call				
Micro-phone	*Audio rec*	*Easy voice rec*	*Smart voice rec*	*Snd/voice rec*	*Smart voice rec*
	Empty File	**Empty File**	**Empty File**	**Empty File**	**Empty File**

Memory updates are ephemeral

- Guest device can violate host's usage policies by simply rebooting to undo host's memory updates!
- Once device checked in, secure world must:
  - Mediate all low-battery and power-off interrupts
  - Checkpoint device memory to disk
  - Upon power up, must restore device memory from checkpoint

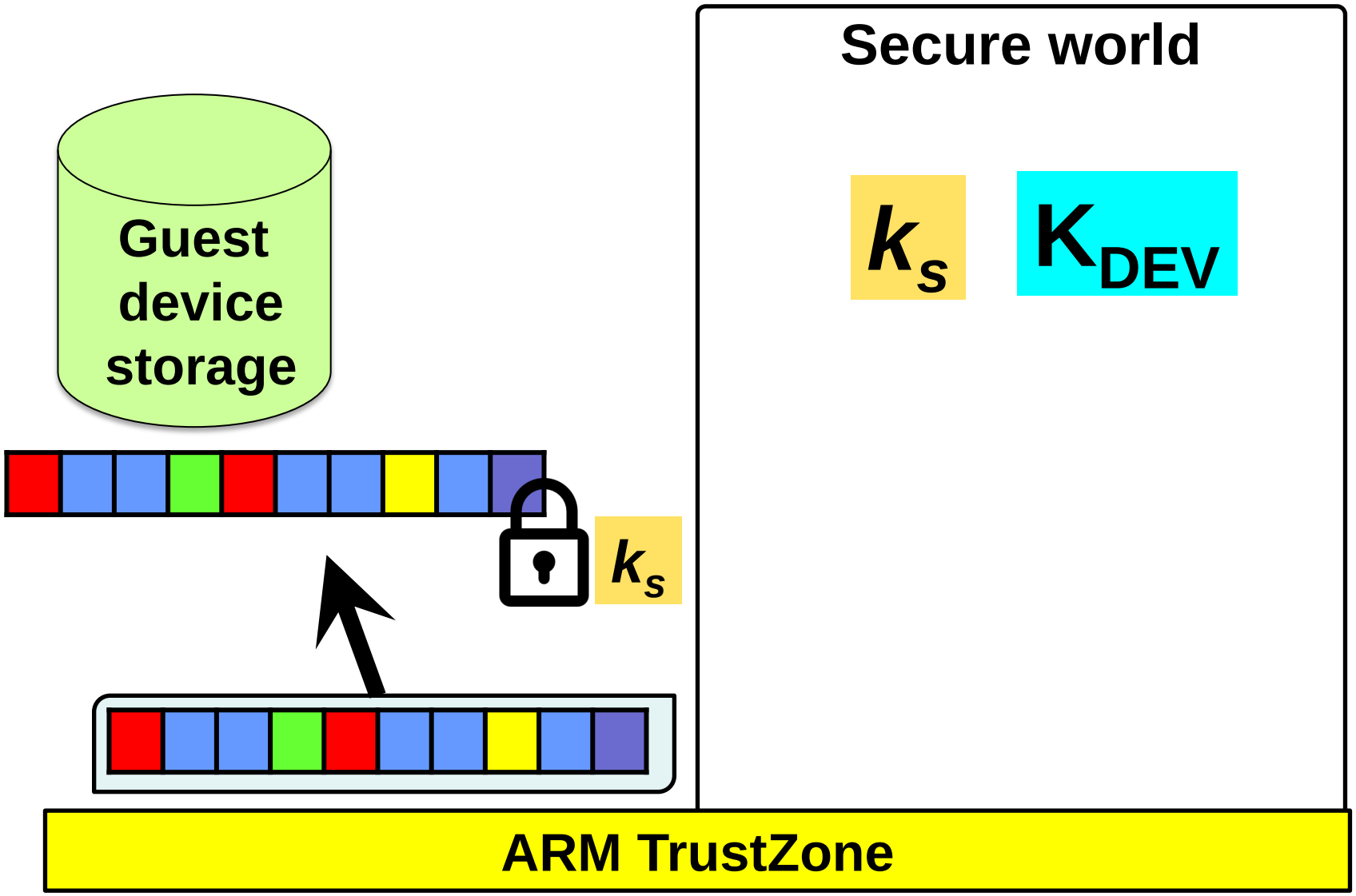
Device checkpoint

- **Problem**: Checkpoint stored on disk
  - Readable by untrusted end-user
  - But session key k_s must not be stored in clear
  - Otherwise, malicious end-user can use it to impersonate guest's trusted secure world!
- **Solution**: **REM-suspend** protocol

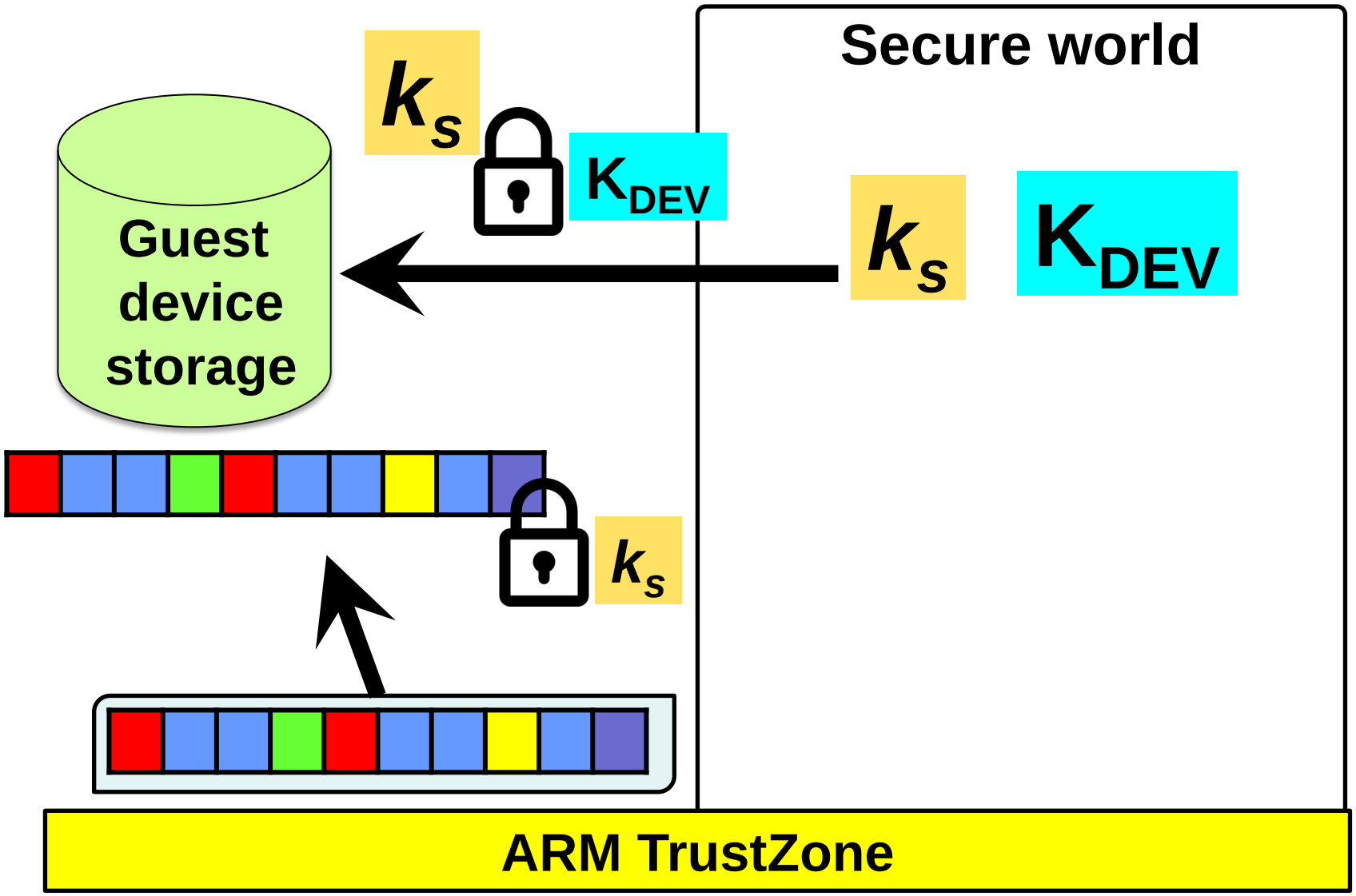
REM-suspend

- ARM TrustZone equips each device with a device-specific key K_{DEV}
- The key K_{DEV} is only accessible from the secure world
- We use K_{DEV} to encrypt k_s in device checkpoint
- When device is powered again, secure world uses K_{DEV} to decrypt and restore k_s

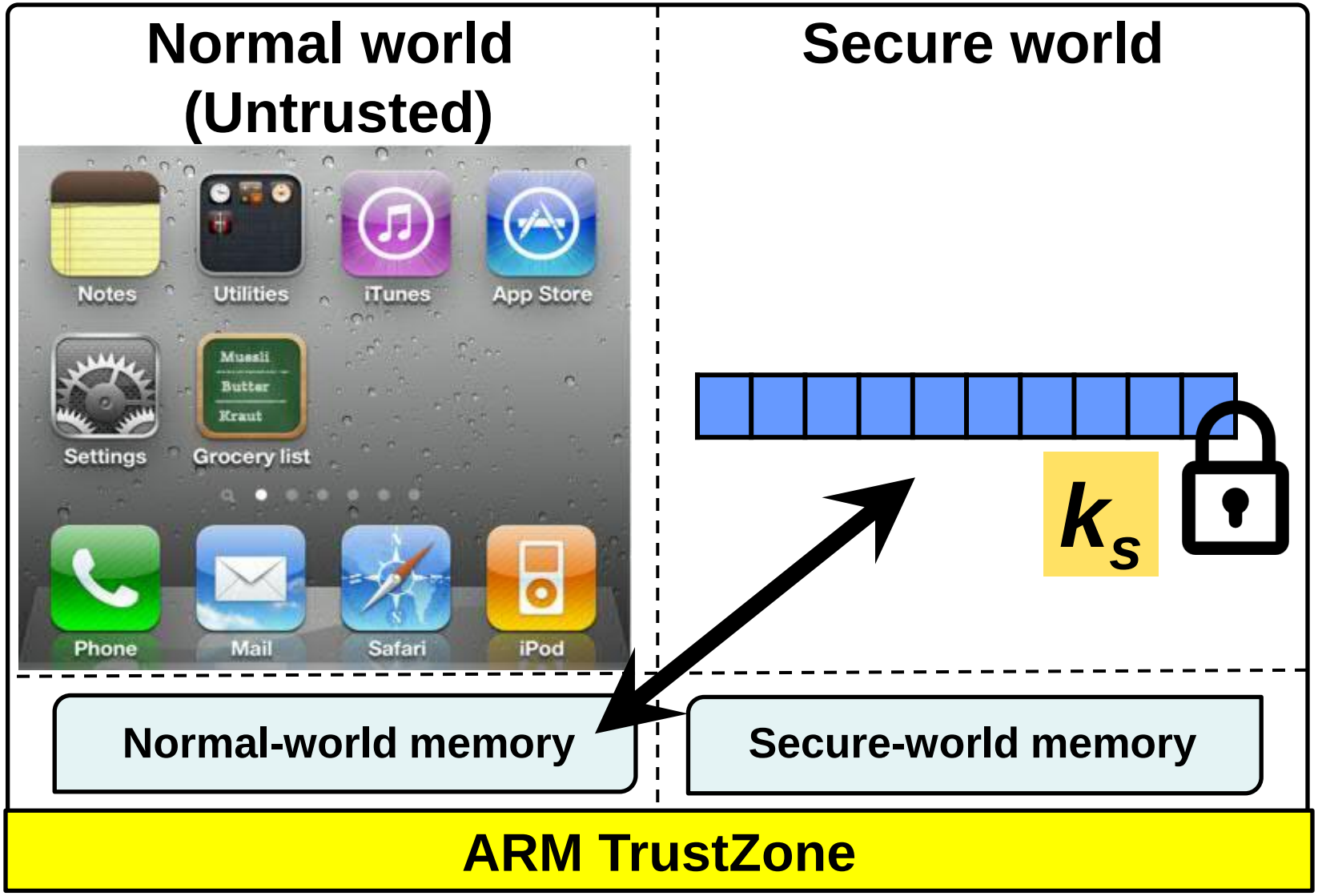
REM-suspend



REM-suspend


Secure world
k_s
K_{DEV}
Guest device storage
k_s
K_{DEV}
k_s
ARM TrustZone

SW reads NW memory



Classroom and exam setting