An MTM based Watchdog for Malware Famishment in Smartphones

Osman Ugus and Dirk Westhoff
HAW Hamburg

Presented by
Benjamin Vetter
Juni 17, 2011

Smartphones (SPs)

- Combination of mobile phones and PDAs
- Rich functionalities & features
- Support of installation 3rd party software
- Multiple networking interfaces
  - GSM, UMTS, WLAN, Bluetooth, etc.

Source: www.apple.com

Smartphones (SPs)

- Not a cell phone
  - Online banking, Internet access, Business contacts etc.
  - Security and privacy risks increasing
- Appetizing target for attackers
  - Viruses, rootkits, and sophisticated malwares already appeared

Source: www.almostlikeeverything.com

Terminology

- **Trusted App (TAP):** An App which is assumed to be secure, not malicious and allowed to be run a Smartphone (SP). Other Apps must be prevented from execution.

- **Watchdog-ed executable (WDEX):** An executable which is critical for the system security whose integrity is checked during the boot of an SP.
  - e.g., system functions enforcing security checks

Malware injection

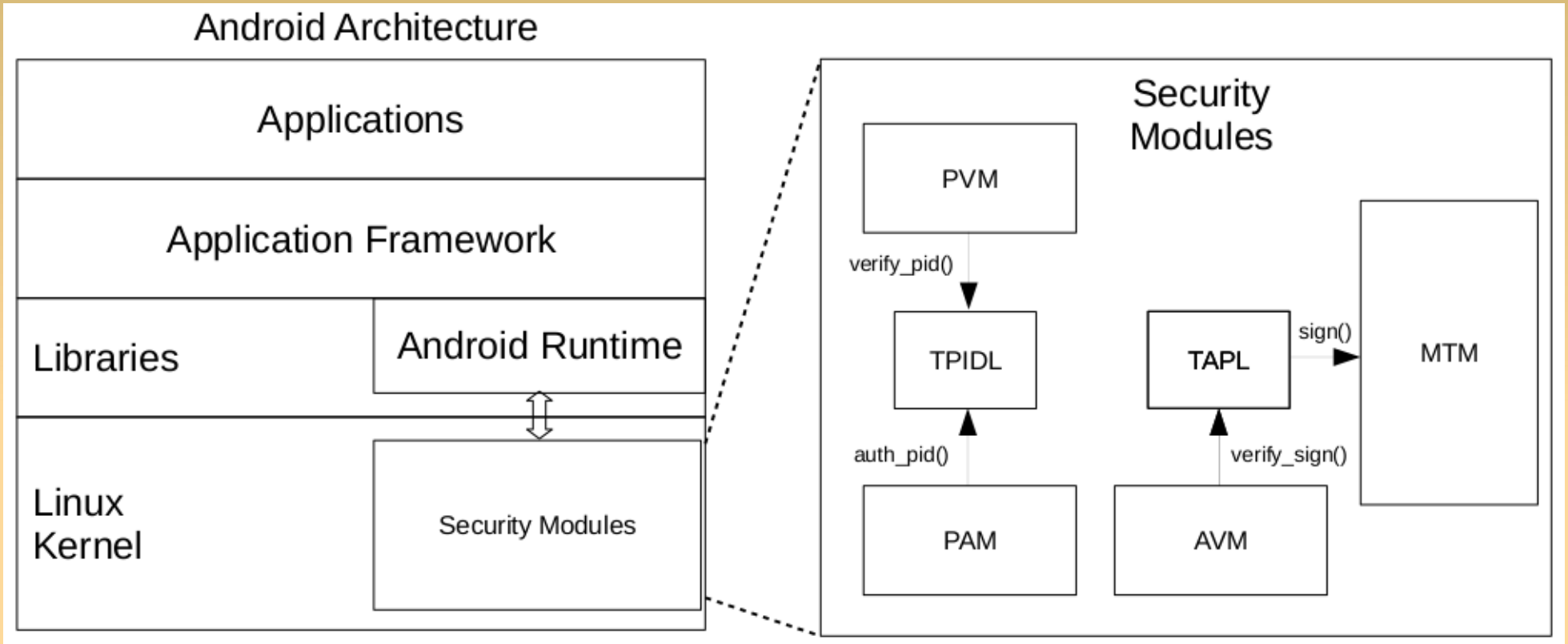
- Possible via
  - Third-party Apps
  - Vulnerabilities at Communication Protocols
  - Vulnerabilities at OS Level

- ***Our solution detects and prevents execution of Malware types that leave traces on WDEXs and TAPs***
  - e.g., computer viruses

Use Case Scenario

- Employees use SPs
  - to pursue their works while being on a business trip
  - SPs possibly store confidential customer data, price lists, offers, contracts, etc.
- Employer wants
  - non-TAPs that may contain malicious code and danger company's confidential data must never be executed on the SP
  - ***TAPs which are infected after their installation must never be executed on the SP***

Our Security Architecture



MTM: Mobile Trusted Module
PVM: Process Verification Module
PAM: Process Authentication Module
AVM: App Verification Module

TPIDL: Trusted PID List
TAPL: Trusted App List

Our Security Architecture

- MTM
  - Integrity check during boot, secure key management and storage, App signing
- TAPL
  - List of TAPs allowed to run on the SP
  - Signed with a private key stored on the MTM
  - Encrypted with a symmetric key stored on the MTM
- Key Management
  - Signature key is never revealed outside of MTM
  - Encryption key is available only if SP's integrity is intact

Our Security Architecture

- Makes it possible to run only TAPs
  - TAPs: customized Apps, secure and not malicious Apps, executables needed for the basic functionality, etc.
- Requires no trusted-third party involvement
  - MTM takes care of secure key storage and APP signing
  - Avoids unnecessary cost and complexity, easy administration
  - Apps can be declared as TAP locally
  - TAPs can be revoked locally

HAW HAMBURG

Our Security Architecture

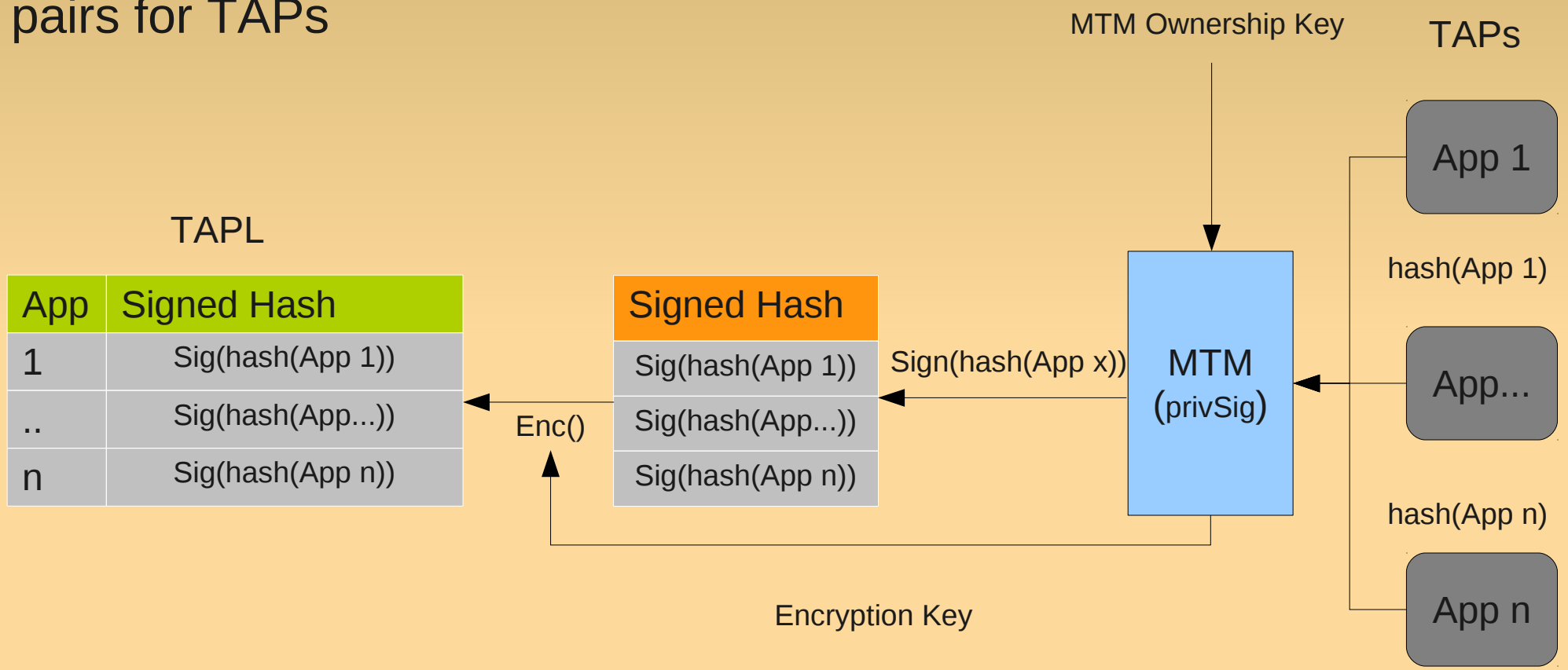
- Protects against Malware types
  - that leave traces on TAPs or WDEXs
  - i.e., that must be appended to some TAPs to perform their malicious activities (e.g., computer viruses)
- Prevents execution of TAPs that are infected after their installation
  - avoids possible damages on the SP
  - prevents further propagation of Malware

Our Security Architecture

- What is better than Antivirus tools?
  - Antivirus tools detect only Malwares with known signatures
  - Our solution detects even *zero-day* Malwares
  - Our solution ensures that Malware is never executed!
- What is better than existing Attestation protocols?
  - Attestation protocol checks the integrity. No immediate countermeasure
  - Our solution prevents polluted App from execution

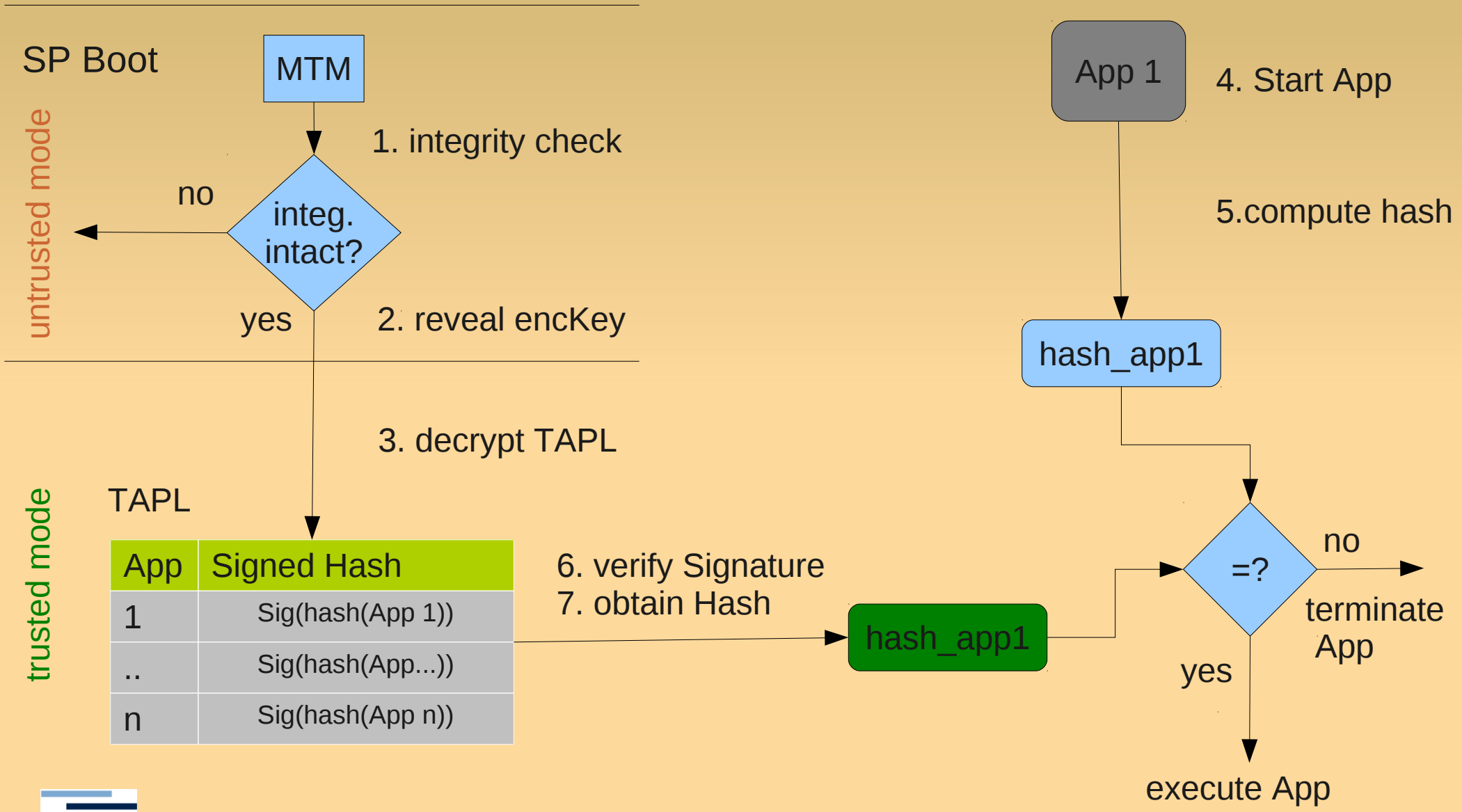
Signing TAPs

TAPL stores signature-hash pairs for TAPs



PrivSig: private signature key.

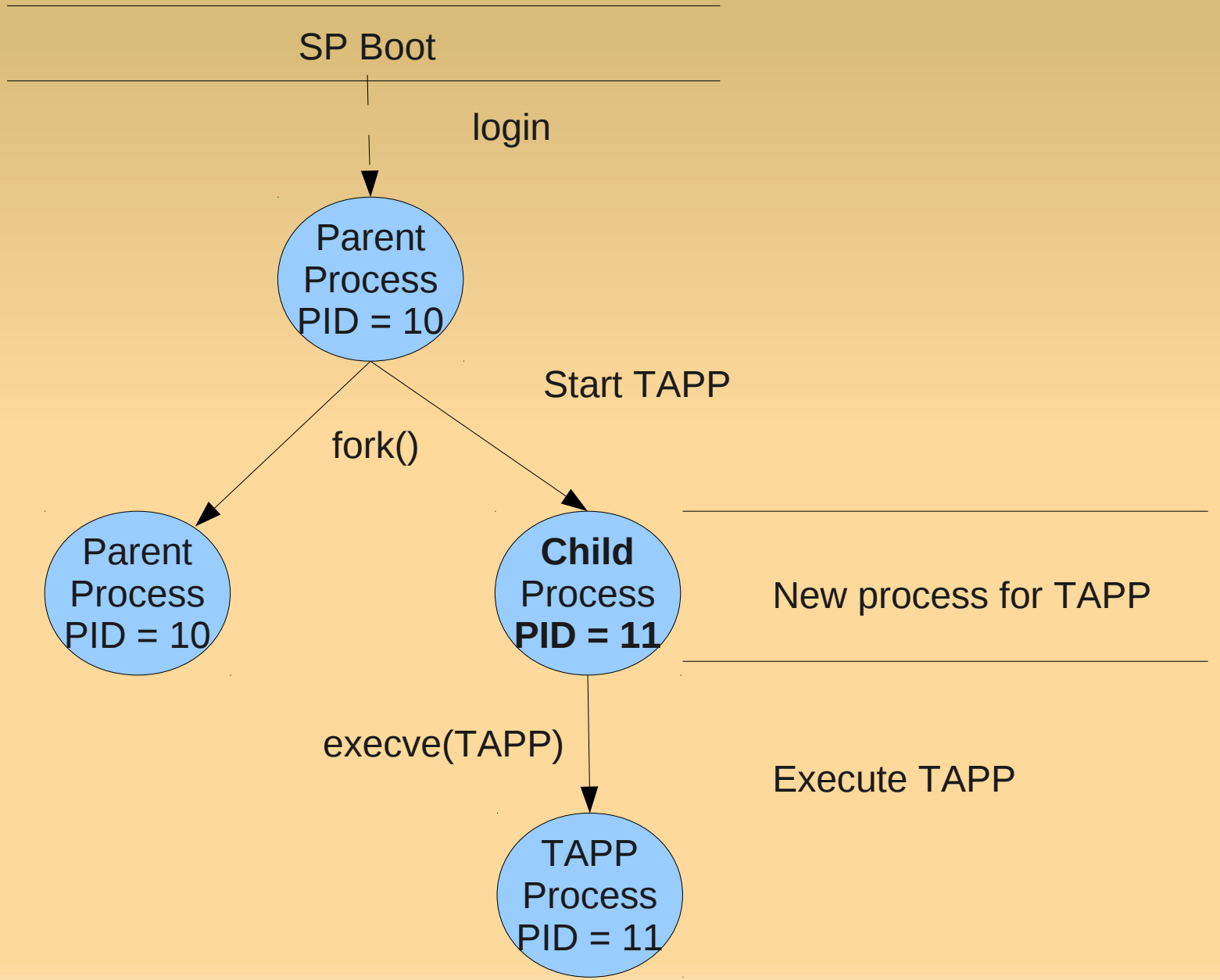
TAP Execution



TAPP Execution

- Security
  - Single bit change in TAPP results in a different hash
  - TAPP is executed *iff* hash is equal to signed hash
- Attacker needs to
  - forge signature
    - Implies breaking e.g., RSA
  - skip security checks
    - Implies breaking security of MTM

Process Creation



Trusted Process Chain

- init() is the first process.
- MTM ensures the trust on init()
- All processes are derived from init() via fork() and execve() calls
- Propagate trust from MTM through processes

Trusted Process Chain

- Each process is authenticated during its creation
- Each process is verified before its execution by execve()
- Each process is verified before being assigned with CPU by Scheduler
- fork(), execve(), scheduler needs to be extended with required security checks

Trusted Process Chain

- A Process is authenticated with MAC(PID)
- MAC Key is sealed on MTM
- MAC Key revealed if the integrity of WDEXs are intact

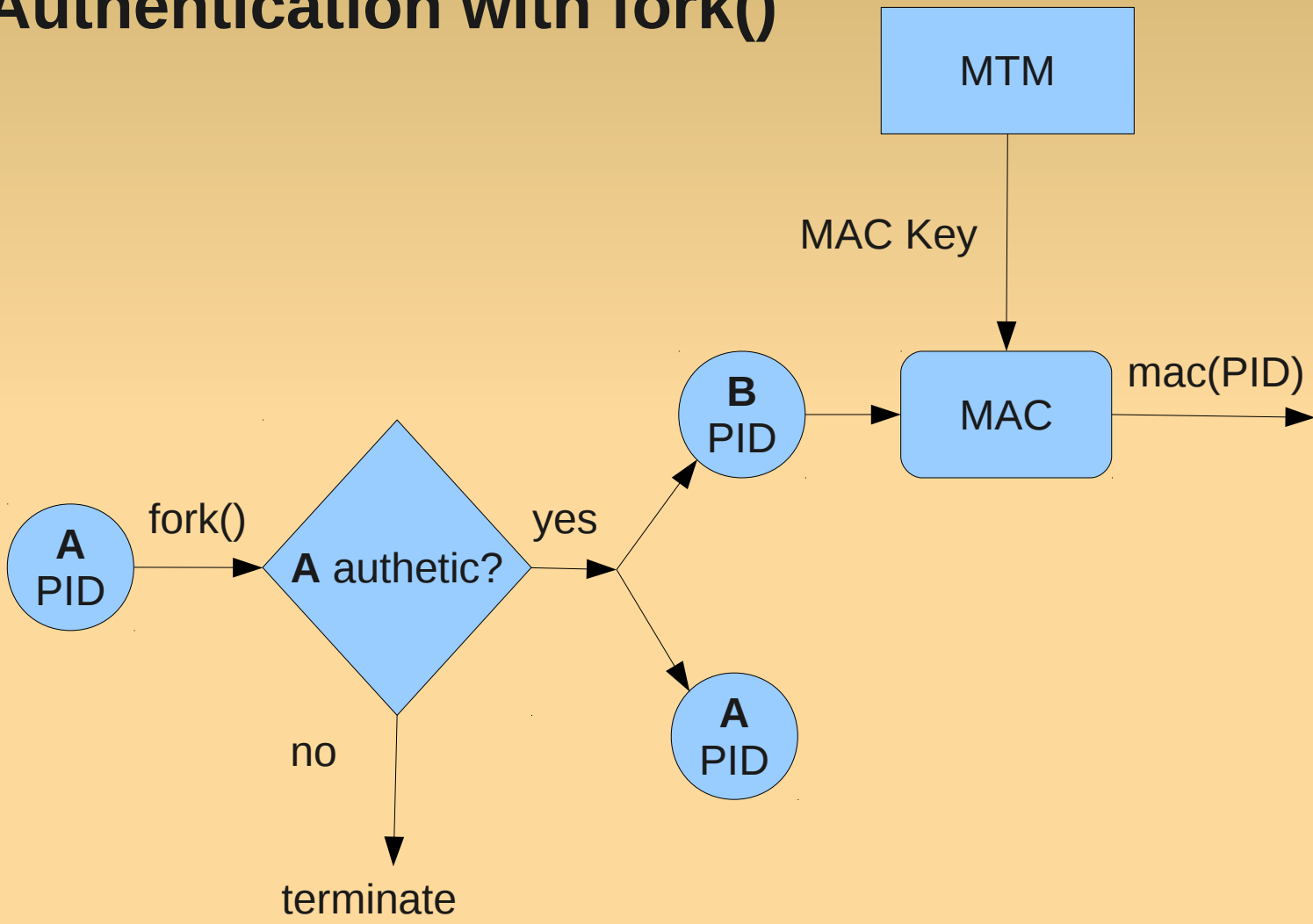
- **TPIDL:** the list of PIDs with MACs

TPIDL

PID	MAC(PID)
1	1
..	mac(...)
n	mac(n)

Process Authentication

Authentication with fork()



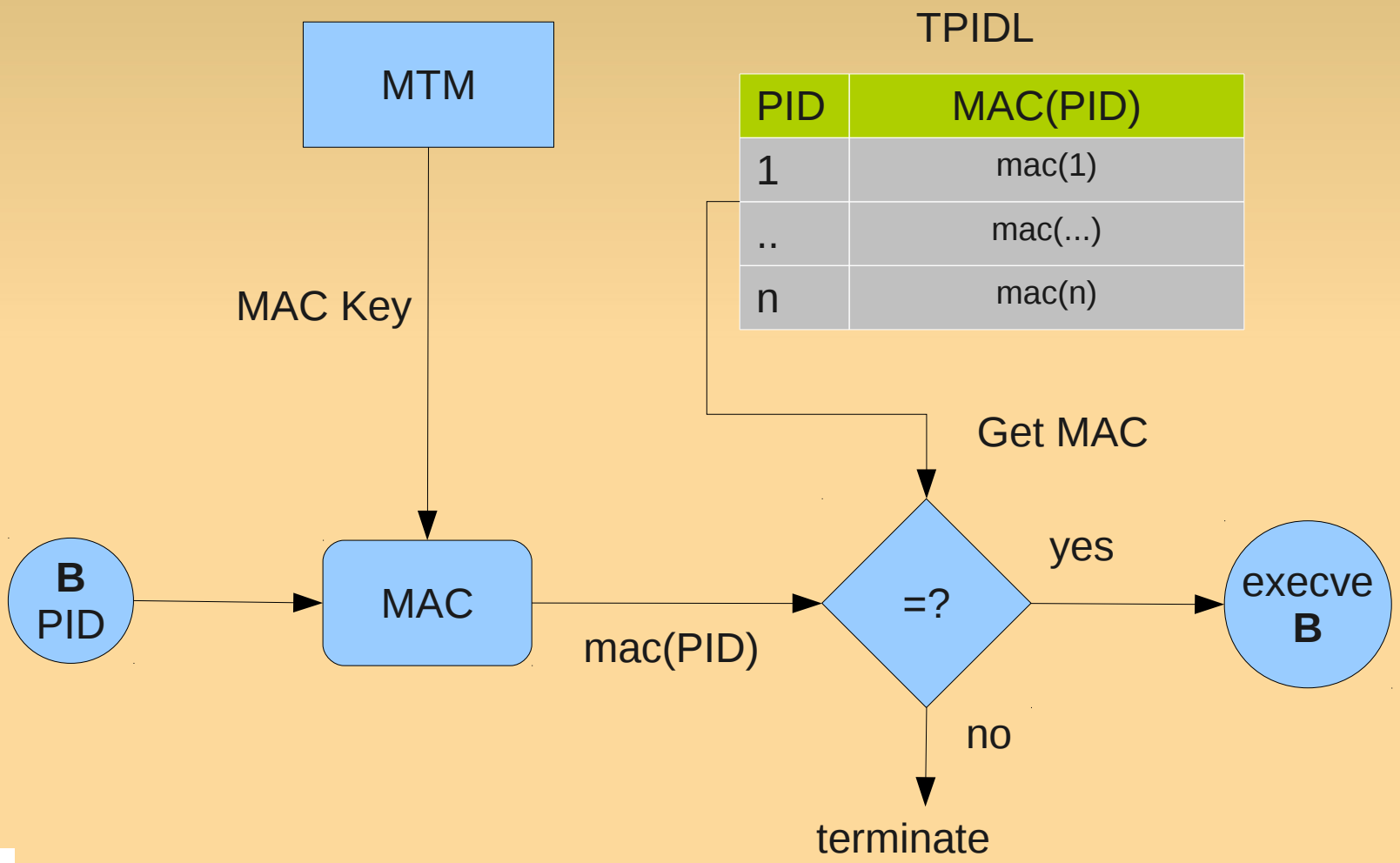
TPIDL

PID	MAC(PID)
1	1
..	mac(...)
n	mac(n)

MAC Key: revealed upon integrity verification during boot

Process Verification

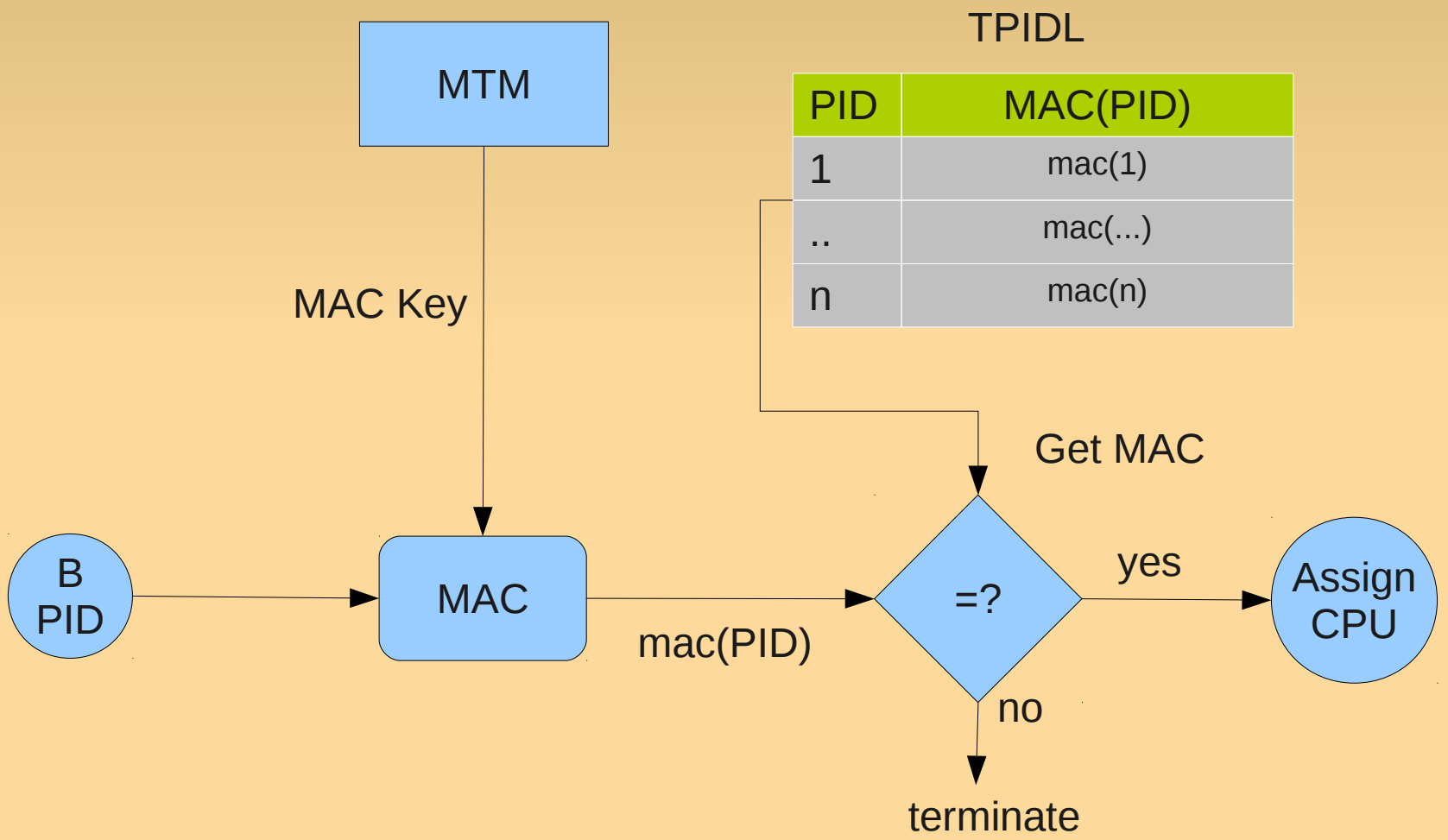
Verification with execve()



MAC Key: revealed upon integrity verification during boot

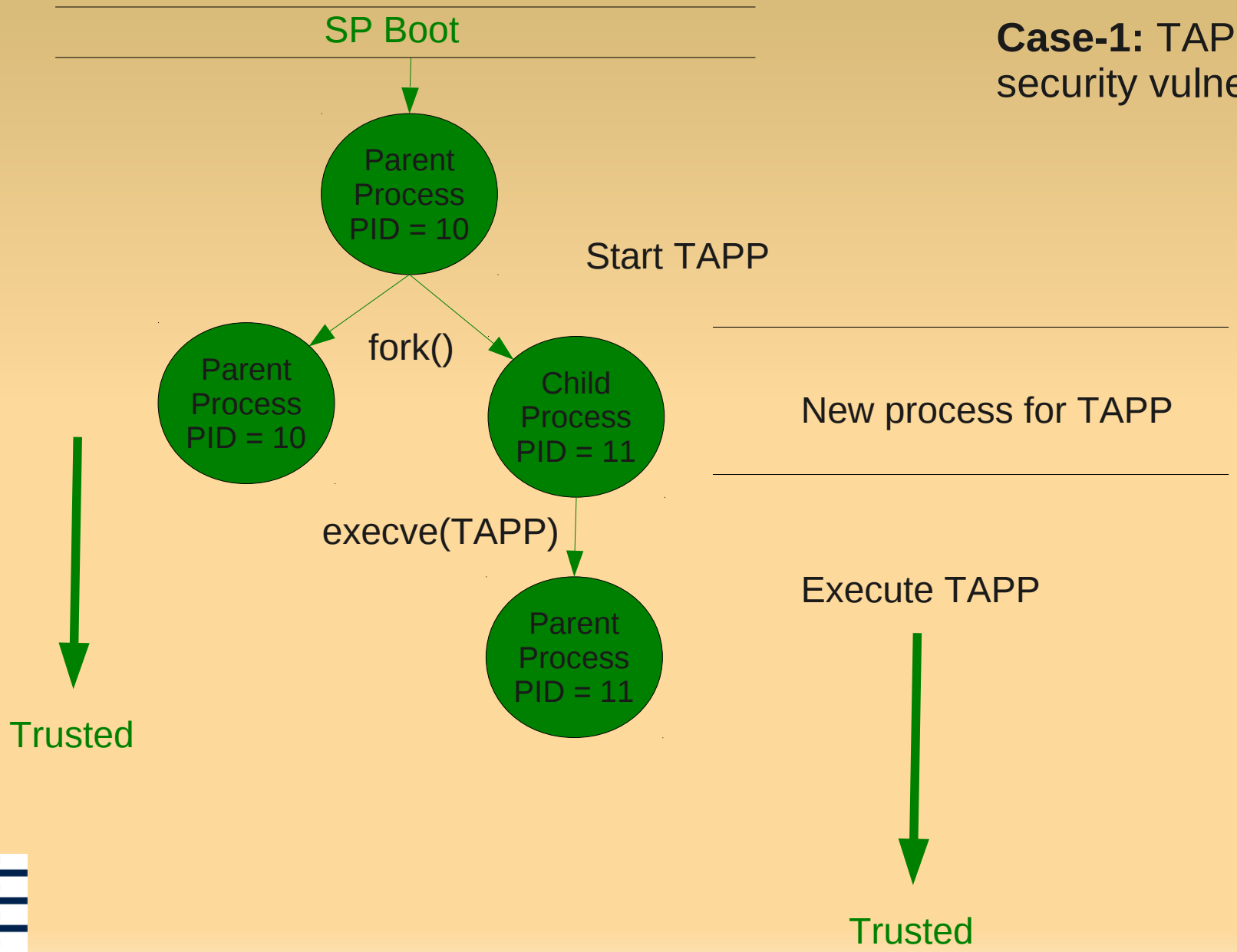
Process Verification

Verification with Scheduler()



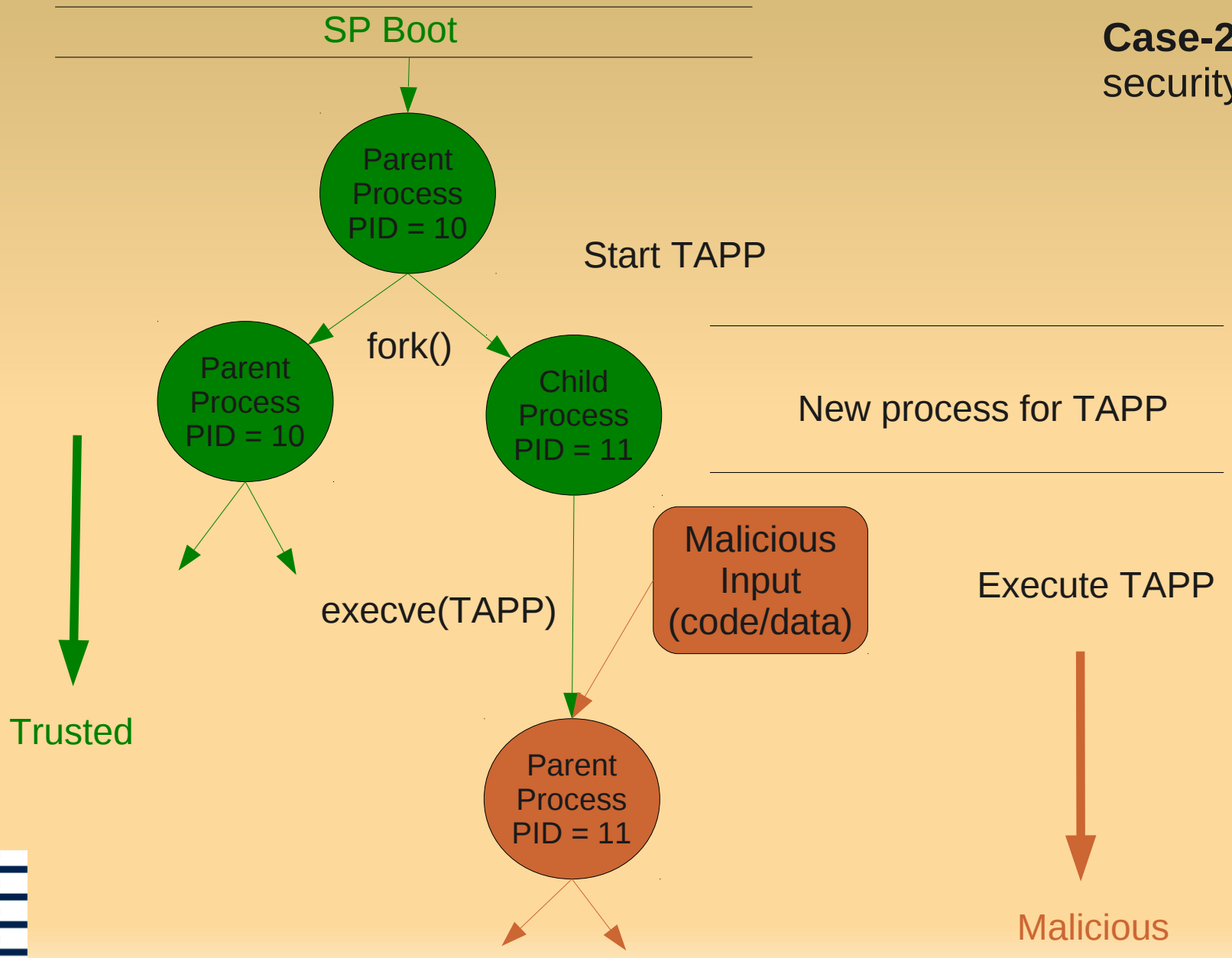
MAC Key: revealed upon integrity verification during boot

Trusted Process Chain - Security



Case-1: TAPP has no security vulnerability

Trusted Process Chain - Security



Case-2: TAPP has security vulnerability

Trusted Process Chain - Security

- TAPP has a security weakness
- Malicious input (code/data) exploits it for execution under trusted process
- Policies may be used to limit the rights for malicious code
- **Is deterministic (non-heuristic) security against such attacks possible at all?**

Conclusion

- Our solution **always detects and prevents** execution of Malware types that leave traces on the SP

- Our solution provides ***no*** protection against Malware types
  - that leave no traces on the SP
  - that exploit run-time data of TAP to run malicious code

HAW HAMBURG

Future Work

- Implementation on Google Nexus S
- MTM software based emulator
- Performance analysis and evaluation

Thank you very much for your attention!