An advanced embedded Key Management System

Flavia Caforio
Samuele Yves Cerini
Sergio Mazzola

Cybersecurity for Embedded Systems, Politecnico di Torino

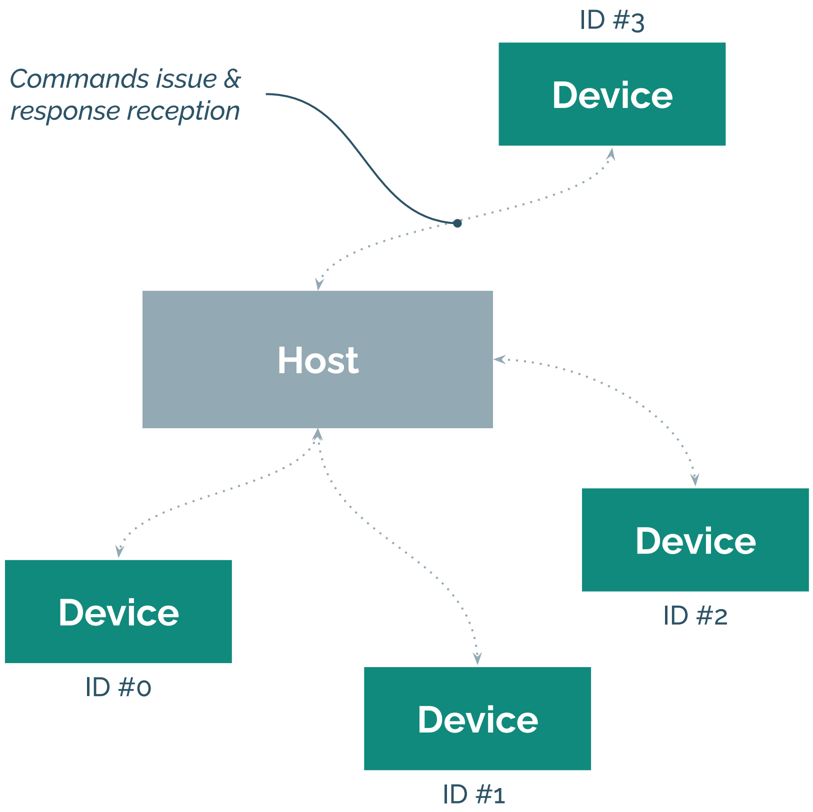
February 2021

Project features

- Development of a **cryptosystem** from scratch
    - SECube open-source **secure** platform, emulated Cortex-M4
    - **Embedded**: low-power, low-resource
- Complete **Key Management System**
    - Generate, store, delete, use keys & cryptoperiod management
    - Tampering & glitching detection
    - **Key Agreement Protocol** among two parties

Project features

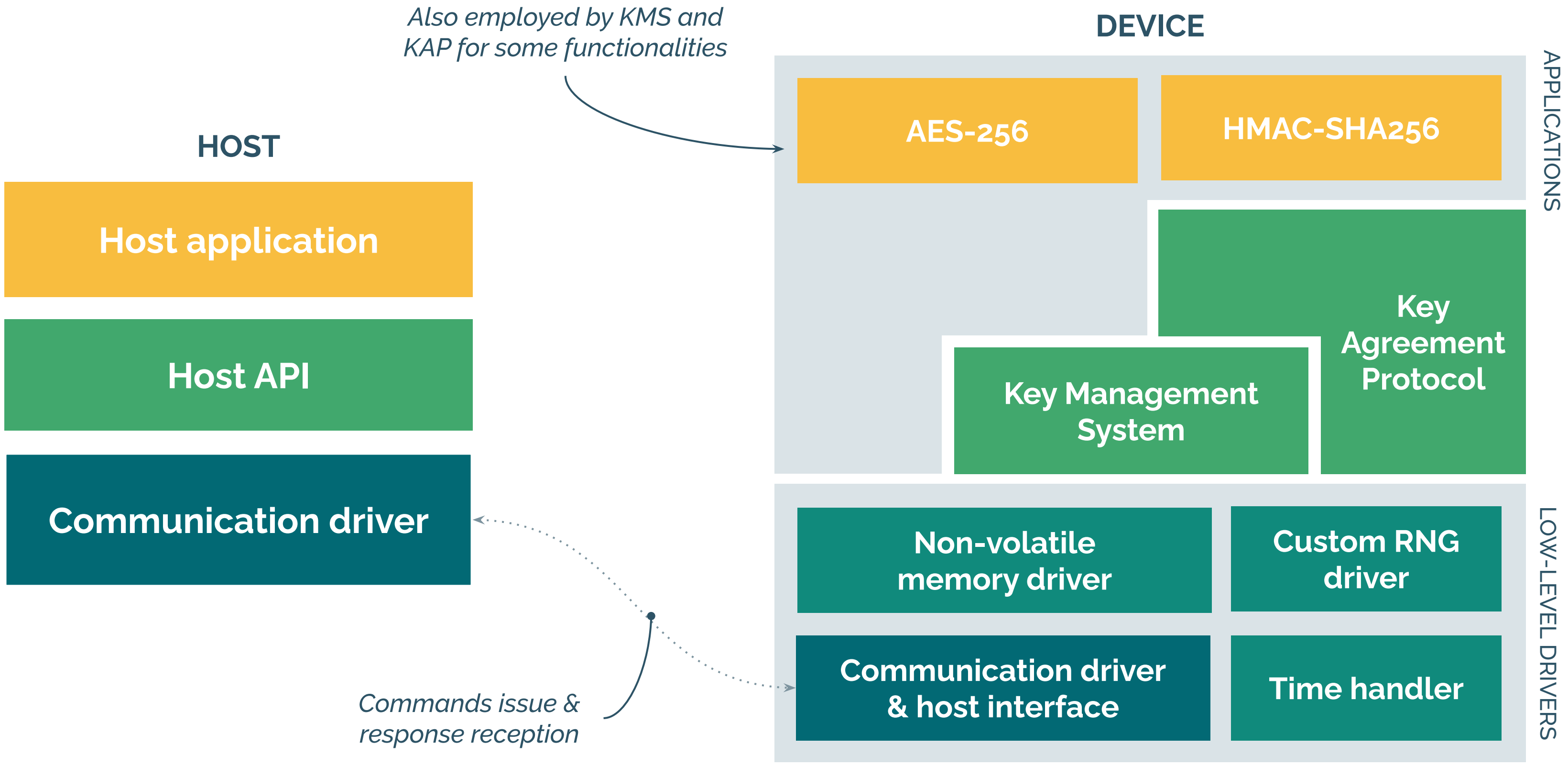
- Example cryptographic applications
  - **AES-256** encryption & decryption
  - **HMAC-SHA256** signature computation & checking
- Additional goodies
  - **Communication** framework & low-level **drivers**
  - **Layered** architecture: modular, flexible, scalable, extensible
  - Full **errors traceback** for device software
  - Complete host-side **API** to request functionalities to device
  - Fully automatic framework for **multiple devices**

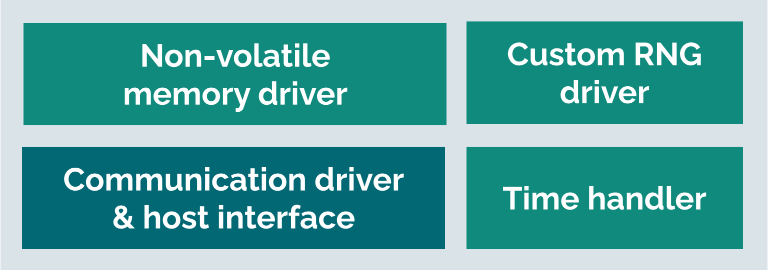


System-level architecture
communication framework

- Host **API**, device as black-box
- Automatic framework for **multiple devices** management
- Host = orchestrator: devices not aware of each other
- **Layered** communication stack
    - Easy to switch **platform**
    - Easily **extensible** with new functionalities

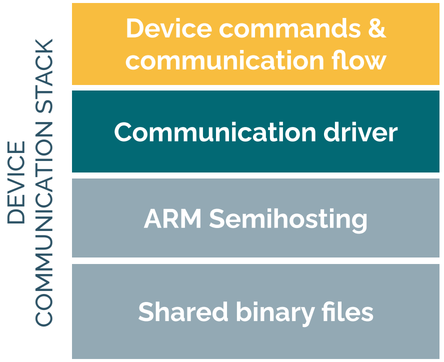
Software architecture

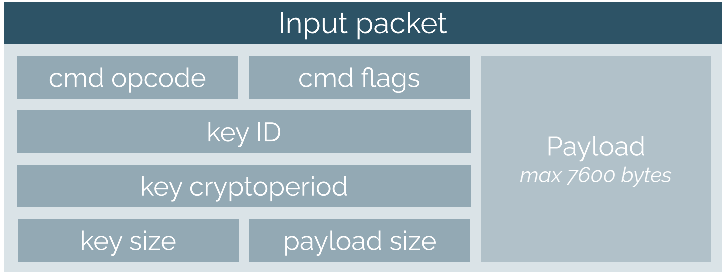


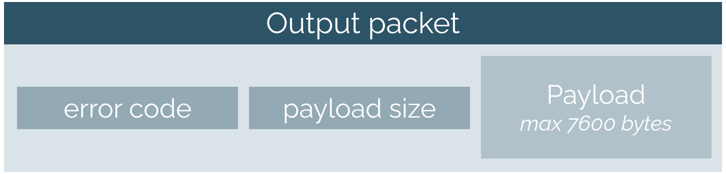


Low-level drivers

- Communication with the host
- Non-volatile memory driver
- Custom Random Number Generator
- Unix epoch Time handler

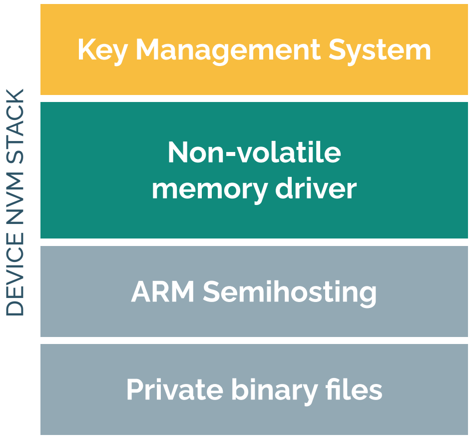






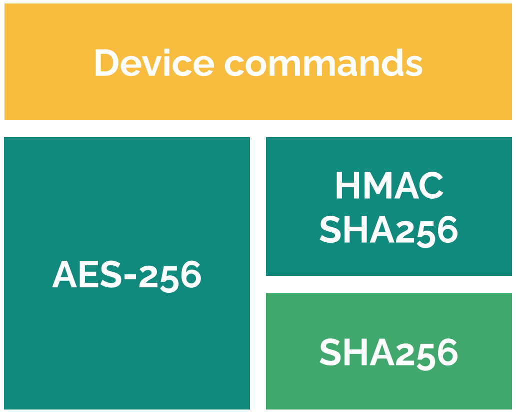
Low-level drivers

- **Communication** with the host
  - High-level commands interface towards the host
  - Low-level communication driver
    - ARM Semihosting shared files
    - Packets with fixed header
- Non-volatile memory driver
- Custom Random Number Generator
- UNIX epoch Time handler



Low-level drivers

- **Communication** with the host
- **Non-volatile memory** driver
    - ARM Semihosting private file
    - Size of 1 MB, byte-addressable
    - Static memory map; only used by the KMS database
- Custom **Random Number Generator**
    - ARM Semihosting: /dev/urandom
- Unix epoch **Time handler**
    - ARM Semihosting host time functionalities

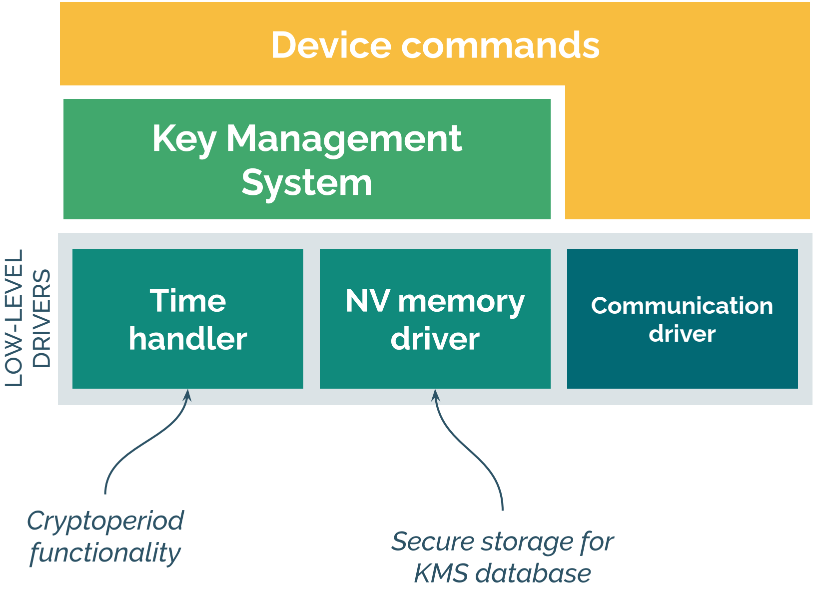


Applications

cryptographic functionalities

- **AES-256** encryption & decryption
  - ECB or CBC modes
  - Block length of 16 bytes, padding done with zeros
- **HMAC-SHA256** signature computation & check

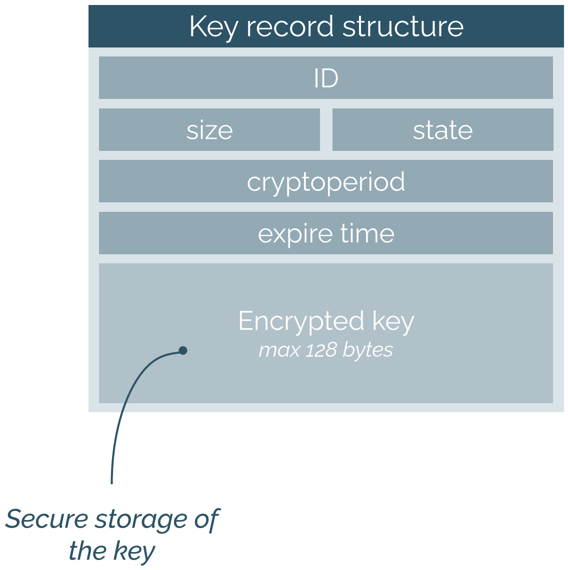
1. Requested by host; use KMS keys
2. Used also by KMS and KAP implementations



Key Management System

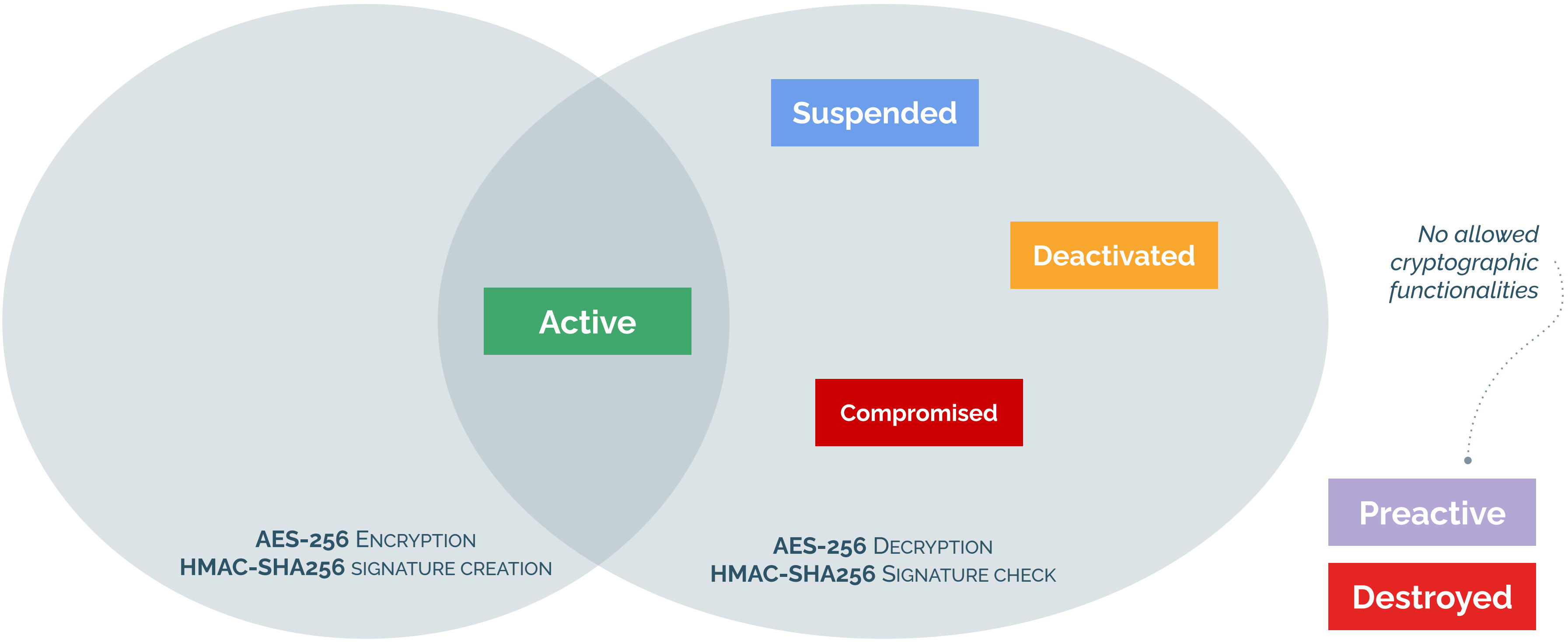
- Generate, add, remove, update, list keys
- Keys lifecycle management
- Device cryptographic functionalities use keys from KMS database
- Secure storage in Non-volatile memory
- Tampering & glitching detection

KMS - Functionalities

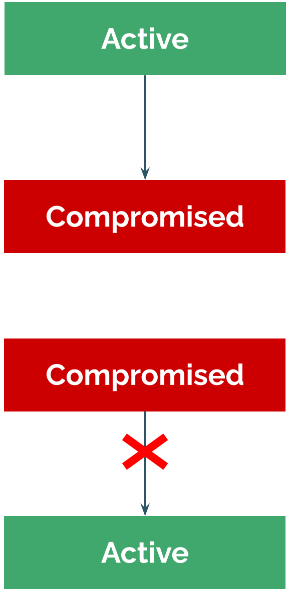


- Manage a key for its entire **lifecycle**
  - From its creation and use to its destruction
- Introduction of the **cryptoperiod** concept
  - A key has limited time of use
    - After which is considered to be expired…
    - … and its privileges are reduced
  - Goal: enforce a periodic substitution of the key
    - Limits the damages in case of key disclosure
- Host-side **API**
  - Key state commands
    - Activate, suspend, deactivate, compromise, destroy
  - Key management commands
    - Add, remove, update, list

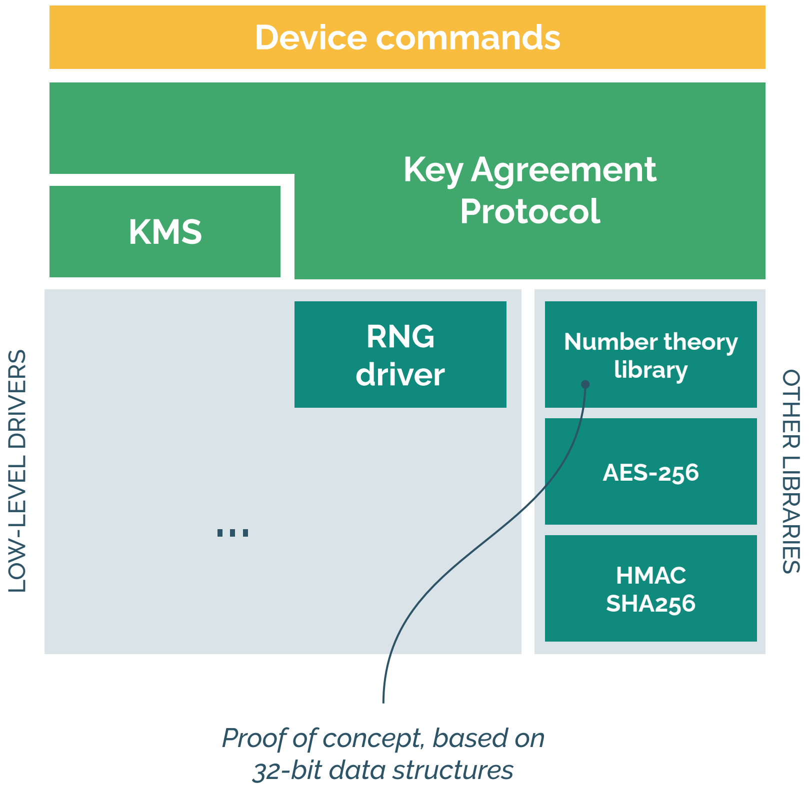
KMS - State privileges



KMS - Tampering & glitching detection



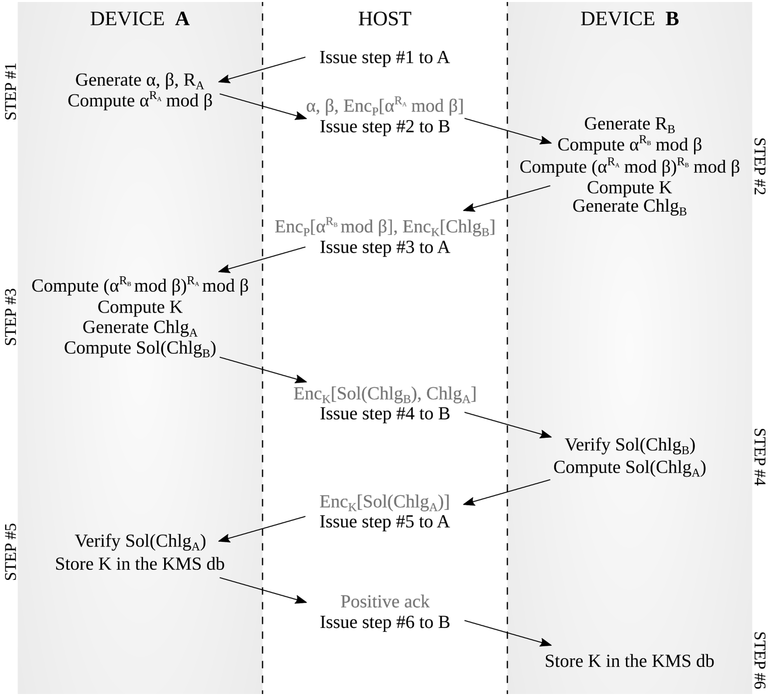
- Goal: enforce a safe state graph
  - Some transitions are normally not allowed
  - The device itself can change a key state automatically
- Problem: Malicious activity may exist
  - Attacker goal: move a key from a low-privileged state to a high-privileged one
  - How: stolen device + non-invasive attacks
- Solution:
  - Detect external tamper and glitching attempts
    - Leverage the compromised state
    - Block any action and inform the host of suspicious activity



Key Agreement Protocol

- Password-based Encrypted Key Exchange with Exponential Key Exchange
- Aim: solve Diffie-Hellman criticalities
  - Keys distribution in symmetric key cryptography
  - Secure against MitM, replay and offline dictionary attacks
- Agreement of keys up to 256 bit
  - Between two distributed devices
  - The host orchestrates the protocol flow

KAP - Protocol flow & security



- Every exchanged quantity is encrypted with P
  - Man-in-the-middle, offline dictionary attacks
- Exchange of random challenges
  - Replay attacks

Choice of parameters

- β large enough prime number
- β-1 must have at least one large factor
- α primitive root in GF(β)
  - against discrete logarithm computation

KAP - Implementation details

- Protocol based on 32-bit data structures
- Data exchanged among parties encrypted with AES-256
- Shared secret hashed with HMAC-SHA256 to generate a shared key K of up to 256 bits
- P and H shared secrets hardcoded in every device
- Generation of β (large random prime number)
  - generate large random number and perform fast primality test
- Communication framework integration
  - 6 new commands (one for each step) + KAP reset command
  - KAP-related commands have pre-defined payload encoding

- Demo
- Q&A