上海科技大学
ShanghaiTech University

Verifying Hardware Security Modules with Information-Preserving Refinement

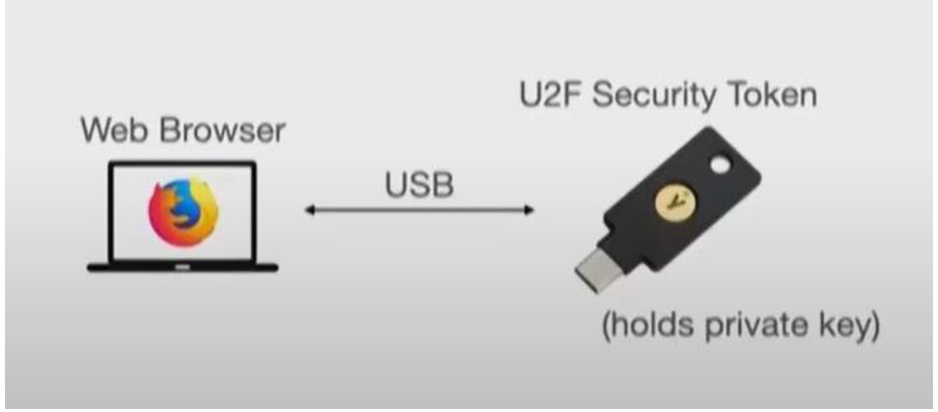
papers for reading group
Cunhan You
2023/3/3

Anish Athalye, M. Frans Kaashoek, Nickolai Zeldovich
MIT CSAIL
OSDI22

立志成才报国裕民

HSMs: powerful tools for securing system

- Factor out core security operations
- Provide security under host compromise
- Many types of HSMs
  - U2F token
  - PKCS#11
  - Hardware wallet
  - iPhone Secure Enclave
- Hundreds of millions of deployed HSMs



HSMs suffer from bugs

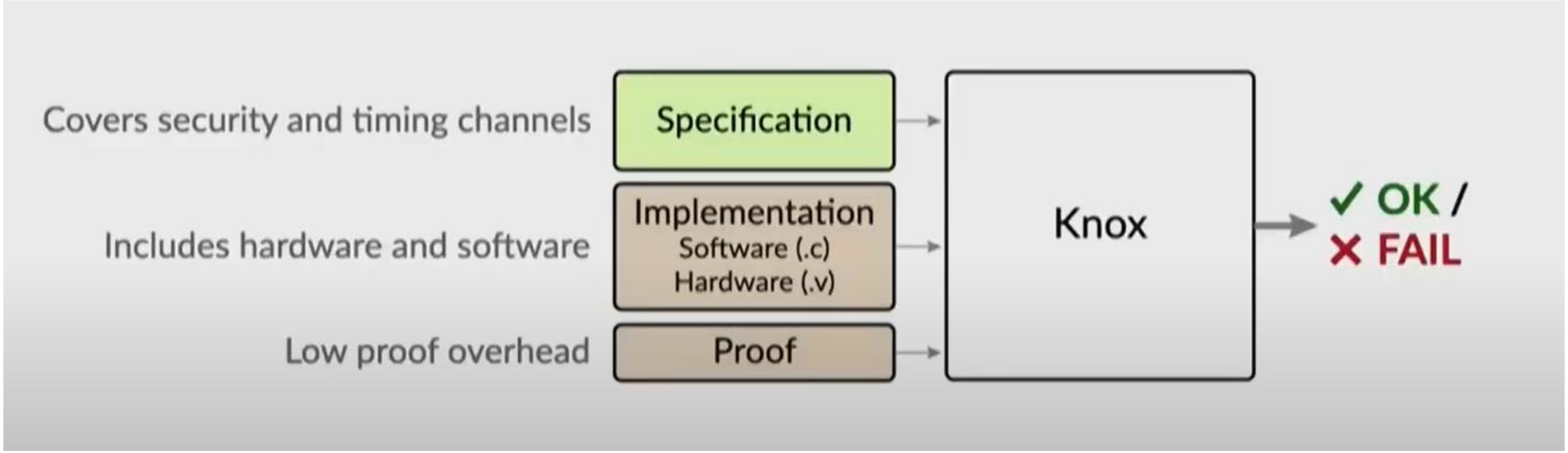
- Hardware
- Software
- Timing side channels

Goal: HSMs without security vulnerabilities

- Rule out hardware, software, and timing side-channel vulnerabilities

- Threat model
  - Powerful adversary that gains control of host machine
  - Full control over I/O interface to HSM
  - Physical attacks and other side channels: out of scope

Approach: formal verification



Related work

- Knox is the first to verify correctness and security of hardware and software including timing side channels

- Hardware/software co-verification: Bedrock2 [PLDI'21], CakeML [PLDI'19]
  - Focused on correctness, not security
- Application security verification: Ironclad Apps [OSDI'14]
  - Doesn't cover hardware or side channels

Contributions

上海科技大学
ShanghaiTech University

- Information-preserving refinement (IPR)
  - a new security definition
- Knox framework
  - for verifying HSMs using IPR
- Case studies
  - built and verified 3 simple HSMs
    - PIN-protected backup HSM
    - Password-hashing HSM
    - TOTP token
- Approach rules out hardware bugs, software bugs, and timing side channels

立志成才报国裕民

Example: PIN-protected backup HSM

上海科技大学 ShanghaiTech University

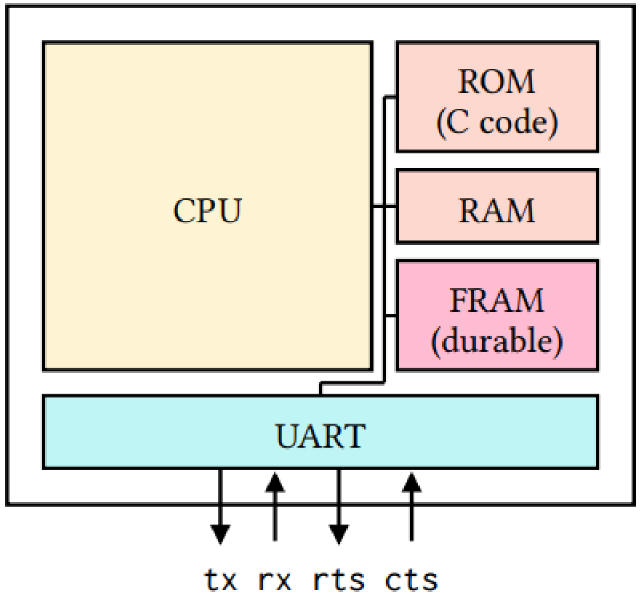
- Functional specification
- Describes input-output behavior
  - No notion of timing

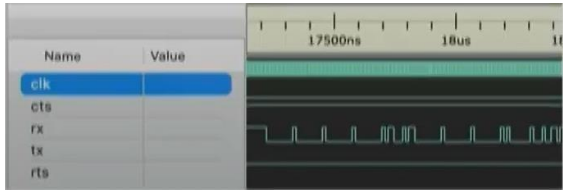
```
var bad_guesses = 0, secret = 0, pin = 0

def store(new_secret, new_pin):
  secret = new_secret
  pin = new_pin
  bad_guesses = 0

def retrieve(guess):
  if bad_guesses >= 10:
    return 'No more guesses'
  if guess == pin:
    bad_guesses = 0
    return secret
  bad_guesses = bad_guesses + 1
  return 'Incorrect PIN'
```

立志成才报国裕民

Implementation

- Implementation includes hardware/software
  - CPU
  - Code that runs on it
  - Peripherals
  - Persistent memory
  - …
- Interface: wires
  - Read output wires
  - Write input wires
  - Wait for a cycle





How to relate impl to spec

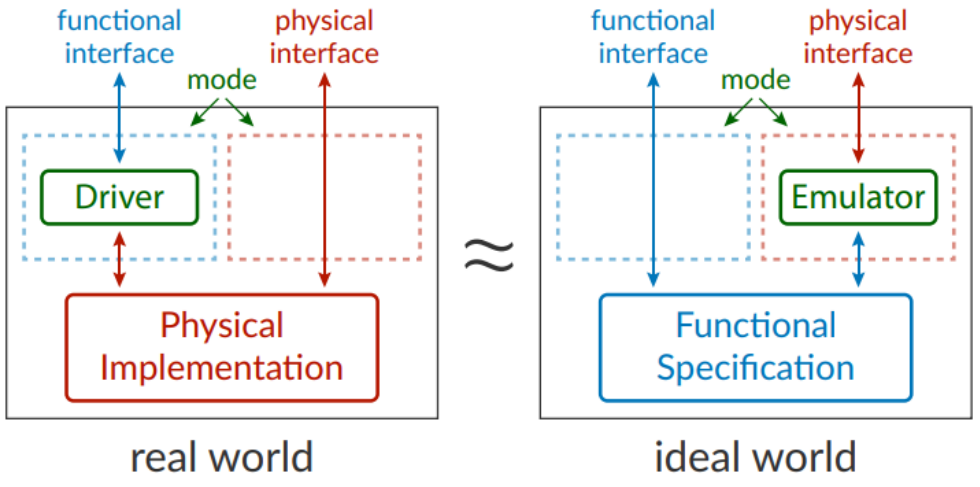
- Want to capture:
  - (1) Functional correctness: implementation implements spec
  - (2) Non-leakage: Wire-level interface leaks no more than specIncluding timing, e.g., PIN comparison with strcmp()

- Implementation is at the level of wires
- Specification is at the level of functions (has no notion of wires)

Information-preserving refinement (IPR)

上海科技大学 ShanghaiTech University

- Defined as indistinguishability between a real and an ideal world
- Inspired by formalization of zero knowledge in cryptography
- Interface adapters in each direction



立志成才报国裕民

IPR: driver

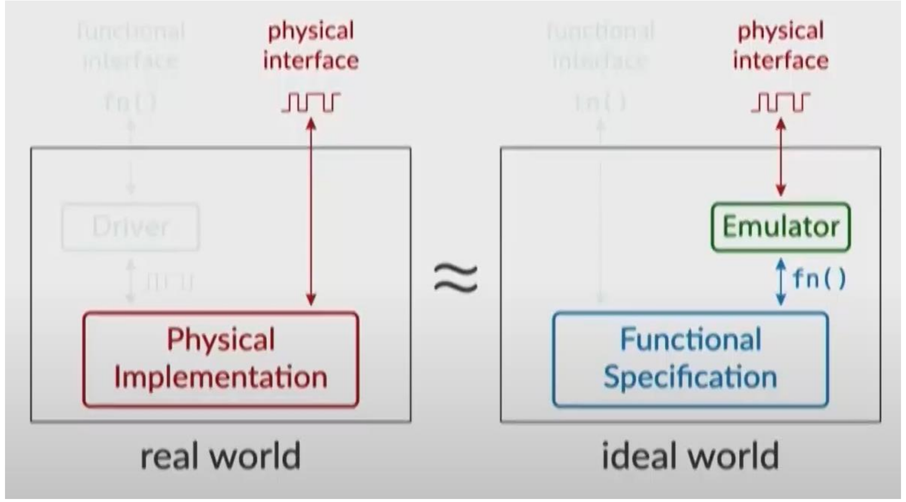
- Driver: translates spec-level operations to wire-level I/0
- Like a device driver in an OS
- Trusted, part of the specification
- Captures functional correctness

```
(define (store secret pin)
  (send-byte #x02) ; command number
  (send-bytes pin)
  (send-bytes secret)
  (recv-byte)) ; wait for ack

(define (wait-until-clear-to-send)
  (while (get-output 'rts))
    (tick))) ; wait a cycle

(define (send-bit bit)
  (set-input 'rx bit)
  (for ([i (in-range BAUD-RATE)])
    (tick)))

(define (send-byte byte)
  (wait-until-clear-to-send)
  (send-bit #b0) ; send start bit
  ;; send data bits
  (for ([i (in-range 8)])
    (send-bit (extract-bit byte i)))
  (send-bit #b1)) ; send stop bit

(define (send-bytes bytes)
  (for ([byte bytes])
    (yield) ; wait for arbitrary number of cycles
    (send-byte byte)))
```

IPR: emulator

上海科技大学
ShanghaiTech University

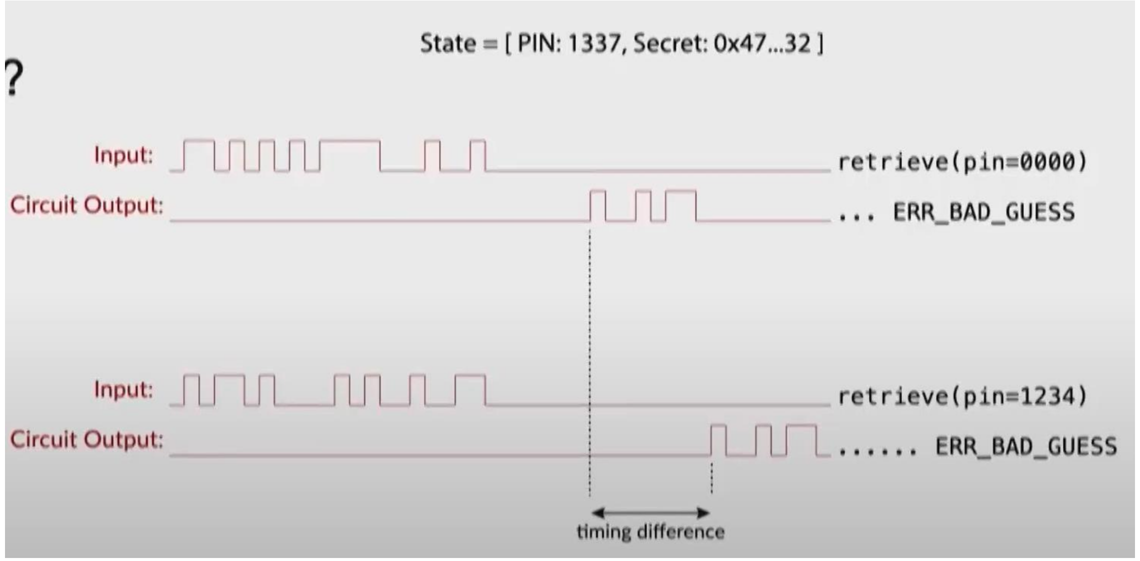
- Emulator mimics wire-level behavior
  - Without direct access to secrets
  - With queries to spec-level operations
- Proof artifact, constructed by developer(just needs to exist)
- Captures non-leakage



立志成才报国裕民

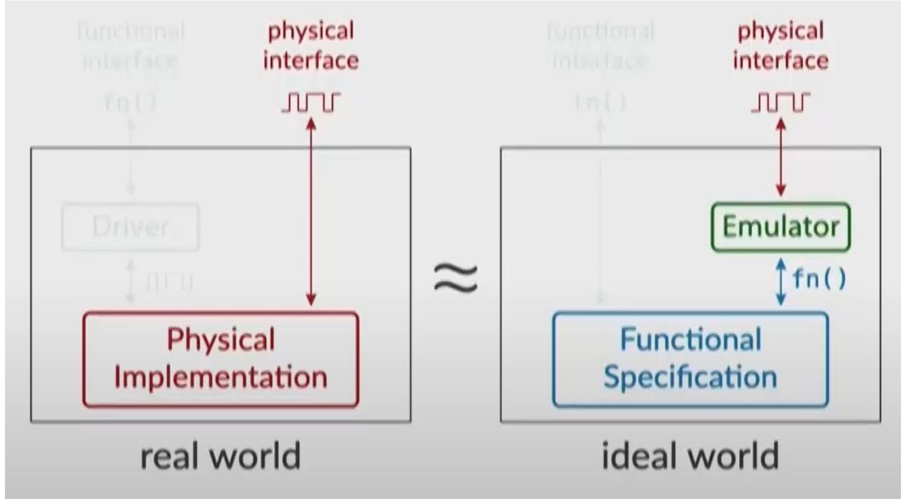
IPR rules out timing channels

- What if circuit leaked info through timing, e.g., strcmp()?
- Emulator does not exist: can get return value using query to retrieve(), but can't reproduce timing behavior



IPR: emulator construction

- Copy circuit, but replace operations on secret state with queries to spec



IPR transfers security properties from spec

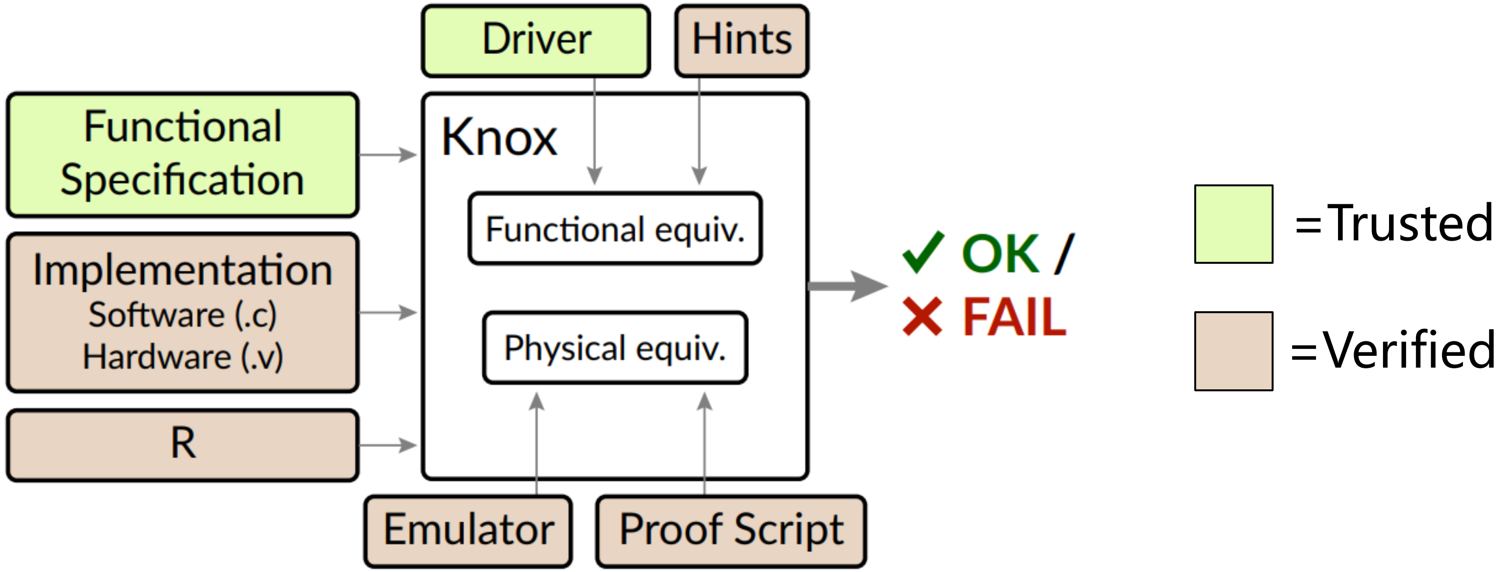
- Only reveals secret when correct PIN supplied
- Enforces guess limits
- Forgets old secret/pin when store() is called
- Doesn't leak past PIN guesses

```
var bad_guesses = 0, secret = 0, pin = 0

def store(new_secret, new_pin):
  secret = new_secret
  pin = new_pin
  bad_guesses = 0

def retrieve(guess):
  if bad_guesses >= 10:
    return 'No more guesses'
  if guess == pin:
    bad_guesses = 0
    return secret
  bad_guesses = bad_guesses + 1
  return 'Incorrect PIN'
```

Knox framework

上海科技大学 ShanghaiTech University

- ~3000 LOC on top of Rosette [PLDI'14]
- Symbolically execute entire circuit + code
- Relies on human guidance through *hints*



立志成才报国裕民

Evaluation: case studies

上海科技大学 ShanghaiTech University

- 3 simple HSMs, run on an FPGA
- Hardware: minimal RISC-V CPU, cryptographic accelerator, UART, …
- Software: control logic, peripheral drivers, HOTP, HMAC, …
- Succinct specifications
- Low proof overhead

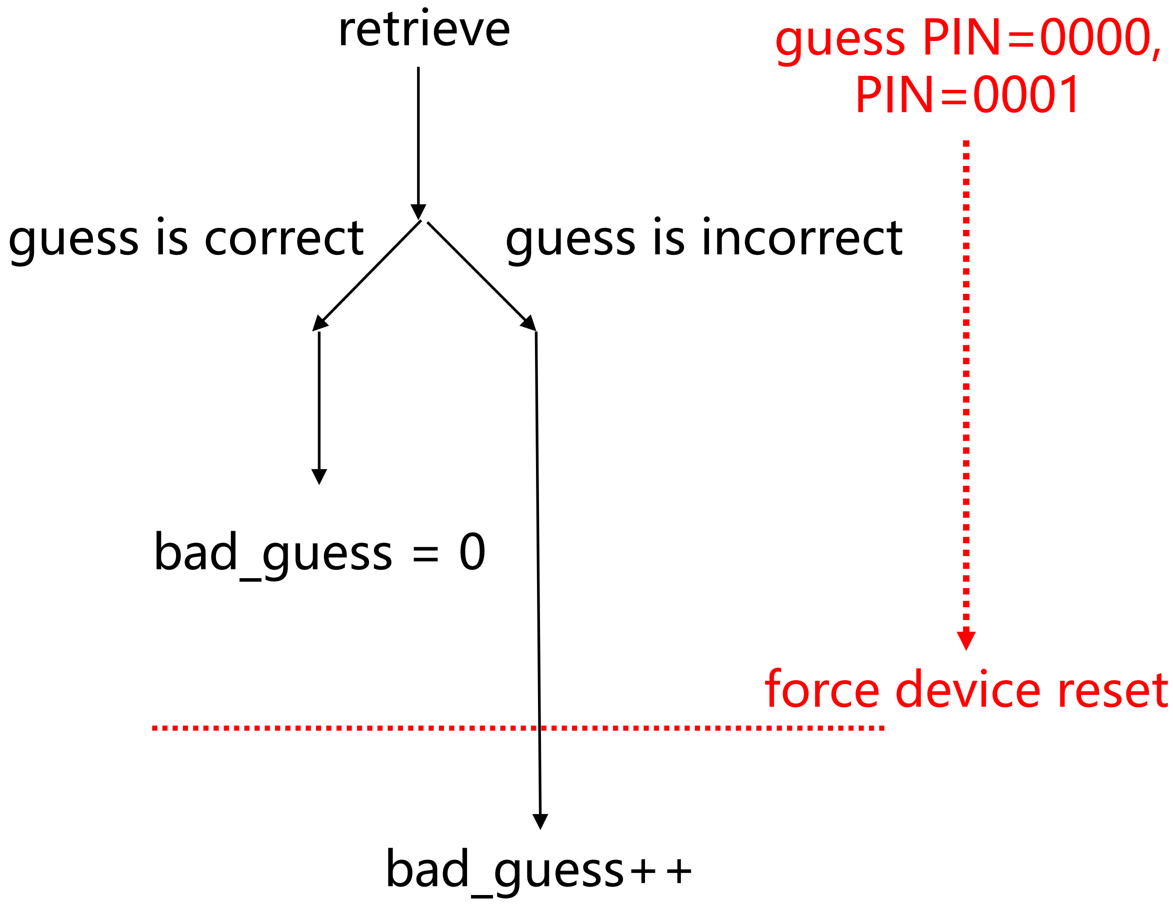
HSM	Spec core	Spec total	Driver	HW	SW	Proof
PIN backup	32	60	110	2670	190	470
PW hasher	5	150	90	3020	240	650
TOTP	10	180	80	2950	360	830

Lines of code for case studies

立志成才报国裕民

Subtle bug involving persistence & timing

```c
// return error if PIN guess limit exceeded
// ...

// check PIN guess and update guess_count accordingly
if (!constant_time_cmp(&entry->pin, guess)) {
    entry->bad_guesses++;
    uart_write(ERR_BAD_PIN);
    return;
}
entry->bad_guesses = 0;

// output secret
// ...
```



Adversary can't tell which branch was taken
(no outputs up to this point) but still, security bug!
Resets guess count to 0

Real implementations have similar code

- SoloKey: pattern similar to the bug
- Other HSMs like OpenSK have more robust code to avoid this issue

```c
    int8_t ret = verify_pin_auth_ex(CM->pinAuth, (u

    if (ret == CTAP2_ERR_PIN_AUTH_INVALID)
    {
        ctap_decrement_pin_attempts();
        if (ctap_device_boot_locked())
        {
            return CTAP2_ERR_PIN_AUTH_BLOCKED;
        }
        return CTAP2_ERR_PIN_AUTH_INVALID;
    }
    else
    {
        ctap_reset_pin_attempts();
    }
```

Conclusion

- Information-preserving refinement (IPR)
  - Implementation reveals no more information than specification
- Knox framework
  - For verifying HSMs using IPR
- Case studies
  - Built and verified 3 simple HSMs

anish.io/knox