Verifying Hardware Security Modules with Information-Preserving Refinement

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

HSMs: powerful tools for securing systems

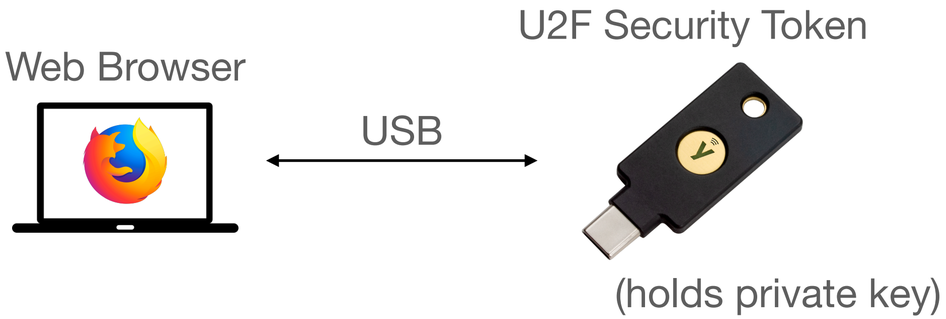
Factor out core security operations

Provide security under host compromise

Many types of HSMs

- U2F token
- PKCS#11 HSM
- Hardware wallet
- iPhone Secure Enclave

Hundreds of millions of deployed HSMs



HSMs suffer from bugs

Hardware

Software

Timing side channels

STM32F303xB/C — Description of device errata

2.2.2 Data Read when the CPU accesses successively SRAM address “A” and SRAM address “A + offset of 16 KBytes (0x4000)”

Description

If the CPU writes to an address A in the SRAM memory and immediately (the cycle after) reads an address B in the SRAM memory, while B = A+0x4000, the read operation will return the content at address A instead of the content of address B.

CVE-ID	
CVE-2019-18672	Learn more … • CVSS Severity … Mappings • CPE
Description	
Insufficient checks in the finite state machine o … 6.2.2 allow a partial reset of cryptographic secr … breaks the security of U2F for new server regis… vulnerability can be exploited by unauthenticat…	

TPM-FAIL: TPM meets Timing and Lattice Attacks

Daniel Moghimi and Berk Sunar, *Worcester Polytechnic Institute, Worcester, MA, USA;* Thomas Eisenbarth, *University of Lübeck, Lübeck, Germany;* Nadia Heninger, *University of California, San Diego, CA, USA*

https://www.usenix.org/conference/usenixsecurity20/presentation/moghimi-tpm

This paper is included in the Proceedings of the 29th USENIX Security Symposium.

August 12–14, 2020

978-1-939133-17-5

Nitrokey / nitrokey-pro-firmware Public — Watch 16

Code | Issues 37 | Pull requests 6 | Actions | Projects

Fix off by one error in OTP slot range check

Merged — szszszsz merged 1 commit into Nitrokey:master from FlorianUekermann

Security advisory YSA-2018-01 – Security issue with password pro… applet on YubiKey NEO

Published date: **2018-01-16**

Tracking IDs: **YSA-2018-01**

Summary

Oscar Mira and Roi Martin from the Schibsted security team informed us of a se… Open Authentication) applet on the YubiKey NEO. The YubiKey OATH applet is … password (TOTP) and HMAC-based one-time password (HOTP) codes that are t… Authenticator app. To provide an extra layer of protection against unauthorize… applet can be protected with an optional password; a feature unique to the Yu… password (OTP) code generators. The issue may allow an individual in physica… remove the password protection of the OATH applet and view the TOTP/HOTP … companion Yubico Authenticator app, without knowing the password.

CVE-ID	
CVE-2019-18671	Learn more at National Vulnerability Database (NVD) • CVSS Severity Rating • Fix Information • Vulnerable Software Versions • SCAP Mappings • CPE Information
Description	
Insufficient checks in the USB packet handling of the ShapeShift KeepKey hardware wallet before firmware 6.2.2 allow out-of-bounds writes in the .bss segment via crafted messages. The vulnerability could allow code execution or other forms of impact. It can be triggered by unauthenticated attackers and the interface is reachable via WebUSB.	

CVE-ID	
CVE-2021-31…	
Description	
Insufficient length checks in the ShapeShift KeepKey hardware wallet firm… buffer overflow via crafted messages. The overflow in ethereum_extractTh… can circumvent stack protections and lead to code execution. The vulnerab… over WebUSB.	

CVE-ID	
CVE-2018-6875	Learn more at National Vulnerability Database (NVD) • CVSS Severity Rating • Fix Information • Vulnerable Software Versions • SCAP Mappings • CPE Information
Description	
Format String vulnerability in KeepKey version 4.0.0 allows attackers to trigger information display (of information that should not be accessible), related to text containing characters that the device's font lacks.	

SecurityAdvisory 2015-04-14

Tracking IDs: YSA-2015-1 and CVE-2015-3298.

Summary

Minerva: The curse of ECDSA nonces

Systematic analysis of lattice attacks on noisy leakage of bit-length of ECDSA nonces

Ján Jančár[1], Vladimír Sedláček[1,2], Petr Švenda[1] and Marek Sýs[1]

[1] Masaryk University,
[2] Ca' Foscari University of Venice
{j08ny,vlada.sedlacek}@mail.muni.cz;{svenda,syso}@fi.muni.cz

Abstract. We present our discovery[1] of a group of side-channel vulnerabilities in implementations of the ECDSA signature algorithm in a widely used Atmel AT90SC FIPS 140-2 certified smartcard chip and five cryptographic libraries (libgcrypt, wolfSSL, MatrixSSL, SunEC/OpenJDK/Oracle JDK, Crypto++). Vulnerable implementations leak the bit-length of the scalar used in scalar multiplication via timing. Using leaked bit-length, we mount a lattice attack on a 256-bit curve, after observing enough signing operations. We propose two new methods to recover the full private key requiring just 500 signatures for simulated leakage data, 1200 for real cryptographic

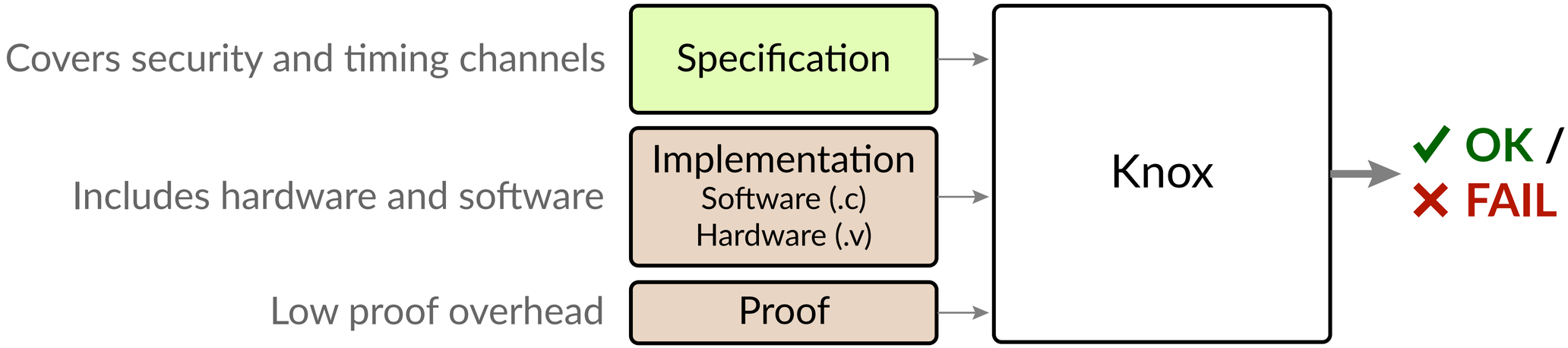
Goal: HSMs without security vulnerabilities

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

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

Example: PIN-protected backup HSM

```
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 = bad_guesses + 1
    return 'Incorrect PIN'
  bad_guesses = 0
  return secret
```

Functional specification

Describes input-output behavior

No notion of timing

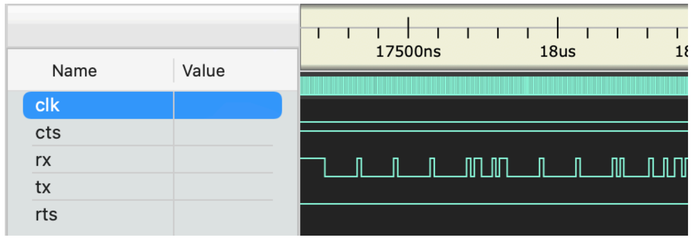
Implementation

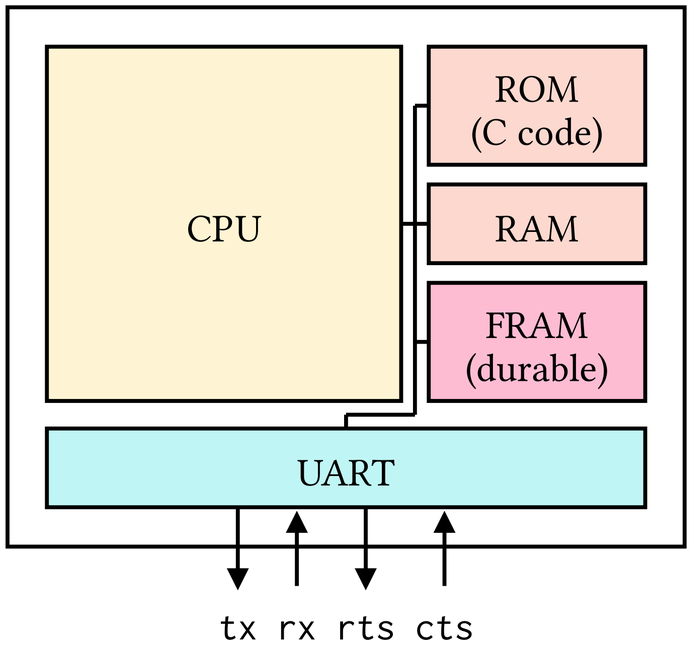
Implementation includes hardware/software

- CPU
- Code that runs on it
- Peripherals
- Persistent memory
- ...

Interface: wires

- Read output wires
- Write input wires
- Wait for a cycle





How to relate implementation to spec?

Want to capture:

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

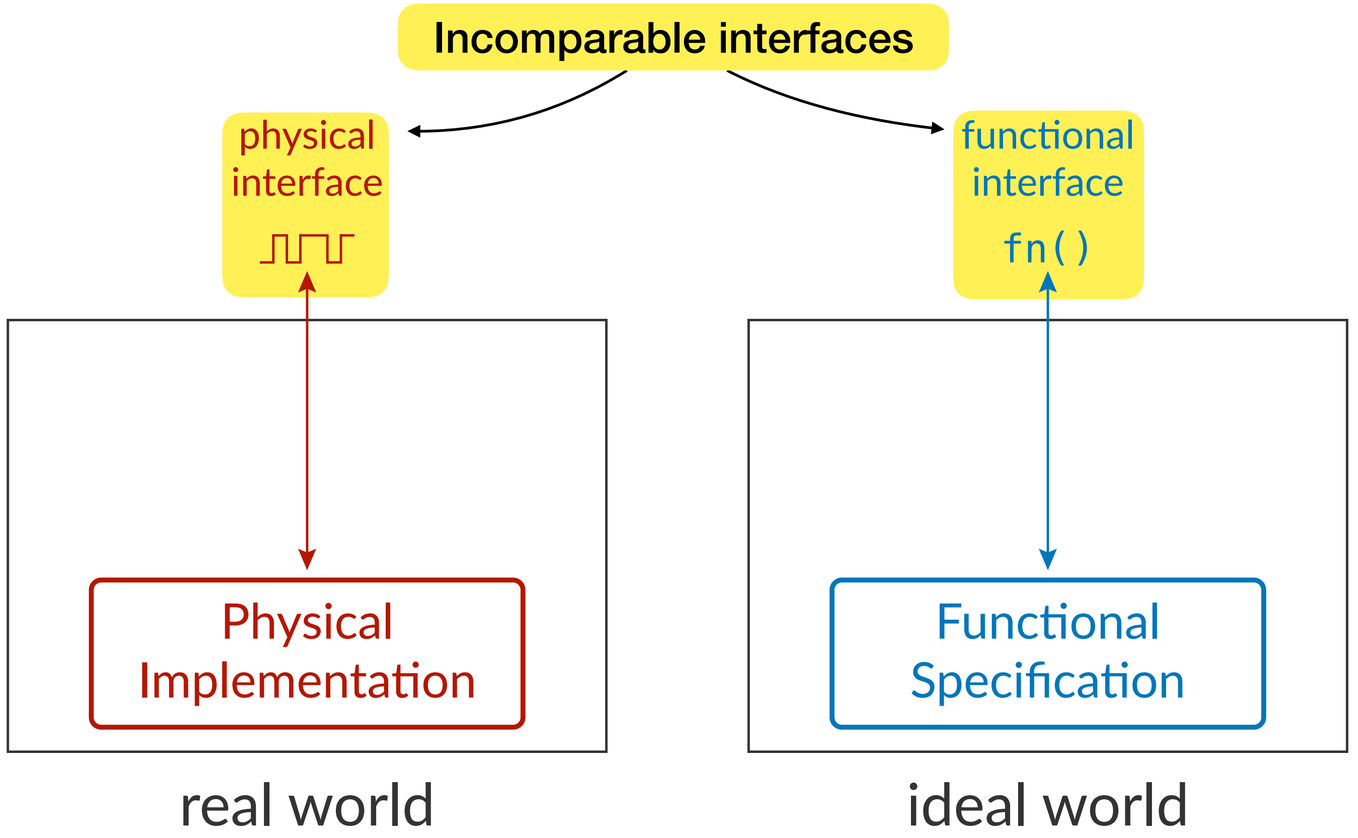
Implementation is at the level of wires

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

Information-preserving refinement (IPR)

Defined as indistinguishability between a real and an ideal world

Inspired by formalization of zero knowledge in cryptography

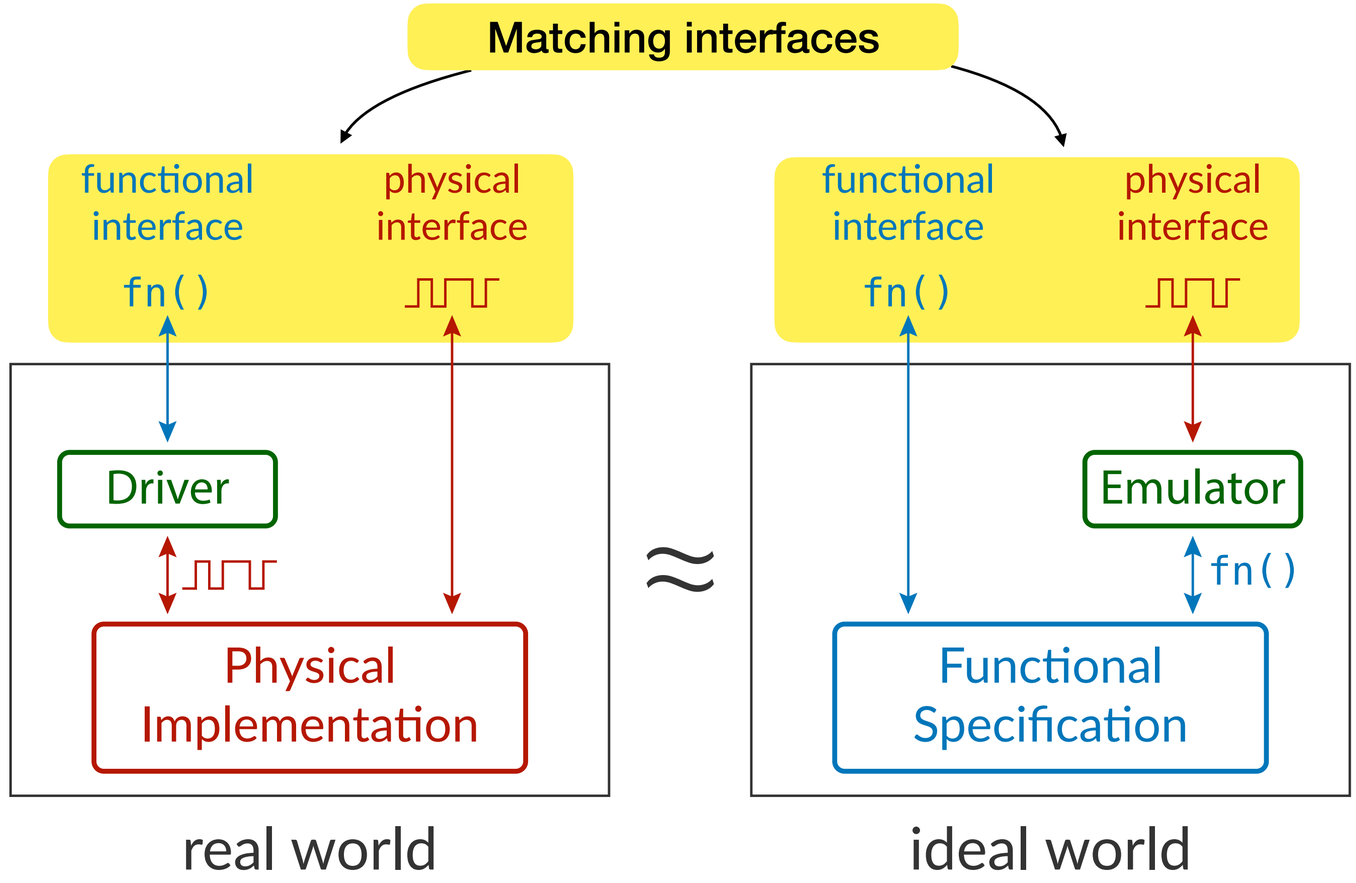


Information-preserving refinement (IPR)

Defined as indistinguishability between a real and an ideal world

Inspired by formalization of zero knowledge in cryptography

Interface adapters in each direction



IPR: driver

Driver: translates spec-level operations to wire-level I/O

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
```

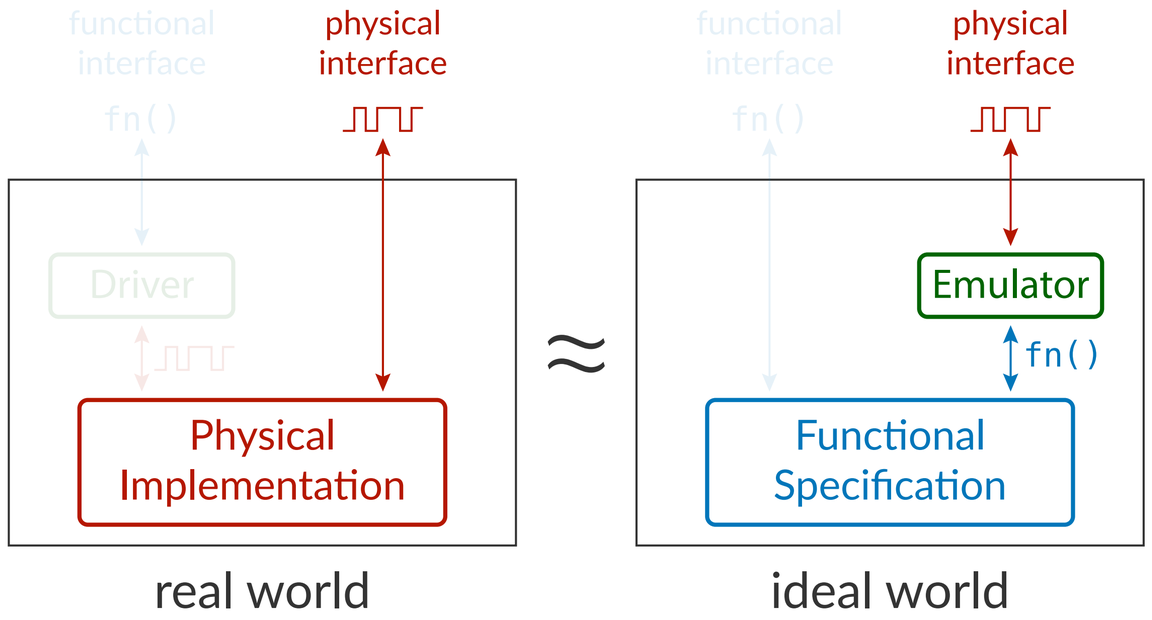
IPR: emulator

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)

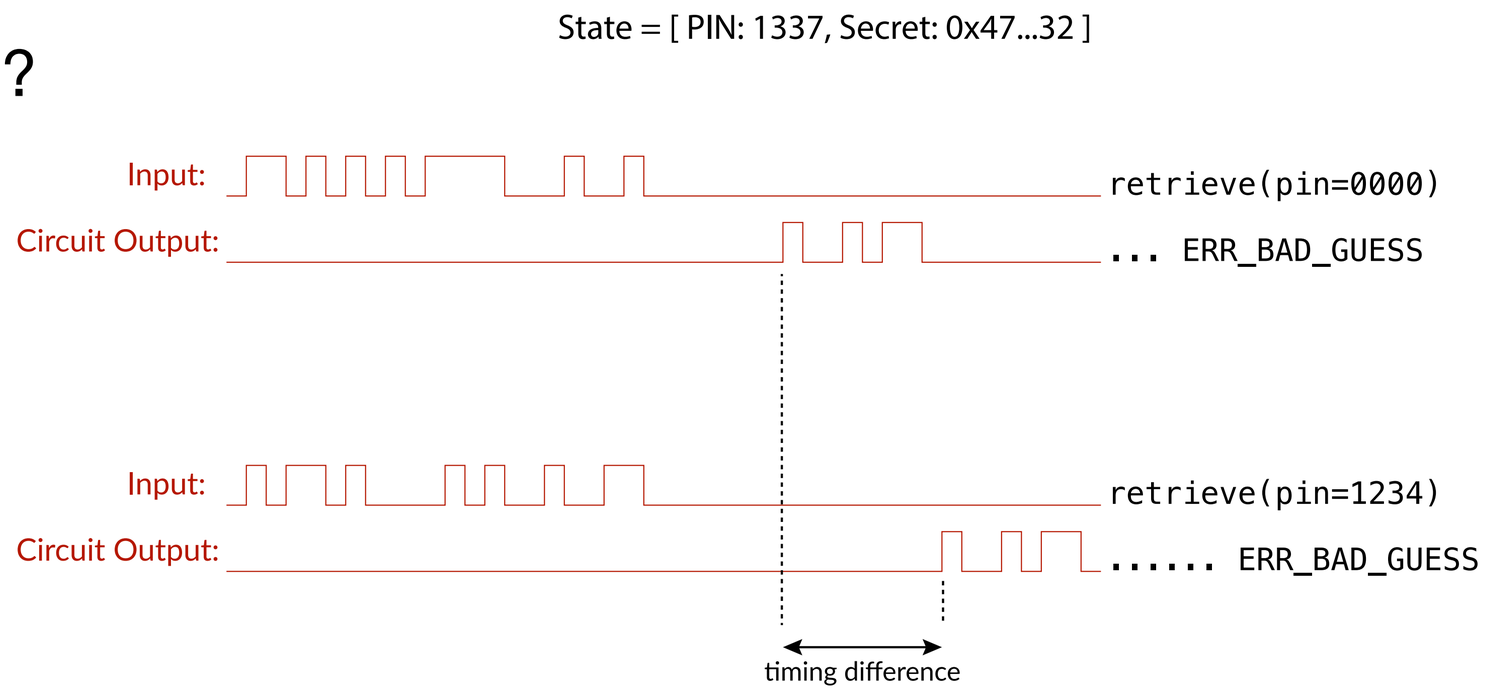
Captures non-leakage



IPR rules out timing channels

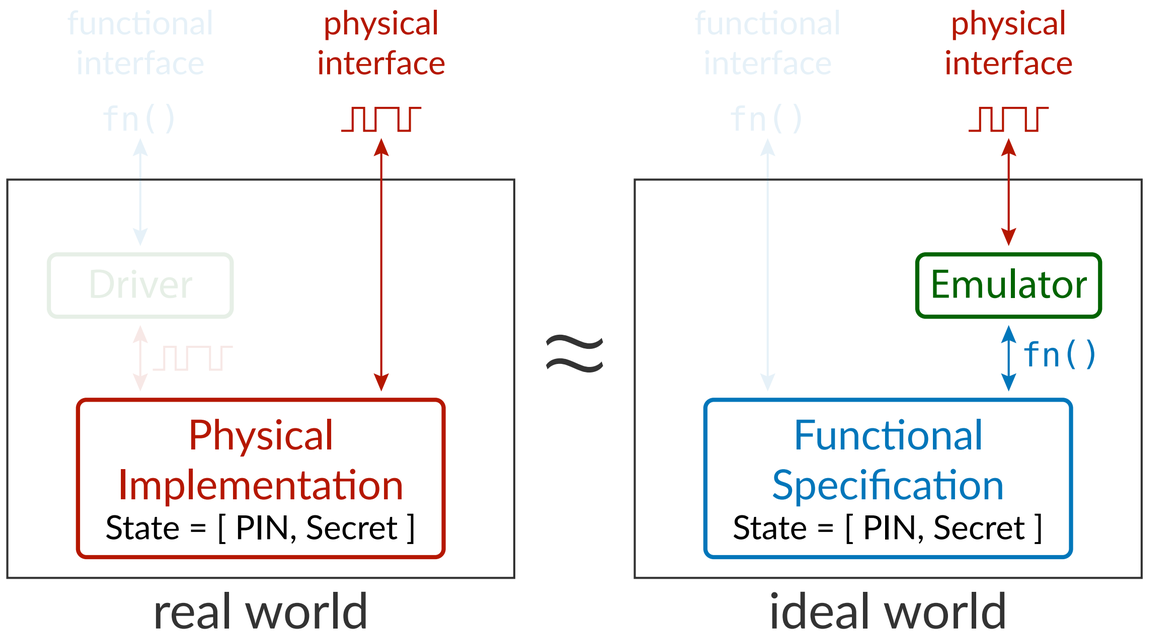
What if circuit leaked info through timing, e.g., `strcmp()`?

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 to impl

Only reveals secret when correct PIN supplied

Enforces guess limits

Forgets old secret/pin when `store()` is called

Doesn't leak past PIN guesses

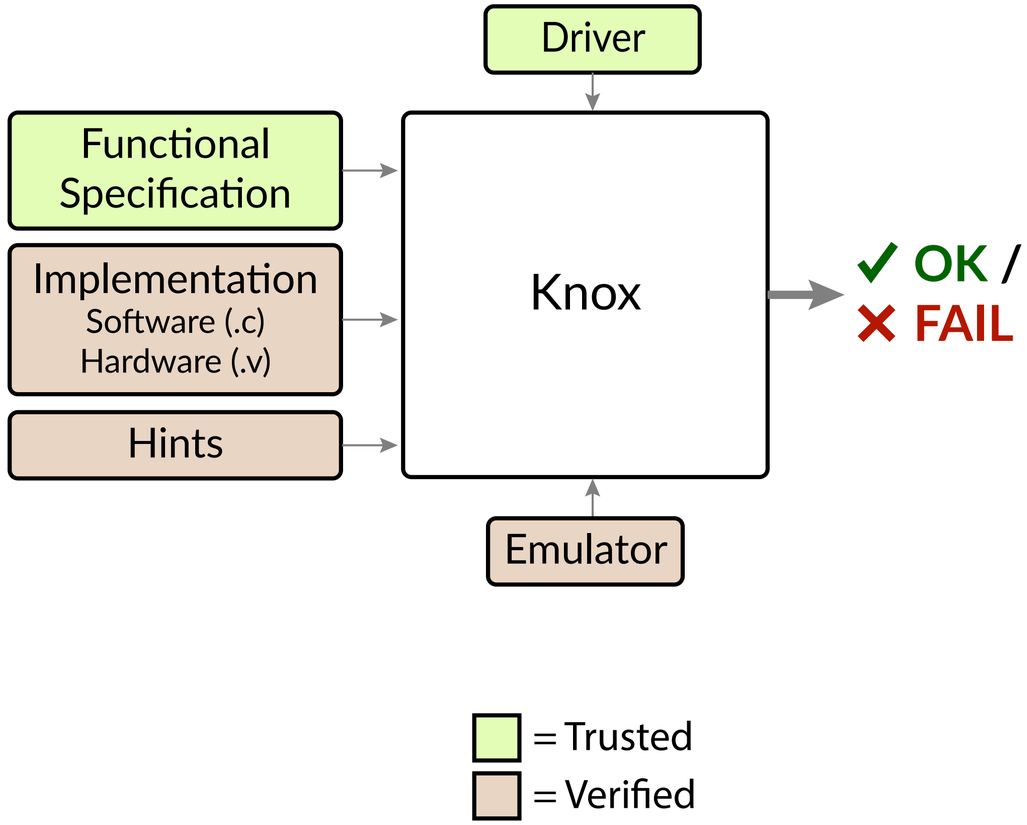
```
var bad_guesses = 0, secret = 0, pin = 0

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 = bad_guesses + 1
    return 'Incorrect PIN'
  bad_guesses = 0
  return secret
```

Knox framework

~ 3000 LOC on top of Rosette [PLDI'14]

Symbolically execute entire circuit + code

Relies on human guidance through *hints*



Evaluation: case studies

3 simple HSMs, run on an FPGA

Hardware: minimal RISC-V CPU, cryptographic accelerator, UART, ...

Software: control logic, peripheral drivers, HOTP, HMAC, ...

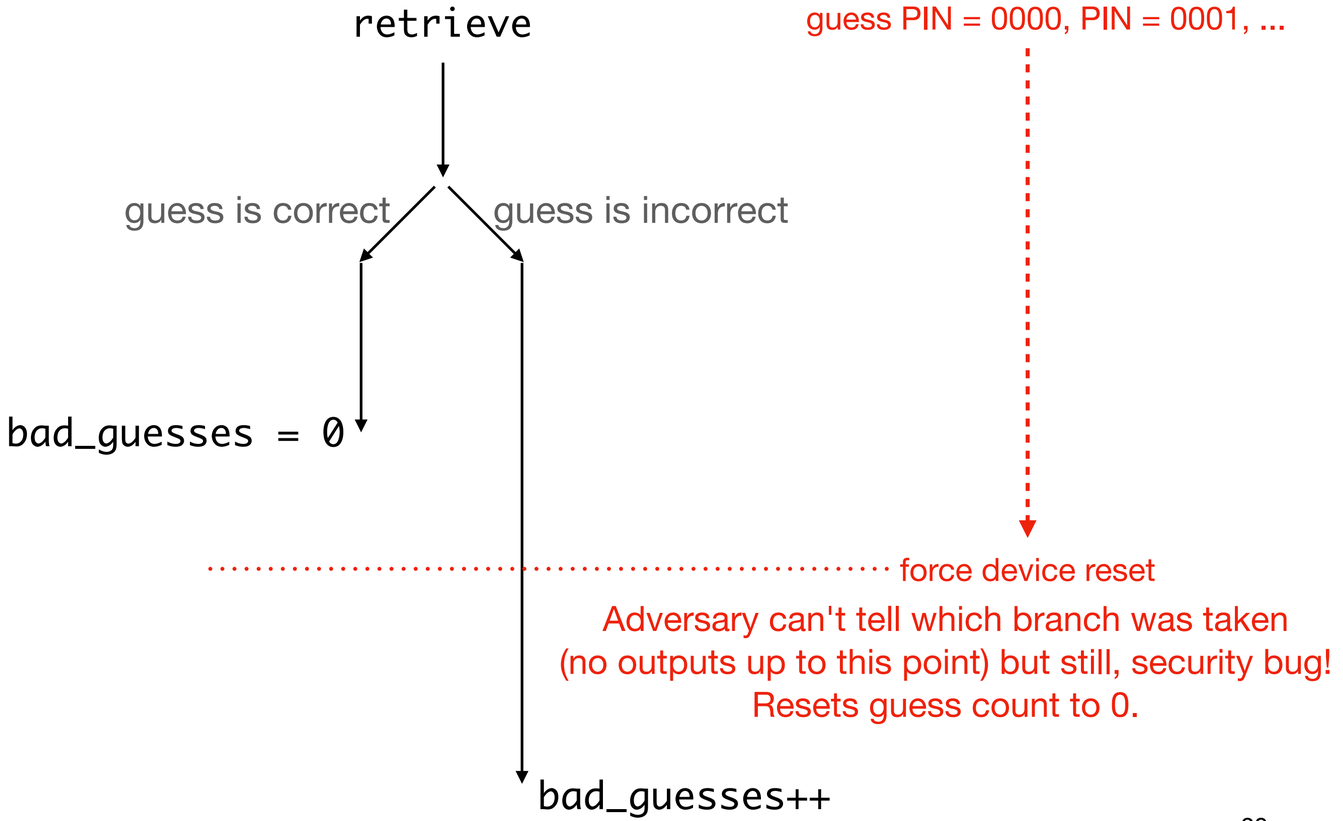
Succinct specifications

Low proof overhead

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

Lines of code for case studies

Subtle bug involving persistence and timing




```c
void retrieve(uint8_t *guess) {
  // return error if PIN guess limit exceeded
  // ...

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

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

Real implementations have similar code

SoloKey: pattern similar to our bug

Other HSMs like OpenSK have more robust code to avoid this issue

```
1568        int8_t ret = verify_pin_auth_ex(CM->pinAuth, (u
1569
1570        if (ret == CTAP2_ERR_PIN_AUTH_INVALID)
1571        {
1572            ctap_decrement_pin_attempts();
1573            if (ctap_device_boot_locked())
1574            {
1575                return CTAP2_ERR_PIN_AUTH_BLOCKED;
1576            }
1577            return CTAP2_ERR_PIN_AUTH_INVALID;
1578        }
1579        else
1580        {
1581            ctap_reset_pin_attempts();
1582        }
```

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