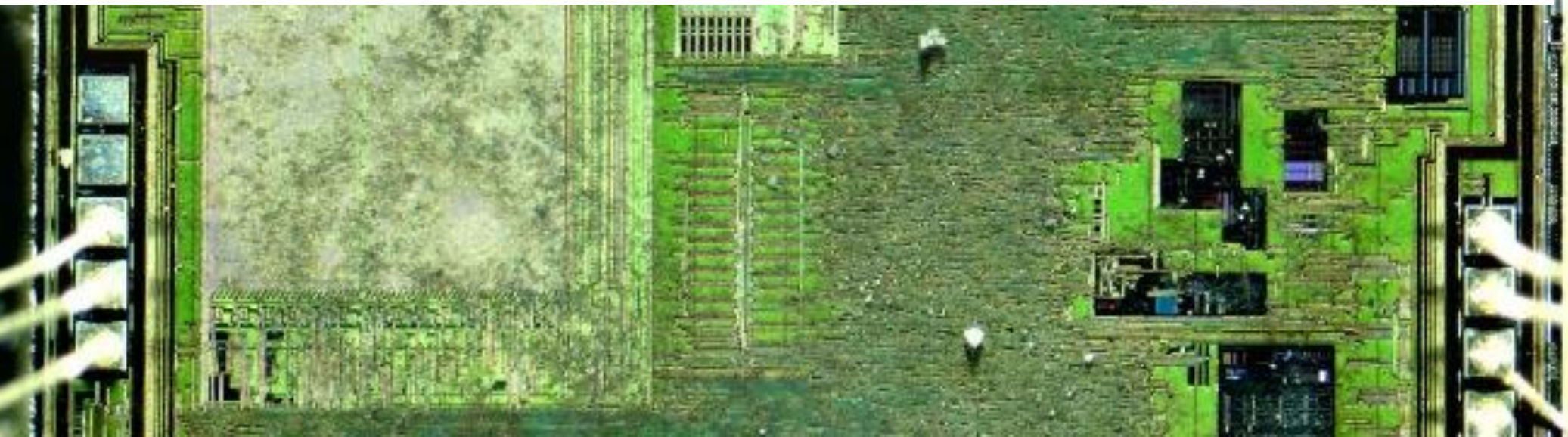
User Controlled Hardware Security Anchors: Evaluation and Designs

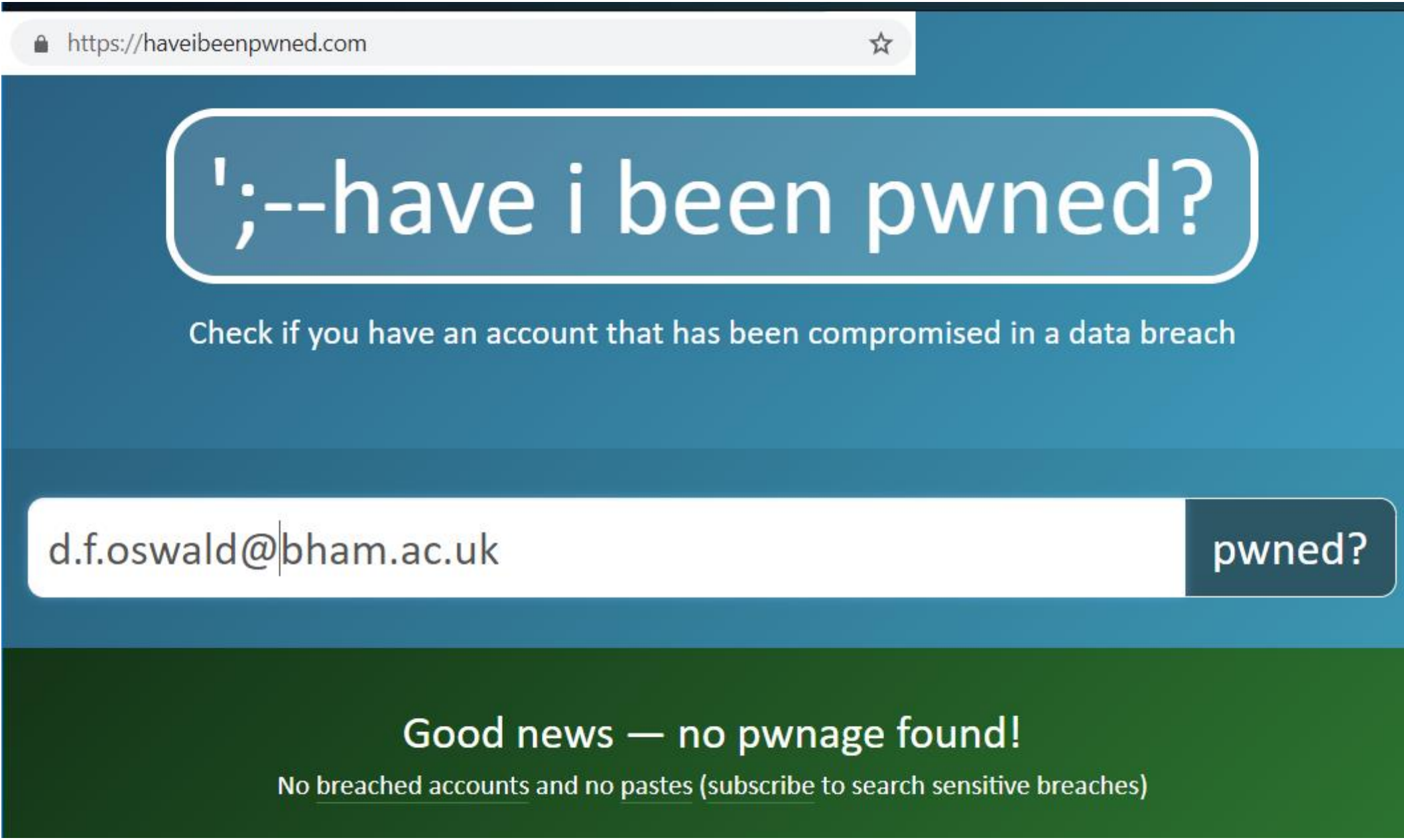
Dr David Oswald, Prof Mark Ryan, Prof Flavio Garcia

The University of Birmingham

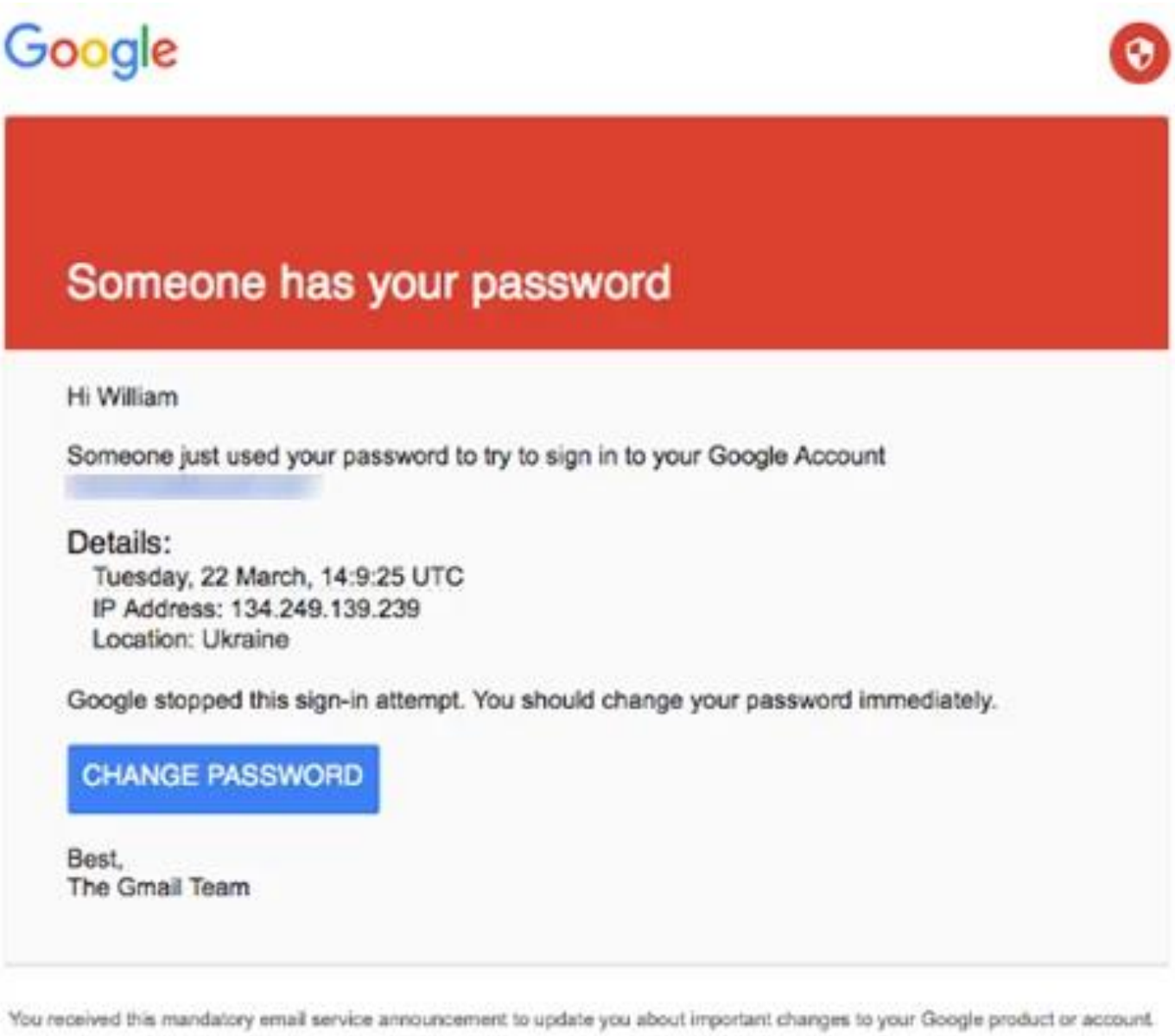
Industry partners: HP Labs, Yubico



Why Hardware Security Anchors?



Why Hardware Security Anchors?



Google

Someone has your password

Hi William

Someone just used your password to try to sign in to your Google Account

Details:
Tuesday, 22 March, 14:9:25 UTC
IP Address: 134.249.139.239
Location: Ukraine

Google stopped this sign-in attempt. You should change your password immediately.

CHANGE PASSWORD

Best,
The Gmail Team

You received this mandatory email service announcement to update you about important changes to your Google product or account.

User Controlled Hardware Security Anchors: Evaluation and Designs (1)

- **WP1:** Evaluate the security of available security anchors and Trusted Execution Environments (more later)
- **WP2:** Establishing secure channels between TEE and the user through …
  - Auxiliary devices
  - Platform features for secure I/O

User Controlled Hardware Security Anchors: Evaluation and Designs (2)

- **WP3:** Enhancing user authentication
  - Basis: FIDO(2) and U2F
  - Addressing enrollment and revocation
  - Authentication policies (e.g. location, …)
  - Formal modelling and verification
- **WP4:** Demonstrators
  - TEE implementation
  - Smartphone app
  - Authentication token

Evaluating the state of TEE security

An overview

Trusted Execution Environments in a nutshell

- Main technologies at present:
  - Trusted Platform Module (separate chip or firmware)
  - Intel Software Guard eXtensions (microcode w/ HW)
  - AMD Platform Security Processor (separate core)
  - ARM TrustZone (software w/ HW support)
  - Apple Secure Enclave Processor (separate core, same die)
- All provide some form of running code or crypto operations in isolation
- Most require cooperation with the silicon/device manufacturer (to different extent)

Relevant attack vectors

- "Classical" vulnerabilities, e.g. buffer overflows
- Microarchitecture (e.g. cache timing, Spectre and Meltdown, etc)
- Software-driven fault attacks (RowHammer, CLKSCREW[1], …)
- Hardware-level attacks (JTAG, faults, EM and power side channels)

[1] https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/tang

Intel SGX

- Highest flexibility for the user, can run arbitrary code in “enclaves” – interesting for SW TPM
- Currently “dead” from a security perspective
  - Cache-timing side channels (https://arxiv.org/pdf/1703.06986.pdf, https://arxiv.org/abs/1702.07521, https://arxiv.org/pdf/1702.08719.pdf)
  - MemJam (https://arxiv.org/abs/1711.08002)
  - Spectre and Meltdown variants
  - More?

SGX vs Spectre / Meltdown

GitHub, Inc. [US] | https://github.com/lsds/spectre-attack-sgx

Spectre attack against SGX enclave

sgx spectre attack enclave speculative-execution

1 commit | 1 branch | 0 releases | 1 contributor | Apache-2.0

Branch: master | New pull request | Find file | Clone or download

danokeeffe Initial commit of proof of concept SGX Spectre attack. Latest commit d759e91 2 days ago

SGXSpectre	Initial commit of proof of concept SGX Spectre attack.	2 days ago
.gitignore	Initial commit of proof of concept SGX Spectre attack.	2 days ago
LICENSE	Initial commit of proof of concept SGX Spectre attack.	2 days ago
README.md	Initial commit of proof of concept SGX Spectre attack.	2 days ago

README.md

spectre-attack-sgx

Sample code demonstrating a Spectre-like attack against an Intel SGX enclave.

Overview

Given our ongoing research on Intel SGX here in the LSDS group at Imperial College London, a question that occurred to us immediately on first hearing of the recent Meltdown and Spectre attacks is *what are the security implications of speculative*

SGX

GitHub,
Spectre at
sgx spe
Branch: ma
danok
SGXSpe
.gitignc
LICENS
READM
READM
sp
Sam
Ov
Give

FORESHADOW: Extracting the Keys to the Intel SGX Kingdom with Transient Out-of-Order Execution

Jo Van Bulck[1], Marina Minkin[2], Ofir Weisse[3], Daniel Genkin[3], Baris Kasikci[3], Frank Piessens[1], Mark Silberstein[2], Thomas F. Wenisch[3], Yuval Yarom[4], and Raoul Strackx[1]

[1]*imec-DistriNet, KU Leuven,* [2]*Technion,* [3]*University of Michigan,* [4]*University of Adelaide and Data61*

Abstract

Trusted execution environments, and particularly the Software Guard eXtensions (SGX) included in recent Intel x86 processors, gained significant traction in recent years. A long track of research papers, and increasingly also real-world industry applications, take advantage of the strong hardware-enforced confidentiality and integrity guarantees provided by Intel SGX. Ultimately, enclaved execution holds the compelling potential of securely offloading sensitive computations to untrusted remote platforms.

We present Foreshadow, a practical software-only microarchitectural attack that decisively dismantles the security objectives of current SGX implementations. Crucially, unlike previous SGX attacks, we do not make any assumptions on the victim enclave's code and do not necessarily require kernel-level access. At its core, Foreshadow abuses a speculative execution bug in modern Intel processors, on top of which we develop a novel exploitation methodology to reliably leak plaintext enclave secrets from the CPU cache. We demonstrate our attacks by extracting full cryptographic keys from Intel's vetted architectural enclaves, and validate their correctness by launching rogue production enclaves and forging arbitrary local and remote attestation responses. The extracted remote attestation keys affect millions of devices.

distrusting enclaves with a minimal Trusted Computing Base (TCB) that includes only the processor package and microcode. Enclave-private CPU and memory state is exclusively accessible to the code running inside it, and remains explicitly out of reach of all other enclaves and software running at any privilege level, including a potentially malicious operating system and/or hypervisor. Besides strong memory isolation, TEEs typically offer an *attestation* primitive that allows local or remote stakeholders to cryptographically verify at runtime that a specific enclave has been loaded on a genuine (and hence presumed to be secure) TEE processor.

With the announcement of Intel's Software Guard eXtensions (SGX) [2, 27, 43] in 2013, hardware-enforced TEE isolation and attestation guarantees are now available on off-the-shelf x86 processors. In light of the strong security guarantees promised by Intel SGX, industry actors are increasingly adopting this technology in a wide variety of applications featuring secure execution on adversary-controlled machines. Open Whisper Systems [50] relies on SGX for privacy-friendly contact discovery in its Signal network. Both Microsoft and IBM recently announced support for SGX in their cloud infrastructure. Various off-the-shelf Blu-ray players and initially also the 4K Netflix client furthermore use SGX to enforce Digital Rights Management (DRM) for high-resolution video

AMD Platform Security Processor

- Separate ARM core running PSP in Trustzone
- Firmware e.g. here
  https://github.com/coreboot/blobs/tree/master/southbridge/amd/avalon/PSP
- Buffer overflow in firmware TPM (fTPM) discovered on Jan 3, 2018 (bad timing…), leading to code execution via crafted certificate

FULL DISCLOSURE Full Disclosure mailing list archives

By Date By Thread Google Custom Search Search

AMD-PSP: fTPM Remote Code Execution via crafted EK certificate

From: Cfir Cohen via Fulldisclosure <fulldisclosure () seclists org>
Date: Wed, 3 Jan 2018 09:40:40 -0800

Trusted Platform Module

- Separate chip, limited functionality
- Chen & Ryan showed issues w/ authData
- Tarnovsky demonstrated microprobing of SLE 66
- Nemec et al: ROCA vulnerability in key generation of secure elements / TPMs
- Boone: MITM to exploit PC-side bugs[1]
- Side-channel attacks?

[1] https://github.com/nccgroup/TPMGenie

ARM TrustZone

```
                DCB 0, 0, 0
aStartOfRawMeta DCB "Start of Raw Metallica OTP Collected Data",0xA,0
                                        ; DATA XREF: sub_30CE6+C↑o
                DCB 0
aBoot0Data      DCB "Boot 0 Data",0     ; DATA XREF: sub_30CE6+20↑o
aBoot1Data      DCB "Boot 1 Data",0     ; DATA XREF: sub_30CE6+32↑o
aSectorData     DCB "Sector Data",0     ; DATA XREF: sub_30CE6+48↑o
aEndOfRawMetall DCB "End of Raw Metallica OTP Collected Data",0xA,0
                                        ; DATA XREF: sub_30CE6+58↑o
```

ARM TrustZone

- HW-supported TEE in bigger ARM chips
- The OS running in TZ is up to the OEM, examples include:
  - Trustonic Kinibi (aka t-base, proprietary)
  - Qualcomm QSEE (proprietary)
  - Trusty (open, https://source.android.com/security/trusty/)

Previous attacks on Samsung TZ

- Long history of SW attacks on TZ, https://googleprojectzero.blogspot.co.uk/2017/07/trust-issues-exploiting-trustzone-tees.html
- Up to Galaxy S7, attacker can roll back to old (vulnerable) versions of trustlets
- Beniamini discovered buffer overflow in OTP trustlet, allowing code execution in the context of this trustlet
- Lapid & Wool showed that KeyMaster Key Encryption Key can be extracted via OTP vuln or cache-timing side channel

Example: Samsung Galaxy S6

- Galaxy S6 runs Trustonic TEE OS
- Trustlets are `.tlbin` files in `/data/app/mcRegistry`:
  - Biometry / fingerprint matching
    `2150-ffffffff000000000000000000000000e.tlbin`
  - KeyMaster
    `2178-ffffffff000000000000000000000003e.tlbin`
  - Samsung Pay
    `2172-ffffffff000000000000000000000028.tlbin`
- Can be loaded into IDA (custom loader) and "easily" be analysed

Example: fingerprint matching trustlet

IDA - 2150-ffffffff000000000000000000000000e.i64 (2150-ffffffff000000000000000000000000e.tlbin) Z:\s6_trustzone\s6-trustlets\2150-ffffffff000000000000000000...

File Edit Jump Search View Debugger Options Windows Help

No debugger

Library function | Regular function | Instruction | Data | Unexplored | External symbol

Functions window

Function name

- sub_4998
- sub_49C6
- sub_49E4
- sub_4A38
- internal_hmac
- internal_cmp_hmac
- internal_encrypt
- decrypt_wrapper
- GetKey
- CreateAuthToken
- IntegrityCheckAuthToken
- generate_template_id
- encode_metadata
- decode_metadata
- tl_do_identify_stub
- decode_each_templ
- decode_all_templates
- sub_5ABC
- sub_5ECA

Line 39 of 1181

IDA View-A | Enums | Imports | Exports

```
t:00004AE0                 MOV             R1, R6
t:00004AE2                 STR             R0, [SP,#0x38+var_38]
t:00004AE4                 MOV             R0, R5
t:00004AE6                 BL              internal_hmac
t:00004AEA                 MOVS            R4, R0
t:00004AEC                 BEQ             loc_4AF8
t:00004AEE                 MOV             R1, R0
t:00004AF0                 ADR             R0, aInternalCmpHma ; "internal_cmp_hmac internal_hmac
t:00004AF2                 BL              debug_printf ; format string in R0
t:00004AF2                                         ; Args in R1,  R2, ...
t:00004AF6                 B               loc_4B0C
t:00004AF8 ; ---------------------------------------------------------------------------
t:00004AF8
t:00004AF8 loc_4AF8                                ; CODE XREF: internal_cmp_hmac+24↑j
t:00004AF8                 ADDS            R1, R5, R6
t:00004AFA                 MOVS            R2, #0x20
t:00004AFC                 ADD             R0, SP, #0x38+var_34
t:00004AFE                 BL              memcmp_probably ; return 0 if equal
t:00004B02                 CBZ             R0, loc_4B0C
t:00004B04                 ADR             R0, aHmacCmpFailed ; "hmac cmp failed\n"
```

00003AFE 0000000000004AFE: internal_cmp_hmac+36

Output window

```
Loading type libraries...
Autoanalysis subsystem has been initialized.
Database for file '2150-ffffffff000000000000000000000000e.tlbin' has been loaded.
-------------------------------------------------------------------------------
```

Example: fingerprint trustlet

- Reverse-engineered data flow for encryption
- Note: TrustZone has no separate storage
- Some open questions remaining …

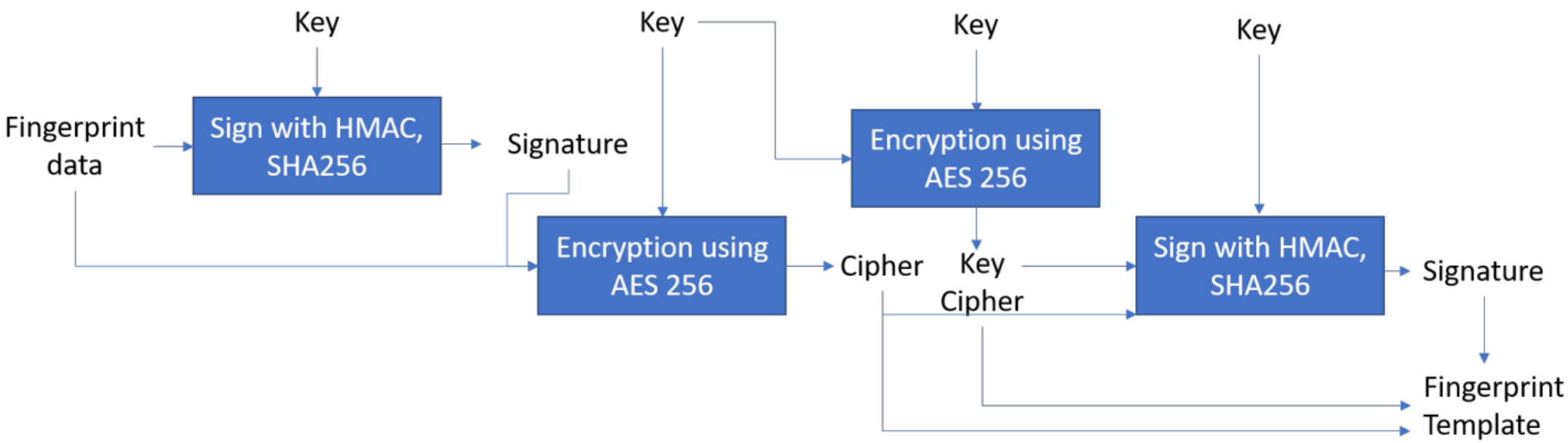


FIGURE 3.9: Diagram shows fingerprint encryption flow.

Future attack vectors

- Vulnerabilities in interesting trustlets (e.g. biometrics, payment)
- Automatic detection (e.g. missing bound checks)?
- Hardware (and software) side channels and fault attack vulnerabilities (obtain Root Encryption Key)

Apple SEP

```
__const:0004BB60 ; Segment type: Pure data
__const:0004BB60                 AREA __const, DATA, ALIGN=4
__const:0004BB60                 ; ORG 0x4BB60
__const:0004BB60 aDerivedKey     DCB "derived_key",0          ; DATA XREF: sub_
__const:0004BB6C aSepDerivedKey  DCB "SEP derived_key",0
__const:0004BB7C aSeWhat         DCB "SE what?",0             ; DATA XREF: sub_
__const:0004BB85                 ALIGN 4
__const:0004BB88                 DCD sub_14716+1
```

Apple SEP

- Separate ARMv7A core in iOS devices and newer Macs (cf. touchbar)
- Security anchor for
  - Biometrics
  - Storage encryption
  - Device unlocking
  - Apple Pay (together with separate Javacard chip)
  - Selected crypto operations for apps

Using SEP in apps

GitHub, Inc. [US] | https://github.com/trailofbits/SecureEnclaveCrypto

Branch: master | New pull request | Find file | Clone or download

withzombies Merge pull request #12 from trailofbits/alex ··· Latest commit 7e22bdd on Mar 14, 2017

SecureEnclaveObjective-C	better structure in Swift project	10 months ago
SecureEnclaveSwift	better structure in Swift project	10 months ago
.gitignore	Initial commit	2 years ago
LICENSE	Initial commit	2 years ago
README.md	Update README.md	a year ago
key_builder.rb	move key_builder.rb	a year ago

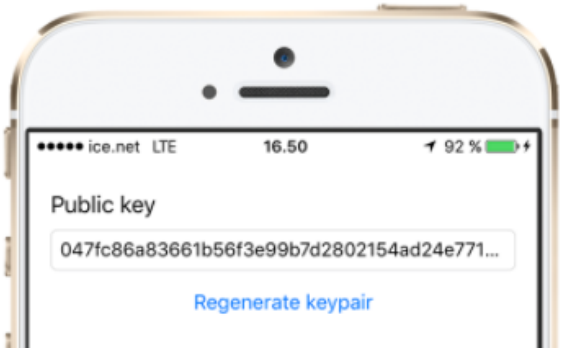
README.md

SecureEnclaveCrypto

Secure?

This project shows you how to

- create a keypair where as the private key is stored in the secure enclave
- sign a string / some data with the private key
- use the security functions like SecKeyRawVerify, SecKeyGeneratePair and SecItemCopyMatching in Swift 3 and Objective-C
- store the public key in the keychain



Understanding Apple SEP

- OS and firmware format documented at BH'16[1] in detail, but no attacks published
- Firmware encrypted, but decryption keys for iPhone 5S published in 2017
- Firmware image (IMG4) can be parsed and loaded into IDA using open tools

[1] https://www.blackhat.com/docs/us-16/materials/us-16-Mandt-Demystifying-The-Secure-Enclave-Processor.pdf

Decrypting SEP firmware (iPhone 5S)

sepdump00_boot
sepdump01_kerne
sepdump02_SEPC
sepdump03_SEPD
sepdump04_sepS
sepdump05_ARTM
sepdump06_sks
sepdump06_sks.i6
sepdump07_sbio
sepdump07_sbio.
sepdump08_scrd

IDA - sepdump07_sbio.i64 (sepdump07_sbio) Z:\ios_sep\sep_dec\sepdump07_sbio.i64

File Edit Jump Search View Debugger Options Windows Help

No debugger

Library function | Regular function | Instruction | Data | Unexplored | External symbol

Functions wi... | IDA ... | Strings ... | Hex ... | Str... | En... | Im... | Ex...

Function name
sub_F916
sub_F9B2
sub_FA08
sub_FADE
sub_FB44
sub_FC22
sub_FC44
sub_FD1A
sub_FD4C
sub_FE68
sub_FF2E
sub_FF5A
sub_FFEC
sub_1008A
sub_1014E
sub_1024A
sub_10250
sub_1032A
sub_10428
sub_10642
sub_106E4
sub_10AC8
sub_10ED8
sub_10F90
sub_11042

Address	Length	Type	String
__text:00054...	00000005	C	?;C^-
__text:0007B...	00000011	C	[[[[[[[[[[[[[[[[
__cstring:00...	0000003B	C	sbio: SecureBiometricEngine application starting (%s, %s)\n
__cstring:00...	00000005	C	Mesa
__cstring:00...	00000008	C	release
__cstring:00...	0000001D	C	Could not locate AKF driver.
__cstring:00...	0000001E	C	Could not locate TRNG driver.
__cstring:00...	00000020	C	Could not locate expert driver.
__cstring:00...	0000001E	C	Could not locate SKG service.
__cstring:00...	00000015	C	Could not locate sks
__cstring:00...	00000014	C	sbio: %s sks found\n
__cstring:00...	00000006	C	_main
__cstring:00...	00000025	C	Could not locate ARTManager service.
__cstring:00...	0000001B	C	sbio: %s ARTManager found\n
__cstring:00...	0000001F	C	max ctx size estimate is wrong
__cstring:00...	00000069	C	/BuildRoot/Library/Caches/com.apple.xbs/Sources/Mesa_Firmware/Mesa-424.18/A
__cstring:00...	00000018	C	workloop RPC error r=%d
__cstring:00...	00000025	C	Could not allocate object for stack.
__cstring:00...	00000015	C	Could not map stack.
__cstring:00...	0000002D	C	RNG is failing to produce requested entropy.
__cstring:00...	0000001E	C	_state == kSessionEstablished
__cstring:00...	0000000B	C	outDataOut
__cstring:00...	0000000E	C	outDataLength
__cstring:00...	00000006	C	patch
__cstring:00...	0000000F	C	encryptedPatch

Line 773 of 1363 | Line 1 of 3287

```
- open file "sepdump07_sbio"

  offset   num  description [bits.endian.size]
  ---------------------------------------------
  0007b1f0 874  SHA256 Hash constant words K (0x428a2f98) [32.le.256]
  000bc5cc 536  CRC-16-IBM maxim/usb [crc16.0xa001 lenorev 1.512]
  000bc5cc 529  CRC-16-IBM maxim/usb [crc16.0x8005 le rev int_min.512]
  000bc7cc 648  CRC-32-IEEE 802.3 [crc32.0xedb88320 lenorev 1.1024]
  000bc7cc 641  CRC-32-IEEE 802.3 [crc32.0x04c11db7 le rev int_min.1024]
  000bd20c 897  Rijndael Te0 (0xc66363a5U) [32.be.1024]
  000bd60c 906  Rijndael Td0 (0x51f4a750U) [32.be.1024]
  000bda0c 894  AES Rijndael S / ARIA S1 [..256]
  000bdb0c 895  AES Rijndael Si / ARIA X1 [..256]
  000bdc30 878  Hash constant words K for SHA-384 and SHA-512 [64.le.640]
  000bdeb0 1036 SHA1 / SHA0 / RIPEMD-160 initialization [32.le.20&]
  000bdeb0 2402 Lucifer (outerbridge) DFLTKY [..16]
  000bdebc 2053 RIPEMD-128 InitState [32.le.16&]
  000bdee4 1030 SHA256 [32.le.288&]
  000bdee4 876  SHA256 Initial hash value H (0x6a09e667UL) [32.le.32&]
  000bdee8 2364 Crypton kp [32.le.16]
```

Future possible vulnerabilities

- Understand implementations of relevant applets (fuzzing, static/dynamic analysis)
- Side-channel vulnerabilities with physical access (BH'16 authors recommend: *"Stick to the A7 (newer ones are more resistant)"*)
- Software side channels and faults

Conclusions

- Hardware security anchors and TEEs solve many important security problems (e.g. user auth) …
- … but are hard to get right (all TEEs covered in this talk have vulnerabilities)
- Potential issues include
  - Software vulnerabilities
  - Side channels and shared resources
  - Large flexibility/complexity = large attack surface

UNIVERSITY OF BIRMINGHAM

Thanks for your attention!

Questions?

d.f.oswald@bham.ac.uk