Practical Cryptographic Systems

Hardware Security & Tamper Resistance*

* Much of this lecture based on Anderson, Chap 14

Instructor: Matthew Green

Housekeeping: Midterm

News?

Conditional branch (Variant 1) attack

```
if (x < array1_size)
    y = array2[array1[x]*512];
```

Attacker calls victim code with `x`=N (where N > 8)

- Speculative exec while waiting for `array1_size`
  - Predict that if() is true
  - Read address (`array1` base + `x`) w/ out-of-bounds `x`
  - Read returns secret byte = **09** (fast – in cache)
  - Request memory at (`array2` base + **09***512)
  - Brings `array2[`**09**`*512]` into the cache
  - Realize if() is false: discard speculative work
- Finish operation & return to caller

Attacker times reads from `array2[i*512]`

- Read for `i`=**09** is fast (cached), revealing secret byte

Memory & Cache Status

`array1_size = 00000008`

Memory at `array1` base address:
8 bytes of data (value doesn't matter)
[... lots of memory up to `array1` base+N...]
09 F1 98 CC 90... (something secret)

```
array2[ 0*512]
array2[ 1*512]
array2[ 2*512]
array2[ 3*512]
array2[ 4*512]
array2[ 5*512]
array2[ 6*512]
array2[ 7*512]
array2[ 8*512]
array2[ 9*512]
array2[10*512]
array2[11*512]
  ...
```

Contents don't matter
only care about cache ***status***

Uncached Cached

Many slides and examples borrowed and inspired by Kocher: Spectre attack slides
https://www.ieee-security.org/TC/SP2019/SP19-Slides-pdfs/Paul_Kocher_01_-_Spectre_Attacks-IEEE-SecurityPrivacy_v05.pdf

Why does this matter?

- We can learn information about data that we should not be able to access
- For example:
  - Kernel secrets
  - Secrets in the same process (keys)
  - Other applications
  - It <u>should</u> run user space code
    But attacker can cause it to leak information about data in the kernel

Intel SGX

- "Virtual trusted hardware" inside of an Intel CPU
- Basic ideas:
  - To run an SGX program ("enclave") turn in an isolation mode that is even stronger than Ring0
  - OS cannot read enclave memory, enclave cannot read OS/app memory
  - Enforced by microcode

What software can we run?

- Enclave software has no access to I/O (network, keyboard, mouse). It has to interface with a real application via a strict API
- Every SGX (“enclave program”) is hashed, then digitally signed under a certificate chain held by Intel

How does the enclave store data persistently?

- Enclave can’t access disks directly
- It has to send data out to an application via the API
- But if that program gets compromised, how does the enclave verify it’s safe?

How does the enclave store data persistently?

- Answer: lots of crypto
- Ever enclave gets its own secret encryption keys (generated by SGX base OS)
- Keys are generated as a combination of:
  Enclave software hash +
  System unique identifier +
  A system root secret key known to SGX

How does the enclave store data persistently?

- Each enclave can “seal” (encrypt/MAC) data under this secret key, send it out to application
- Application can store the encrypted/authenticated data for the enclave (e.g., on disk), hand back to enclave next time it boots

How does the enclave store data persistently?

- Each enclave can “seal” (encrypt/MAC) data under this secret key, send it out to application
- Application can store the encrypted/authenticated data for the enclave (e.g., on disk), hand back to enclave next time it boots
- What about replay attacks?

Intel SGX attestation

- This all works great when I'm running software on my own computer
- What happens if I want to use Intel SGX in the cloud?
- How do I know my software is really being operated by SGX and not some malicious software pretending to be SGX?

Solution: attestation

- Every Intel processor has a secret signing key baked in at the factory*
- An enclave can ask the SGX OS to sign any block of data ("attestation")
- Signature includes:
  - Enclave hash (enforced by SGX OS)
  - Arbitrary data blob specified by application
  - Any other measurements, e.g., RAM hash
  - Certificate signed by Intel

So TL;DR

- As long as every single Intel signing key stays secret, a signature produced by a random SGX-enabled processor can be trusted
- Converse: if anyone steals a single SGX signing key (e.g., by attacking hardware) they can fake all these signatures, pretend to be an SGX enclave when they're not
- How do we deal with that?

Foreshadow

- As long as every single Intel signing key stays secret, a signature produced by a random SGX-enabled processor can be trusted
- Converse: if anyone steals a single SGX signing key (e.g., by attacking hardware) they can fake all these signatures, pretend to be an SGX enclave when they're not
- How do we deal with that?

FORESHADOW

Breaking the Virtual Memory Abstraction with Transient Out-of-Order Execution

Read the paper | Cite | Watch a demo

Introduction

Foreshadow is a speculative execution attack on Intel processors which allows an attacker to steal sensitive information stored inside personal computers or third party clouds. Foreshadow has two versions, the original attack designed to extract data from SGX enclaves and a Next-Generation version which affects Virtual Machines (VMs), hypervisors (VMM), operating system (OS) kernel memory, and System Management Mode (SMM) memory.

Foreshadow AaaS @ForeshadowAaaS · Aug 15, 2018

Hi, I'm the Foreshadow AaaS bot! :-) I react to tweets I'm tagged in by providing a genuine Intel attestation that can be verified against Intel Attestation Service (IAS).

8 23 50

Show this thread

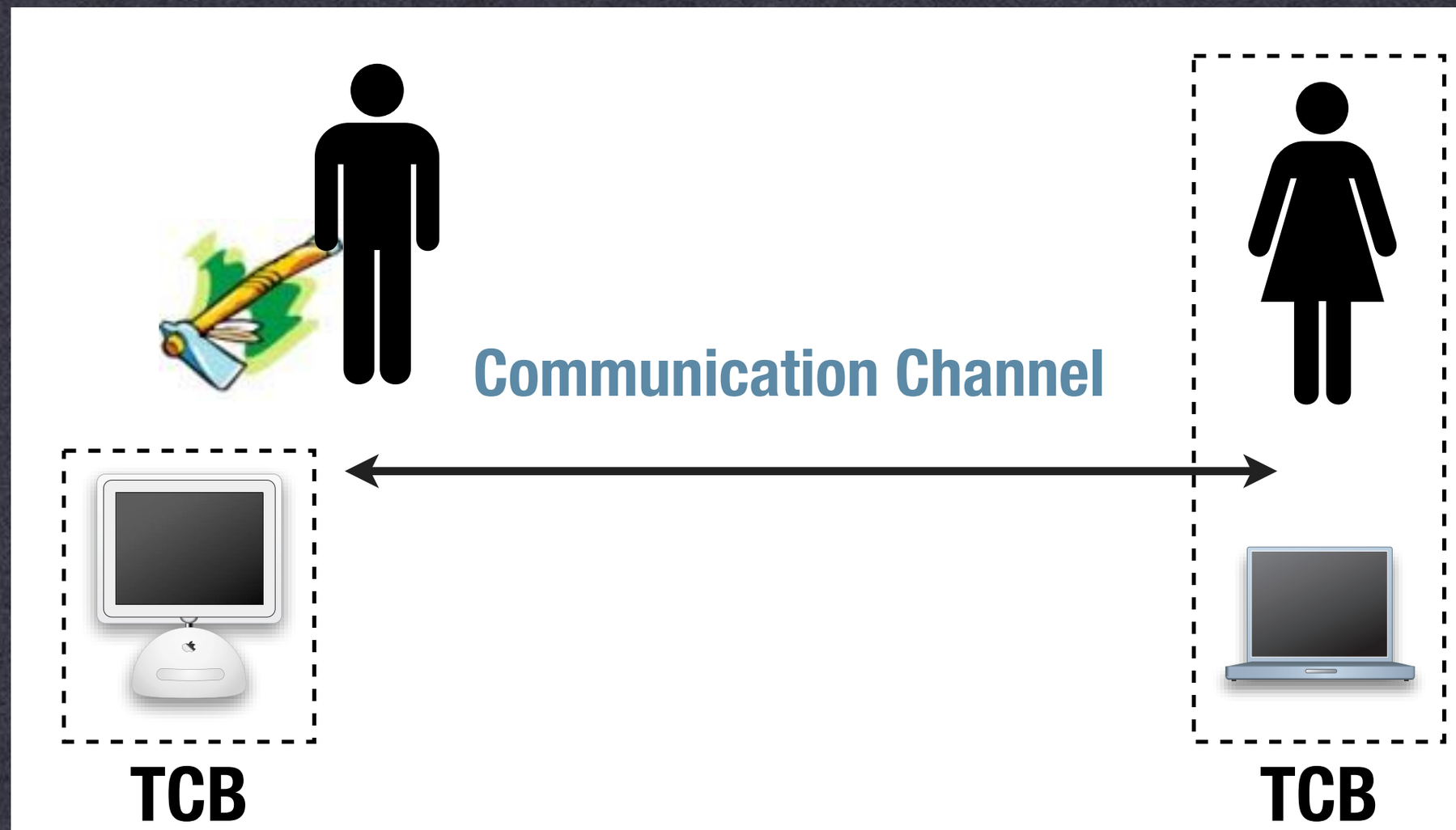
Replies

Foreshadow AaaS @ForeshadowAaaS · Aug 15, 2018

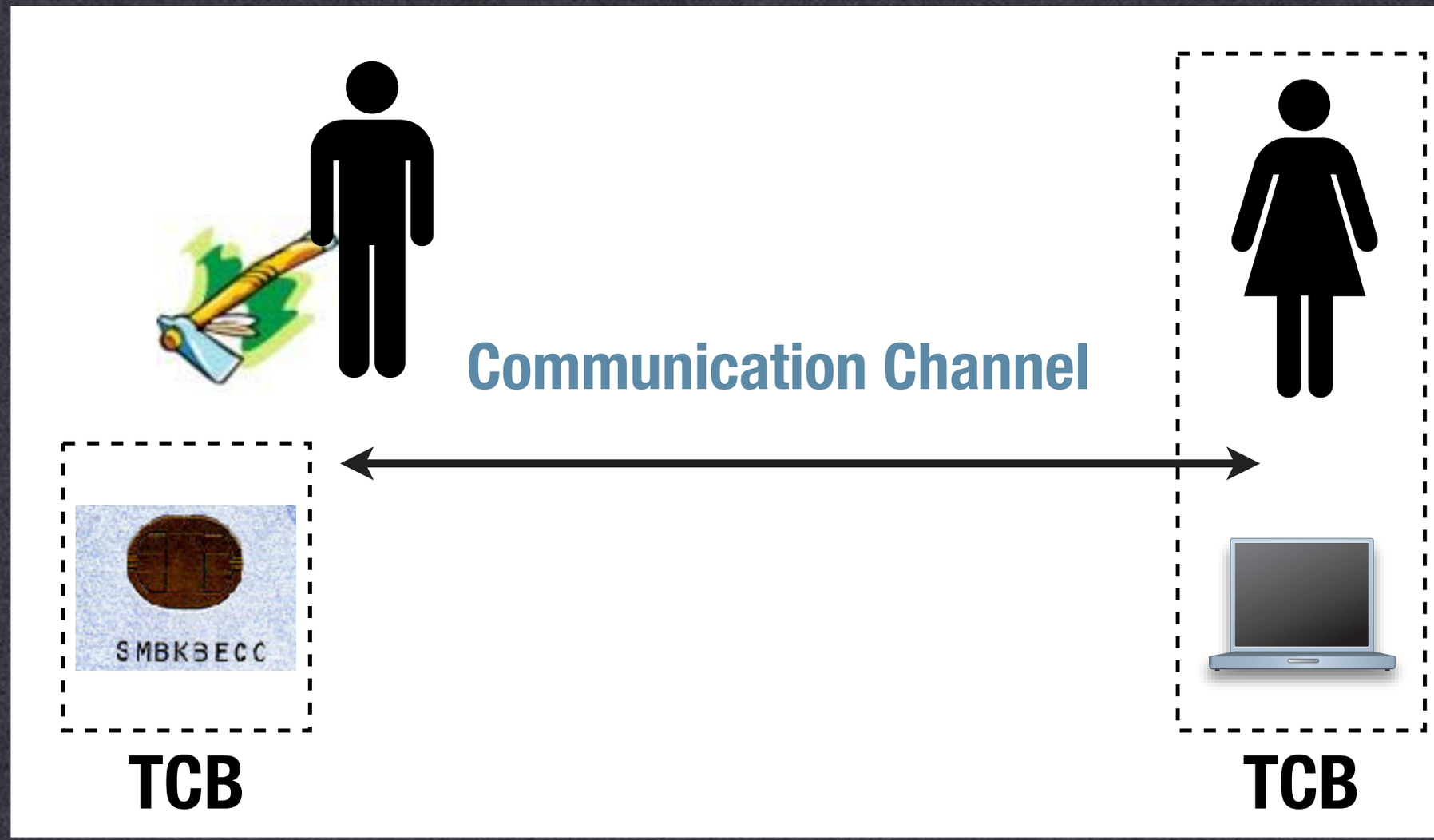
Here is your attestation that "RT: Hi, I'm the Foreshadow AaaS bot! :-) I react to tweets I'm tagged in by providing a genuine " is a genuine SGX enclave github.com/TeeAaas/Foresh...

1 1

Today



Today



Physical Security

- Devices in a hostile environment
  - Military cryptoprocessors
  - Financial services

-ATM machines

-Credit-card smartchips

  - Digital Rights Management devices

-Trusted Computing

Image of IBM Cryptoprocessor: www.cl.cam.ac.uk/~rnc1/ descrack

Threat Models

- Various levels of sophistication
  - Class 1: Clever Outsiders
    Moderate knowledge of system, will usually attack via a known weakness
  - Class 2: Knowledgeable Insiders
    Specialized technical education & experienc
    Access to the system & tools to exploit it
  - Class 3: Funded Organizations
    Team of attackers backed by significant funding,

Source: Abraham, Dolan, Double, Stevens: Transaction Security System, pp.206-229 (via Anderson Chap. 14)

FIPS Levels

- **FIPS 140-2: Cryptographic Modules**
  - **FIPS 140, Level 1**

-Software only, no physical security

  - **FIPS 140, Level 2**

-Tamper evidence or pick-resistant locks

  - **FIPS 140, Level 3**

-Tamper resistance

  - **FIPS 140, Level 4**

-Strong physical security around device

System Examples

- Classical ATM machine:
  - Armor, temperature sensors, tilt sensors
  - Attacker has to drill through the shell, defeat any tamper sensors, avoid setting off alarms
  - Might lose $$, but can often protect cryptographic secrets

System Examples

- Modern ATM machine:
  - No armor, limited physical security
  - Owner/operator might be untrustworthy
  - Limited $$ amounts, still worry about loss of key material
  - Interface based on commodity OS (Windows CE?)

System Examples

- Back-end server:
  - Strong physical security around server room
  - Performance is critical
  - Insider attacks a major concern
  - Large potential $$ losses if key material compromised
  - Linux/Windows/Legacy

System Examples

- **Client-side smartcard:**
  - **User/owner is potentially hostile**
  - **Worse if card is stolen**
  - **Attacker-supplied power source**

Two approaches

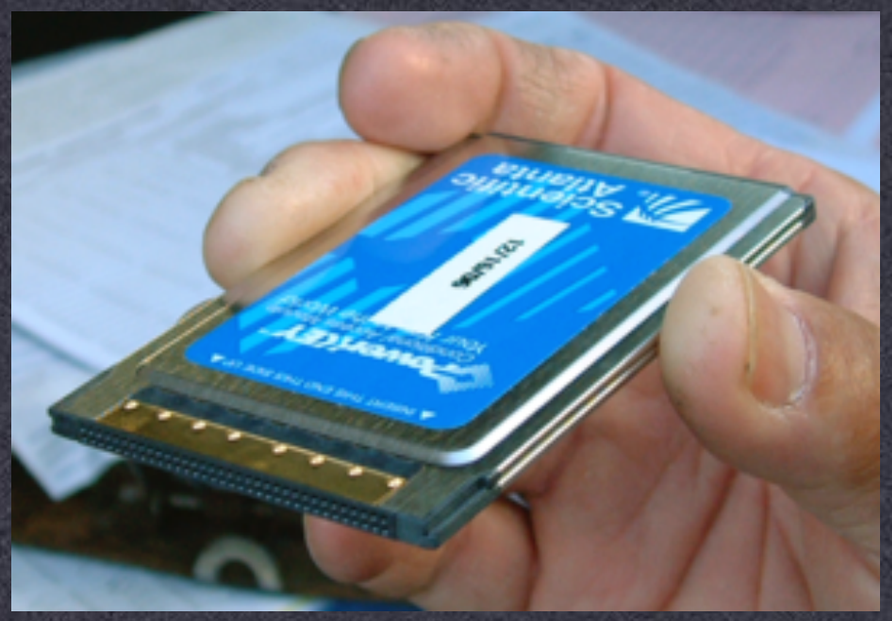
- Tamper-evidence
  - Make tampering obvious
  - Allow for recovery/renewability (e.g., re-generate cryptographic keys
- Tamper-resistance:
  - Keep attackers out
  - Wipe cryptographic key material

A General Approach

- Minimize the size of the “T” in our TCB
  - I.e., putting an ATM in a safe is expensive
  - Anchor trust in this area, build a secure system from there
  - Might involve untrusted storage/RAM/ROM/ processing, all connected to one trusted component

A General Approach

- Might also be necessary for practical reasons:
  - Support multiple “untrusted” manufacturers
  - E.g., SIM chips
  - E.g., CableCARD



Protecting Cryptoprocessors

- Metal
  - Can be cut/drilled
- Epoxy
  - Can be scraped off, penetrated with a logic analyzer probe

IBM 4758

IBM 4758

- High-security Cryptoprocessor
  - RAM, CPU, cryptographic circuitry
  - Dedicated SRAM for key memory
  - Battery powered
  - All wrapped in:

-Aluminum EM shielding

-Tamper-sensing mesh

-Potting material

Source: Anderson Chap 14

IBM 4758

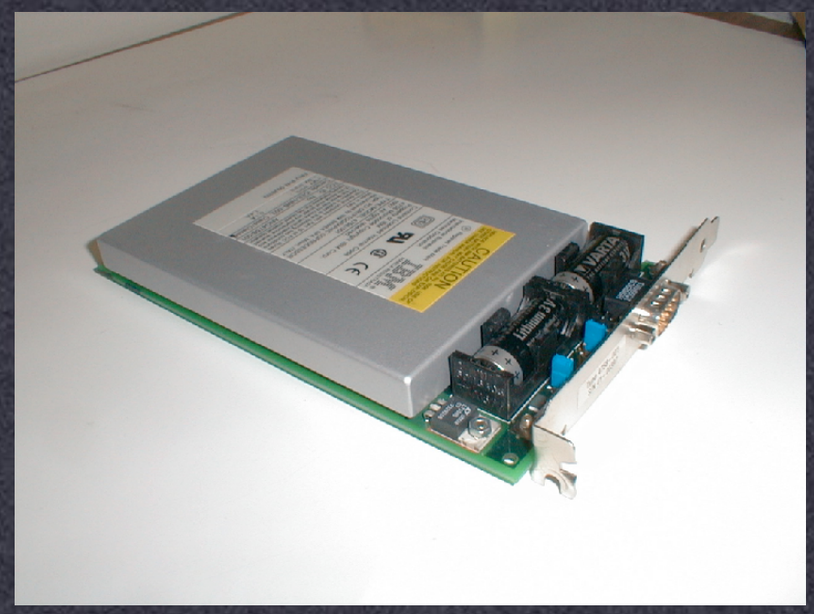
- Tamper sensing mesh:
  - Thousands of small wires wrapped around device
  - Interrupting any circuit (i.e., drilling) wipes the contents of key SRAM

-v1: copper wires

-v2: printed circuits

IBM 4758

- Memory remanence attacks:
  - Key “burn in”
  - After storing keys for too long, they may become permanent default states of RAM
  - NSA Forest Green Book has guidelines for this
  - Solution: “RAM savers”



IBM 4758

- Memory remanence attacks:
  - Attacker might try to freeze the device
  - Thus keys would survive wipe
  - Bursts of X-Rays can “lock” RAM in
  - Protections: temperature sensor, radiation sensor

-Gets too cold, keys go away

-Though what if we ship UPS!

Capstone/Clipper

- Proposed national voice encryption device
  - Designed as a compromise between need for strong crypto, and US's need to eavesdrop
  - Contained a secret cipher (Skipjack) w/80-bit key
  - Tamper-resistance:

-Difficult to extract embedded secret keys

-Difficult to R.E. Skipjack design

-Difficult to alter operation

  - Currently used in Fortezza card

Capstone/Clipper

- Threat model & design considerations
  - Extremely hostile environment
  - Range of well-funded adversaries (probably non-military)
  - Protecting secrets & design & operation
  - Very small device

Clipper/Capstone

- Tamper-resistance:
  - Extremely sophisticated
  - Metal/epoxy top
  - Vialink Read-Only Memory (VROM)

-Bits set by blowing antifuses using electrical charges

Clipper/Capstone

- **Attacking Clipper & QuickLogic:**
  - **Remove upper metal layer (difficult)**
  - **Use an electron microscope to read VROM (expensive & difficult, requires extra analysis)**
  - **Monitor circuit while chip is in use (more promising)**

Capstone/Clipper

- Operation:
  - Each ciphertext accompanied by LEAF (Law Enforcement Access Field)
  - Essentially, key escrow under gov't key
  - LEAF contains a "checksum" (MAC) to prevent tampering/removal
- Break:
  - Matt Blaze - found that LEAF bound to message w/ 16-bit checksum

Smartcards

- Very widely used
  - Contact/Contactless varieties
  - Small microprocessor/RAM/Serial bus
  - Became major targets of attack due to:

-Satellite TV

-GSM phones

-Lately: Payment Cards

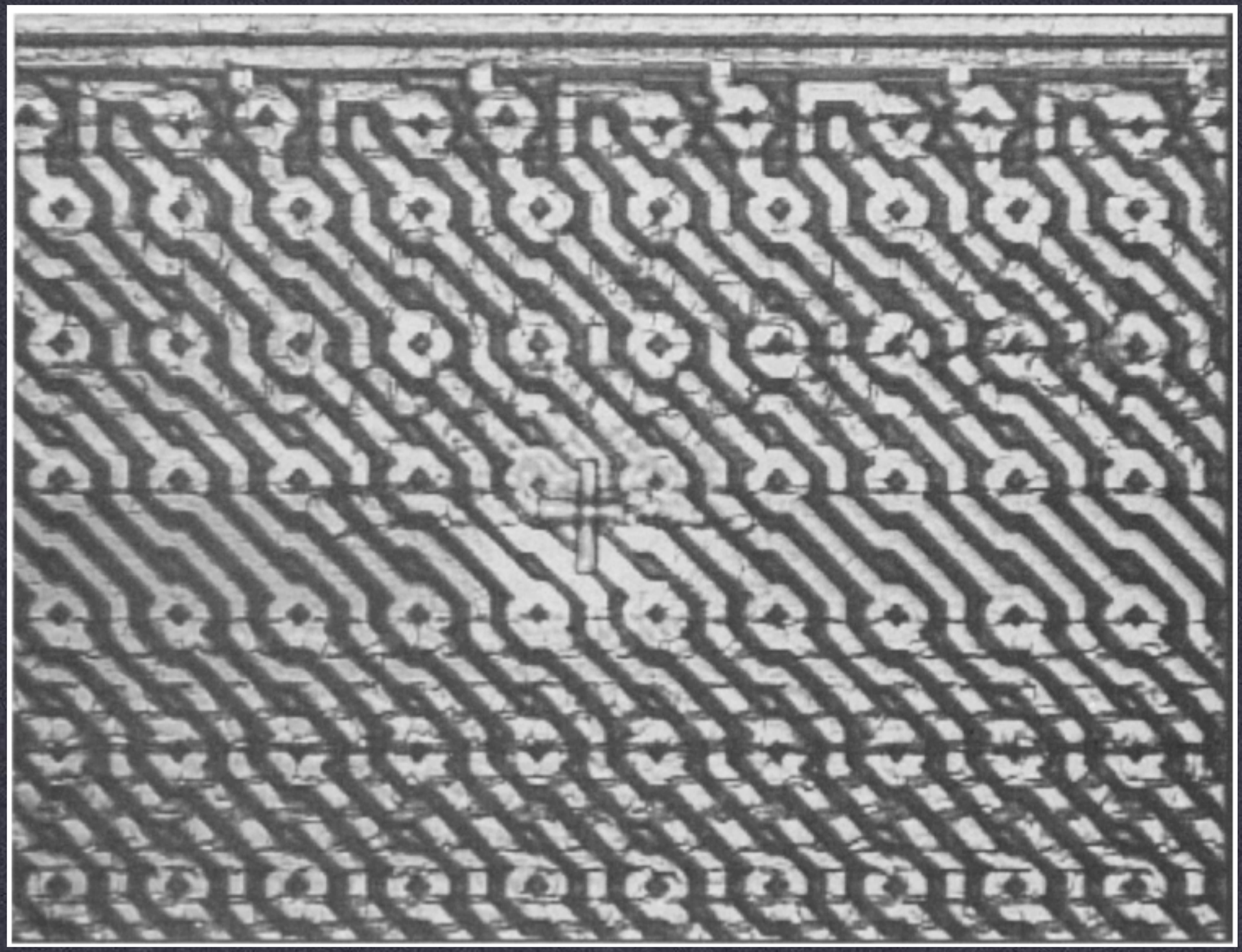
Attacking Smartcards

- Protocol-level attacks
  - Intercept messages, control access
- Side-channel attacks
  - Attacker controls the power source
  - DPA countermeasures introduces in 1990s

Attacking Smartcards

- Attacks on voltage supply
  - Memory read/write attacks
  - Fault injection
- Attacks on hardware
  - Take it apart, use an electron microscope
  - Same countermeasures, though:

-protective mesh, potting



Tamper-evidence

- Seals/locks
  - Make sure you can detect and renew security after an attack occurs
- Can even be implemented in software (under certain assumptions)

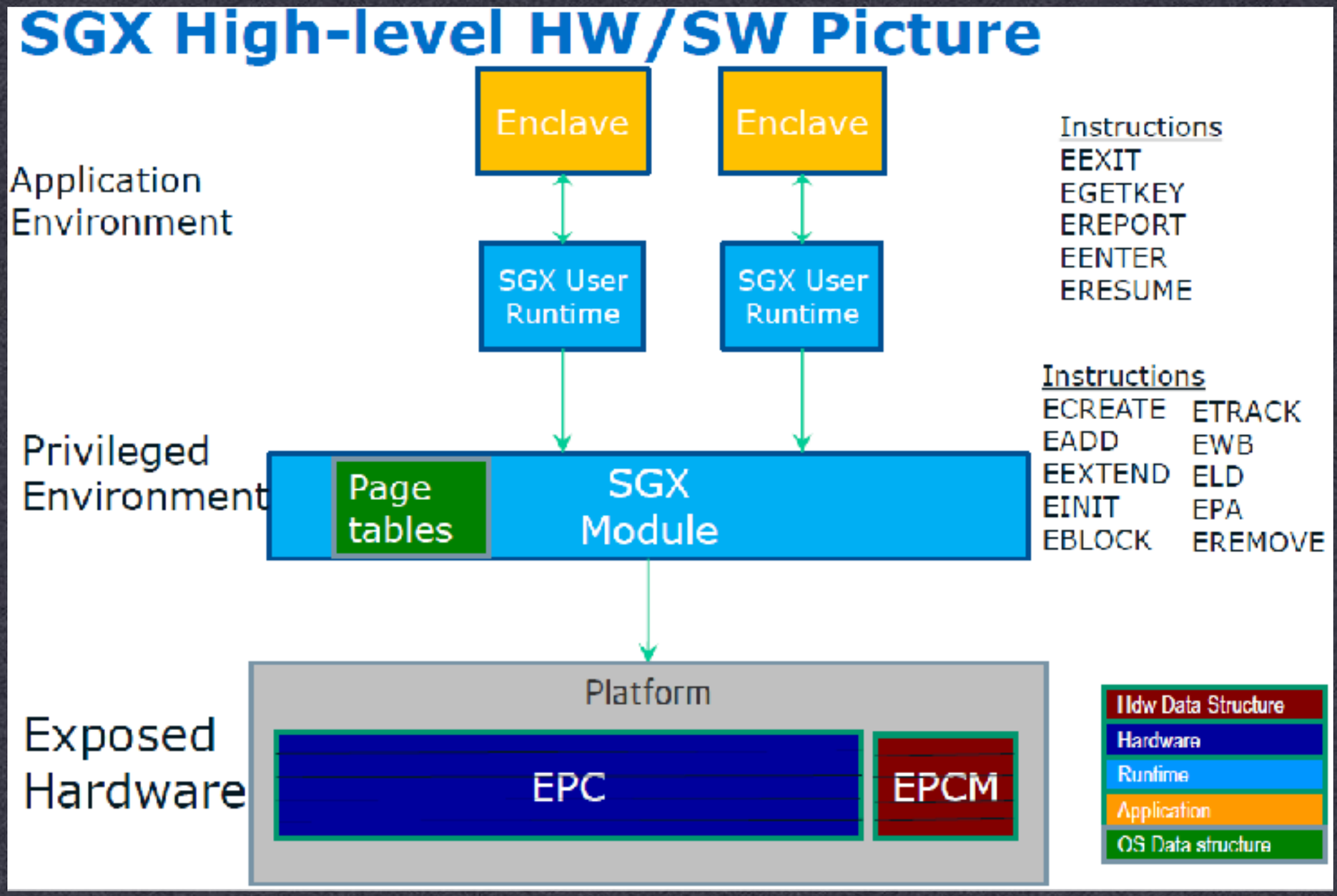
Trusted Computing

- Effort to put a tamper-resistant security processor into every PC
  - Useful to verify software integrity
  - Key management for access control & DRM
  - Minimize co-processor footprint by bootstrapping secure software
- Newest incarnation: Intel SGX

Trusted Computing

- Effort to put a tamper-resistant security processor into every PC
  - Useful to verify software integrity
  - Key management for access control & DRM
  - Minimize co-processor footprint by bootstrapping secure software
- Newest incarnation: Intel SGX

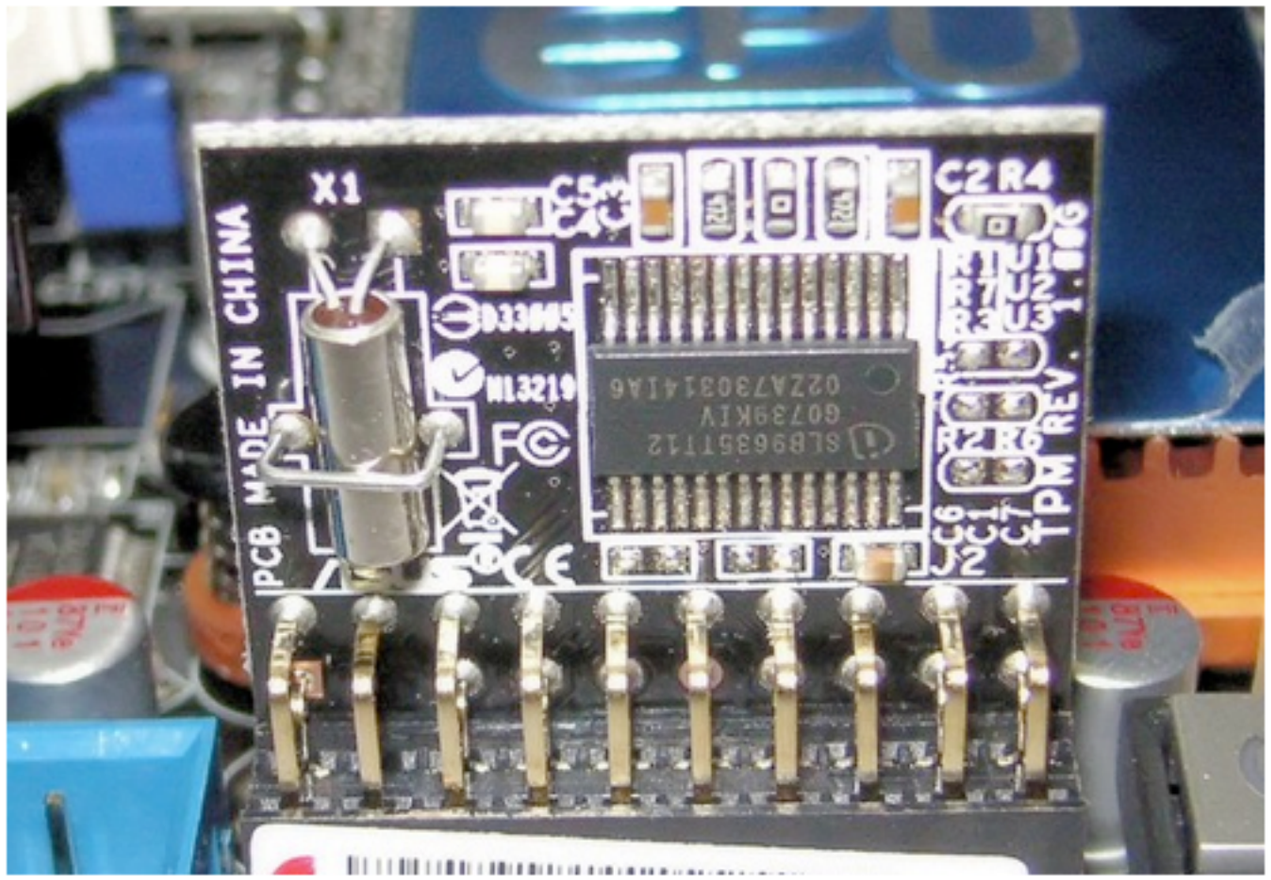
SGX


SGX High-level HW/SW Picture
Application Environment
Privileged Environment
Exposed Hardware
Enclave
Enclave
SGX User Runtime
SGX User Runtime
Page tables
SGX Module
Platform
EPC
EPCM
Instructions
EEXIT
EGETKEY
EREPORT
EENTER
ERESUME
Instructions
ECREATE
EADD
EEXTEND
EINIT
EBLOCK
ETRACK
EWB
ELD
EPA
EREMOVE
Hdw Data Structure
Hardware
Runtime
Application
OS Data structure

FILED UNDER *Desktops*, *Laptops*

Christopher Tarnovsky hacks Infineon's 'unhackable' chip, we prepare for false-advertising litigation

By Tim Stevens posted Feb 12th 2010 10:31AM



As it turns out, Infineon may have been a little bit... *optimistic* when it said its SLE66 CL PE was "unhackable" -- but only a little. The company should have put an asterisk next to the word, pointing to a disclaimer indicating something to the effect of: "Unless you have an electron microscope, small conductive needles to intercept the chip's internal circuitry, and the acid necessary to expose it." Those are some of the tools available to researcher Christopher Tarnovsky, who perpetrated the hack and presented his findings at the Black Hat DC Conference earlier this month. Initially, Infineon claimed what he'd done was impossible, but now has taken a step back and said "the risk is manageable, and you are just attacking one computer." We would tend to agree in this case, but Tarnovsky still deserves serious respect for this one. Nice work, Big Gun.

Trusted Computing

- Implications
  - Same chip used in the XBox 360
    (allegedly, Tarnovsky offered $100k to hack it)
  - Illustrates the most important lesson
    of tamper resistant systems:

-Individual devices can and will be hacked

-Must ensure that single-device hack
(or easily replicable attack)
does not lead to system-wide compromise!