Trusted Computing and SGX

Acknowledgments: Lecture slides are from the Computer Security course thought by Dan Boneh and John Mitchell at Stanford University. When slides are obtained from other sources, a a reference will be noted on the bottom of that slide. A full list of references is provided on the last slide.

TCG: Background

TCG consortium. Founded in 1999. Lots of companies.

Goals:

- **Hardware protected (encrypted) storage:**
  - Only "authorized" software can decrypt data
  - e.g.: protecting key for decrypting file system

    ⇒ only "authorized" software can boot

- **Attestation:** Prove to remote server what software started on my machine.

TCG: changes to the PC

Extra hardware: Trusted Platform Module (**TPM**) chip (33Mhz)

- Available on many laptops

Software changes:

Hardware layer: BIOS, EFI (UEFI)

Software: OS and apps

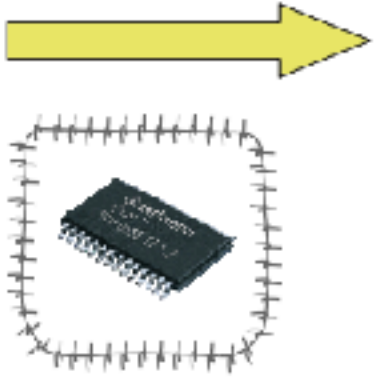
Trusted Computing

What is the TPM?

Integrating Trust and Security into Computing Platforms using a Security Chip

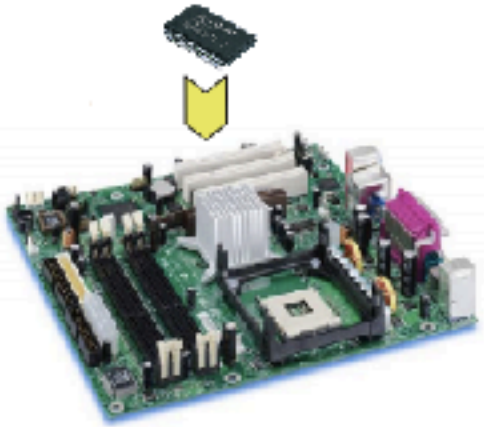
Standard Processor System

- Easy to program
- Easy to change
- Easy to attack



TPM-Security Module

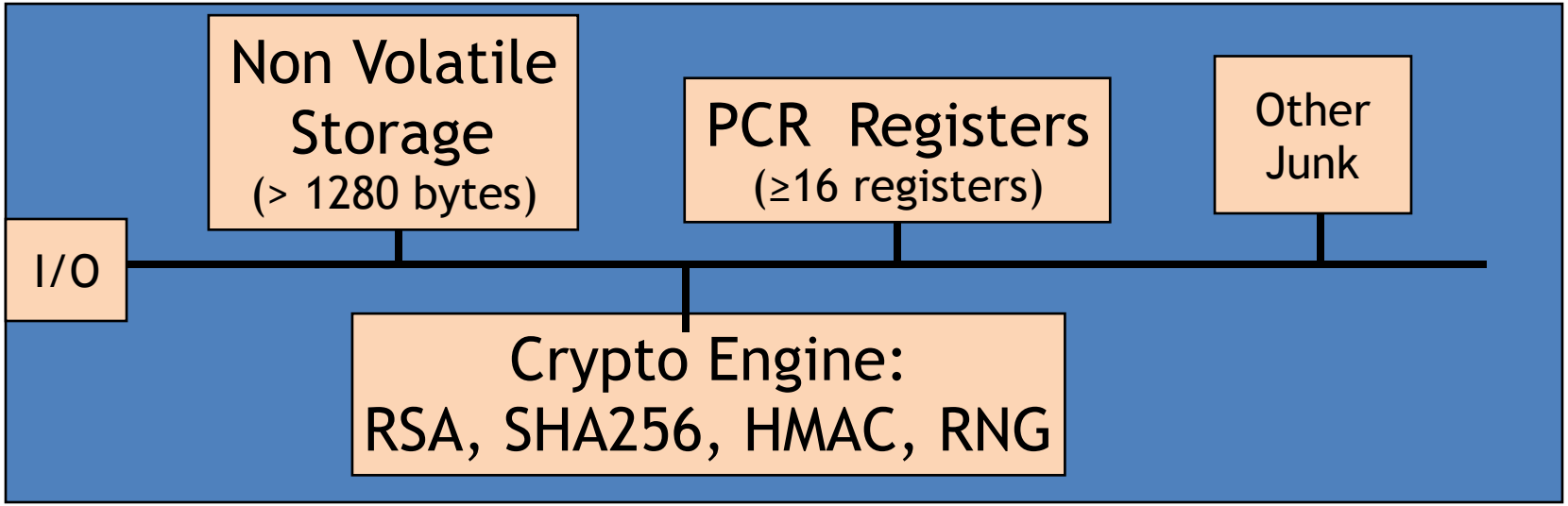
- Shielded and encapsulated chip
- Controlled interface to external
- Trusted software in a protected hardware



Trusted platform

=> Security functions, protected against manipulations

Components on TPM chip



RSA: 1024, 2048 bit modulus

SHA256: Outputs 32 byte digest

Non-volatile storage

1. Endorsement Key (EK) (2048-bit RSA)
   - Created at manufacturing time. Cannot be changed.
   - Used for "attestation" (described later)

2. Storage Root Key (SRK) (2048-bit RSA)
   - Used for encrypted storage. Created after running
     TPM_TakeOwnership(OwnerPassword, …)
   - Can be cleared later with TPM_ForceClear from BIOS

3. OwnerPassword (160 bits) and persistent flags

Private: EK, SRK, and OwnerPwd never leave the TPM

PCR: the heart of the matter

PCR: Platform Configuration Registers

- Many PCR registers on chip (at least 16)
- Contents: 32-byte SHA256 digest (+junk)

Updating PCR #n :

- TPM_Extend(n,D): PCR[n] ⟵ SHA256 (PCR[n] || D)
- TPM_PcrRead(n): returns value(PCR(n))

PCRs initialized to default value (e.g. 0) at boot time

Using PCRs: the TCG boot process (SRTM)

On power-up: TPM receives a TPM_Init signal from LPC bus.

BIOS **boot block** executes:

- Calls TPM_Startup (ST_CLEAR) to initialize PCRs to 0
  [can only be called once after TPM_Init]
- Calls PCR_Extend(n, <BIOS code>)
- Then loads and runs BIOS post boot code

BIOS executes: Calls PCR_Extend(n, <MBR code>)

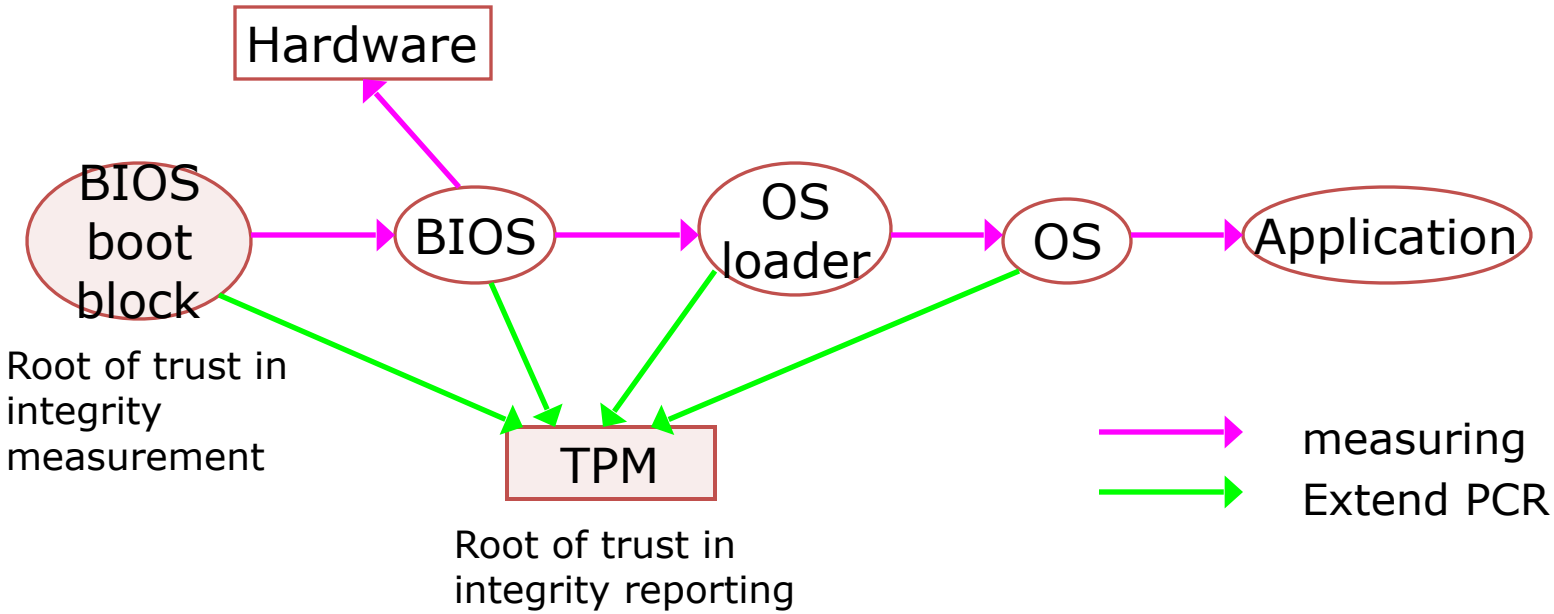
- Then runs MBR (master boot record), e.g. GRUB.

MBR executes: Calls PCR_Extend(n, <OS loader code, config>)

- Then runs OS loader

… and so on

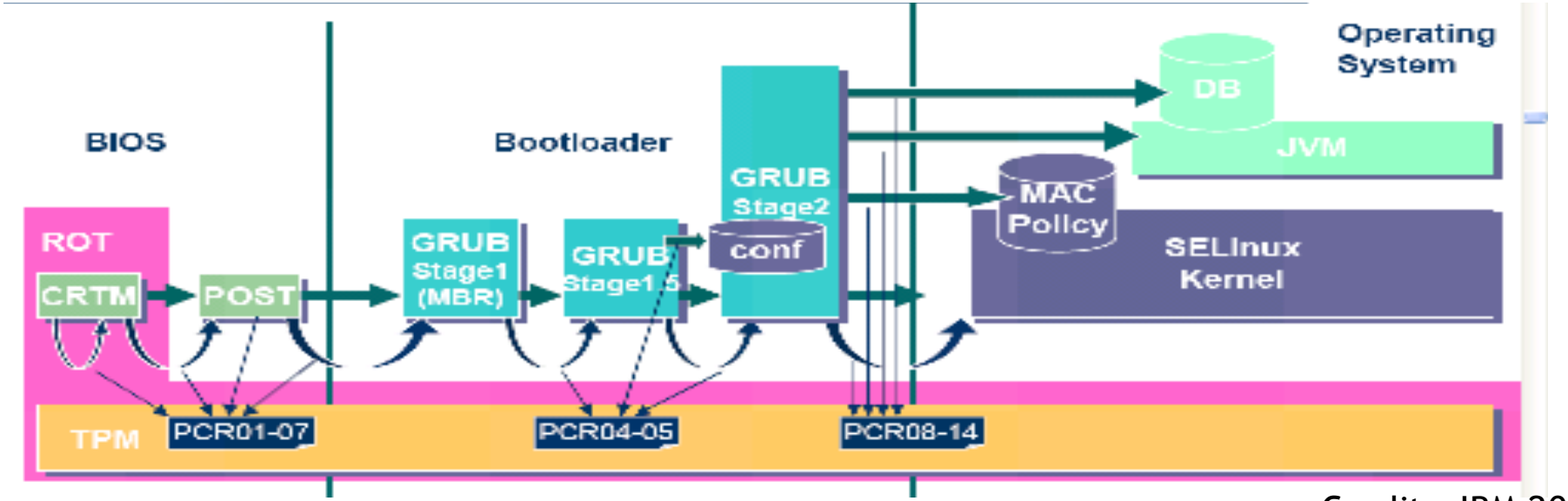
In a diagram



After boot, PCRs contain hash chains of booted software

Collision resistance of SHA256 ensures commitment

Example: Trusted GRUB



Credit: IBM 2005

PCR # to use and what to measure is specified in GRUB config file

The main point

After boot completes, PCR registers measure the entire software stack that booted on the machine:

- BIOS and hardware configuration
- Boot loader and its configuration
- Operating system
- Running apps

What would go wrong if TPM_Startup (ST_CLEAR) could be called at any time after boot?

Malicious OS could reset PCRs post-boot and then set them to a valid OS hash. PCRs would then look as if valid OS loaded.

TPM Counters

- TPM must support at least four hardware counters
  - Increment rate: every 5 seconds for 7 years.

- Applications:
  - Provide time stamps on blobs.
  - Supports “music will pay for 30 days” policy.

Trusted Computing

Using PCRs after boot

Using PCRs after boot

Application: **encrypted (a.k.a sealed) storage.**

Setup step 1: TPM_TakeOwnership(OwnerPassword, …)

- Creates 2048-bit RSA Storage Root Key (SRK) on TPM
- Cannot run TPM_TakeOwnership again without OwnerPwd:
  - Ownership Enabled Flag ⟵ False
- Done once by IT department or laptop owner.

(optional) Step 2: TPM_CreateWrapKey / TPM_LoadKey

- Create more RSA keys on TPM protected by SRK
- Each key identified by 32-bit keyhandle

Implementing Protected Storage

TPM_Seal: Encrypt data using RSA key on TPM. (some) Arguments:

- keyhandle: which TPM key to encrypt with
- KeyAuth: Password for using key `keyhandle'
- PcrValues: PCRs to embed in encrypted blob (named by PCR num.)
- data block: at most 256 bytes [e.g. an AES key]

Returns encrypted blob.

Main point: blob can only be decrypted with TPM_Unseal when **PCR-reg-vals = PCR-vals** in blob. TPM_Unseal fails otherwise

Protected Storage

Embedding PCR values in blob ensures that only specific apps can decrypt data.

– Changing MBR or OS kernel will change PCR values

⇒ data cannot be decrypted

Sealed storage: applications

Lock software on machine:

- Suppose OS and apps are sealed with MBRs PCR value
- Any changes to MBR will prevent sealed OS from loading
- Prevents modifying or inspecting OS (or loading other OS)

Web server: seal server's SSL private key

- Goal: only unmodified Apache can access SSL key
- Problem: updates to Apache or Apache config

Example: BitLocker drive encryption

tpm.msc: utility to manage TPM (e.g TakeOwnership)

- Auto generates 160-bit OwnerPassword
- Stored on TPM and in file computer_name.tpm

Volume Master Key (VMK) encrypts disk volume key

- VMK is sealed (encrypted) under TPM SRK using
  - BIOS, extensions, and optional ROM (PCR 0 and 2)
  - Master boot record (MBR) (PCR 4)
  - NTFS Boot Sector and block (PCR 8 and 9)
  - NTFS Boot Manager (PCR 10), and
  - BitLocker Access Control (PCR 11)

BitLocker

Many options for **VMK** recovery: disk, USB, paper (enc. with pwd)

- Recovery needed after legitimate system change:
  - Moving disk to a new computer
  - Replacing system board containing TPM
  - Clearing TPM (with TPM_ForceClear)

At system boot (before OS boot)

- Optional: OS loader requests PIN or USB key from user
- TPM unseals VMK, only if PCR and PIN are correct

Suppose BIOS code is updated by a firmware update.

How would the system enable access to blobs previously sealed to current BIOS version?

Patch process must re-seal all blobs with new PCR values

Trusted Computing

Security?

Security? [Kauer 2007]

Attack 1: reset TPM after boot with a wire

- Connect LRESET pin to ground -- mimics TPM_Init on LPC bus
  - then extend PCRs arbitrarily
- Harder in TPM 2.0 due to "locality"

Attack 2: block TPM until after boot, then extend PCRs arbitrarily

Better root of trust

- DRTM – Dynamic Root of Trust Measurement
  - AMD: **skinit** Intel: **senter**
  - Atomically does:
    - Reset CPU. Reset PCR 17 to 0.
    - Load the given Secure Loader (SL) code into I-cache (locked)
    - Extend PCR 17 with SL
    - Jump to SL

- BIOS boot loader is no longer root of trust. Processor microcode is.
- Avoids TPM_Init attack: TPM_Init sets PCR 17 to -1

Trusted Computing

Attestation

Attestation: what it does

Goal: prove to remote party what software loaded on my machine

Good applications:

- Bank allows money transfer only if customer's machine runs "up-to-date" OS patches
- Enterprise allows laptop to connect to its network only if laptop runs "authorized" software
- Gamers can join network only if their game client is unmodified

DRM: MusicStore sells content for authorized players only.

Attestation: how it works

Recall: EK private key on TPM.

- Cert for EK public-key issued by TPM vendor.

Step 1: Create Attestation Identity Key (AIK)

- Details not important here
- AIK Private key known only to TPM
- AIK public cert issued only if EK cert is valid

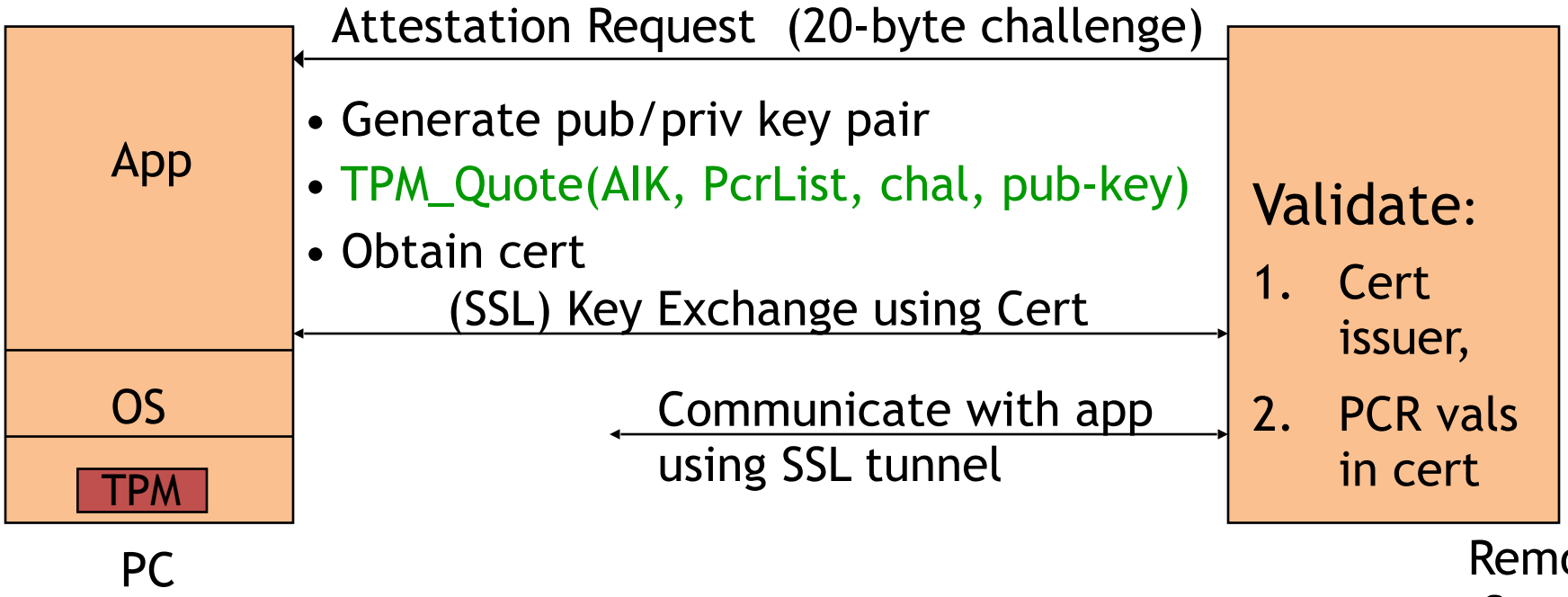
Attestation: how it works

Step2: sign PCR values (after boot) with TPM_Quote. Arguments:

- keyhandle: which AIK key to sign with
- KeyAuth: Password for using key `keyhandle'
- PCR List: Which PCRs to sign.
- Challenge: 20-byte challenge from remote server
  - Prevents replay of old signatures.
- Userdata: additional data to include in sig.

Returns signed data and signature.

Attestation: how it works



- Attestation typically includes key-exchange
- App must be isolated from rest of system

What would go wrong if communication between app. and server were done in the clear?

User can reboot machine after attestation and run arbitrary software pretending to be app.

Trusted Computing

Attestation: challenges

1. Attesting to Current State

- Attestation only attests to what code was loaded.

- Does not say whether running code has been compromised.
  - Problem: what if Quake vulnerability exploited after attestation took place?

- Can we attest to the current state of a running system?

2. Encrypted viruses

Suppose malicious music file exploits bug in video player.

- Video file is encrypted.
- TCG prevents anyone from getting video file in the clear.

- Can anti-virus companies study virus without seeing its code in the clear?

- How would you solve this?

3. TPM Compromise

Suppose one TPM Endorsement Private Key is exposed

– Destroys all attestation infrastructure:
  - Embed private EK in TPM emulator.
  - Now, can attest to anything without running it.

⇒ Certificate Revocation is critical for TCG Attestation.

Intel SGX

SGX: Goals

- Extension to Intel processors that support:
- **Enclaves**: running code and memory isolated from the rest of system
- **Attestation**: prove to local/remote system what code is running in enclave
- **Minimum TCB**: only processor is trusted

  nothing else: DRAM and peripherals are untrusted

  ⇒ all writes to memory must be encrypted

Applications

Server side:

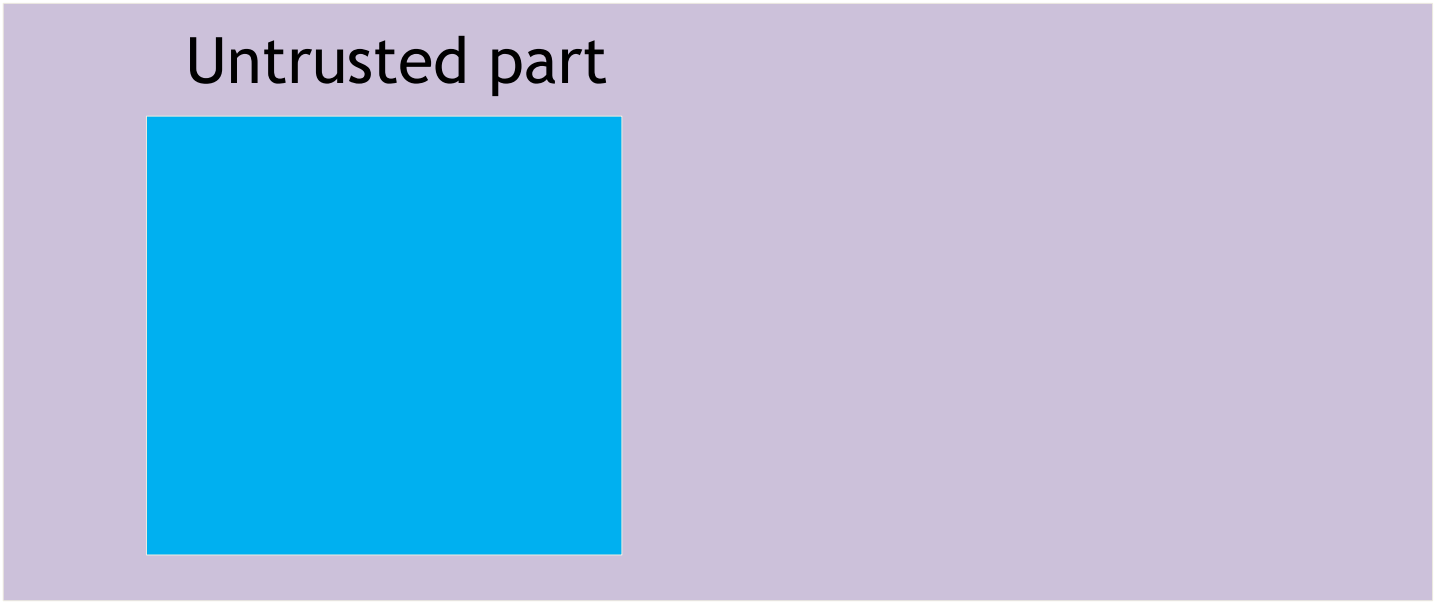
- Storing a Web server HTTPS secret key
  - secret key only opened inside of an enclave
  - malware cannot get the key
- Running a private job in the cloud: job runs in enclave
  - Cloud admin cannot get code or data of job

Client side:

- Hide anti-virus (AV) signatures:
  - AV signatures are only opened inside an enclave
  - not exposed to adversary in the clear

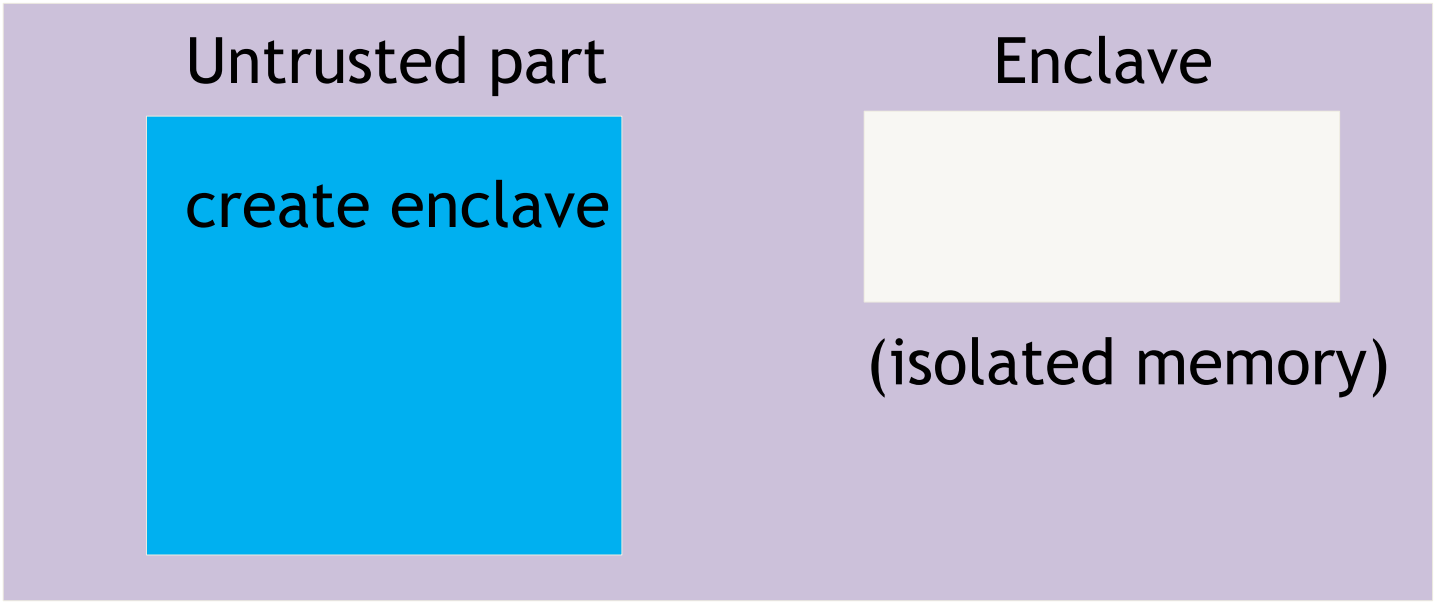
How does it work?

- An application defines part of itself as an enclave



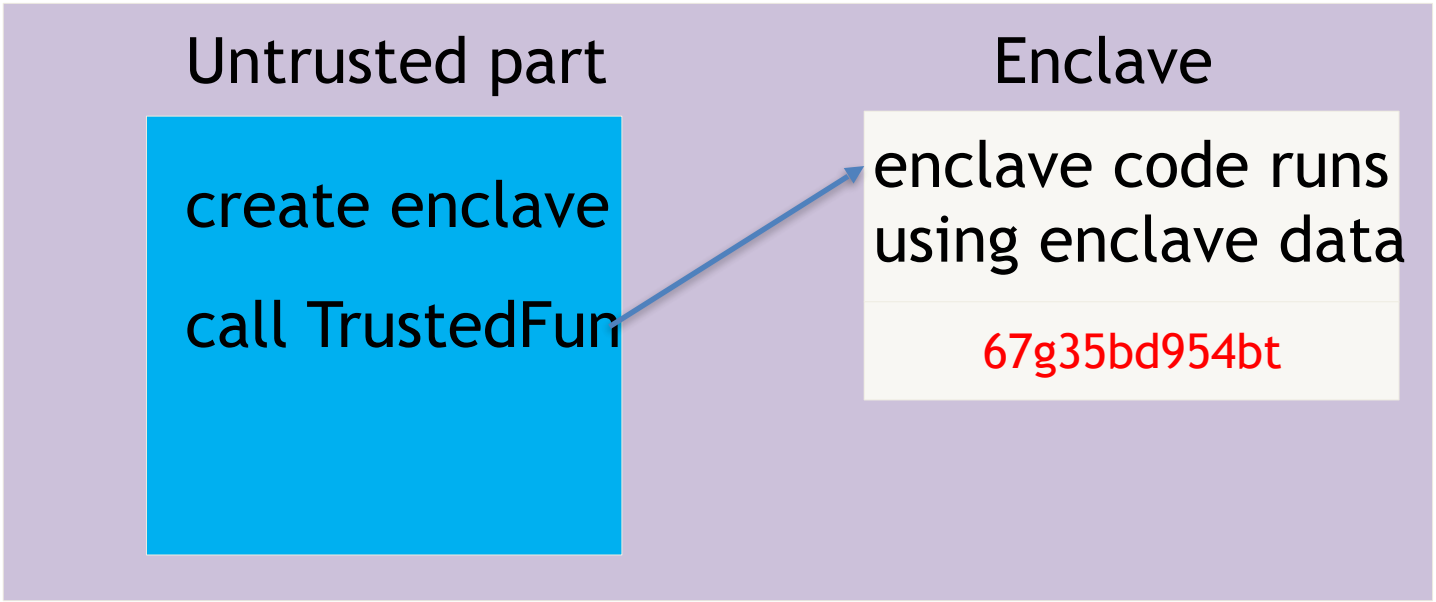
How does it work?

- An application defines part of itself as an enclave



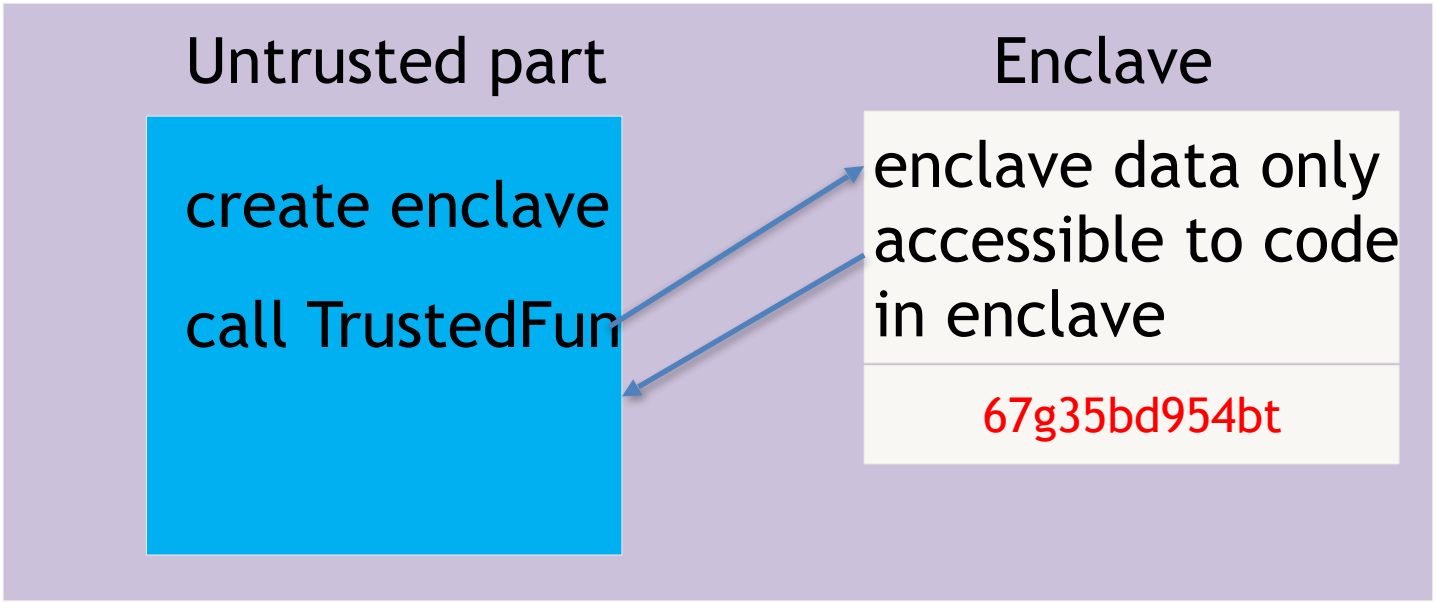
How does it work?

- An application defines part of itself as an enclave



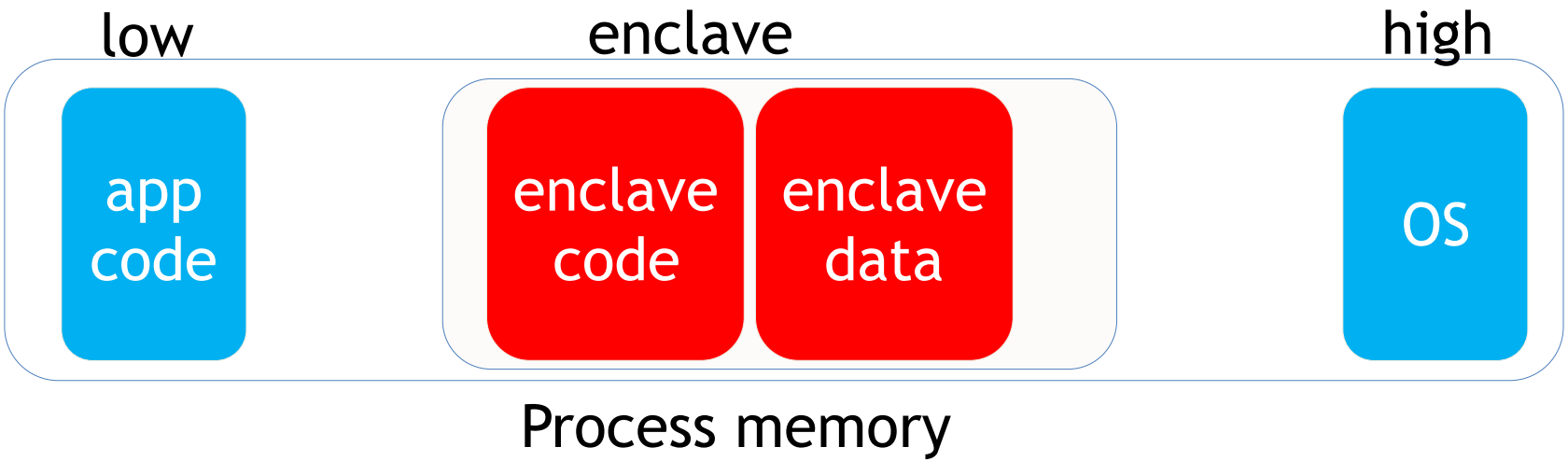
How does it work?

- An application defines part of itself as an enclave



How does it work?

- Part of process memory holds the enclave:



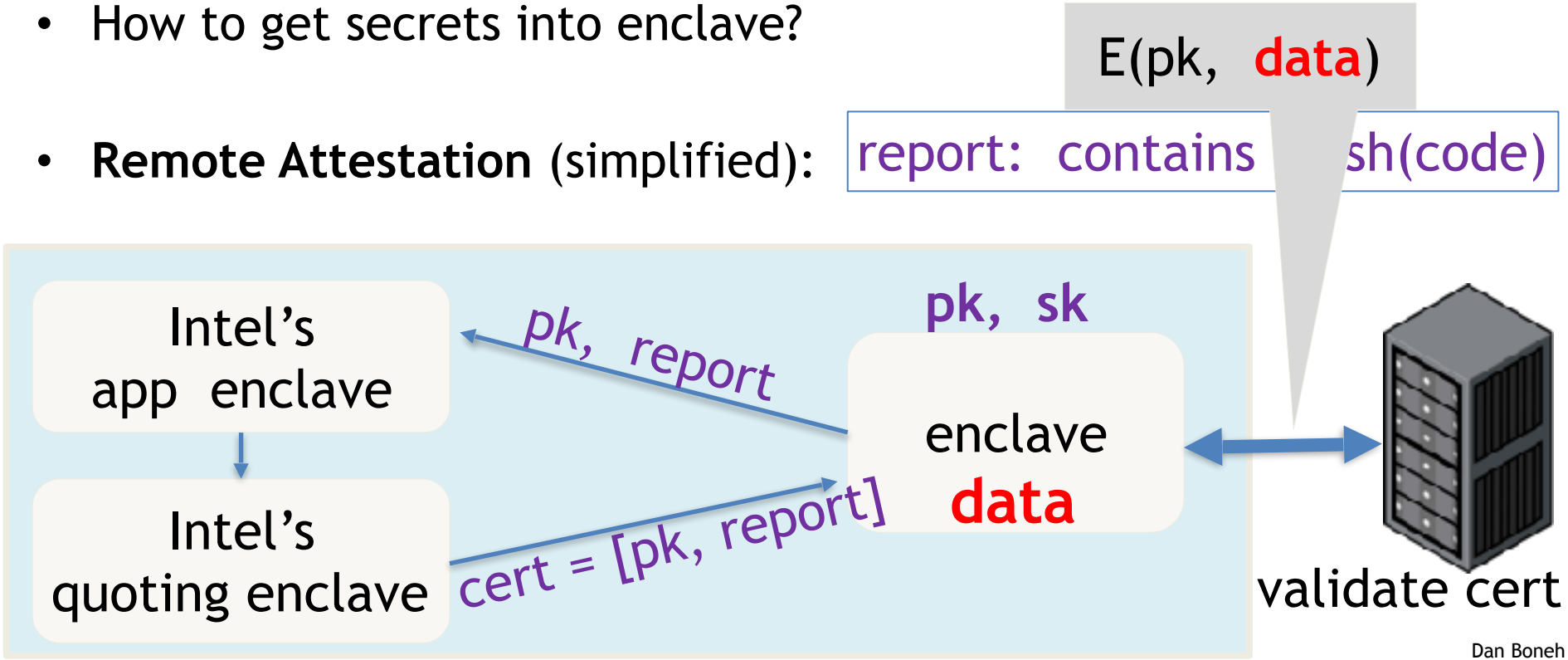
Creating an enclave: new instructions

- **ECREATE**: establish memory address for enclave
- **EADD**: copies memory pages into enclave
- **EEXTEND**: computes hash of enclave contents (256 bytes at a time)
- **EINIT**: verifies that hashed content is properly signed
- if so, initializes enclave (signature = RSA-3072)

- **EENTER**: call a function inside enclave
- **EEXIT**: return from enclave

Provisioning enclave with secrets: attestation

- The problem: enclave is in the clear prior to activation (EINIT)
- How to get secrets into enclave?
- **Remote Attestation** (simplified):



Summary

- SGX: a powerful architecture for managing secret data
- Enables processing of data that cannot be read by anyone, except for code running in enclave
- Minimal TCB: nothing trusted except for x86 processor
- Not suitable for legacy applications

THE END