CS 5430:
Measured Principals and Gating Functions

Fred B. Schneider
Samuel B Eckert Professor of Computer Science

Department of Computer Science
Cornell University
Ithaca, New York 14853
U.S.A.

Overview

- New abstractions
  - Measured principal
  - Gating function
- Example implementations
  - TPM Trusted Platform Module
- Applications
  - Whole disk encryption
  - Cloud-hosted services
  - Digital rights management
  - Remote Attestation

Keys as Principals

Let K_P/k_P be a public/private key pair where k_P is accessible only to a principal P. We then would have:

- K_P **speaksfor** P
- K_P **says** S using: k_P-**sign**(S)

"Accessible only to…"

- Store k_P in processor memory?
  - How to block attacker access?

- Use external hardware security module (HSM)?
  - HSM as secure storage? Exported key is vulnerable.
  - HSM as remote eval of crypto function (using key).
    - HSM must authenticate caller.
    - HSM must implement authorization for using key.

  … what caller name is authenticated and authorized?

Overview

- New abstractions
  - Measured principal
  - Gating function
- Example implementations
  - TPM Trusted Platform Module
- Applications
  - Remote Attestation
  - Whole disk encryption
  - Cloud-hosted services
  - Digital rights management

What's in a name?

Authorization based on name presumes:

Translation: Name → properties of executions

- Name must reflect or depend on:
  - Actual bits that will be executed.
  - Execution environment for that code:
    - Initialization data read.
    - Code already executing as available services.

What’s in a name?

Authorization based on name presumes:

Translation: Name → properties of executions

- Name must reflect or depend on:
  - Actual bits that will be executed.
  - Execution environment for that code:
    - Initialization data read.
    - Code already executing as available services.
      - Binary code that will be executed
      - Execution environment for that code:
        - Initialization data read
        - Code already available as services
          - Binary code that will be executed…

Name construction

A name for App would involve other names:

Hardware processor + I/O
→ Boot firmware
→ Boot code & data read
→ OS IPL code & data read
→ OS
→ App

Measured Principals

Properties of a **measured principal**:

- Name derived from code, data read at startup, and environment.
- Change any bit(s) and get unpredictably different name.

- Name for a measured principal serves as a label for properties satisfied by principal's execution.
- Name for a measured principal could serve as a basis for trust.

Descriptions and Descriptors

- Name N(D) is a name generated for a **description** D.
- D is a sequence

$$\langle d_1\ d_2\ \ldots\ d_n \rangle$$

  of **descriptors** d_i such that
  - change to any descriptor d_i produces new description D′ with unpredictably different name N(D′)
  - d_i derived from all details of resource at the time of first access by measured principal with name N($\langle \ldots d_i \ldots \rangle$)
    - Resources include: processor, i/o devices, executables, storage regions, …
  - descriptors are listed in order of first access.
- **Goal**: Description indicates whether associated principal can be trusted.

Completeness of Descriptions

- Incomplete description: Leads to inaccurate predictions of possible behavior by principal.
- Complete description:
  - Blocks attacks by modified versions that spoof.
  - Prevent attacker persistence (APT) by file modifications.
  - Inconvenient: Customization, patches. upgrades change file contents… change descriptors… change name.

Properties of Naming Schemes

Properties of N(·):

- Collision resistance:
  - $D \neq D'$ implies $N(D) \neq N(D')$ with high probability.
- Preimage resistance:
  - Given D, it is infeasible to construct D′ where $D \neq D'$ and $N(D) = N(D')$ hold.

Implementation of Naming Schemes

$N(\langle d_1\ d_2\ \ldots\ d_n \rangle)$: Implement as a hash chain…

- $N(\langle\ \rangle) = 0$
- $N(D \cdot d_i) =$ **hash**$(N(D) \cdot$ **hash**$(d_i))$

Note, incremental calculation $N(D \cdot d_i)$:

- Allows files (in D) to become inaccessible after use.
  E.g., boot loader, IPL, …

If we assume a trusted source for integrity of names

- Only allowed change is: extension by d_i.

then no need to protect integrity of D_P for P.

- Simply check whether D_P satisfies $N(D_P) = N_P$

Descriptors for Code and Data

Code and data are bit strings.

Descriptor d_{Obj} for Obj is **hash**(Obj)

- Complication:
  - Copies of objects that incorporate addresses will have different descriptors when loaded.

Descriptors for HW Processor

Naïve approach: Include ROM with a unique id in each processor.

- Must be able to read id.
- If id can be read, then emulation is possible.

Descriptor details: HW Processor

Better approach: For a processor id,

- include unique signing key k_{id} in ROM.
- include instruction to generate k_{id}-**sign**(M).
- trusted party (manufacturer) has public key K_C

Distribute public key K_{id} for use as descriptor / name for processor.

k_C-**sign**(K_{id} **speaksfor** id)

Distribute certificate for ISA, too.

k_C-**sign**(K_{id} **speaksfor** ISA_{x86})

For privacy …

… might want a single processor to have multiple names.

- Prevents correlation of attributes by attackers who monitor requests at services.
- Prevents detection that two measured principals are executing on the same processor.

Solution: Processor invents new attestation identify key (signing key) e.g., k_{id2} for each different identity. A trusted third party certifies authenticity of corresponding public key K_{id2}.

Descriptors for Properties

Avoid brittleness of object descriptors by using descriptors for properties of the object rather than for implementation of the object.

Properties don't change (much) due to upgrades etc.

- E.g., signed certificate from trusted org about "linux" property.
- E.g., signed output of an analyzer chkr

Descriptor Auxiliary Information

Descriptors are opaque bit strings

- Hash of object   or   public key of processor

Trust in object might depend on object details, to allow:

- Identification and retrieval of objects associated with descriptor.
- Verification of descriptor by recalculating it from objects.
- Assessment of whether those objects should be trusted.

… So include **auxiliary information** with a descriptor.

Examples: d_i auxiliary information is

- Linux4.8.0.36-generic   … name of a system in a public repository
- /user/fbs/cs5432/finalExam.txt   … name of a file

Auxiliary information allows object to be independently downloaded and analyzed.

Overview

- New abstractions
  - Measured principal
  - Gating function
- Example implementations
  - TPM Trusted Platform Module
- Applications
  - Remote Attestation
  - Whole disk encryption
  - Cloud-hosted services
  - Digital rights management

Gating Functions Defined

Gating function [K-F](·) (*FBS notation*) associates access control with use of a key K and a crypto function K-F(·).

- K can be accessed **only** for evaluating gating functions [K-F](·).
  - Ensures confidentiality and integrity of K
- [K-F](·) requires system to satisfy Config([K-F]), which specifies a set of measured principals that must be executing for calculation of K-F(·) to proceed.

N.b. The brackets [...] are intended to suggest that crypto function K-F(·) has been wrapped with access control.

Uses for Gating Functions?

- Authentication / attestation of a system.
- Isolation?
  - Confidentiality by encryption.
  - Integrity by signatures or MAC or authenticated encryption.
  - Comparison: processes, virtual machines, containers.
    - GF weaker: Achieve integrity by creating unavailability.
    - GF stronger: Restricts what code can have access.
    - GF stronger: Supports attestation.

Overview

- New abstractions
  - Measured principal
  - Gating function
- Example implementations
  - TPM Trusted Platform Module
- Applications
  - Remote Attestation
  - Whole disk encryption
  - Cloud-hosted services
  - Digital rights management

Hardware Support

Simplified version of Trusted Platform Module (TPM) has:

- **Measurement registers** and instructions to update them.
  - Measurement registers are volatile.
  - Values in measurement registers are what Config(·) checks.
- **Key registers** and instructions for provisioning a key register with a fresh key.
  - Key registers are not volatile.
- Instructions to perform certain crypto operations using key in a given key register if certain Config(·) exists in measurement registers:
  - **sealing**: protect confidentiality and integrity of local content.
  - **quoting**: to establish authenticity of locally produced content.
  - **binding**: to import remote content if local system is proper.

TPM Design Precis

Confidentiality of keys follows because:

- Unencrypted keys born in key registers and never leave key registers in plaintext form.
- Instructions that use values in key registers compute functions that do not reveal the key.
- Key register values persist across boots but measurement register values don’t.
  - Access to keys requires the same measured principals to be running after a reboot

Measurement Registers

Measurement registers: mr_0, mr_1, …, mr_N.

- mr_0 auto incremented with each reboot.
  - Enables creation of ephemeral keys (if mr_0 is in Config)
  - Ephemeral keys defend against replay attacks.

Measurement Registers

Measurement registers: mr_0, mr_1, …, mr_N.

- mr_0 auto incremented with each reboot.
  - Enables creation of ephemeral keys (if mr_0 is in Config)
  - Ephemeral keys defend against replay attacks.
- mr_1, …, mr_N reset to 0 on reboot
- Instructions (with semantics):

  MRreset(mr_i): mr_i := 0

  MRextend(mr_i, *mem*): mr_i := **hash**(mr_i , **hash**(mem))

Sets of measurement registers are used to create names for measured principals.

Trust in Measurement Registers

Trust principals that execute MRextend if

- they are measured principals, and
- their names correspond to descriptions we have analyzed, and
- they were loaded by systems we trust.

… Result is chain of trust back to boot loader, firmware, processor hardware.

- Trust each subsequent link by trusting its predecessor
- Trust first link (=root of trust) based on information from an external source.

Configuration Constraints: Config

$$C = \{\langle 1,v_1\rangle \ldots \langle i,v_i\rangle \ldots \}$$

defines **configuration constraint** that is satisfied during execution if

$$mr_1 = v_1 \land \ldots\ mr_i = vi \land \ldots$$

holds. C may name only a subset of the measurement registers.

A configuration constraint C_{kr} is associated with each key register kr when a new key is generated there. So there is a configuration constraint associated with each gating function.

seal and unseal: Basics

Authenticated shared-key encrypt/decrypt.

- Shared key K generated into sealing key register skr_i.
- Configuration constraint C associated with skr_i.

seal creates a C/K-sealed value.

unseal recovers v from a C/K-sealed value v.

Properties of sealed values:

- **read** a C/K-sealed value v reveals nothing about v.
- **update** causes subsequent unseal to fail.
- … availability is compromised by write (unlike other forms of isolation).

seal and unseal: Instructions

sealing key registers: skr_1, …, skr_N, store: skr_i.key and skr_i.config

crSet is bit vector of length N: crSet[1]=1 iff $mr_i \in$ crSet

SKRgen(skr_i, crSet):

skr_i.key := fresh symmetric key;

skr_i.config := $\{ \langle j, v_j \rangle \mid crSet[j] \wedge mr_j = v_j \}$

seal and unseal: Instructions

sealing key registers: skr_1, …, skr_N, store: skr_i.key and skr_i.config

crSet is bit vector of length N: crSet[1]=1 iff $mr_i \in$ crSet

SKRgen(skr_i, crSet):

skr_i.key := fresh symmetric key;

skr_i.config := $\{ \langle j, v_j \rangle \mid crSet[j] \wedge mr_j = v_j \}$

seal(skr_i, in, out): ← Any principal can invoke!

out := shr_i.key-$Encrypt^A$(in)

unseal(skr_i, in, out): ← Only certain invokers succeed!

if ConfigSat(skr_i.config)

then out := shr_i.key-$Decrypt^A$(in)

else fail

seal and unseal: Uses

- **seal** can save state between executions / sessions.
- Protocol now needed to perform a software upgrade:
  - **unseal** all data;
  - Upgrade the software;
  - Reset and reload measurement registers;
  - Reprovision sealing key registers (uses updated values in measurement registers);
  - **seal** all data (uses updated sealing key registers);

… seal/unseal are slow, but ”data” might just be a single key that is used to encrypt/decrypt full state.

Key Archives

Coping with a small fixed number of key registers: Time-multiplexing

- Cannot extract raw key values from key registers.
- Store and restore key registers (with configuration constraints) using a key archive.
  - KRseal: invokes **seal** for a set of key registers (values and config constraints) and stores the result as a key archive.
  - KRunseal: invokes unseal for key archive and reloads the key registers.
    - By including mr_0 in kr.config for key register kr stale key values in old key archive don't work when reloaded.

quoting: Basics

quoted bit string: Signed using (private) key in some quoting key register qkr_i.

- Configuration constraint for qkr_i means quoted bit string is generated by a specific system (and thus can be trusted).
- qkr_{id}: special key register having a fixed value and no configuration constraints.
  - qkr_{id} contains the unique signing key k_{id} associated with processor.

quoting: Instructions

QKRgen(qkr_i, crSet, mem):

qkr_i.config := { ⟨ i, v_i ⟩ | crSet[i] ∧ $mr_i = v_i$}

let k/K be a fresh private/pubic key pair

in qkr_i.key := k;

mem := qkr_{id}.key-**sign**(**qkr key**: i | K)

Quote(qkr_i, in, out):

if ConfigSat(qkr_i.config)

then out := qkr_i.key-**sign**(**sig:** i | in)

else fail

Note disambiguating prefix is signed strings.

Note K not being stored in key register.

What Configuration?

… is currently in effect for a key register?

KRgetConf(kr_i, r, out):

out := qkr_{id}.key-**sign**(**keyConfig:** i | r | kr_i.config)

… is in effect now?

KRgetCurConf(crSet, r, out):

cc := $\{\langle i, v_i \rangle \mid crSet[i] \wedge mr_i = v_i\}$

out := qkr_{id}.key-**sign**(**curConfig:** r | cc)

By including mr_0 in crSet, the resulting certificate can be included in an immutable data object, thus incorporated into its descriptors. This descriptor avoids replay attacks for old versions of the object.

bind and unbind

Goal: Ensure that information sent from outside a system S can be read only by S.

Solution:

- Distribute public encryption key K_S far and wide.
- Content sent to S is encrypted: K_S-**encrypt**(msg)
- S uses gating function -- where Config is for S --- to recover plaintext

  plain := [k_S-**decrypt**](….)

sealing vs binding

- **seal** and **unseal** both access the same key register on a single machine.
  - **unseal** requires a specific configuration.
- **bind** uses a public key, so it can be executed on any machine.
  - **unbind** requires a specific configuration on a specific machine.

bind and unbind: Instructions

UKRgen(ukr_i, crSet, mem):

ukr_i.config : ={ ⟨ i, v_i ⟩ | crSet[i] ∧ mr_i = v_i}

let k/K be a fresh private/pubic key pair

in ukr_i.key := k;

mem := qkr_{id}.key-**sign**(**ukr key**: i | K)

UKRdec(ukr_i, in, out):

if ConfigSat(ukr_i.config)

then out := ukr_i.key-**decrypt**(in)

else fail

TPM Summary

- measurement registers
  - Configuration constraints Config(·)
- seal/unseal
  - Key archives
- quote
  - Configuration retrieval
- unbind

Applications

- Full disk encryption (BitLocker)
- Cloud-hosted services
- Digital rights management (DRM)
- Remote attestation

Full Disk Encryption

Goal: Protect stored content against device theft.

- Use sealing on each disk block?
  - TPM operations are too slow.

Full Disk Encryption

Goal: Protect stored content against device theft.

- Use sealing on each disk block?
  - TPM operations are too slow.
- Use software-implemented shared key encryption.
  - Generate *disk key* when first boot system.
  - Use seal/unseal to protect *disk key* when stored on disk after power-down.
    - Sealing key stored in key register.
  - Also copy *disk key* to some secure device for disk recovery after failure.
  - Use OS memory protection for *disk key* while computer is running.
    - Assumes memory is obliterated at power down.
  - Must encrypt memory when memory is stored for hibernation mode.
  - Use length-preserving encryption for disk driver compatibility.
    - Protects confidentiality but cannot protect integrity

Full Disk Encryption: Implementation

Where to locate encrypt/decrypt routines for disk blocks?

- In application? (Limits app developers)
- In disk driver? (Limits disk developers)
- In operating system!
  - 1 cache → 2 caches of disk blocks
    - Cache for encrypted blocks (disk driver access this)
    - Cache for decrypted blocks (I/O system calls access this)
    - OS copies from one cache to the other.

Boot block: not encrypted

Cloud-Hosted Services: Servers

Goal: Server is a measured principal.

- sealing key used to protect server state while server is not running
- quoting key allows clients to authenticate responses from server. (Public key must be known).
- binding key used to protect content sent by client to server.

Cloud-Hosted Services: Environment

The environment must support:

- Memory isolation for server.
  - E.g., processor, virtual machine, …
- Measured principals and gating functions
  - E.g., hardware TPM, virtual machine TPM, …

Digital Rights Management (DRM)

Goal: Enforce access control for digital objects located anywhere in the network and on any host.

Enables:

- monetize content in digital form.
  - Non-interactive: Music and texts. Pirate can still record sound and images, though some loss of fidelity.
  - Interactive: Games and simulations.
- mandatory access control of institutional documents.

DRM: Implementation

- Distribute content in encrypted form.
  - Use separate encryption key for each copy.
- Bind decryption key and forward to client
  - Client is a measured principal.
  - Client generates bind/unbind and forwards bind key to server.
  - Server checks client description to ensure authorization will be enforced.
  - Server forwards decrypt key using bind key.

New locus of control

Measured principals and gating functions enable software producers to control:

- What programs are run.
- What information can be accessed.
- What programs can process a given digital object.

… Compare with today: Computer owner and operator have control over these things.

Abuses now facilitated

- [Vendor Lock-in] Software designed to prevent competitors software from executing on a platform.
  - Limits competition
  - Discourages new entrants to market

Abuses now facilitated

- [Vendor Lock-in] Software designed to prevent competitors software from executing on a platform.
  - Limits competition
  - Discourages new entrants to market
- Automation of access control that is today grounded in human judgement.
  - Fair use (for copyright)
  - Obscenity
  - Fake news

In favor …

Benefits of ceding control to software producers:

- Experts can evaluate software and prevent installation of vulnerabilities. Users don't and most can't.
  - App stores can support vendor lock-in, too.
- Protects individual machines but also protects the ecosystem. Compromised machine anywhere can attack yours.

Transfer of **rights** comes with transfer of **responsibilities**. Network-connected implies responsibilities not to host attackers… Should/could random users shoulder that?

Back to authentication of things (= HW + SW) …

Overview

- New abstractions
  - Measured principal
  - Gating function
- Example implementations
  - TPM Trusted Platform Module
- Applications
  - Whole disk encryption
  - Cloud-hosted services
  - Digital rights management
  - Remote Attestation

Remote Attestation

Provide:

- Name P for a measured principal executing on a remote host.
- Attestation public key K_P for verifying messages signed by P.

Given a description D_P obtained from remote host or elsewhere.

- Can check whether $P = N(D_P)$ holds.
- Can use D_P to decide whether to trust P (and K_P).

TOCTOU attacks

If signing key k_P not refreshed at each reboot…

Attack:

- Remote processor sends P, D_P, K_P to client.
- Attacker reboots remote processor and runs new code.

Defense:

- Include mr_0 or current time in D_P.

… Old k_P will no longer work after reboot for software that satisfies D_P or for attacker's software.

Protocol 1 for Remote Attestation

Assumptions:

A1: R trusts S and has K_S **speaksfor** S.

A2: S is exec environment for P.

A3: S implements a gating function [k_P-**sign**].

1. R → S: ⟨r, P⟩, where r is fresh nonce
2. S: Generate K_P/k_p where Config([k_P-**sign**]) = {P}
3. S → R: [k_S-**sign**](r, P, K_P)
4. R: Accept K_P provided:
   - Msg 3 verified as from S (by using K_S) and $N(D_P)=P$ holds.

Discharging Assumptions

Assumption A1: R trusts S and K_S **speaksfor** S.

- R sends S a fresh challenge r
- S uses **quote** and KRgetConf to construct certificate

  k_{id}-**sign**(r, S, D_S, K_S, Config[k_S-**sign**])
- S sends certificate to R
- R checks:
  - Source of cert (using K_{id}) and timeliness (using r).
  - Whether N(D_S) = Config[ks-**sign**]) holds.
  - Uses knowledge of D_S to decide whether to trust S.
  - Concludes: K_S **speaksfor** N(D_S)

    = K_S **speaksfor** S

Discharging Assumptions

Assumption A2: S is exec environment for P.

- Check that D_S is a prefix of D_P.

Assumption A3: S implements a gating function [k_P-**sign**].

- Check D_S to see if processor id appears as initial descriptor.
- Obtain manufacturers certificate

  k_C-**sign**(K_{id} **speaksfor** ISA_{x86})

  and check ISA_{x86}.

Attestation at System Startup

- Startup involves stages D_1, D_2, ... D_n
- Startup Attestation Protocol
  - associates K_i/k_i with each stage $N(D_i)$.
  - generates set AttCerts from which

    K_i **speaksfor** $N(D_i)$

    can be inferred.

Protocol 2 for Remote Attestation

k_0=k_{id}, K_0=K_{id}, N(D_0)=N(hw)=id;

for i:= 0 **to** n-1 **do**

N(D_i) loads software, creating D_{i+1}

N(D_i) generates fresh k_{i+1}/K_{i+1} to support

Config([k_{i+1}-**sign**]) = N(D_{i+1})

AttCerts := AttCerts ∪ {K_i **says** K_{i+1} **speaksfor** N(D_{i+1})}

N(D_i) relinquishes control to N(D_{i+1})

N.b. Trust in N(D_i) must imply N(D_i) will relinquish control to an N(D_{i+1}) that can be trusted.

Avoiding HW support

Goal: Ensure k_i not revealed or abused without using key registers or gating functions.

Solution: D_i deletes k_i just before D_i relinquishes control to D_{i+1}.

Stale AttCerts?

Idea: Incorporate k_{id}-sign(mr0) into AttCerts.

Implementations:

- **Option 1:** Include mr_0 in D_0
- **Option 2**: Include in each D_i a certificate signed by a trusted 3rd party and including timestamp and challenge.

Boot Attestation

- **Trusted boot**: Software establishes trust in its exec environment by checking whether AttCerts contains expected content.
  - Processor register rt reset on boot.
  - rt is updated whenever AttCerts is updated, so

    rt = **hash**(AttCerts).

- **Secure boot**: Processor check successive values of rt against predetermined allowable sequence (in firmware). Halt if values diverge.

General Principle

A layer is responsible for

- measuring and validating the target of a control transfer
- updating a summary measurement

before transferring control.