Computer Security in the Real World

Butler Lampson

Microsoft

August 2005

Real-World Security

It's about risk, locks, and deterrence.

- **Risk** management: cost of security < expected value of loss
  - Perfect security costs way too much
- **Locks** good enough that bad guys don't break in often.
- Bad guys get caught and punished often enough to be **deterred**, so police and courts must be good enough.
- You can **recover** from damage at an acceptable cost.

Internet security is similar, but **little accountability**

- It's hard to identify the bad guys, so can't deter them

Accountability

- Can't identify the bad guys, so can't deter them
- How to fix this? End nodes enforce accountability
  - They refuse messages that aren't accountable enough
    - » or strongly isolate those messages
  - *All trust is local*
- Need an ecosystem for
  - Senders becoming accountable
  - Receivers demanding accountability
  - Third party intermediaries
- To stop DDOS attacks, ISPs must play

How Much Security

Security is expensive—buy only what you need.
- You pay mainly in inconvenience
- If there's no punishment, you pay a lot

People *do* behave this way

We don't *tell* them this—a big mistake

The best is the enemy of the good
- Perfect security is the worst enemy of real security

Feasible security
- Costs less in inconvenience than the value it protects
- Simple enough for users to configure and manage
- Simple enough for vendors to implement

Dangers and Vulnerabilities

Dangers

- Vandalism or sabotage that
  - » damages information — *integrity*
  - » disrupts service — *availability*
- Theft of money — *integrity*
- Theft of information — *secrecy*
- Loss of privacy — *secrecy*

Vulnerabilities

- Bad (buggy or hostile) *programs*
- Bad (careless or hostile) *people*
  giving instructions to good programs

Defensive strategies

Locks: Control the bad guys

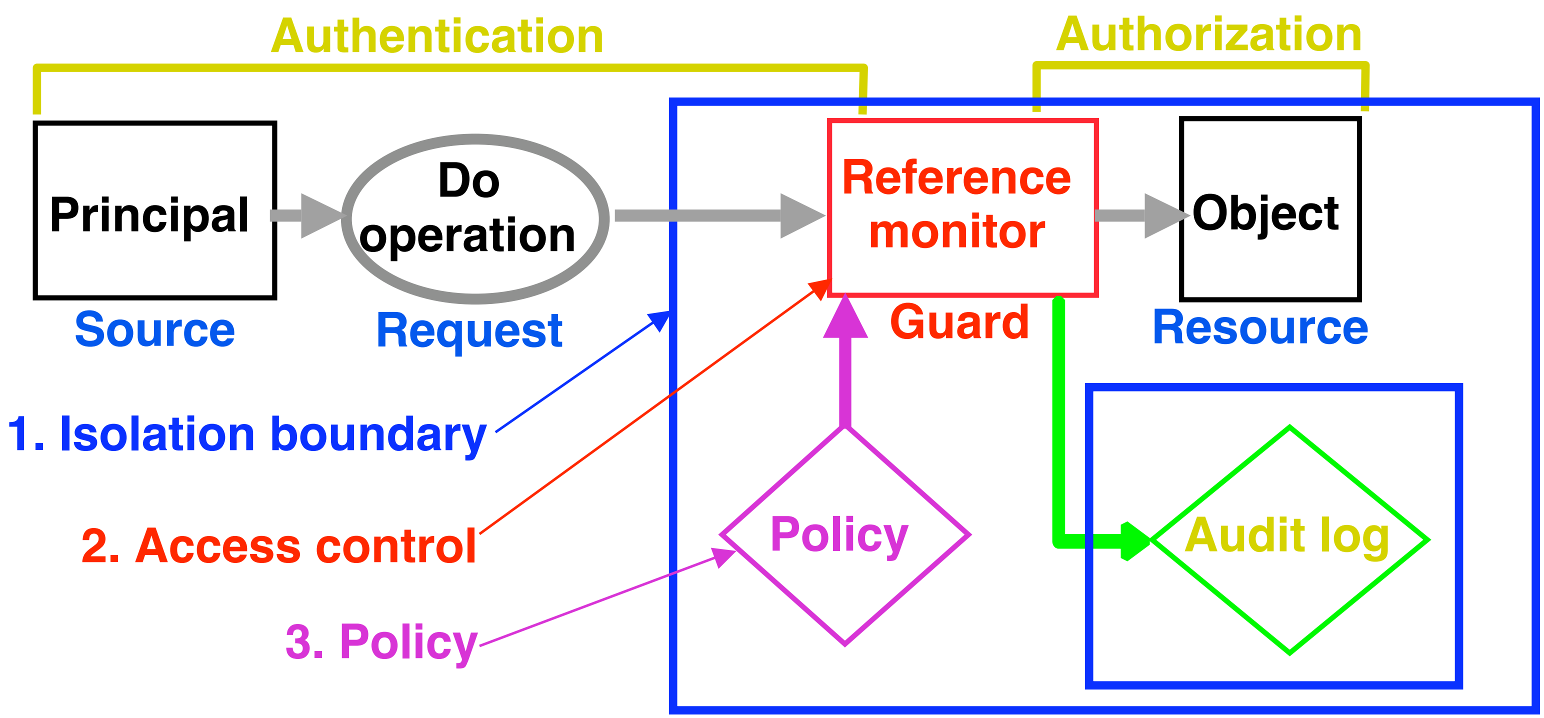
- Coarse: Isolate—keep everybody out
- Medium: Exclude—keep the bad guys out
- Fine: Restrict—Keep them from doing damage
  Recover—Undo the damage

Deterrence: Catch the bad guys and punish them

- Auditing, police, courts or other penalties

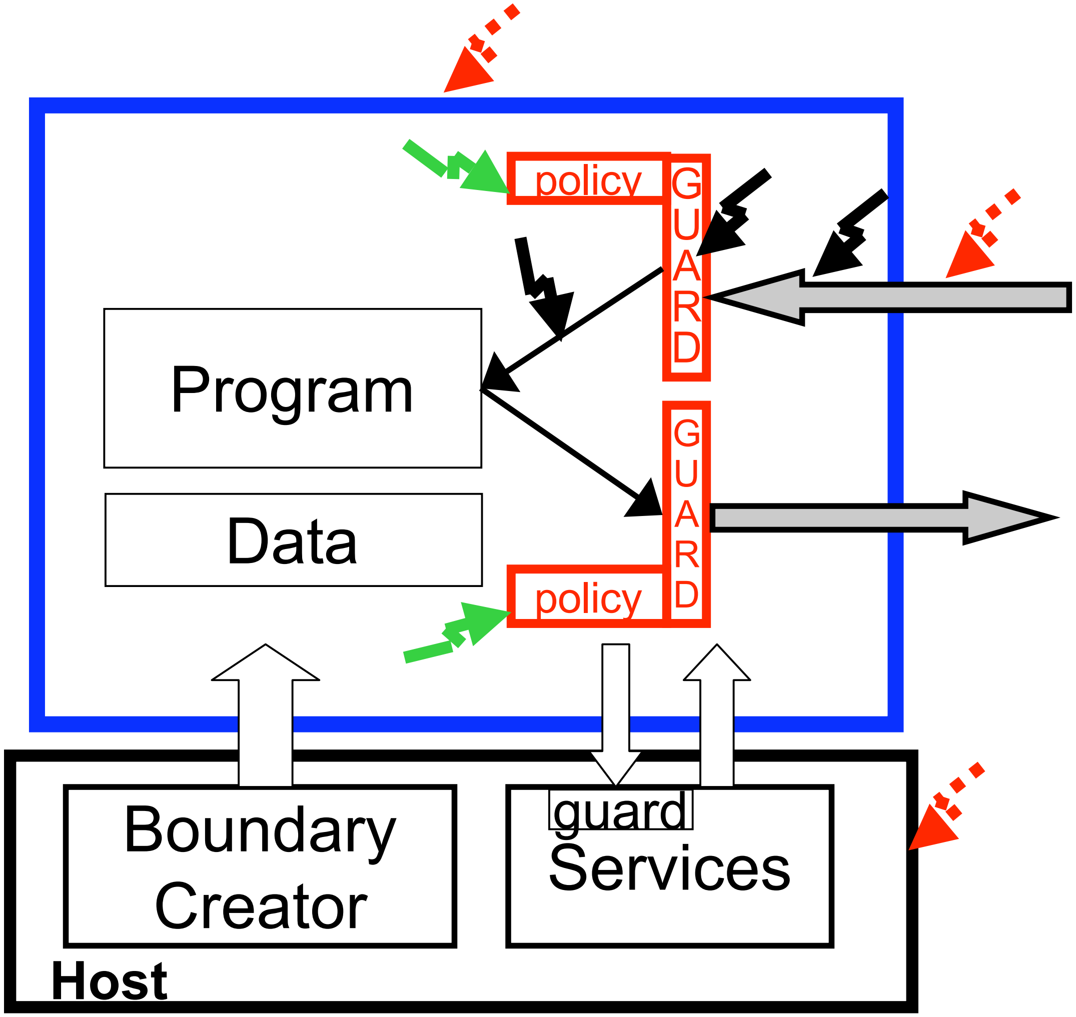
The Access Control Model

1. **Isolation Boundary** to prevent attacks outside access-controlled channels
2. **Access Control** for channel traffic
3. **Policy** management



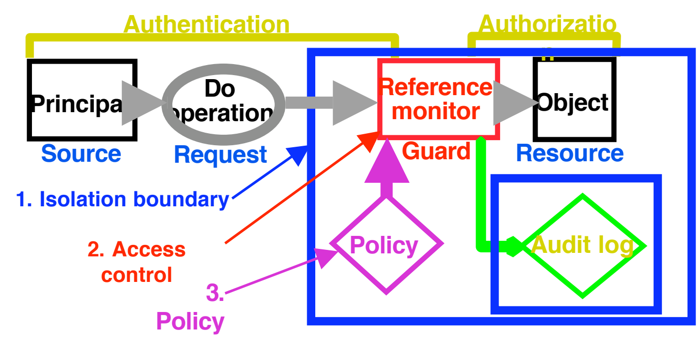
Isolation

I am isolated if whatever goes wrong is my (program's) fault



Attacks on:

- Program
- Isolation
- Policy



Mechanisms—The Gold Standard

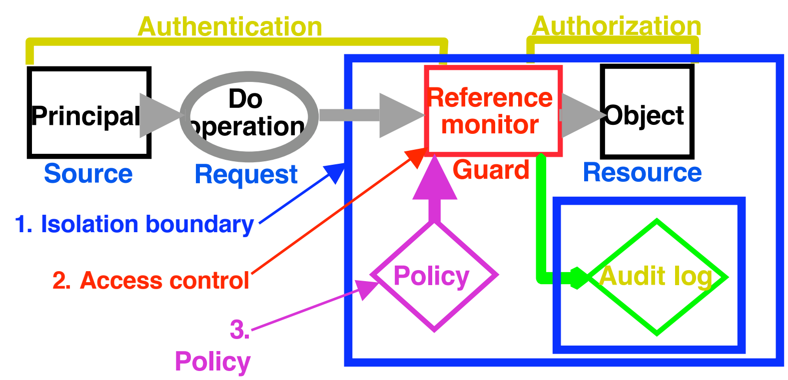
Authenticate principals: Who made a request

- Mainly people, but also channels, servers, programs (encryption implements channels, so key is a principal)

Authorize access: Who is trusted with a resource

- *Group* principals or resources, to simplify management
  - Can be defined by a property, such as "type-safe" or "safe for scripting"

Audit: Who did what when?



Lock = *Authenticate* + *Authorize*

Deter = *Authenticate* + *Audit*

Making Security Work

Assurance

- Does it really work as specified by policy?
- Trusted Computing Base (TCB)
  - Includes everything that security depends on: Hardware, software, and **configuration**

Assessment

- Does formal policy say what I mean?
  - Configuration and management

The unavoidable price of reliability is simplicity.—Hoare

Resiliency: When TCB Isn't Perfect

Mitigation: stop bugs from being tickled

- Block known attacks and attack classes
  - » Anti-virus/spyware, intrusion detection
- Take input only from sources believed good
  - » Red/green; network isolation. Inputs: code, web pages, …

Recovery: better yesterday's data than no data

- Restore from a (hopefully good) recent state

Update: today's bug fix installed today

- Quickly fix the inevitable mistakes
- As fast and automatically as possible
  - » Not just bugs, but broken crypto, compromised keys, …

Why We Don’t Have “Real” Security

A. People don’t buy it:

- Danger is small, so it’s OK to buy features instead.
- Security is expensive.
  - » Configuring security is a lot of work.
  - » Secure systems do less because they’re older.
- Security is a pain.
  - » It stops you from doing things.
  - » Users have to authenticate themselves.

B. Systems are complicated, so they have bugs.

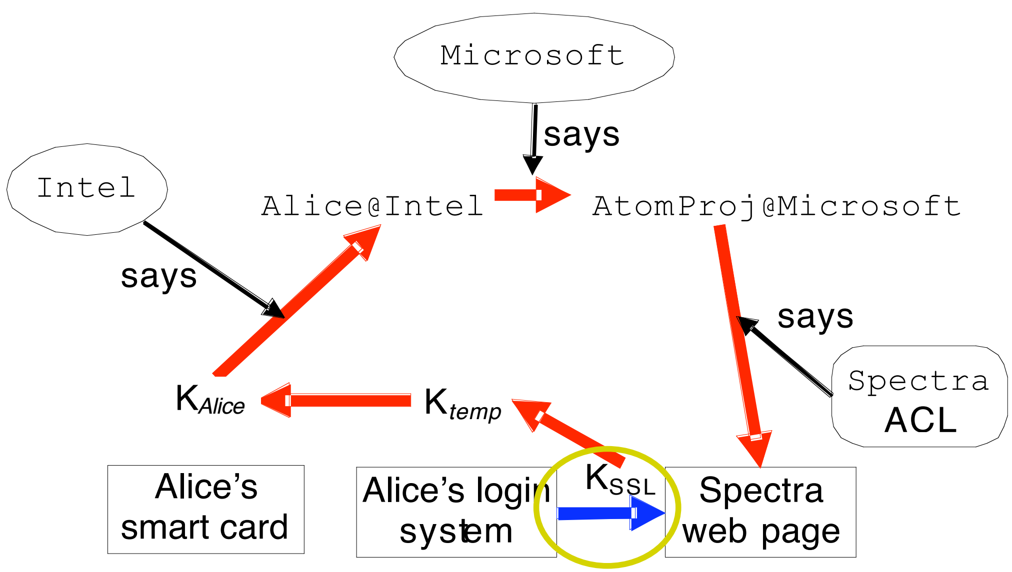
- Especially the configuration

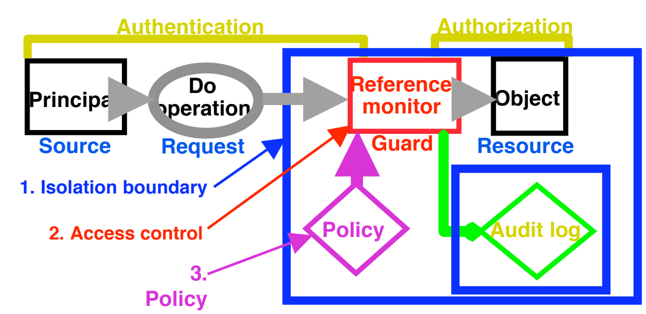
Authentication and Authorization

- `Alice` is at Intel, working on `Atom`, a joint Intel-Microsoft project
- `Alice` connects to `Spectra`, `Atom`'s web page, with SSL

Chain of responsibility:

$K_{SSL} \Rightarrow K_{temp} \Rightarrow K_{Alice} \Rightarrow$ `Alice@Intel` $\Rightarrow$ `Atom@Microsoft` $\Rightarrow_{r/w}$ `Spectra`





Principals

Authentication: Who sent a message?

Authorization: Who is trusted?

Principal — abstraction of "who":

- People `Alice`, `Bob`
- Services `microsoft.com`, `Exchange`
- Groups `UW-CS`, `MS-Employees`
- Secure channels key `#678532E89A7692F`, `console`

Principals say things:

- "Read file `foo`"
- "Alice's key is `#678532E89A7692F`"

Trust: The “Speaks For” Relation

- Principal *A* speaks for *B* about *T*: $A \Rightarrow_T B$
  - Meaning: if *A* says something in set *T*, *B* says it too.
- Thus *A* is **as powerful as** *B*, or **trusted like** *B*, about *T*
- These are the links in the chain of responsibility
  - Examples
    - » `Alice` ⇒ `Atom` *group of people*
    - » `Key #7438` ⇒ `Alice` *key for* `Alice`

Delegating Trust: Evidence

How do we establish a link in the chain?

- A link is a fact $Q \Rightarrow R$. Example: `Key#7438 ⇒ Alice@Intel`

The "verifier" of the link needs **evidence**:

"P **says** $Q \Rightarrow R$". Example: K_{Intel} **says** `Key#7438 ⇒ Alice@Intel`

Three questions about this evidence:

- How do we know that *P* says the delegation?
  - » It comes on a secure channel from *P*, or signed by *P*'s key
- Why do we trust *P* for this delegation?
  - » If *P* speaks for *R*, *P* can delegate this power
- Why is *P* willing to say it?
  - » It depends: *P* needs to know *Q*, *R* and their relationship

Secure Channel

Says things	directly	C **says** s	K_{SSL} **says** read Spectra
Has known	possible receivers	Confidentiality	
	possible senders	Integrity	
If P is the only	possible sender	$C \Rightarrow P$	$K_{Alice} \Rightarrow$ Alice@Intel

Examples

- Within a node — Operating system (pipes, LPC, etc.)
- Between nodes — Secure wire (hard if > 10 feet)
  - IP Address (fantasy for most networks)
  - Cryptography (practical)

Secure channel does **not** mean physical network channel or path

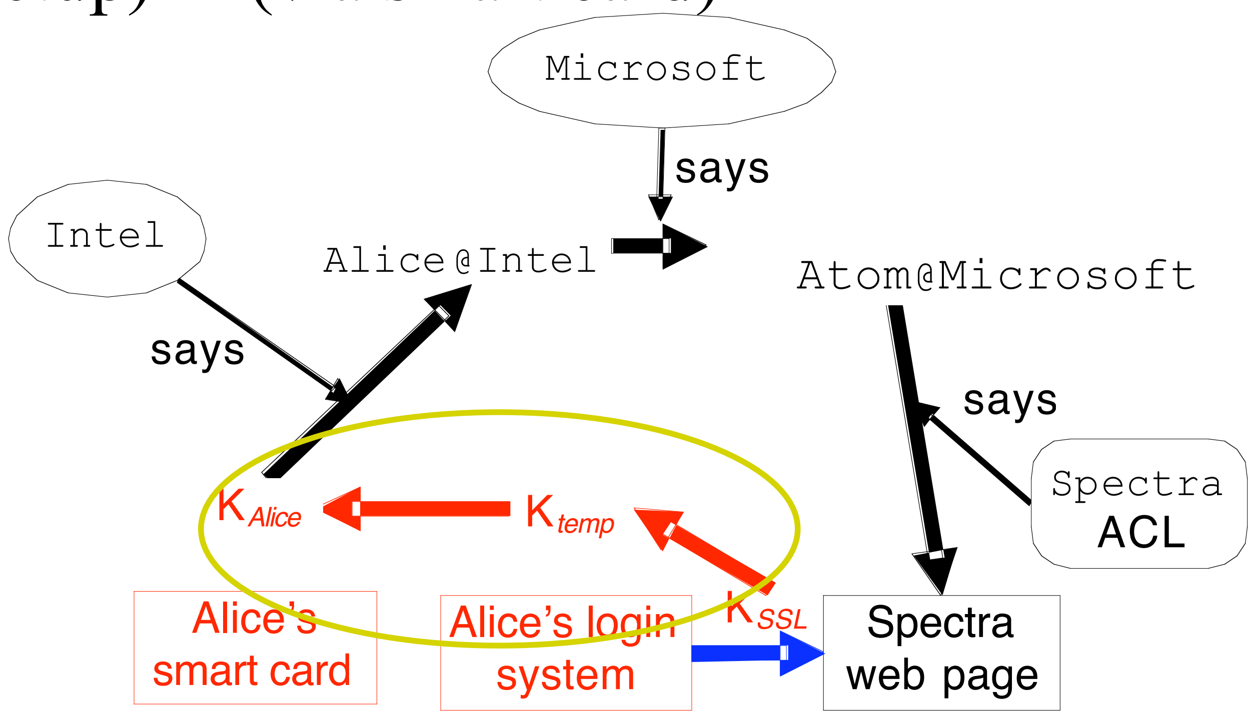
Authenticating Channels

Chain of responsibility:

$K_{SSL} \Rightarrow K_{temp} \Rightarrow K_{Alice} \Rightarrow$ `Alice@Intel` $\Rightarrow \ldots$

K_{temp} **says** (SSL setup)

K_{Alice} **says** (via smart card)



Authenticating Names: SDSI/SPKI

A name is in a name space, defined by a principal *P*

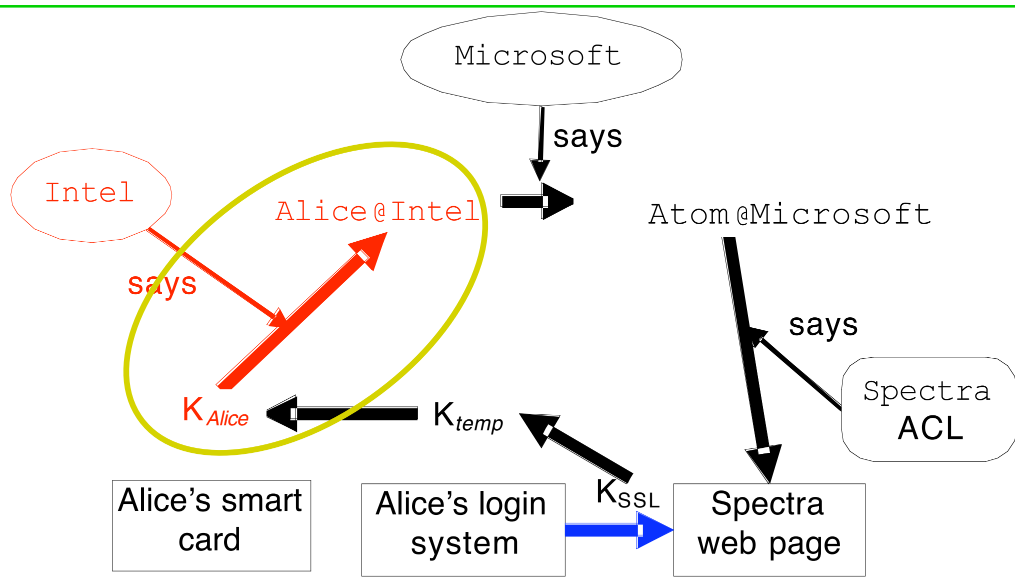
- *P* is like a directory. The root principals are keys.

P speaks for *any* name in its name space

$K_{Intel} \Rightarrow K_{Intel}$ / `Alice` (which is just `Alice@Intel`)

K_{Intel} **says**

… $K_{temp} \Rightarrow K_{Alice} \Rightarrow$ `Alice@Intel` $\Rightarrow$ …



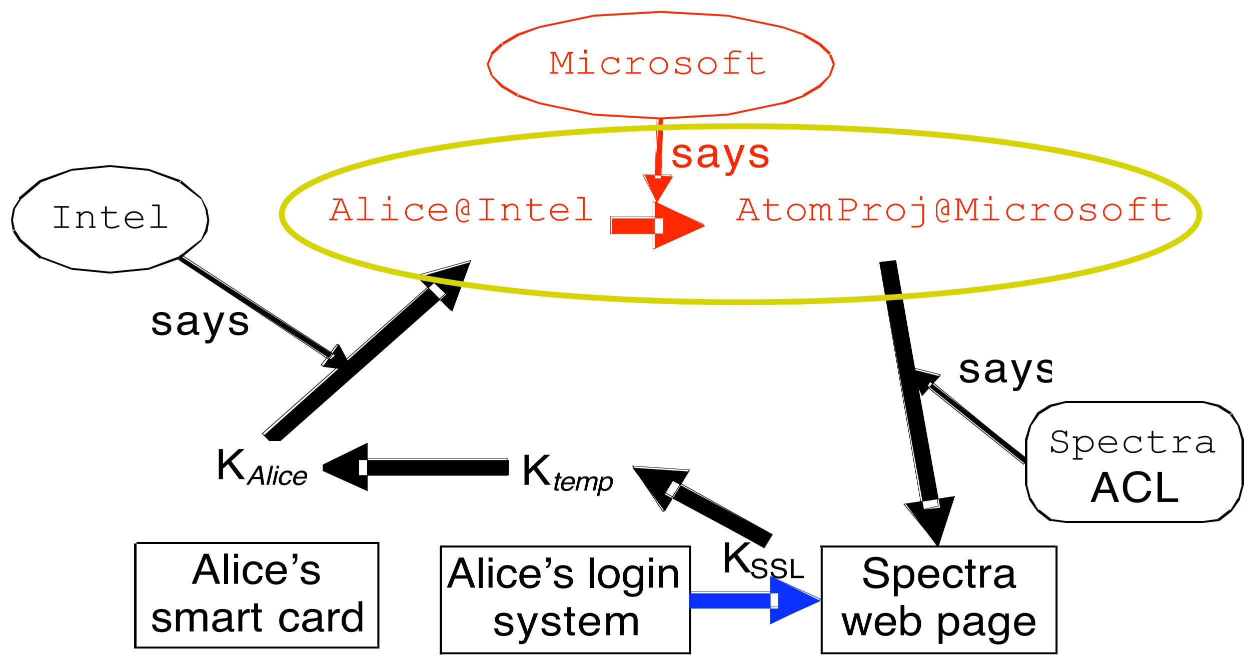
Authenticating Groups

A group is a principal; its members speak for it

- Alice@Intel ⇒ Atom@Microsoft
- Bob@Microsoft ⇒ Atom@Microsoft
- …

Evidence for groups: Just like names and keys.

… K_{Alice} ⇒ Alice@Intel ⇒ Atom@Microsoft $\Rightarrow_{r/w}$ …



Authorization with ACLs

View a resource object *O* as a principal

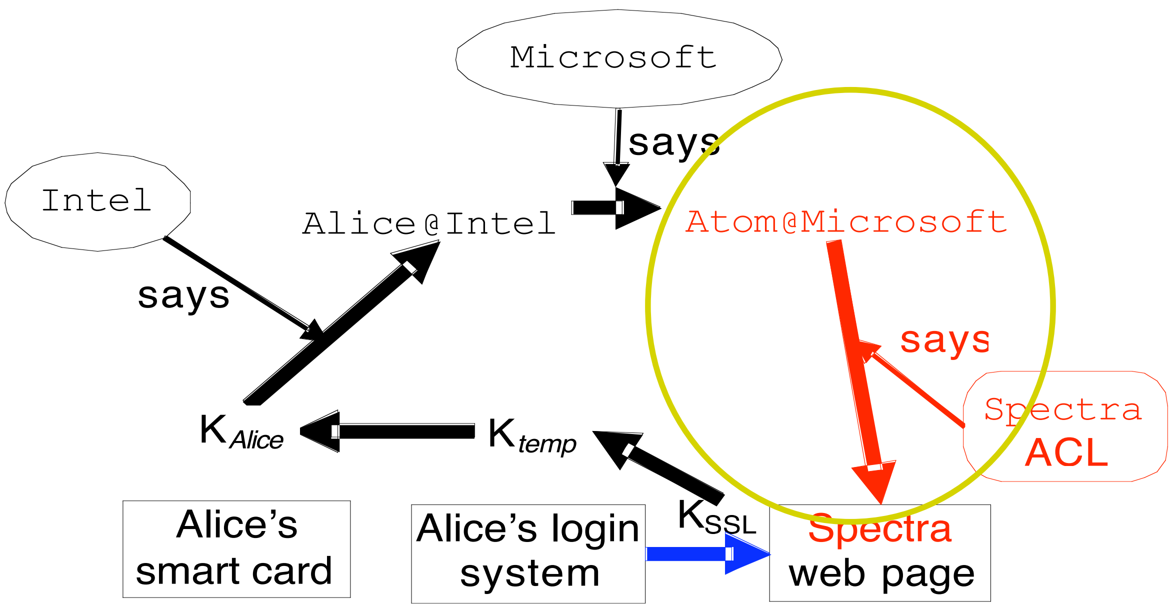
An ACL entry for *P* means *P* can speak for *O*

- Permissions limit the set of things *P* can say for *O*

If `Spectra`'s ACL **says** `Atom` **can** `r/w`, that means

`Spectra` **says**

`... Alice@Intel` $\Rightarrow$ `Atom@Microsoft` $\Rightarrow_{r/w}$ `Spectra`

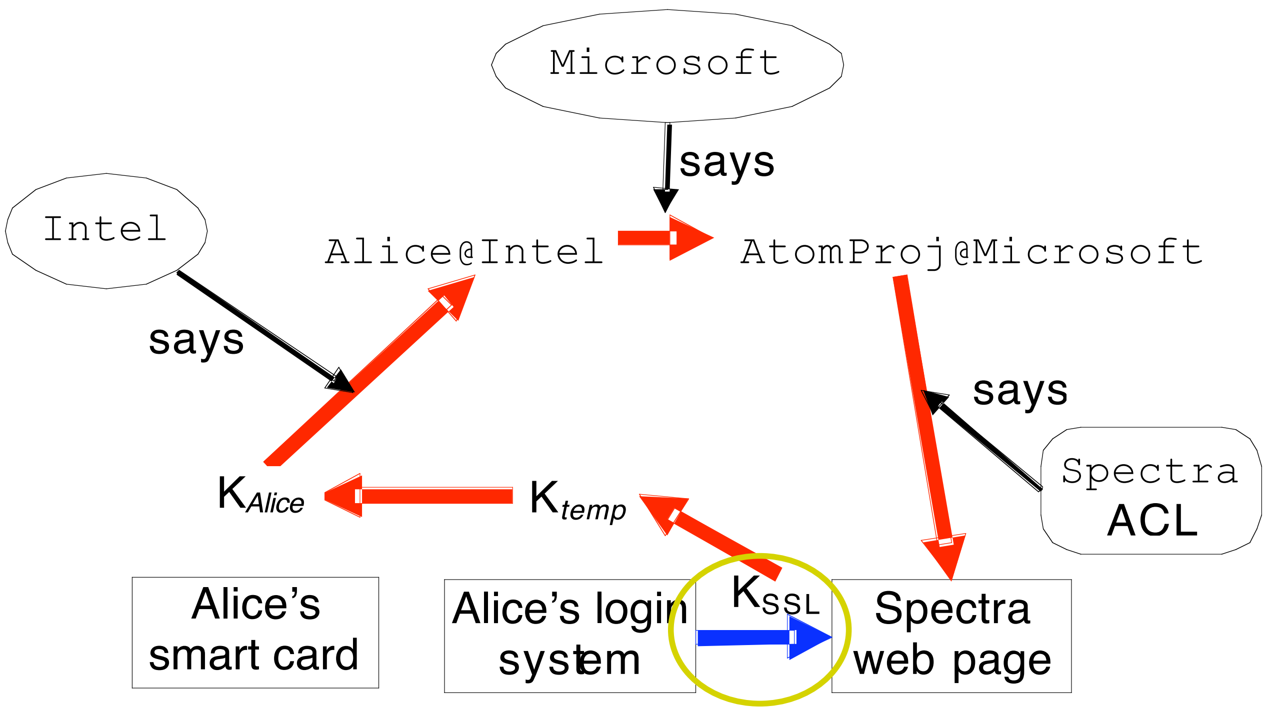


End-to-End Example: Summary

Request on SSL channel: K_{SSL} **says** "read `Spectra`"

Chain of responsibility:

$K_{SSL} \Rightarrow K_{temp} \Rightarrow K_{Alice} \Rightarrow$ `Alice@Intel` $\Rightarrow$ `Atom@Microsoft` $\Rightarrow_{r/w}$ `Spectra`



Authenticating Programs: Loading

Essential for extensibility of security

A digest X can authenticate a program `SQL`:

- $K_{Microsoft}$ **says** "If file I has digest X then I is `SQL`"
- formally $X \Rightarrow K_{microsoft}$/`SQL` like $K_{Alice} \Rightarrow$ `Alice@Intel`

To be a principal, a program must be **loaded**

- By a **host** H into an execution environment
- Examples: booting OS, launching application

$X \Rightarrow$ `SQL` makes H —want to run I if H approves `SQL`

—willing to assert H / `SQL` is running

But H must be trusted to run `SQL`

- $K_{BoeingITG}$ **says** H / `SQL` $\Rightarrow K_{BoeingITG}$/`SQL`

Auditing

Auditing: Each step is logged and justified by

- A statement, stored locally or signed (certificate), or
- A built-in delegation rule

Checking access:

– Given	a request	K_{Alice} **says** "`read Spectra`"
	an ACL	`Atom` **may** `r/w Spectra`
– Check	K_{Alice} speaks for `Atom`	$K_{Alice} \Rightarrow$ `Atom`
	rights suffice	`r/w` $\geq$ `read`

Assurance: NGSCB/TPM

A cheap, convenient, physically separate machine

A high-assurance OS stack (we hope)

A systematic notion of program identity

- Identity = digest of (code image + parameters)
  - » Can abstract this: K_{MS} **says** digest $\Rightarrow K_{MS}$ / `SQL`
- Host certifies the running program's identity:
  H **says** $K \Rightarrow H / P$
- Host grants the program access to sealed data
  - » H seals (data, ACL) with its own secret key
  - » H will unseal for P if P is on the ACL

Learn more

Computer Security in the Real World

at research.microsoft.com/lampson

(slides, paper; earlier papers by Abadi, Lampson, Wobber, Burrows)

Also in IEEE Computer, June 2004

Ross Anderson – www.cl.cam.ac.uk/users/rja14

Bruce Schneier – *Secrets and Lies*

Kevin Mitnick – *The Art of Deception*