Understanding Authentication and Access Control in Distributed Systems

Mike Reiter

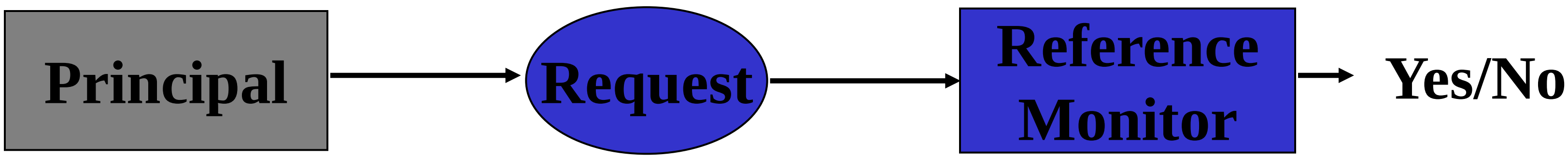
University of North Carolina at Chapel Hill

Background Reading

- B. Lampson, M. Abadi, M. Burrows and T. Wobber. Authentication in distributed systems: Theory and practice. ACM TOCS 10(4), Nov 1992.

Access Control

- Principal makes a request for an object
- Reference monitor grants or denies the request



Ex: Editor — Send file — File server

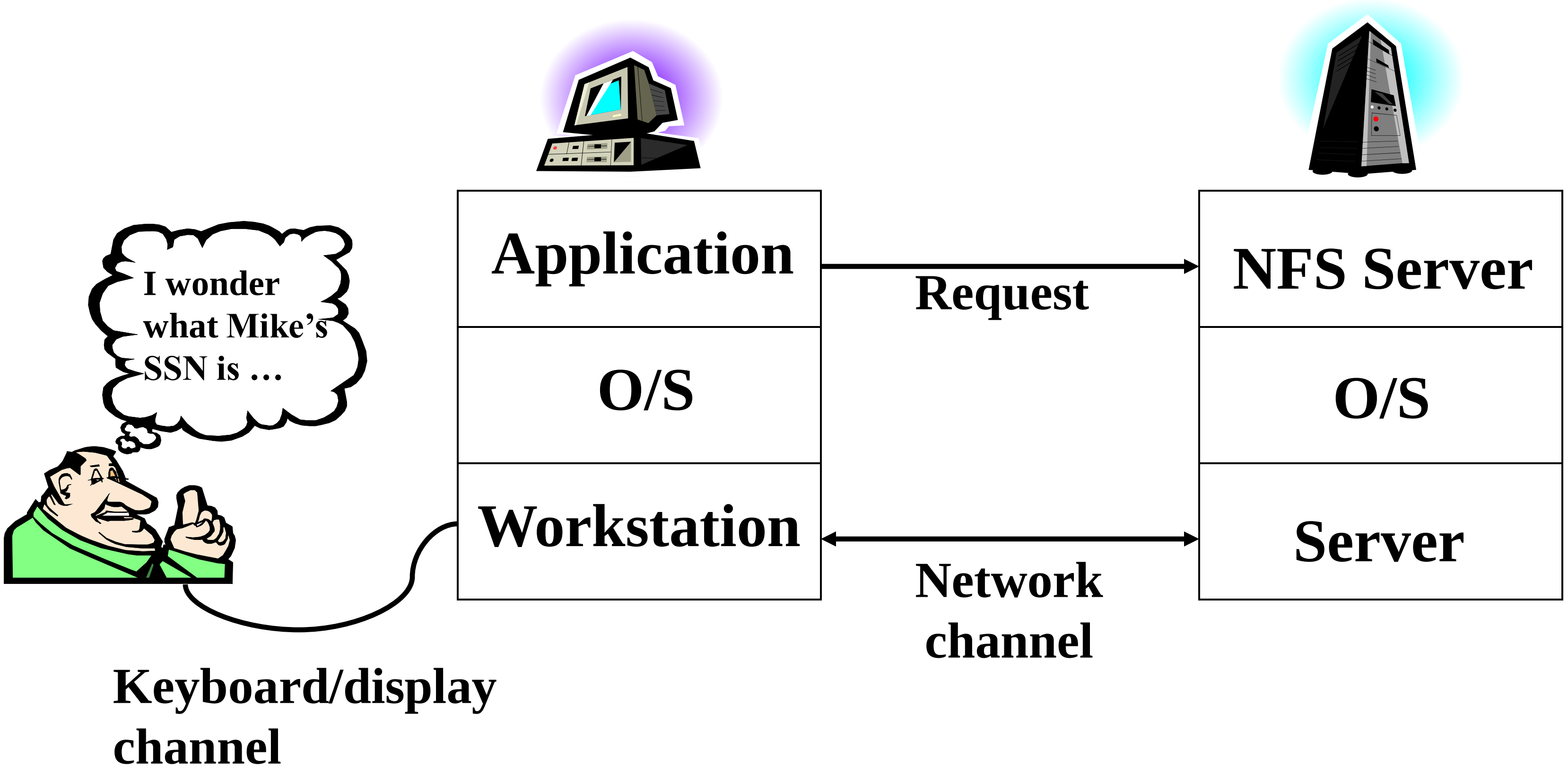
Ex: Host — Forward packet — Firewall

- Authentication: Determining who made request
- Authorization: Determining whether requestor is trusted to access an object
  - The "decision" the reference monitor must make

Authenticating a Channel

- **Each request arrives on some channel, e.g.,**
  - Kernel call from a user process
  - Network connection
  - A channel defined by a cryptographic key
- **Reference monitor must authenticate the channel, i.e., determine whom the request is from**
- **Easy in a centralized system**
  - OS implements all channels and knows the principal responsible for each process
- **Harder in a distributed system**
  - Request may have traversed different, not-equally-trusted machines
  - Different types of channels
  - Some parts of the system may be faulty or broken

The Challenge



- Who is the request "from"?
  - The user? The workstation? The application?
  - All of the above?

Our Approach to Studying the Problem

- **Explain authentication and access control using a logic**
- **The logic forces us to make assumptions explicit and teaches us how to think about access control**

- **Logic helps us to reason about principals and the statements they make**
- **Principals can be**
  - Keys
  - People
  - Machines
  - Principals in roles
  - Groups
  - …

Trusted Computing Base (TCB)

- **Logic will help us identify the "trusted computing base", i.e., the collection of hardware and software that security depends on**
  - Compromise or failure of a TCB element may result in an incorrect "Yes" access-control decision
- **Thus, TCB should be as small as possible**
  - Must be carefully tested, analyzed and protected
- **Benign failure of an untrusted (non-TCB) element may produce more "No" answers, not more "Yes" ones**
  - This is called "fail secure" or "fail safe"
- **Ex: An untrusted server holding a digitally signed credential**
  - Failure prevents credential from being retrieved (more "Nos")
  - Cannot undetectably modify the credential (due to the signature)

The Logic

- The logic is inhabited by
  - Terms that denote principals and strings
  - Formulas that are either “true” or “false”
- Terms:

$$t \ ::= \ s \mid p$$

$$p \ ::= \ \mathrm{key}(s) \mid p.s$$

where s ranges over strings and p over principals

- Formulas:

$$\phi \ ::= \ s \text{ signed } \phi' \ \mid \ p \text{ says } \phi'$$

$$\phi' \ ::= \ \mathrm{action}(s) \ \mid \ p \text{ speaksfor } p \mid \mathrm{delegate}(p, p, s)$$

where s ranges over strings and p over principals

A Logic of Authorization (cont.)

- **Inference rules**

$$\frac{keystr\ \textbf{signed}\ F}{\textbf{key}(keystr)\ \textbf{says}\ F} \qquad \text{(says-I)}$$

$$\frac{A\ \text{says}\ (A.S\ \textbf{says}\ F)}{A.S\ \textbf{says}\ F} \qquad \text{(says-LN)}$$

A Logic of Authorization (cont.)

- **Inference rules**

$$\frac{F}{A \textbf{ says } F} \qquad \text{(says-I2)}$$

$$\frac{A \textbf{ says } (F \rightarrow G) \qquad A \textbf{ says } F}{A \textbf{ says } G} \qquad \text{(impl-E)}$$

A Logic of Authorization (cont.)

- **Inference rules**

$$\frac{A \textbf{ says } (B \textbf{ speaksfor } A) \qquad B \textbf{ says } F}{A \textbf{ says } F} \qquad \text{(speaksfor-E)}$$

$$\frac{A \textbf{ says } (B \textbf{ speaksfor } A.S) \qquad B \textbf{ says } F}{A.S \textbf{ says } F} \qquad \text{(speaksfor-E2)}$$

$$\frac{A \textbf{ says } \textbf{delegates}(A, B, U) \qquad B \textbf{ says } \textbf{action}(U)}{A \textbf{ says } \textbf{action}(U)} \qquad \text{(delegate-E)}$$

Digital Signatures (Informal Definition)

- A digital signature scheme is a triple $<G, S, V>$ of efficiently computable algorithms
  - G outputs a "public key" K and a "private key" K^{-1}

    $$< K, K^{-1}> \leftarrow G(\cdot)$$

  - S takes a "message" m and K^{-1} as input and outputs a "signature" σ

    $$\sigma \leftarrow S_{K^{-1}}(m)$$

  - V takes a message m, signature σ and public key K as input, and outputs a bit b

    $$b \leftarrow V_K(m, \sigma)$$

  - If $\sigma \leftarrow S_{K^{-1}}(m)$ then $V_K(m, \sigma)$ outputs 1 ("valid")
  - Given only K and message/signature pairs $\{<m_i, S_{K^{-1}}(m_i)>\}_i$, it is computationally infeasible to compute $<m, \sigma>$ such that

    $$V_K(m, \sigma) = 1$$

    any new $m \neq m_i$

Cryptographic Keys as Channels

- Let σ be a digital signature on x such that $V_K(x, \sigma) = 1$
- Interpret t or σ as "K signed x" (for respective K)

Authenticating a Channel

- **Reference monitor receives a request *C* says action(*s*)**
- **An access-control list usually specifies named principals**
- **Thus, reference monitor must collect certificates to prove that *A* says action(*s*) for some *A* on the access control list**

- **Two general methods**
  - Push: The sender on the channel *C* collects *A*'s credentials and presents them to authenticate the channel to the receiver.
  - Pull: The receiver looks up *A* in some database to get credentials for *A* when it needs to authenticate the sender.

Certification Authorities

- Credentials typically come from "certification authorities"
- A certification authority is a named principal *CA*
- *CA* issues statements of the form

$$K_{CA} \text{ signed } (\text{key}(K_A) \text{ speaksfor key}(K_{CA}).A)$$

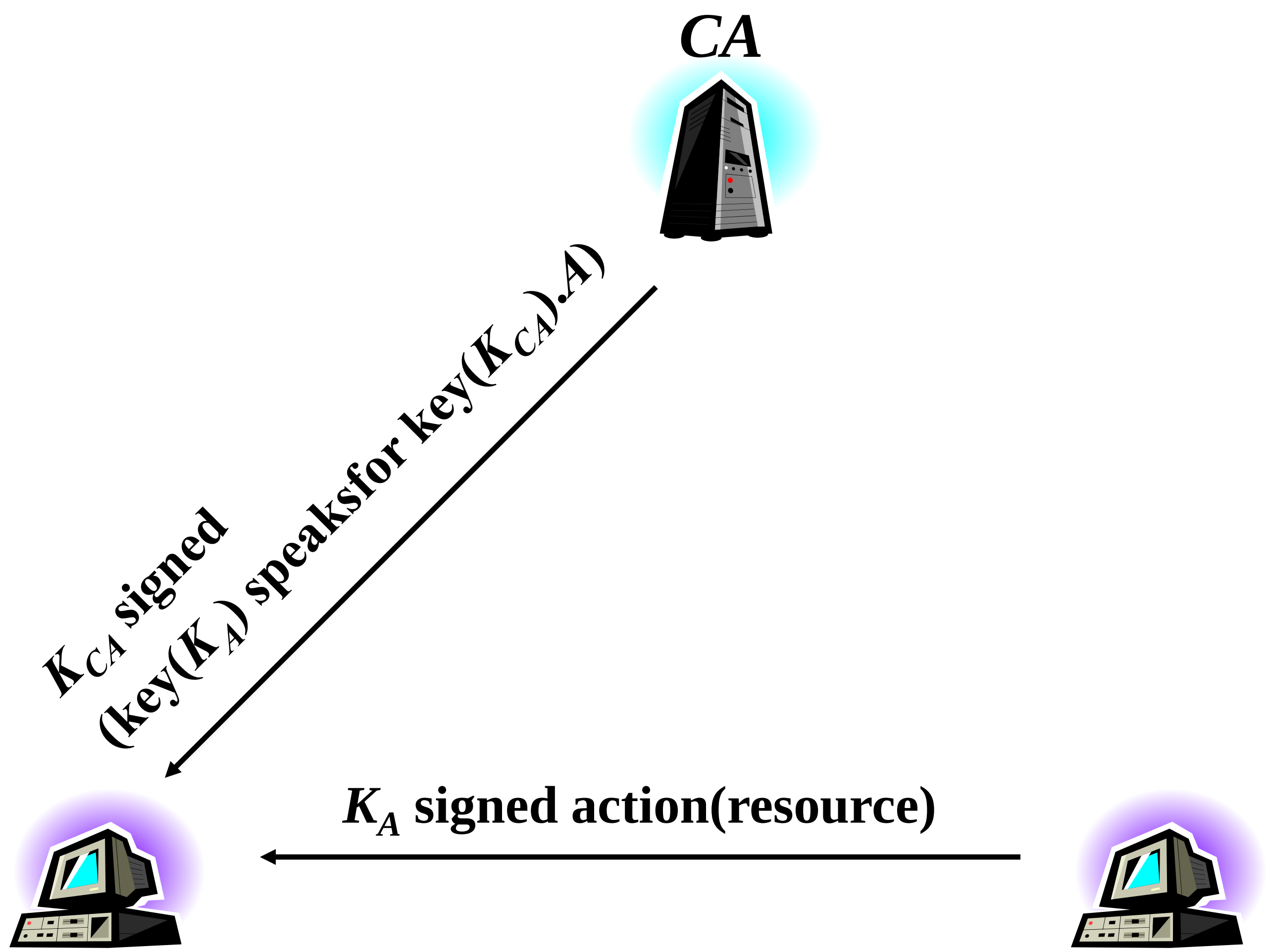
- If K_{CA} is a public key, this statement is called a *certificate*
  - But K_{CA} can be a symmetric key, too

An Example Proof

1. K_{CA} signed (key(K_A) speaksfor key(K_{CA}).A)
2. K_A signed action(resource)

3. key(K_{CA}) says (key(K_A) speaksfor key(K_{CA}).A) says-I(1)
4. key(K_A) says action(resource) says-I(2)
5. key(K_{CA}).A says action(resource) speaksfor-E2(3, 4)

A Certification Authority



Infers key(K_{CA}).A says action(resource)

Groups

- A group is a principal whose members speak for it
- Simplest way to define a group G is for a defining CA to issue certificates

$$\text{key}(K_{CA})\ \textbf{says}\ P_1\ \textbf{speaksfor}\ \text{key}(K_{\text{CA}}).G$$
$$\text{key}(K_{CA})\ \textbf{says}\ P_2\ \textbf{speaksfor}\ \text{key}(K_{\text{CA}}).G$$
$$\ldots$$

for group members P_1, P_2, …

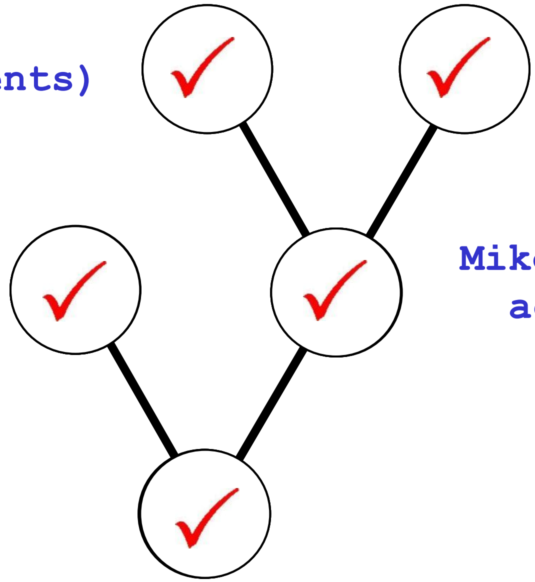
Example Proof

`Mike says (Scott speaksfor Mike.Students)`

`Scott says action(D208)`

`Mike says delegate(Mike, Mike.Students, D208)`

`Mike.Students says action(D208)`



`Mike says action(D208)`

Stored in the reference monitor. Part of the TCB.

Traditional Access Control Lists

Implicitly known to the reference monitor.

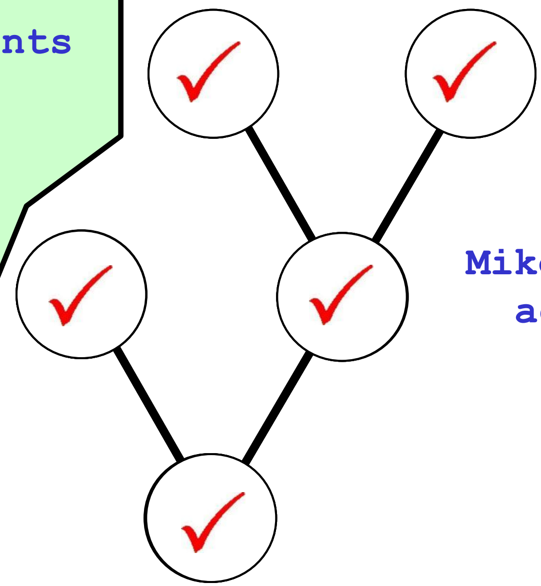
`Scott speaksfor Mike.Students`

`delegate(Mike, Mike.Students, D208)`

Note: not signed

Received in the request.

`Scott says action(D208)`



`Mike.Students says action(D208)`

`Mike says action(D208)`

Stored in the reference monitor. Part of the TCB.

A "Pull" Approach

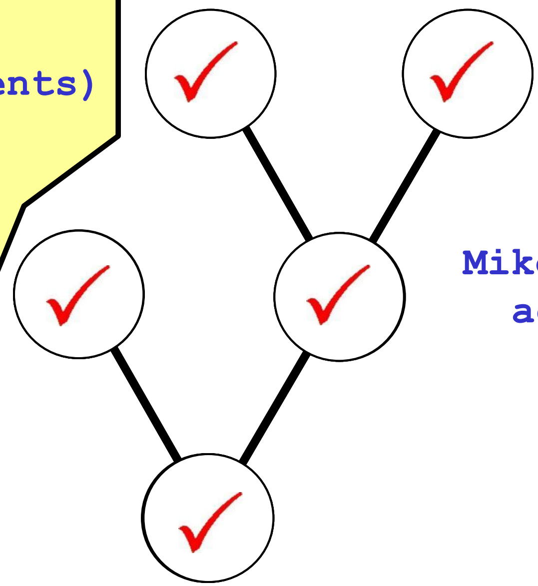
Retrieved by reference monitor.

```
Mike says (Scott
  speaksfor Mike.Students)
```

```
Mike says delegate(Mike,
  Mike.Students, D208)
```

Received in the request.

```
Scott says action(D208)
```

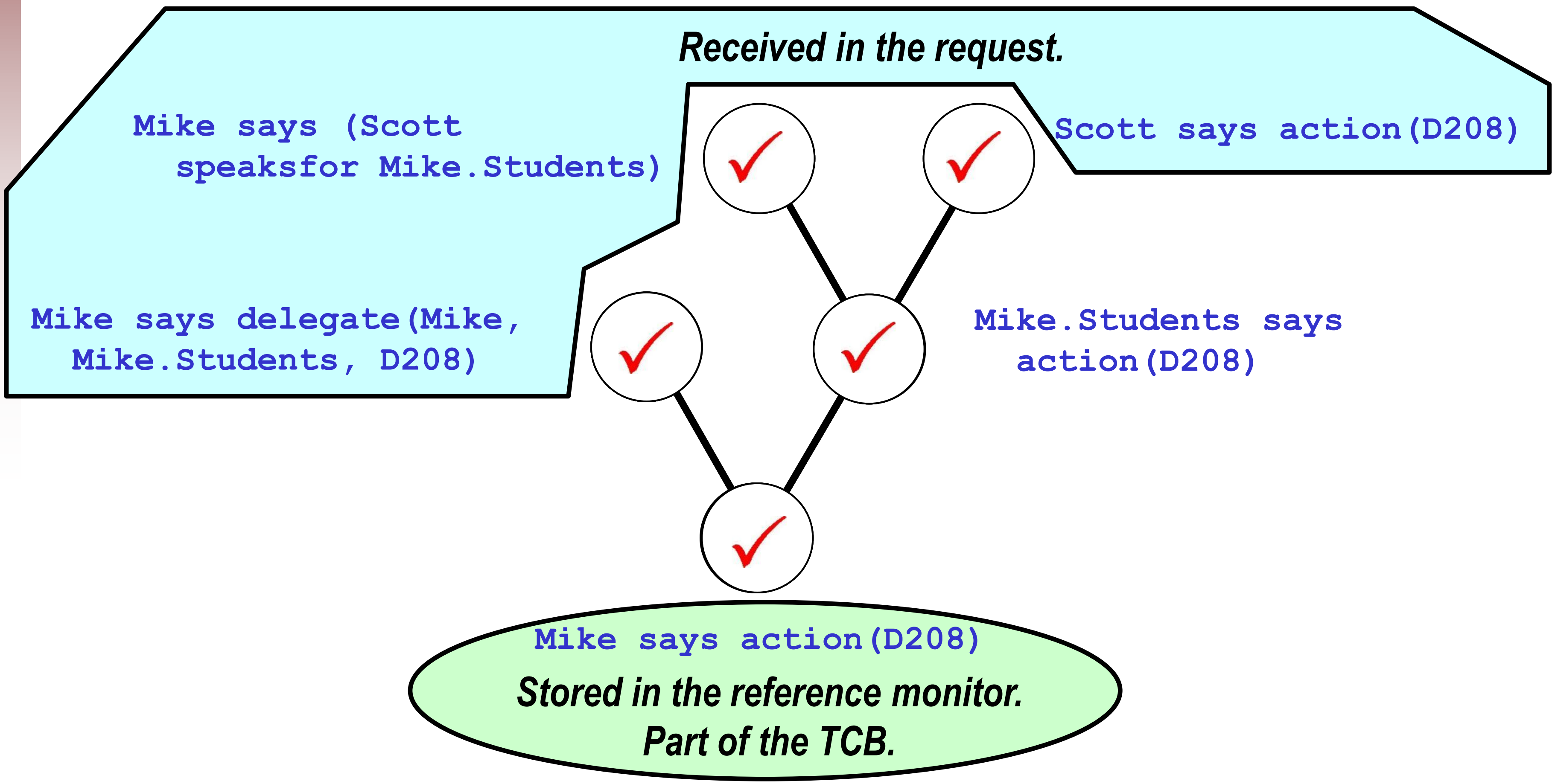


```
Mike.Students says
  action(D208)
```

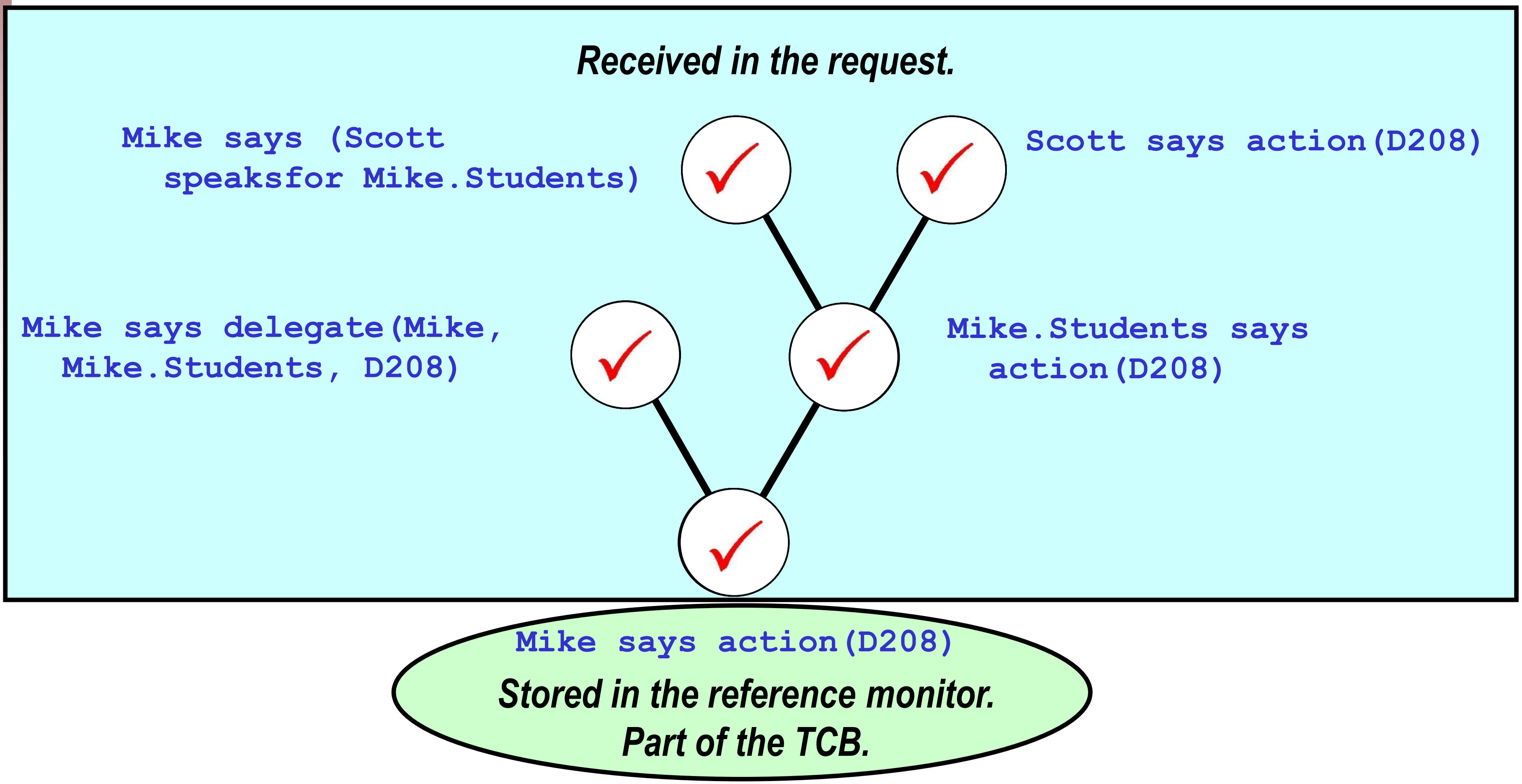
```
Mike says action(D208)
```

Stored in the reference monitor.
Part of the TCB.

A "Push" Approach



A “Proof Carrying” Approach



Roles

- Suppose a principal wants to *limit* its authority
  - Reiter "as" GamePlayer
  - Reiter "as" SysAdmin

- Intuition: A "as" R should be weaker than A
- A can accomplish this by enabling statements of the form

$$A.R \text{ says } F$$

to be created

Programs as an Application of Roles

- Acting in a role is like acting according to some program
- If node N is running program with text I, then N can make

  $$N.I \textbf{ says } F$$

  for a statement F made by the process running I

- Instead of using the whole program I, N can instead make

  $$N.D \textbf{ says } F$$

  where $D = h(I)$ for h a collision-resistant and 2nd preimage resistant hash function, and using

  $$D \textbf{ speaksfor } P$$

  where P is the program name

Loading Programs

- **To load program named *P*, node *N***
  - Creates a process *pr*
  - Reads text *I* of file *P* from the file system
  - Finds credentials for *D* **speaksfor** *P* and checks *h*(*I*) = *D*
  - Copies *I* into *pr*
  - Gives *pr* ability to write to channel *C*
  - Emit: *N* **says** *C* **speaksfor** *N*.*P*
- **Now *pr* can issue requests on channel *C***
  - Will be granted if *N*.*P* is on ACL

Virus Control

- **Some viruses alter texts of programs in the file system**
  - If I' is the infected program text, then $D' = h(I')$ will be different from $D = h(I)$, and so D **speaksfor** P will not apply

- **Certification authority *CA* can issue certificates**

  K_{CA} **signed** P **speaksfor** key(K_{CA}).trustedSW

  K_{CA} **signed** N **speaksfor** key(K_{CA}).trustedNodes

  K_{CA} **signed** (P **speaksfor** key(K_{CA}).trustedSW $\rightarrow$

  (N **speaksfor** key(K_{CA}).trustedNodes $\rightarrow$

  $N.P$ **speaksfor** key(K_{CA}).trustedNodeAndSW))

  where trustedSW, trustedNodes, and trustedNodeAndSW are group names, *P* is a program name, and *N* is a node name

Secure Booting

- **'trustedNodes' should be computers that**
  - run operating systems validated before booting
  - validate other software before loading it
- **Validating O/S during boot is like validating other software**
  - Machine *W* holds *h*(*I*) in boot ROM, where *I* is O/S image
    - i.e., *h*(*I*) **speaksfor** *P*
- **To create a channel *C* such that *C* speaksfor *W.P*, *W* can**
  - Generate a new signature key pair $K_{W.P}, K_{W.P}^{-1}$, and
  - Give $K_{W.P}^{-1}$ to *P*, along with K_W **signed** key($K_{W.P}$) **speaksfor** key(K_W).*P*
- **Private key for K_W must be protected in secure hardware**
  - Otherwise, O/S can read it

Example: TCG

- **Historically, PC manufacturers have chosen flexibility over security**
  - User can modify the PC in any way she likes
  - PC does not have hardware protection for boot procedure, does not validate O/S before loading it, does not validate other programs
- **Today this is changing with efforts like the Trusted Computing Group (TCG; www.trustedcomputinggroup.org)**
  - Alliance formed in Jan 1999 by Compaq, HP, IBM, Intel & Microsoft
  - More than 150 companies by 2002
  - Developing a standard for a "trusted platform" (TP), based on principles similar to those we've discussed
  - Scope of specs is at hardware, O/S and BIOS levels
    - Main spec released in Aug 2000 (v1.0) and Feb 2001 (v1.1)
    - PC-specific spec released in Sep 2001

Example: TCG

- **Some goals of TP**
  - Enable local and remote users to obtain reliable information about the software running on the platform
  - Provide a basis for secure key storage
  - Enable conditional release of secret information to the TP based on the software running
- **TP enabled by a "trusted processing module" (TPM)**
  - A hardware processing component that is isolated from software attacks and at least partially resistant to hardware tampering
- **Each TPM is equipped with a different private key K_{TPM}^{-1} and a certificate**

$$K_{\mathrm{TPME}}\ \textbf{says}\ \mathrm{key}(K_{\mathrm{TPM}})\ \textbf{speaksfor}\ \mathrm{key}(K_{\mathrm{TPME}}).\mathrm{TrustedProcessingModules}$$

signed by a "trusted platform module entity" (TPME)

  - TrustedProcessingModules is a group

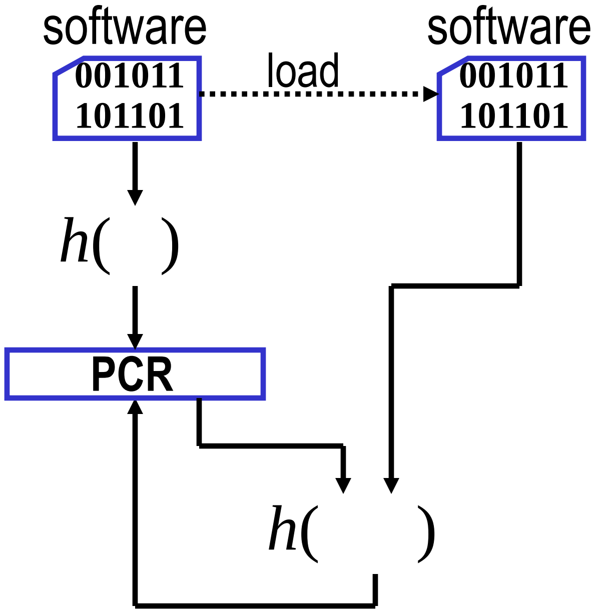
TCG "Roots of Trust"

The standard specifies two logical "roots of trust"

- **<u>Root of trust for measurement (RTM)</u>: A platform-dependent component that starts "measurement" of software running**
  - In a PC, the RTM is the platform itself, which is acceptable only if the RTM cannot be subverted before or during its operation
  - In practice, this means that the RTM must run first (or everything that is run before it is trusted)
    - e.g., BIOS boot block, called the "core root of trust for measurement" (CRTM)
- **<u>Root of trust for reporting (RTR)</u>: A platform-independent component that stores "measurements" as they happen, in such a way that measurements cannot be "undone"**
  - RTR is implemented by the TPM

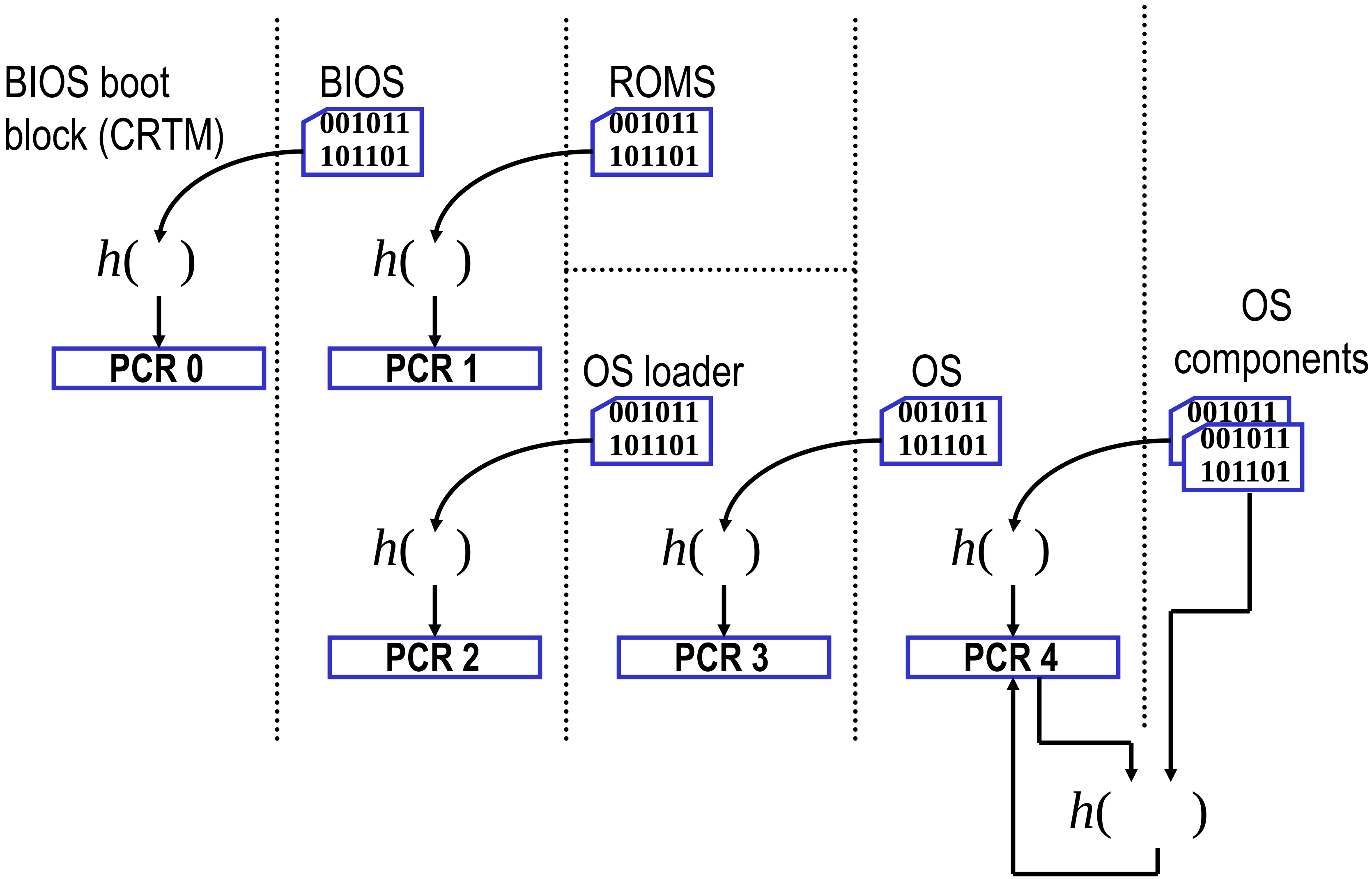
TPM Platform Configuration Registers

- **TPM (version 1.1) contains sixteen 20-byte "platform configuration registers" (PCRs)**
  - 20 bytes in order to store a SHA-1 hash value
- **Each PCR records the last in a sequence of hashes of the software that has been loaded and run**



- **PCR is updated before newly loaded software gets control**
- **PCR cannot be erased except by reboot (or protected processor instruction in v1.2 TPMs)**
- **In this way, PCR contains record of software running**

TCG Authenticated Boot



TCG Secure Boot

- **Non-volatile "data integrity registers" (DIRs) are loaded with expected PCR values**
  - DIRs are contained within TPM and require owner authorization to write

- **If a PCR value, when computed, doesn't match corresponding DIR value, then boot is canceled**

TCG Integrity Challenge and Response

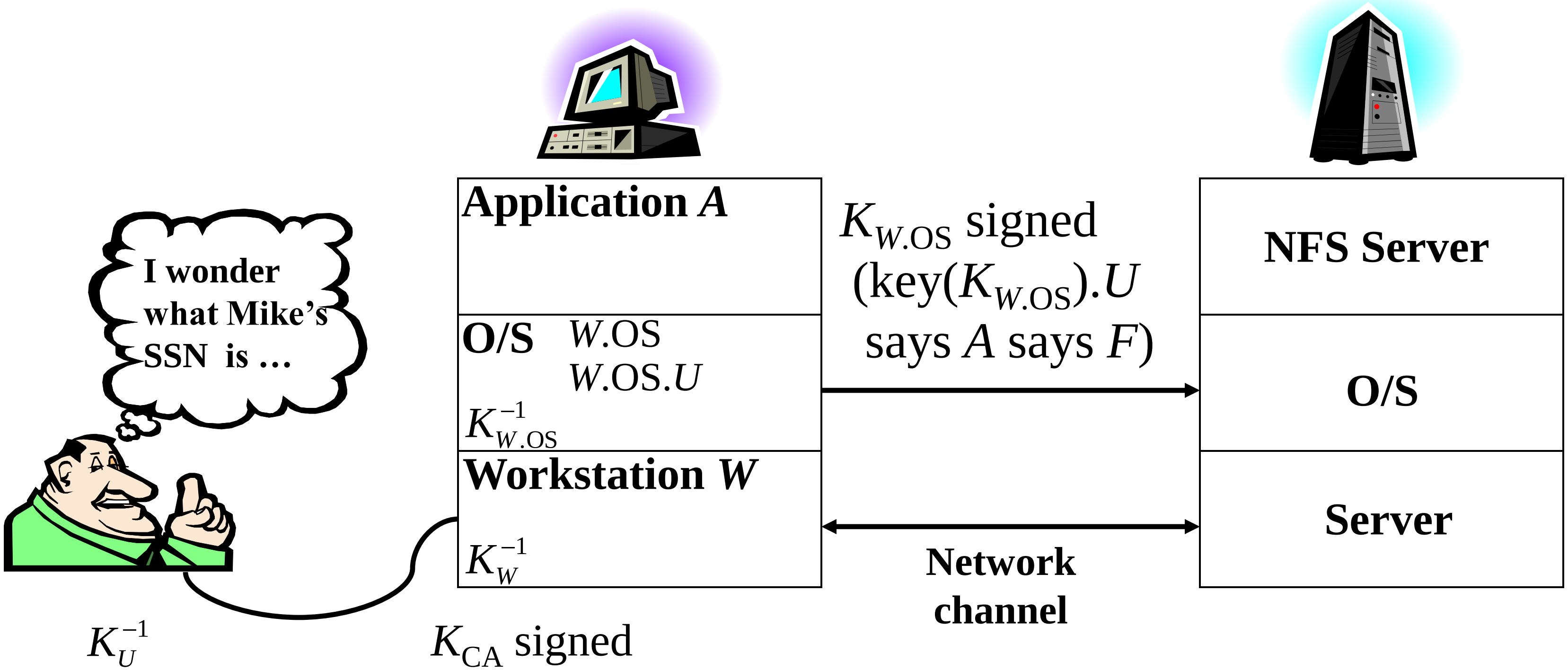
- **Remote machine can query TPM for contents of PCRs**
- **TPM responds with signed PCR values**
  - Think of it as signed with K_{TPM}

    $$K_{TPM} \textbf{ signed } \text{PCRvals} = \ldots$$

  - (In reality, is not signed with K_{TPM} but another "identity key" is used to enhance privacy)

- **TP additionally responds with records (hints) of what is "summarized" in the PCR values**
  - Records could contain software itself, but more likely contains name, supplier, version, and URL for software
  - Enables remote machine to reconstruct and check PCR values
  - Records not trusted and so are stored outside TPM

Example



K_{CA} signed
$\text{key}(K_W)$ speaksfor $\text{key}(K_{\text{CA}}).W$

K_{CA} signed
$\text{key}(K_U)$ speaksfor $\text{key}(K_{\text{CA}}).U$

K_W signed $\text{key}(K_{W.\text{OS}})$ speaksfor $\text{key}(K_{\text{CA}}).W.\text{OS}$
K_U signed $\text{key}(K_{\text{CA}}).W.\text{OS}.U$ speaksfor $\text{key}(K_{\text{CA}}).U$

Example (cont.)

1. K_{CA} signed key(K_W) speaksfor key(K_{CA}).W
2. K_{CA} signed key(K_U) speaksfor key(K_{CA}).U
3. K_W signed key($K_{W.OS}$) speaksfor key(K_{CA}).W.OS
4. K_U signed key(K_{CA}).W.OS.U speaksfor key(K_{CA}).U
5. $K_{W.OS}$ signed (key($K_{W.OS}$).U speaksfor key(K_{CA}).W.OS.U)
6. $K_{W.OS}$ signed (key($K_{W.OS}$).U says A says F)
7. key(K_{CA}) says key(K_W) speaksfor key(K_{CA}).W — says-I(1)
8. key(K_{CA}) says key(K_U) speaksfor key(K_{CA}).U — says-I(2)
9. key(K_W) says key($K_{W.OS}$) speaksfor key(K_{CA}).W.OS — says-I(3)
10. key(K_U) says key(K_{CA}).W.OS.U speaksfor key(K_{CA}).U — says-I(4)
11. key($K_{W.OS}$) says (key($K_{W.OS}$).U speaksfor key(K_{CA}).W.OS.U) — says-I(5)
12. key($K_{W.OS}$) says (key($K_{W.OS}$).U says A says F) — says-I(6)

Example (cont.)

13. key(K_{CA}).W says key($K_{W.OS}$) speaksfor key(K_{CA}).W.OS
    speaksfor-E2(7, 9)
14. key(K_{CA}).U says (key(K_{CA}).W.OS.U speaksfor key(K_{CA}).U)
    speaksfor-E2(8, 10)
15. key(K_{CA}).W.OS says (key($K_{W.OS}$).U speaksfor key(K_{CA}).W.OS.U)
    speaksfor-E2(13, 11)
16. key($K_{W.OS}$).U says A says F — says-LN(12)
17. key(K_{CA}).W.OS.U says A says F — speaksfor-E2(15, 16)
18. key(K_{CA}).U says A says F — speaksfor-E(14, 17)

Example: Web Server Authentication (1)

- **What happens when you access https://www.foo.com?**
- **A protocol called Secure Sockets Layer (SSL) or Transport Layer Security (TLS) is used to authenticate the web server**
  - Also performs other functions that are not important for the moment

HTTP	FTP	SMTP
SSL or TLS		
TCP		
IP		

Example: Web Server Authentication (2)

- **As part of SSL/TLS, web server sends a certificate**

  K_{CA} **signed (key($K_{\text{www.foo.com}}$) speaksfor key(K_{CA}).'www.foo.com')**

  to browser
- **Browser is shipped with public keys for numerous *CA*s:**

  $$K_{CA1} , K_{CA2} , K_{CA3} , \ldots$$

  - Mozilla Firefox 23.0.1 ships with ~200 *CA* keys loaded
  - Reportedly these represent organizations from over 30 countries: AT, BE, BM, CH, CN, CO, DE, DK, EE, ES, EU, FI, FR, GB, GR, HK, HU, IE, IL, IT, JP, NL, NO, PL, RO, SE, SK, TR, TW, US, VE, ZA
- **Should we really trust that** key(K_{CA}).'www.foo.com' **is the "right"** www.foo.com **for all of these CAs?**

Revisiting Trust of *CA*

- **Trusting that for all *CA*s, key(K_{CA}).*A* is the "correct" *A* is too strong**
  - Remember that Firefox comes shipped with ~200 of them!

- **A better approach would reduce this trust**

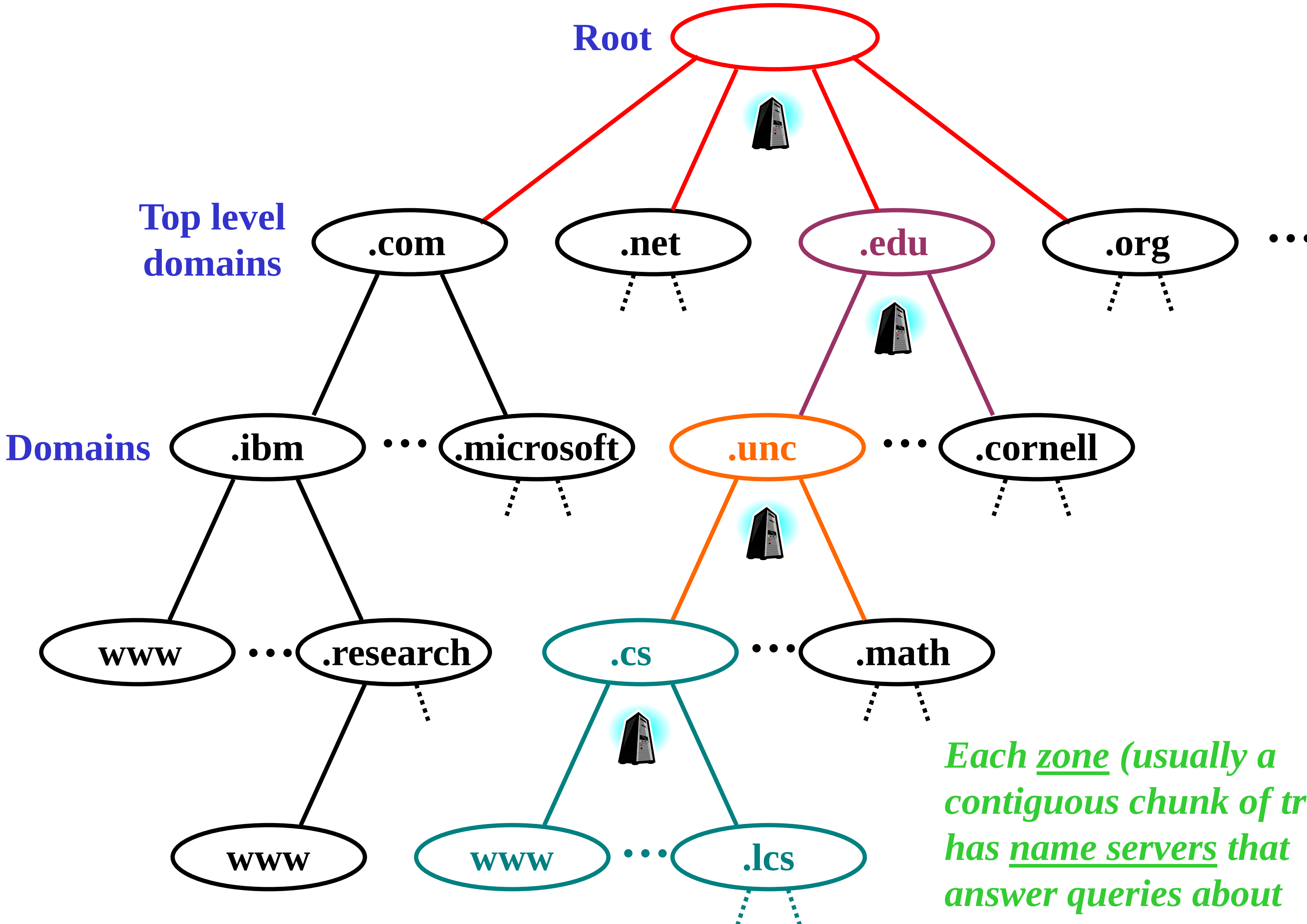
- **If principal names are hierarchical, then this is natural**
  - Many naming schemes are hierarchical, but the most well known one is the Domain Name System ("DNS")

Example: DNS Security

- **DNS translates between human-readable hostnames and IP addresses**
  - Ex: translates www.foo.com to 208.228.229.218
  - Originally specified in RFC 1034 and RFC 1035, and revised by many since

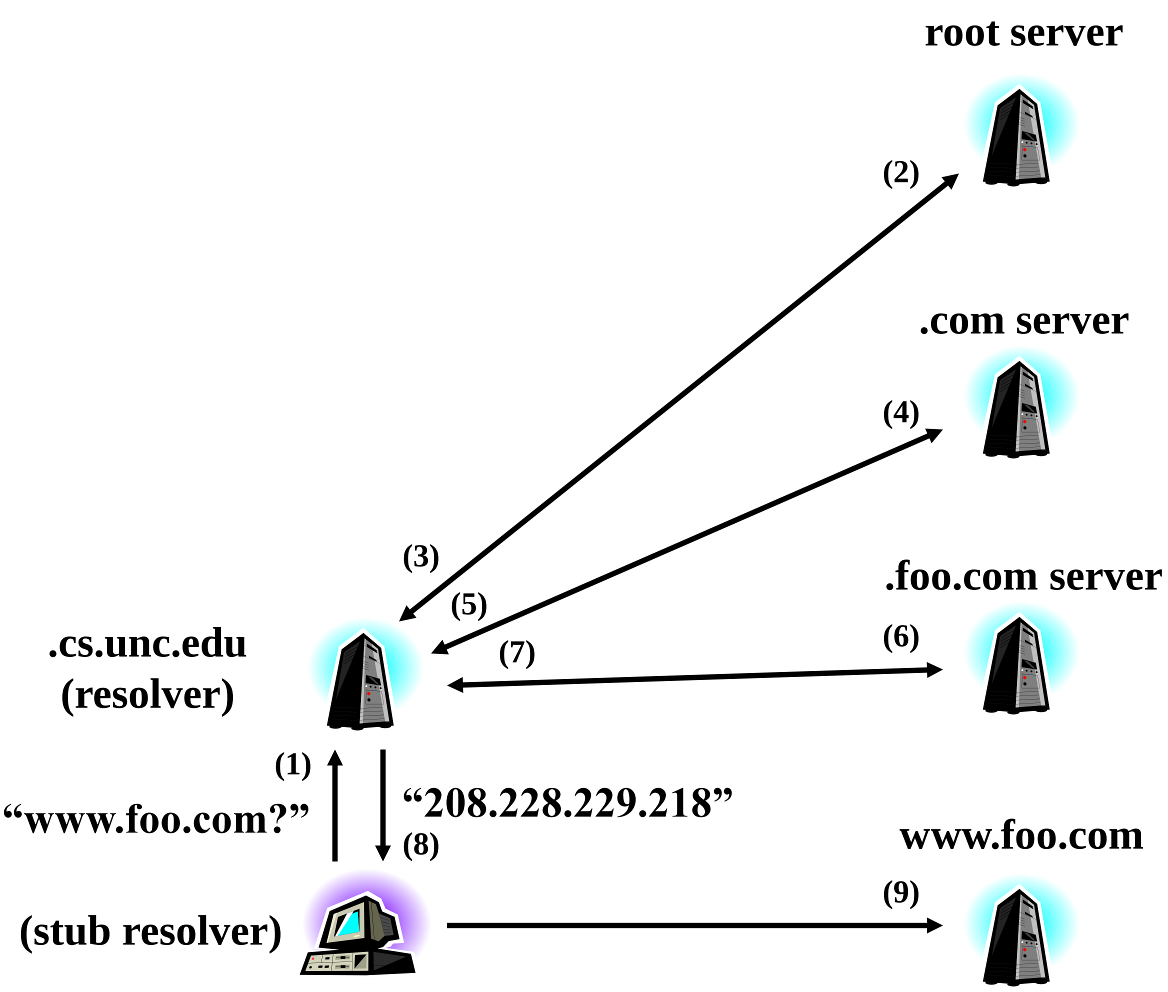
- **DNS Security ("DNSSEC") specifies extensions to DNS to make DNS more secure**
  - "Owned" by the DNSEXT working group in IETF
  - Specified in RFC 2065 (January 1997), revised since

DNS Name Hierarchy



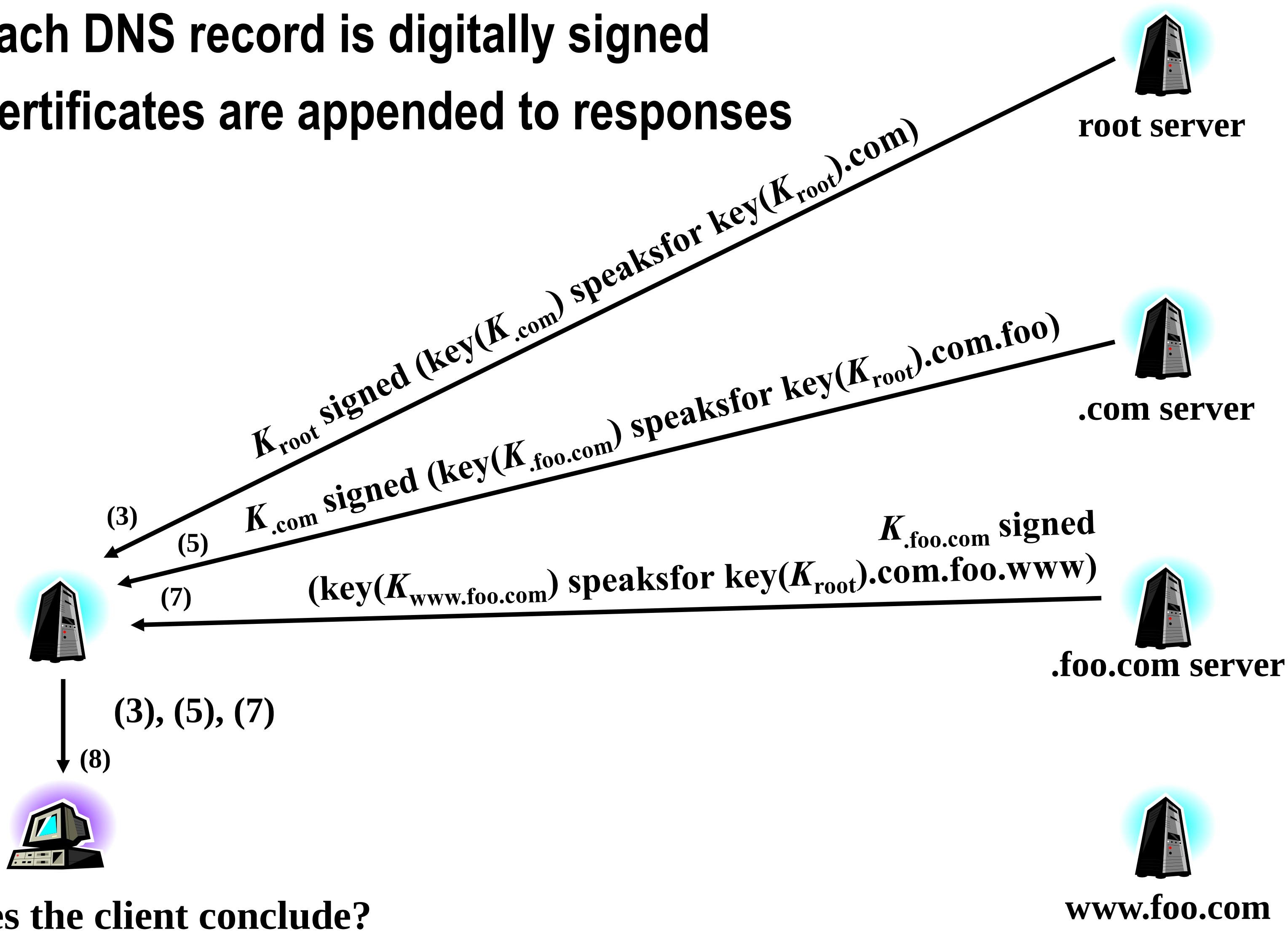
Each __zone__ (usually a contiguous chunk of tree) has __name servers__ that answer queries about names it represents

DNS Name Resolution



DNSSEC

- Each DNS record is digitally signed
- Certificates are appended to responses



What does the client conclude?

Example Proof

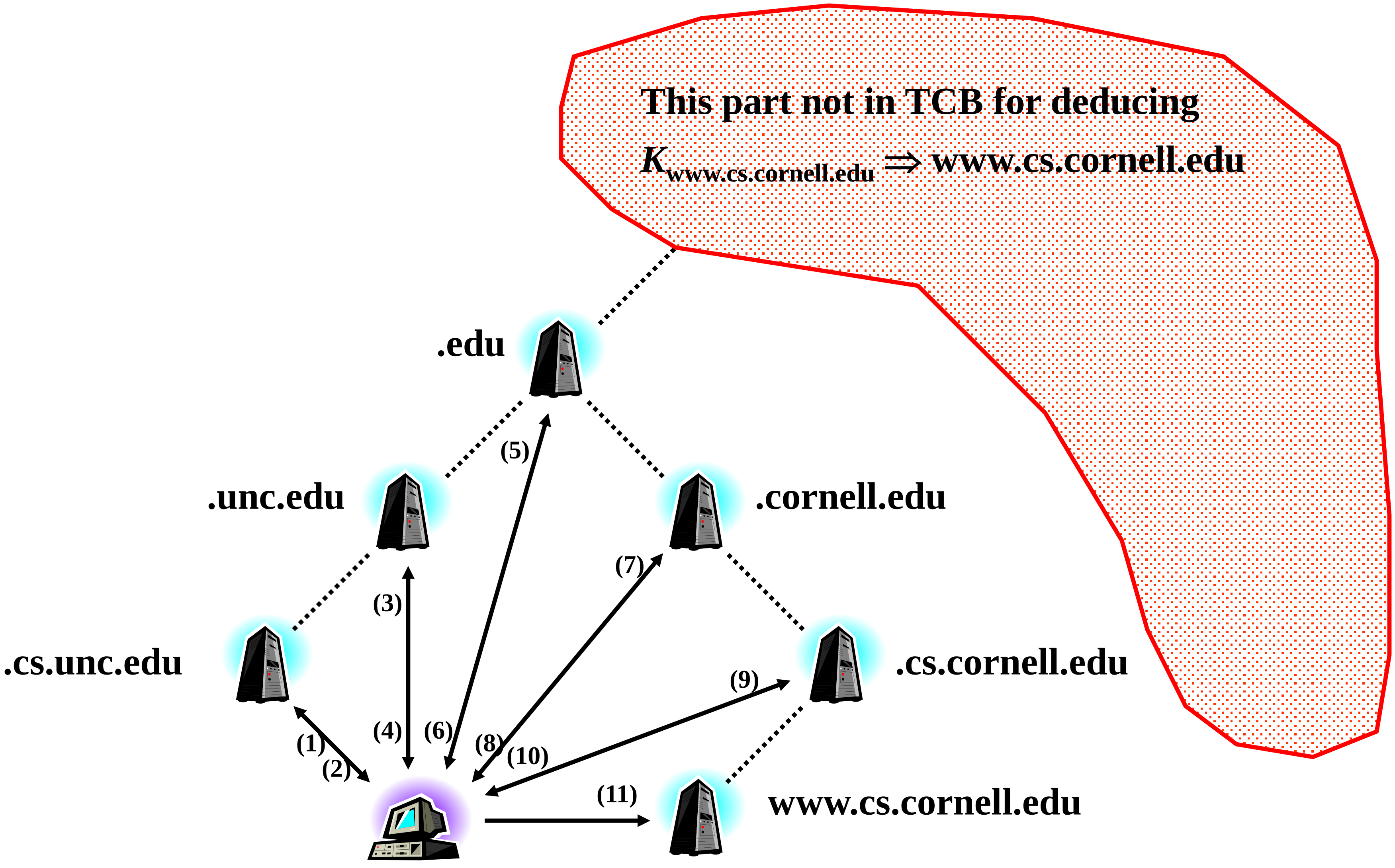
1. K_{root} signed (key($K_{.com}$) speaksfor key(K_{root}).com)
2. $K_{.com}$ signed (key($K_{.foo.com}$) speaksfor key(K_{root}).com.foo)
3. $K_{.foo.com}$ signed (key($K_{www.foo.com}$) speaksfor key(K_{root}).com.foo.www)
4. $K_{www.foo.com}$ signed F
5. key(K_{root}) says (key($K_{.com}$) speaksfor key(K_{root}).com) says-I(1)
6. key($K_{.com}$) says (key($K_{.foo.com}$) speaksfor key(K_{root}).com.foo) says-I(2)
7. key($K_{.foo.com}$) says (key($K_{www.foo.com}$) speaksfor key(K_{root}).com.foo.www) says-I(3)
8. key($K_{www.foo.com}$) says F says-I(4)
9. key(K_{root}).com says (key($K_{.foo.com}$) speaksfor key(K_{root}).com.foo) speaksfor-E2(5, 6)
10. key(K_{root}).com.foo says (key($K_{www.foo.com}$) speaksfor key(K_{root}).com.foo.www) speaksfor-E2(9, 7)
11. key(K_{root}).com.foo.www says F speaksfor-E2(10, 8)

What Went Wrong?

- **We didn't reduce the trust on the root**
  - But that's real life: DNSSEC root is in TCB for every DNS name
- **Is this bad? … The answer depends on your perspective**

- **Optimist: DNS already requires a trusted root, at least DNSSEC is better (but not in this sense)**
- **Pessimist: Could have done better**
  - But probably not without changing how DNS works
  - So, let's try changing how DNS works

Eliminating a Globally Trusted Authority



Conclusion

- **Presented a simple framework for reasoning about authentication and access control in distributed systems**
- **Showed how it can be used to model the propagation of authority in various settings**
  - Web security, secure booting, DNSSEC, …
- **Can also be used to *implement* authentication and access control in systems**
  - L. Bauer, S. Garriss, J. M. McCune, M. K. Reiter, J. Rouse and P. Rutenbar. Device-enabled authorization in the Grey system. *ISC* 2005.
  - L. Bauer, S. Garriss and M. K. Reiter. Efficient proving for practical distributed access-control systems. *ESORICS* 2007.
  - M. L. Mazurek, Y. Liang, W. Melicher, M. Sleeper, L. Bauer, G. R. Ganger, N. Gupta, and M. K. Reiter. Toward strong, usable access control for shared distributed data. In *FAST* 2014.