

Title Goes Here

Understanding Authentication and Access Control in Distributed Systems

Mike Reiter

University of North Carolina at Chapel Hill

Partially based on: Lampson, Abadi, Burrows and Wobber. Authentication in distributed systems: Theory and practice. *ACM TOCS* 10(4), November 1992.

1

1

Access Control

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



Ex: Editor

Send file

File server

Ex: Host

Route packet

Firewall

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

2

2

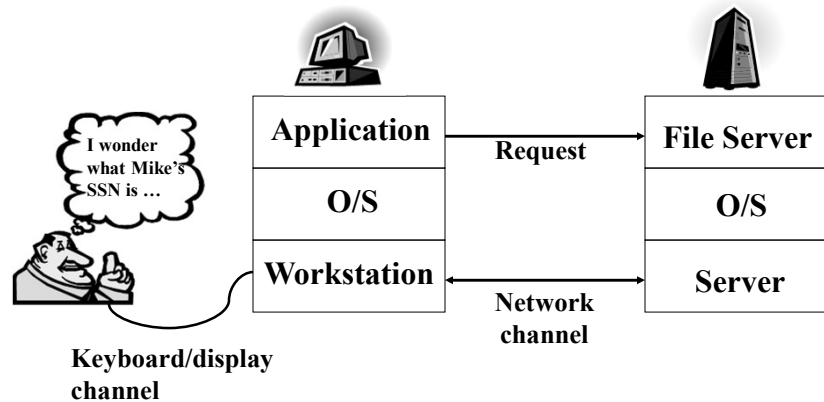
Authenticating a Channel

- 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

3

3

The Challenge



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

4

4

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
 - ▼ ...

5

5

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)

6

6

The Logic

- The logic is inhabited by

- ▮ Terms that denote principals and strings
- ▮ Formulas that are either “true” or “false”

- Terms:

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

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

where s ranges over strings and p over principals

- Formulas:

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

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

where s ranges over strings and p over principals

7

7

A Logic of Authorization (cont.)

- Inference rules

$$\frac{\text{pubkey signed } F}{\text{key(pubkey) says } F} \quad (\text{says-I})$$

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

8

8

A Logic of Authorization (cont.)

■ Inference rules

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

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

9

9

A Logic of Authorization (cont.)

■ Inference rules

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

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

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

10

10

Message Authentication Codes (Informal Defn)

- A message authentication code (MAC) scheme is a triple $\langle G, T, V \rangle$ of efficiently computable functions

- ▼ G outputs a “secret key” K

$$K \leftarrow G(\cdot)$$

- ▼ T takes a key K and “message” m as input, and outputs a “tag” t

$$t \leftarrow T_K(m)$$

- ▼ V takes a message m , tag t and key K as input, and outputs a bit b

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

- ▼ If $t \leftarrow T_K(m)$ then $V_K(m, t)$ outputs 1 (“valid”)

- ▼ Given only message/tag pairs $\{ \langle m_i, T_K(m_i) \rangle \}_i$, it is computationally infeasible to compute $\langle m, t \rangle$ such that

$$V_K(m, t) = 1$$

for any new $m \neq m_i$

11

11

Digital Signatures (Informal Definition)

- A digital signature scheme is a triple $\langle G, S, V \rangle$ of efficiently computable algorithms

- ▼ G outputs a “public key” K and a “private key” K^{-1}

$$\langle K, K^{-1} \rangle \leftarrow G(\cdot)$$

- ▼ 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 $\{ \langle m_i, S_{K^{-1}}(m_i) \rangle \}_i$, it is computationally infeasible to compute $\langle m, \sigma \rangle$ such that

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

any new $m \neq m_i$

12

12

Hash Functions

- A hash function is an efficiently computable function h that maps an input x of arbitrary bit length to an output

$$y \leftarrow h(x)$$

of fixed bit length

- ▼ Preimage resistance: Given only y , it is computationally infeasible to find any x' such that $h(x') = y$.
- ▼ 2nd preimage resistance: Given x , it is computationally infeasible to find any $x' \neq x$ such that $h(x') = h(x)$.
- ▼ Collision resistance: It is computationally infeasible to find any two distinct inputs x, x' such that $h(x) = h(x')$.

13

13

Cryptographic Keys as Channels

- Let t be a MAC tag on message x such that $V_K(x, t) = 1$
- 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)
- Sometimes, public identifiers are needed for keys (channels)
 - ▼ If K is a public key, then $\text{id}(K) = K$
 - ▼ If K is a secret key, then $\text{id}(K) = h(K)$ works if h is a preimage resistant, 2nd preimage resistant, and collision-resistant function
- “ $\text{id}(K)$ signed x ” can be used in place of “ K signed x ” when encoded in a system, if necessary

14

14

Authenticating a Channel

- Reference monitor receives a request C says s
- An access-control list usually specifies named principals
- Thus, reference monitor must collect certificates to prove that C speaks for A 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.

15

15

Certification Authorities

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

K_{CA} signed (key(K_A) speaks for 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

16

16

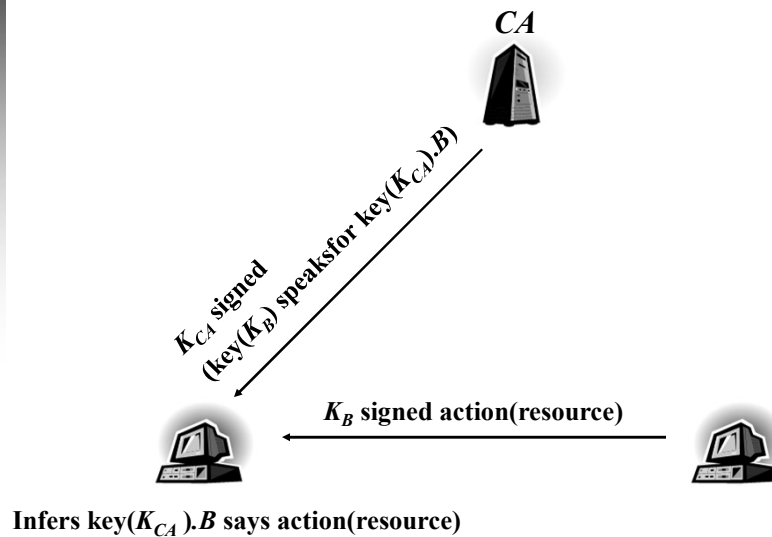
An Example Proof

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)

17

17

A Certification Authority



18

18

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

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

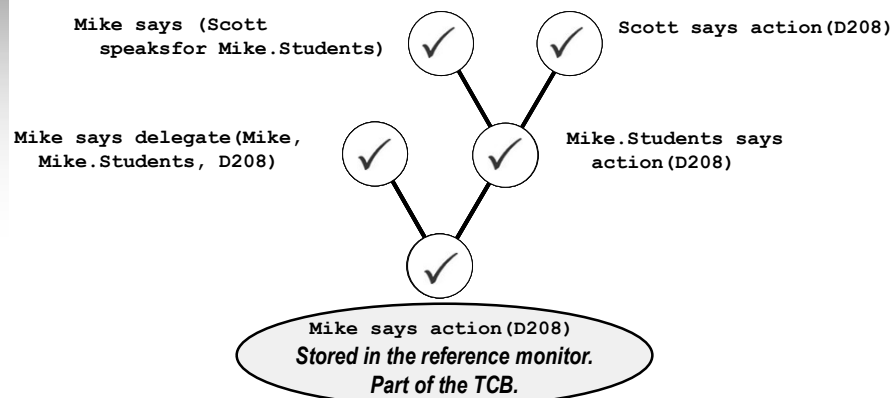
...

for group members P_1, P_2, \dots

19

19

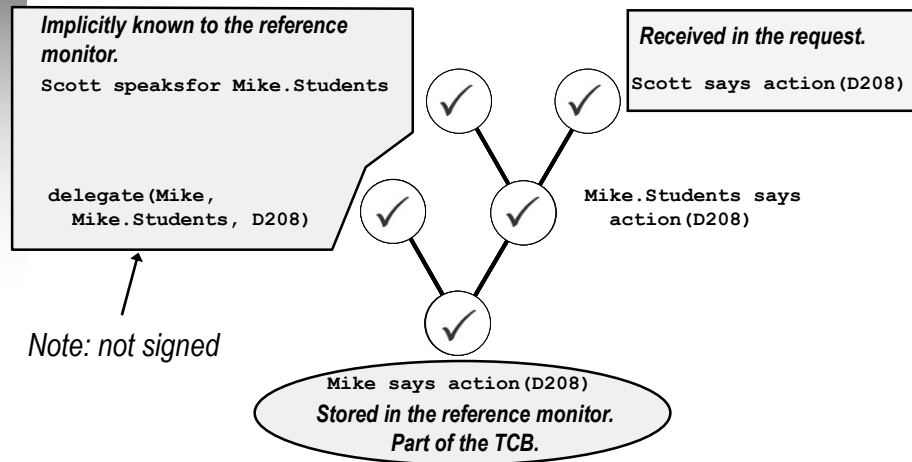
Example Proof



20

20

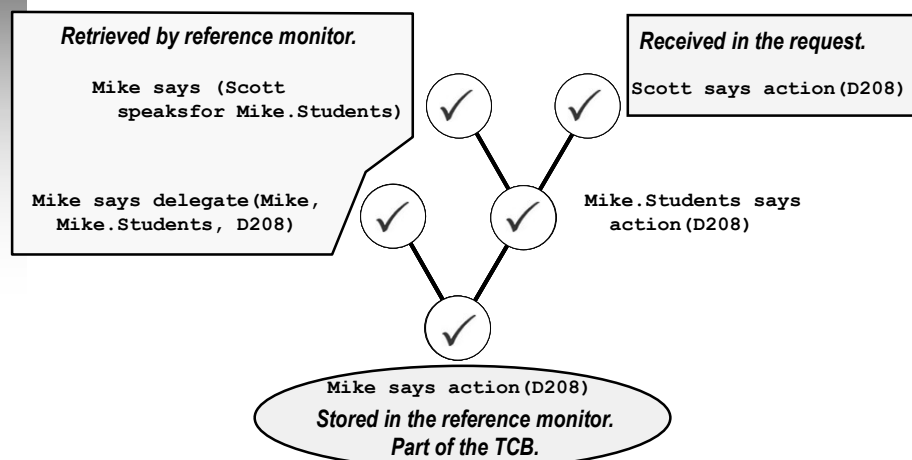
Traditional Access Control Lists



21

21

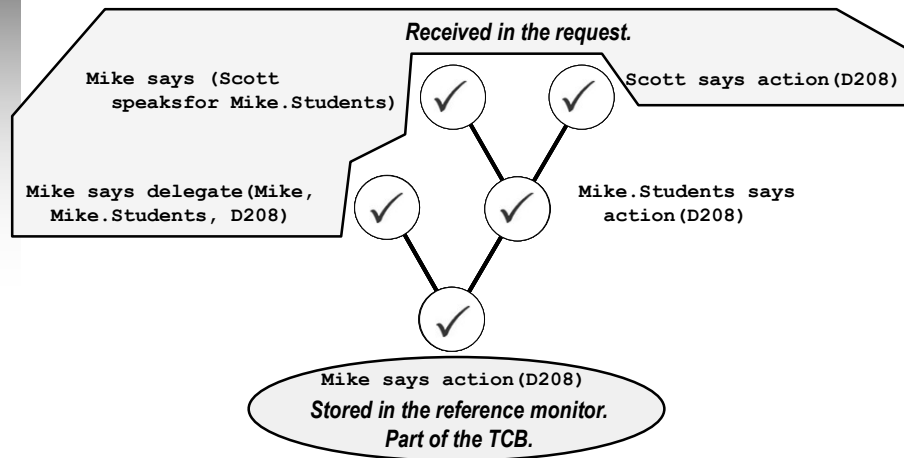
A "Pull" Approach



22

22

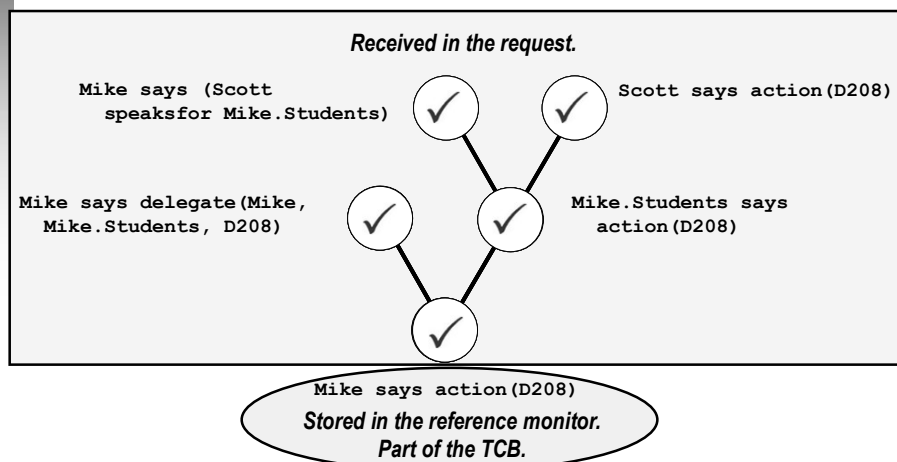
A “Push” Approach



23

23

A “Proof Carrying” Approach



24

24

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

$A.R$ says F

to be created

25

25

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
$$N.I \text{ says } F$$
for a statement F made by the process running I
- Instead of using the whole program I , N can instead make
$$N.D \text{ says } F$$
where $D = h(I)$ for h a collision-resistant and 2nd preimage resistant hash function, and using
$$D \text{ speaksfor } P$$
where P is the program name

26

26

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

27

27

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** $\text{key}(K_{CA}).\text{trustedSW}$
 - K_{CA} **signed** N **speaksfor** $\text{key}(K_{CA}).\text{trustedNodes}$
 - K_{CA} **signed** (P **speaksfor** $\text{key}(K_{CA}).\text{trustedSW} \wedge$
 N **speaksfor** $\text{key}(K_{CA}).\text{trustedNodes} \rightarrow$
 $N.P$ **speaksfor** $\text{key}(K_{CA}).\text{trustedNode}.\text{trustedSW}$)

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

28

28

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

29

29

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
- **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

30

30

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_{TPME} says $\text{key}(K_{\text{TPM}})$ speaks for $\text{key}(K_{\text{TPME}})$. TrustedProcessingModules
signed by a “trusted platform module entity” (TPME)

- ▼ TrustedProcessingModules is a group

31

31

TCG “Roots of Trust”

TCG specifies two logical “roots of trust”

■ Root of trust for measurement (RTM): 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)

■ Root of trust for reporting (RTR): A platform-independent component that stores “measurements” as they happen, in such a way that measurements cannot be “undone”

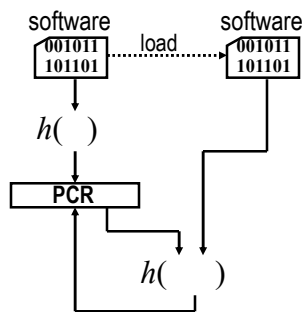
- ▼ RTR is implemented by the TPM

32

32

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

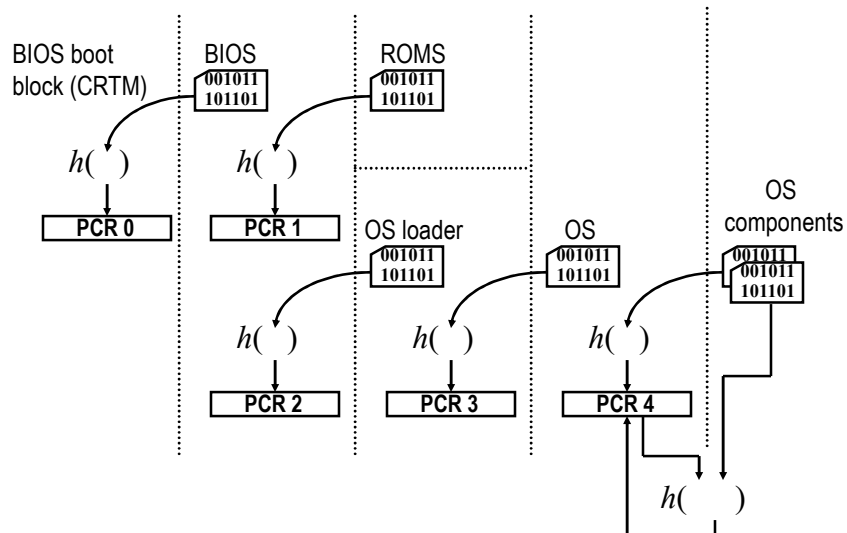


- PCR is updated before newly loaded software gets control
- PCR cannot be erased except by reboot (or protected processor instruction in v1.2 TPMs)
- In this way, PCR contains record of software running

33

33

TCPA Authenticated Boot



34

34

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**

35

35

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} **signed** PCRvals = ...
 - ▼ (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

36

36

The diagram illustrates a secure communication protocol between a Workstation W and a File Server. On the left, a cartoon character representing the user is shown thinking, "I wonder what Mike's salary is ...". The Workstation W is depicted as a computer monitor and keyboard. The File Server is shown as a server rack. The communication channel is labeled "Network channel".

The Workstation W has a stack of components: "Application A ", "O/S", and "Workstation W ". The File Server has a stack of components: "File Server", "O/S", and "Server".

The communication flow is as follows:

- The Workstation W sends a message to the File Server: $K_{W.OS}$ signed (key($K_{W.OS}$). U says A says F).
- The File Server sends a message back to the Workstation W : K_{CA} signed key(K_W) speaksfor key(K_{CA}). W .

Below the Workstation W , there is a note: K_U^{-1} signed (key(K_U) speaksfor key(K_{CA}). U).

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

37

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)

38

Example (cont.)

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

39

39

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		

40

40

Example: Web Server Authentication (2)

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

K_{CA} signed ($\text{key}(K_{\text{www.foo.com}})$ speaks for $\text{key}(K_{CA}).\text{'www.foo.com'}$)
to browser

- Browser is shipped with public keys for numerous CAs:

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

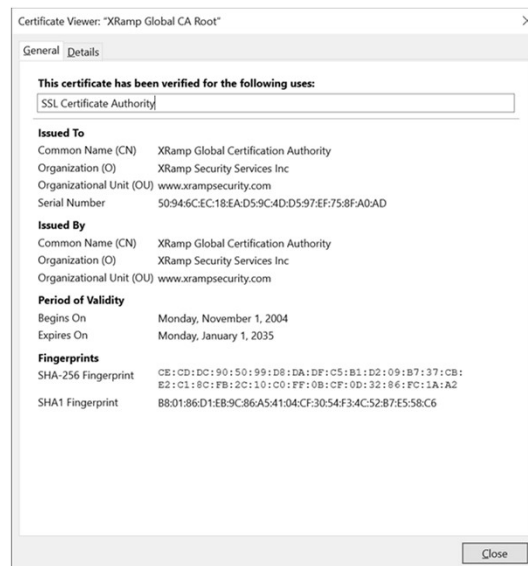
- ▼ E.g., Mozilla Firefox 58.0.1 ships with *hundreds* of CA keys preloaded
- ▼ Reportedly these represent organizations from all over the world

- Should we really trust that $\text{key}(K_{CA}).\text{'www.foo.com'}$ is the “right” www.foo.com for all of these CAs?

41

41

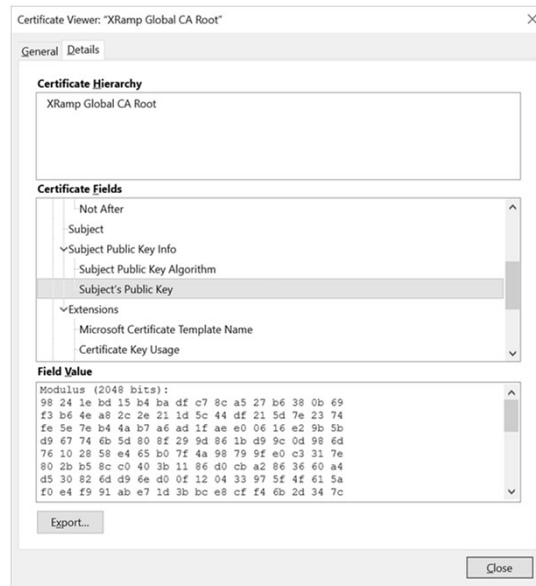
Example CA (from Firefox 58.0.1)



42

42

Example CA (from Firefox 58.0.1)



43

43

What if $K_{\text{www.foo.co m}}^{-1}$ Is Compromised?

- In SSL/TLS, the certificate is sent from the web server
 - ▼ CA sends long-lived certificate to web server in advance
 - ▼ Web server stores it, and forwards it in SSL/TLS handoff protocol
- This structure has a benefit
 - ▼ K_{CA}^{-1} can be kept offline and made more secure
- What if $K_{\text{www.foo.com}}^{-1}$ is exposed?
 - ▼ CA may wish to revoke the statement (certificate)
 - K_{CA} signed (key($K_{\text{www.foo.com}}$) speaksfor key(K_{CA}). 'www.foo.com')

44

44

- For rapid certificate revocation, there needs to be some online authority O that vouches for it

- 45

Certificate Countersigning

- 46

Copyright © 2020 by Michael Reiter
All rights reserved.

Certificate Revocation Lists

- Certificate Revocation Lists (CRLs) are an alternative to countersignatures by an online authority
 - ▼ Also more commonly used
- Each CA periodically produces a digitally signed statement recanting listed certificates

K_{CA} says “certificates 134, 538, and 977 are invalid”
Certificate serial numbers

- CRLs must have limited lifetimes
- All certificate serial numbers must be included in *one* CRL

47

47

Revisiting Trust of CA

- Trusting that for all CAs , $\text{key}(K_{CA}).A$ is the “correct” A is too strong
 - ▼ Remember that Firefox comes shipped with hundreds 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”)

48

48

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

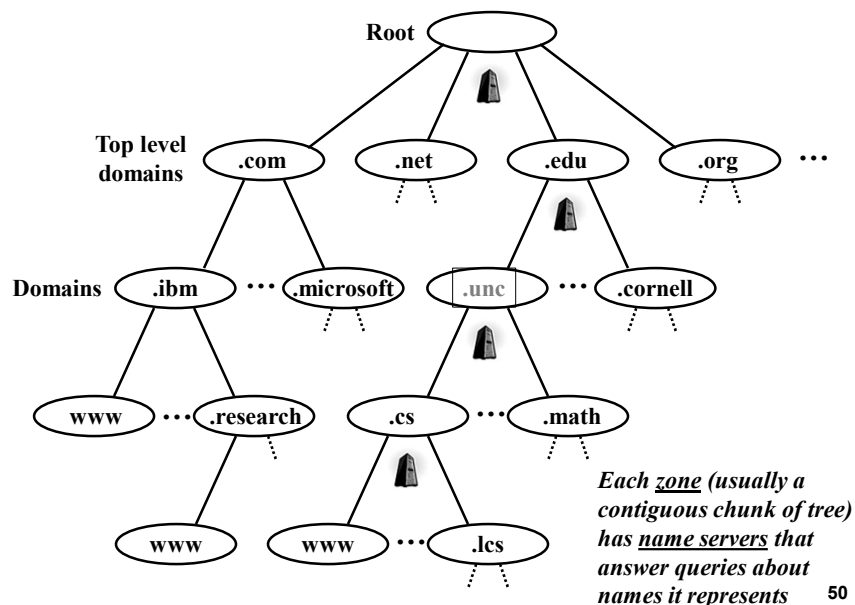
- **DNS Security (“DNSSEC”) specifies extensions to DNS to make DNS more secure**

- ▼ “Owned” by the DNSEXT working group in IETF
- ▼ Specified in RFC 2065 (January 1997), revised since

49

49

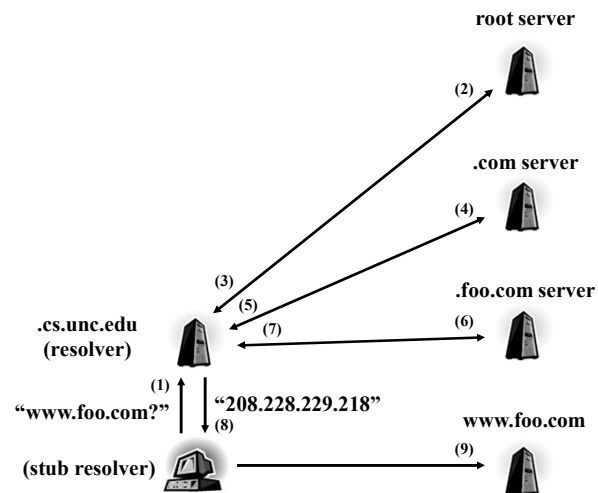
DNS Name Hierarchy



50

50

DNS Name Resolution

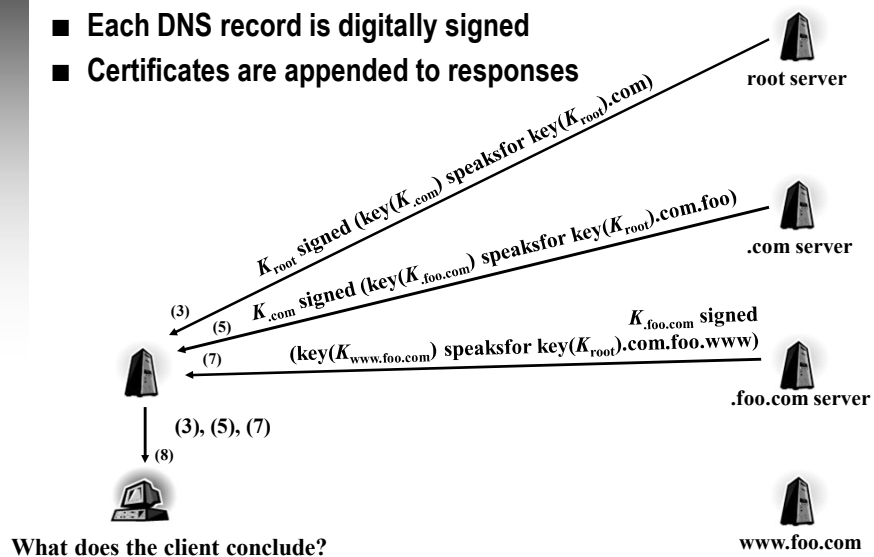


51

51

DNSSEC

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



52

52

Example Proof

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

53

53

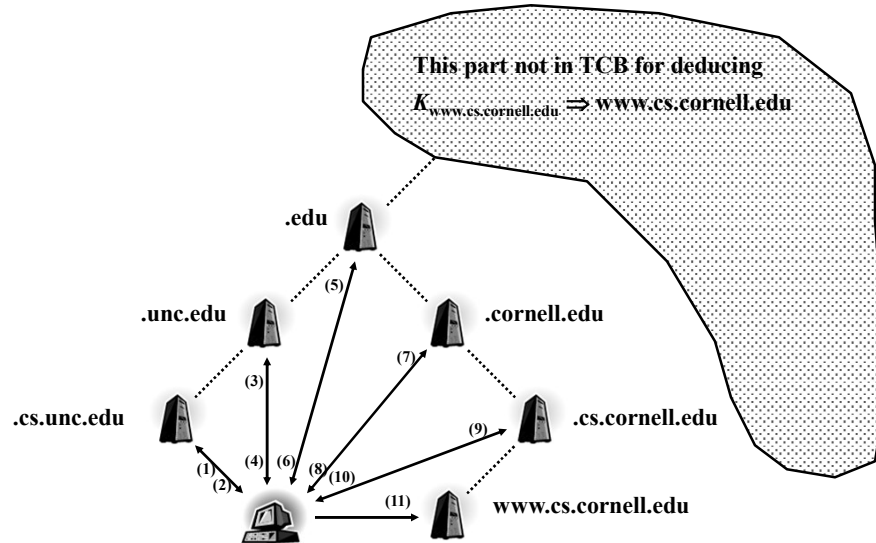
What Went Wrong?

- 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

54

54

Eliminating a Globally Trusted Authority



55

55

Extensions to the Logic

$$A \text{ says ascend}(\text{key}(K_{B,C}), B.C.D)$$

$$\frac{\text{key}(K_{B,C}) \text{ says ascend}(\text{key}(K_B), B.C)}{A \text{ says ascend}(\text{key}(K_B), B.C)}$$

(ascent)

■ If $C \neq D$

$$A \text{ says ascend}(\text{key}(K_B), B.C)$$

$$\frac{\text{key}(K_B) \text{ says descend}(\text{key}(K_{B,D}), B.D)}{A \text{ says descend}(\text{key}(K_{B,D}), B.D)}$$

(a2d)

56

56

Extensions to the Logic (cont.)

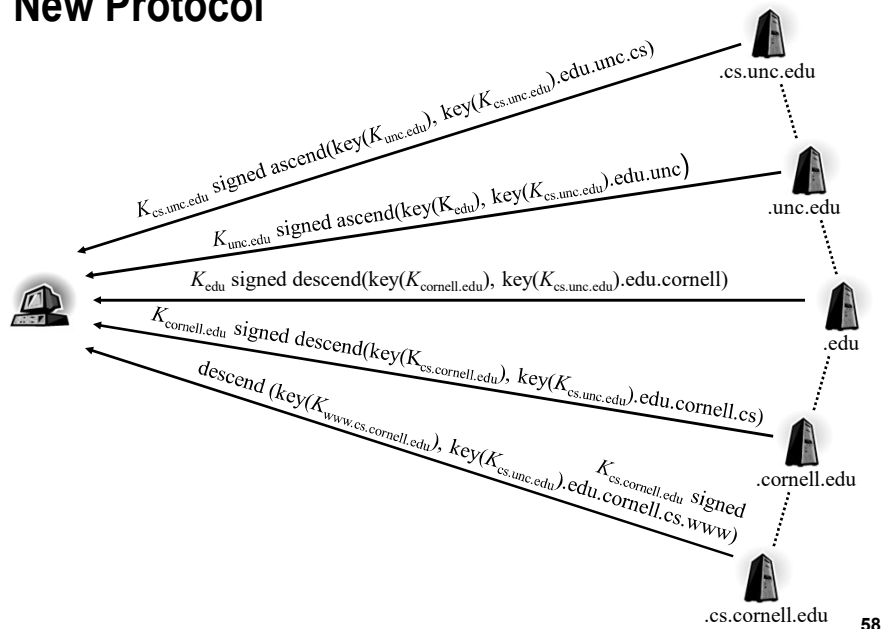
$$\frac{A \text{ says descend}(\text{key}(K_B), B) \quad \text{key}(K_B) \text{ says descend}(\text{key}(K_{B.C}), B.C)}{A \text{ says descend}(\text{key}(K_{B.C}), B.C)} \quad (\text{descent})$$

$$\frac{A \text{ says descend}(\text{key}(K_B), B)}{A \text{ says key}(K_B) \text{ speaksfor } B} \quad (\text{resolve})$$

57

57

New Protocol



58

Analysis

1. $K_{cs.unc.edu}$ signed ascend(key($K_{unc.edu}$), key($K_{cs.unc.edu}$).edu.unc.cs)
2. $K_{unc.edu}$ signed ascend(key(K_{edu}), key($K_{cs.unc.edu}$).edu.unc)
3. K_{edu} signed descend(key($K_{cornell.edu}$), key($K_{cs.unc.edu}$).edu.cornell)
4. $K_{cornell.edu}$ signed
descend(key($K_{cs.cornell.edu}$), key($K_{cs.unc.edu}$).edu.cornell.cs)
5. $K_{cs.cornell.edu}$ signed
descend (key($K_{www.cs.cornell.edu}$), key($K_{cs.unc.edu}$).edu.cornell.cs.www)
6. $K_{www.cs.cornell.edu}$ signed F
7. key($K_{cs.unc.edu}$) says ascend(key($K_{unc.edu}$), key($K_{cs.unc.edu}$).edu.unc.cs)
says-I(1)
8. key($K_{unc.edu}$) says ascend(key(K_{edu}), key($K_{cs.unc.edu}$).edu.unc)
says-I(2)
9. key(K_{edu}) says descend(key($K_{cornell.edu}$), key($K_{cs.unc.edu}$).edu.cornell)
says-I(3)

59

59

Analysis (cont.)

10. key($K_{cornell.edu}$) says descend(key($K_{cs.cornell.edu}$),
key($K_{cs.unc.edu}$).edu.cornell.cs) says-I(4)
11. key($K_{cs.cornell.edu}$) says descend (key($K_{www.cs.cornell.edu}$),
key($K_{cs.unc.edu}$).edu.cornell.cs.www) says-I(5)
12. key($K_{www.cs.cornell.edu}$) says F says-I(6)
13. key($K_{cs.unc.edu}$) says ascend(key(K_{edu}), key($K_{cs.unc.edu}$).edu.unc)
ascent(7, 8)
14. key($K_{cs.unc.edu}$) says
descend(key($K_{cornell.edu}$), key($K_{cs.unc.edu}$).edu.cornell) a2d(13, 9)
15. key($K_{cs.unc.edu}$) says descend(key($K_{cs.cornell.edu}$),
key($K_{cs.unc.edu}$).edu.cornell.cs) descent(14, 10)
16. key($K_{cs.unc.edu}$) says descend (key($K_{www.cs.cornell.edu}$),
key($K_{cs.unc.edu}$).edu.cornell.cs.www) descent(15, 11)

60

60

Analysis (cont.)

17. $\text{key}(K_{\text{cs.unc.edu}})$ says $\text{key}(K_{\text{www.cs.cornell.edu}})$ speaksfor
 $\text{key}(K_{\text{cs.unc.edu}}).\text{edu.cornell.cs.www}$ resolve(16)
18. $\text{key}(K_{\text{cs.unc.edu}}).\text{edu.cornell.cs.www}$ says F speaksfor-E2(12, 17)

61

61

Bibliography

- Lampson, Abadi, Burrows and Wobber. Authentication in distributed systems: Theory and practice. *ACM Transactions on Computer Systems* 10(4), November 1992.

62

62