Software Foundations of Security & Privacy 15315 Spring 2017 Lecture 10: Authentication and Authorization

Matt Fredrikson, Jean Yang
mfredrik@cs.cmu.edu

February 16, 2017

Review: Authorization Logic

Formally, a principal is either an identifier or a key:

$$p \quad ::= \quad \textbf{key}(s) \mid \textit{identifier} \mid p.s$$

where s is a string

Statements, where s is a string and p is a principal

$$
\begin{array}{rcl}
\phi & ::= & \textbf{action}(s) \\
 & \mid & p \textbf{ says } \phi \\
 & \mid & p \textbf{ speaksfor } p \\
 & \mid & s \textbf{ signed } \phi \\
 & \mid & \textbf{delegates}(p, p, s) \\
 & \mid & \phi \rightarrow \phi \\
 & \mid & \phi \wedge \phi
\end{array}
$$

Review: inference rules

says Introduction 1 (Says-I1)

$$\frac{s \ \textbf{signed} \ \phi}{\textbf{key}(s) \ \textbf{says} \ \phi}$$

Intuitively, this rule:

- Creates a principal **key**(s) from a *key string* s
- Establishes **key**(s) said ϕ given the appropriate signature

Review: inference rules

says Introduction 2 (Says-I2)

$$\frac{\phi}{p \textbf{ says } \phi}$$

What does this rule say?

- If ϕ is established to be true, then p says it
- Maybe p didn't say it, but we can proceed as though it did
- Basically, principals will say true things

Review: inference rules

says Introduction 3 (Says-I3)

$$\frac{p \textbf{ says } (p.s \textbf{ says } \phi)}{p.s \textbf{ says } \phi}$$

This rule might seem counterintuitive

- $p.s$ is the "principal that p calls s".
- p can name s to be whomever it wants
- p can just find someone who says ϕ, and call that person s
- We must accept what p says about the person it calls s

Review: inference rules

says Implication (Says-Impl)

$$\frac{p \textbf{ says } (\phi_1 \to \phi_2) \qquad p \textbf{ says } \phi_1}{p \textbf{ says } \phi_2}$$

Recall modus ponens from propositional calculus

- Says-Impl is modus ponens over **says**
- We take principals at their word
- To the logical conclusion

Review: inference rules

speaksfor Elimination 1 (Speaksfor-E1)

$$\frac{p_1 \textbf{ says } (p_2 \textbf{ speaksfor } p_1) \qquad p_2 \textbf{ says } \phi}{p_1 \textbf{ says } \phi}$$

speaksfor Elimination 2 (Speaksfor-E2)

$$\frac{p_1 \textbf{ says } (p_2 \textbf{ speaksfor } p_1.s) \qquad p_2 \textbf{ says } \phi}{p_1.s \textbf{ says } \phi}$$

These rules deal with broad delegations of authority

- When p_1 says p_2 speaks for her...

Review: inference rules

delegates Elimination (Delegate-E)

$$\frac{p_1 \textbf{ says delegates}(p_1, p_2, s) \qquad p_2 \textbf{ says action}(s)}{p_1 \textbf{ says action}(s)}$$

This rule allows more fine-grained delegation of authority

- **delegates** allows p_1 to let p_2 speak-for her
- But only with regard to selected actions

Roles

In some cases, principals want to limit their authority

- Running untrusted code
- Performing mundane, non-critical tasks

Want to model principals acting **as** other principals

In this logic, we can issue statements of the form:

$$p.\textit{role}\ \textbf{says}\ \phi$$

A role is just another principal named by us

Example: programs as roles

If principal p wants to run program a with text t

- Could emit $p.t$ **says** ϕ when program requests ϕ

Alternatively, compute a secure hash h of t

- Then state p **says** $(h$ **speaksfor** $t)$
- What's the benefit of doing it this way?

Reference monitors don't typically want to track code hashes

- Instead, refer to named apps (e.g., "Chrome")
- State that hash values speak for the named app
- Have CA or trusted developer sign this statement

Example due to Mike Reiter

Example: authorizing a program

Suppose we want to execute program a on channel c

1. Read the code t of program a from disk
2. Compute a secure hash h of t
3. Reference monitor attempts to prove that h **speaksfor** a
4. Create process d, copy t into d
5. Emit p **says** c **speaksfor** $p.a$
6. Give d access to channel c

d can now send requests on c

If $p.a$ is authorized for a request, it's granted

Example due to Mike Reiter

Example: malicious code

Malicious code often alters program code on disk

In our example, this means that:

- Altered code t' creates new hash h'
- With high probability, $h \neq h'$!
- Loader can't prove h' **speaksfor** a

Can utilize CA to say which programs are safe

- K_{CA} **signed** $(\text{Chrome } \textbf{speaksfor key}(K_{CA}).\text{trustedSW})$
- trustedSW is a group name
- Authorization rights can be established for group
- Based on trust in *CA*

Example due to Mike Reiter

Secure boot

Idea: establish a group `trustedhosts`

- OS is validated before booting
- OS validates all programs before running them

Validating OS is much the same as validating any other program

- $h(\mathit{OS})$ **speaksfor** *OS*
- $h(\mathit{OS})$ is stored in the system, checked against code before boot

Host (N) can set up channel for OS to use:

1. Generate signing keys $pk_{N.\mathit{OS}}, sk_{N.\mathit{OS}}$
2. Give *OS*: $sk_{N.\mathit{OS}}$ and pk_N **signed key**$(pk_{N.\mathit{OS}})$ **speaksfor key**(pk_N).*OS*

Slide from Mike Reiter

Trusted platform

Effort developed by Trusted Computing Group (TCG)

- Industry consortium with about 100 members founded in 2003
- Standardize a *trusted platform*, using principles we've discussed
- Technical specs relate to hardware, BIOS, and OS features

Some of the main goals of a TP:

- **Software attestation:** users can validate software running on platform
- **Binding:** mechanism for secure storage of crypto keys and encryption
- **Sealing:** specify conditions under which decryption can occur

Slide from Mike Reiter

Trusted platform module (TPM)

Basis for trusted platform is a hardware component called the TPM

- Restricted software interface, isolated from attack “from above”
- (Somewhat) resistant to hardware tampering
- Initialized with a unique key pair $pk_{\textsf{TPM}}, sk_{\textsf{TPM}}$

The manufacturer loads a certificate onto each TPM:

$$pk_{\textsf{TPME}} \textbf{ signed } (\textbf{key}(pk_{\textsf{TPM}}) \textbf{ speaksfor } \textbf{key}(pk_{\textsf{TPME}}).\textit{TrustedModules})$$

TrustedModules is a group

- All TPMs manufactured by that entity
- Trust in TPM-signed statements reduces to trust in the manufacturer

Slide from Mike Reiter

Roots of trust

Background: in TPM-land, you'll often hear of "measurements"

- A "measurement" of a piece of software is a hash of its code
- Why not just say "code hash"?

Root of trust for measurement

- Component that begins measurement of running software
- Usually, the first thing to run on boot
- Can run later, but everything before it must be trusted

Root of trust for reporting

- Component responsible for storing measurements as they are taken
- Must also prevent tampering or undoing of stored measurements

Platform configuration registers

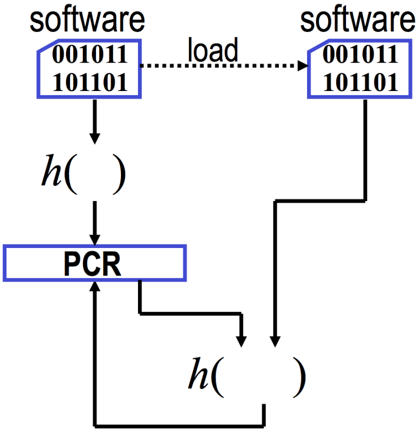
TPM contains 16 20-byte PCRs

PCRs hold measurements of running software

- Updated before software is run
- Can't be reset until reboot

PCRs *extended* to reflect newly-loaded code

Provide a record of all running software



Slide from Mike Reiter

Authenticated boot

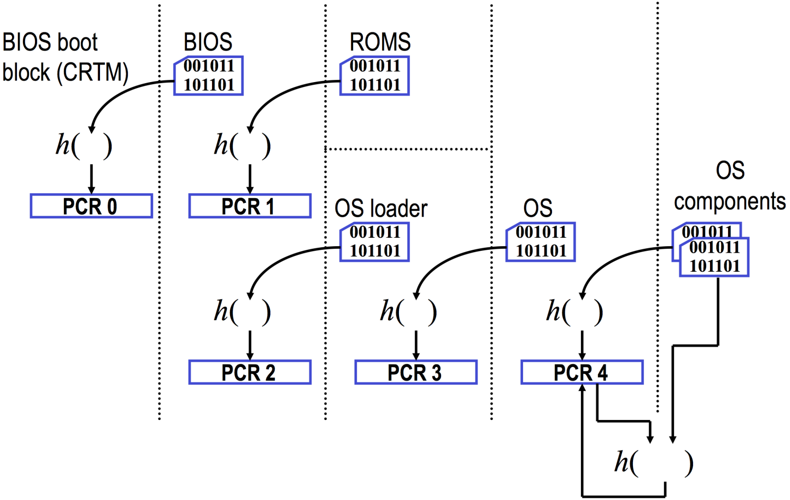


Image credit: Mike Reiter

Remote attestation

Goal: convince a remote machine that we're running the right code

TPM can provide a signed copy of the PCR values

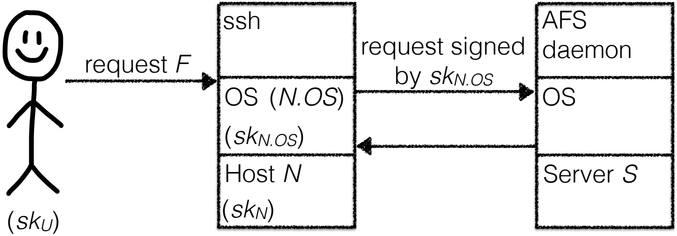
$$pk_{\mathit{TPM}} \textbf{ signed } \mathit{pcr\_vals}$$

TPM will also respond with records of what the measurements summarize

- Name, version number, etc. for loaded software
- Order in which it was loaded
- Enough for remote machine to check PCR values

Any issues with this setup?

Example



1. pk_{CA} **signed** ($\textbf{key}(pk_N)$ **speaksfor** $\textbf{key}(pk_{CA}).N$)
2. pk_{CA} **signed** ($\textbf{key}(pk_U)$ **speaksfor** $\textbf{key}(pk_{CA}).U$)
3. pk_N **signed** ($\textbf{key}(pk_{N.OS})$ **speaksfor** $\textbf{key}(pk_{CA}).N.OS$)
4. $pk_{N.OS}$ **signed** ($\textbf{key}(pk_{N.OS}).U$ **says** F)

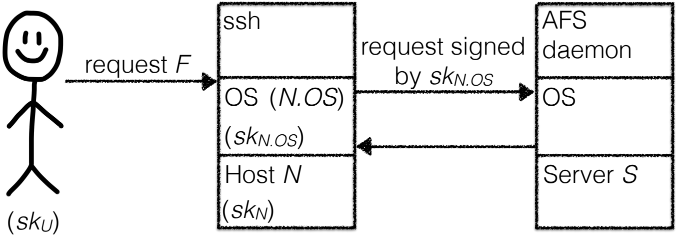
Can we prove $\textbf{key}(pk_{CA}).U$ **says** F?

Example: proof

1. **key**(pk_{CA}) **says** (**key**(pk_N) **speaksfor key**$(pk_{\mathsf{CA}}).N$) (Says-I1)
2. **key**(pk_{CA}) **says** (**key**(pk_U) **speaksfor key**$(pk_{\mathsf{CA}}).U$) (Says-I1)
3. **key**(pk_N) **says** (**key**$(pk_{N.\mathsf{OS}})$ **speaksfor key**$(pk_{\mathsf{CA}}).N.\mathsf{OS}$) (Says-I1)
4. **key**$(pk_{N.\mathsf{OS}})$ **says** (**key**$(pk_{N.\mathsf{OS}}).U$ **says** F) (Says-I1)
5. **key**$(pk_{N.\mathsf{OS}}).U$ **says** F (Says-I3)
6. **key**$(pk_{\mathsf{CA}}).N$ **says** (**key**$(pk_{N.\mathsf{OS}})$ **speaksfor key**$(pk_{\mathsf{CA}}).N.\mathsf{OS}$) (Speaksfor-E2)
7. **key**$(pk_{\mathsf{CA}}).N.OS$ **says** (**key**$(pk_{N.\mathsf{OS}}).U$ **says** F) (Speaksfor-E2)

Can't make any more progress!

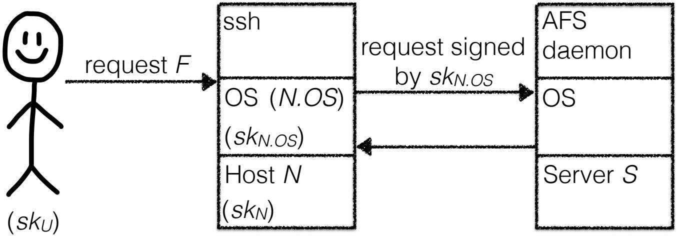
Example



1. pk_{CA} **signed** $($**key**(pk_N) **speaksfor key**$(pk_{CA}).N)$
2. pk_{CA} **signed** $($**key**(pk_U) **speaksfor key**$(pk_{CA}).U)$
3. pk_N **signed** $($**key**$(pk_{N.OS})$ **speaksfor key**$(pk_{CA}).N.OS)$
4. $pk_{N.OS}$ **signed** $($**key**$(pk_{N.OS}).U$ **says** $F)$

What went wrong?

Example



1. pk_{CA} **signed** (**key**(pk_N) **speaksfor key**$(pk_{CA}).N$)
2. pk_{CA} **signed** (**key**(pk_U) **speaksfor key**$(pk_{CA}).U$)
3. pk_N **signed** (**key**$(pk_{N.OS})$ **speaksfor key**$(pk_{CA}).N.OS$)
4. $pk_{N.OS}$ **signed** (**key**$(pk_{N.OS}).U$ **says** F)
5. pk_U **signed** (**key**$(pk_{CA}).N.OS.U$ **speaksfor key**$(pk_{CA}).U$)

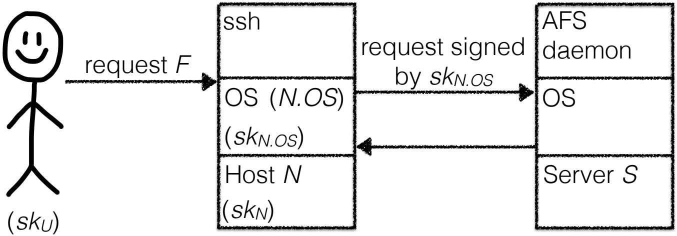
User needs to delegate authority to “who the OS calls U”

Example: proof

1. **key**(pk_{CA}) **says** (**key**(pk_N) **speaksfor key**$(pk_{CA}).N$) (Says-I1)
2. **key**(pk_{CA}) **says** (**key**(pk_U) **speaksfor key**$(pk_{CA}).U$) (Says-I1)
3. **key**(pk_N) **says** (**key**$(pk_{N.OS})$ **speaksfor key**$(pk_{CA}).N.OS$) (Says-I1)
4. **key**$(pk_{N.OS})$ **says** (**key**$(pk_{N.OS}).U$ **says** F) (Says-I1)
5. **key**$(pk_{N.OS}).U$ **says** F (Says-I3)
6. **key**$(pk_{CA}).N$ **says** (**key**$(pk_{N.OS})$ **speaksfor key**$(pk_{CA}).N.OS$) (Speaksfor-E2)
7. **key**$(pk_{CA}).N.OS$ **says** (**key**$(pk_{N.OS}).U$ **says** F) (Speaksfor-E2)
8. **key**(pk_U) **says** (**key**$(pk_{CA}).N.OS.U$ **speaksfor key**$(pk_{CA}).U$) (Says-I1)
9. **key**$(pk_{CA}).U$ **says** (**key**$(pk_{CA}).N.OS.U$ **speaksfor key**$(pk_{CA}).U$) (Speaksfor-E2)

Stuck again!

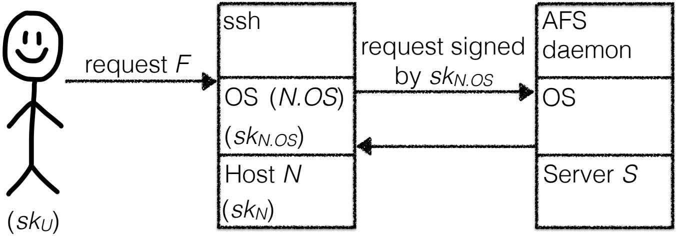
Example



1. pk_{CA} **signed** (**key**(pk_N) **speaksfor key**$(pk_{CA}).N$)
2. pk_{CA} **signed** (**key**(pk_U) **speaksfor key**$(pk_{CA}).U$)
3. pk_N **signed** (**key**$(pk_{N.OS})$ **speaksfor key**$(pk_{CA}).N.OS$)
4. $pk_{N.OS}$ **signed** (**key**$(pk_{N.OS}).U$ **says** F)
5. pk_U **signed** (**key**$(pk_{CA}).N.OS.U$ **speaksfor key**$(pk_{CA}).U$)

Still missing something...

Example



1. pk_{CA} **signed** (**key**(pk_N) **speaksfor key**$(pk_{CA}).N$)
2. pk_{CA} **signed** (**key**(pk_U) **speaksfor key**$(pk_{CA}).U$)
3. pk_N **signed** (**key**$(pk_{N.OS})$ **speaksfor key**$(pk_{CA}).N.OS$)
4. $pk_{N.OS}$ **signed** (**key**$(pk_{N.OS}).U$ **says** F)
5. pk_U **signed** (**key**$(pk_{CA}).N.OS.U$ **speaksfor key**$(pk_{CA}).U$)
6. $pk_{N.OS}$ **signed** (**key**$(pk_{N.OS}).U$ **speaksfor key**$(pk_{CA}).N.OS.U$)

OS needs to vouch for authenticity of “who the OS calls U”

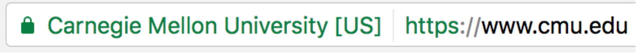
Example: proof

1. **key**$(pk_{\textsf{CA}})$ **says** $($**key**(pk_N) **speaksfor key**$(pk_{\textsf{CA}}).N)$ (Says-I1)
2. **key**$(pk_{\textsf{CA}})$ **says** $($**key**(pk_U) **speaksfor key**$(pk_{\textsf{CA}}).U)$ (Says-I1)
3. **key**(pk_N) **says** $($**key**$(pk_{N.\textsf{OS}})$ **speaksfor key**$(pk_{\textsf{CA}}).N.\textsf{OS})$ (Says-I1)
4. **key**$(pk_{N.\textsf{OS}})$ **says** $($**key**$(pk_{N.\textsf{OS}}).U$ **says** $F)$ (Says-I1)
5. **key**$(pk_{N.\textsf{OS}}).U$ **says** F (Says-I3)
6. **key**$(pk_{\textsf{CA}}).N$ **says** $($**key**$(pk_{N.\textsf{OS}})$ **speaksfor key**$(pk_{\textsf{CA}}).N.\textsf{OS})$ (Speaksfor-E2)
7. **key**$(pk_{\textsf{CA}}).N.OS$ **says** $($**key**$(pk_{N.\textsf{OS}}).U$ **says** $F)$ (Speaksfor-E1)
8. **key**(pk_U) **says** $($**key**$(pk_{\textsf{CA}}).N.\textsf{OS}.U$ **speaksfor key**$(pk_{\textsf{CA}}).U)$ (Says-I1)
9. **key**$(pk_{\textsf{CA}}).U$ **says** $($**key**$(pk_{\textsf{CA}}).N.\textsf{OS}.U$ **speaksfor key**$(pk_{\textsf{CA}}).U)$ (Speaksfor-E2)
10. **key**$(pk_{N.\textsf{OS}})$ **says** $($**key**$(pk_{N.\textsf{OS}}).U$ **speaksfor key**$(pk_{\textsf{CA}}).N.\textsf{OS}.U)$

Example: proof continued

5. **key**$(pk_{N.\mathsf{OS}}).U$ **says** F (Says-I3)
6. **key**$(pk_{\mathsf{CA}}).N$ **says** (**key**$(pk_{N.\mathsf{OS}})$ **speaksfor key**$(pk_{\mathsf{CA}}).N.\mathsf{OS})$ (Speaksfor-E2)
7. **key**$(pk_{\mathsf{CA}}).N.OS$ **says** (**key**$(pk_{N.\mathsf{OS}}).U$ **says** F) (Speaksfor-E2)
8. **key**(pk_U) **says** (**key**$(pk_{\mathsf{CA}}).N.\mathsf{OS}.U$ **speaksfor key**$(pk_{\mathsf{CA}}).U)$ (Says-I1)
9. **key**$(pk_{\mathsf{CA}}).U$ **says** (**key**$(pk_{\mathsf{CA}}).N.\mathsf{OS}.U$ **speaksfor key**$(pk_{\mathsf{CA}}).U)$ (Speaksfor-E2)
10. **key**$(pk_{N.\mathsf{OS}})$ **says** (**key**$(pk_{N.\mathsf{OS}}).U$ **speaksfor key**$(pk_{\mathsf{CA}}).N.\mathsf{OS}.U)$
11. **key**$(pk_{\mathsf{CA}}).N.\mathsf{OS}$ **says** (**key**$(pk_{N.\mathsf{OS}}).U$ **speaksfor key**$(pk_{\mathsf{CA}}).N.\mathsf{OS}.U)$ (Speaksfor-E2)
12. **key**$(pk_{\mathsf{CA}}).N.\mathsf{OS}.U$ **says** F (Speaksfor-E2)
13. **key**$(pk_{\mathsf{CA}}).U$ **says** F (Speaksfor-E1)

Revocation



When you see this, it means that CMU's server sent a certificate

$$pk_{CA}\ \textbf{signed}\ (\textbf{key}(pk_{'\texttt{cmu.edu}'})\ \textbf{speaksfor}\ \textbf{key}(pk_{CA}).'\texttt{cmu.edu}')$$

What happens if $sk_{'\texttt{cmu.edu}'}$ is compromised?

- CA's certificate becomes a lie!
- To maintain trust, CA may need to **revoke** the certificate

Countersigning

Idea: rely on another authority O to sign for validity of the certificate

Now the CA releases a *weaker* certificate

$$pk_{CA}\ \textbf{signed}\ (\textbf{key}(pk_O)\ \textbf{says}\ \textbf{key}(pk_{'\texttt{cmu.edu}'})\ \textbf{speaksfor}\ \textbf{key}(pk_O.\texttt{'cmu.edu'}) \rightarrow (\textbf{key}(pk_{'\texttt{cmu.edu}'})\ \textbf{speaksfor}\ \textbf{key}(pk_{CA})).'\texttt{cmu.edu}'))$$

And O *countersigns*:

$$pk_O\ \textbf{signed}\ (\textit{date}() < '2017.2.17' \rightarrow \textbf{key}(pk_{'\texttt{cmu.edu}'})\ \textbf{speaksfor}\ \textbf{key}(pk_O).'\texttt{cmu.edu}')$$

Slide from Mike Reiter

Example

Assumptions:

1. pk_{CA} **signed** $($**key**(pk_O) **says key**(pk_A) **speaksfor key**$(pk_O.\mathtt{A}) \rightarrow$ $($**key**(pk_A) **speaksfor key**$(pk_{\mathsf{CA}})).A))$
2. pk_O **signed** $(\textit{date} < '2017.2.17' \rightarrow$ **key**(pk_A) **speaksfor key**$(pk_O).A)$
3. $\textit{date} < '2017.2.17'$

---

1. **key**(pk_{CA}) **says** $($**key**(pk_O) **says key**(pk_A) **speaksfor key**$(pk_O.\mathtt{A}) \rightarrow$ $($**key**(pk_A) **speaksfor key**$(pk_{\mathsf{CA}})).A))$
2. **key**(pk_O) **says** $(\textit{date} < '2017.2.17' \rightarrow$ **key**(pk_A) **speaksfor key**$(pk_O).A)$
3. **key**(pk_O) **says** $\textit{date} < '2017.2.17'$
4. **key**(pk_O) **says key**(pk_A) **speaksfor key**$(pk_O).A$
5. **key**(pk_{CA}) **says** $($**key**(pk_O) **says key**(pk_A) **speaksfor key**$(pk_O).A)$
6. **key**(pk_{CA}) **says** $($**key**(pk_A) **speaksfor key**$(pk_{\mathsf{CA}}).A)$

Why countersign?

Why doesn't `cmu.edu` just obtain a new certificate with each connection?

- Efficiency and scalability, perhaps
- CA's private signing key would need to be kept on an online server
- This exposes it to all kinds of risk, better to keep it offline

Does countersigning address the risk?

- If O's signing key becomes compromised, what happens?
- Can extend the period where the compromised certificate is accepted
- But O cannot issue new certificates, so the damage is limited

Further reading

B. Lampson, M. Abadi, M. Burrows. *Authorization in Distributed Systems*. In ACM TOCS, 1992.

M. Abadi. *Logic in Access Control*. In Logic in Computer Science, 2003.

L. Bauer, S. Garriss, M. K. Reiter. *Efficient Proving for Practical Distributed Access-Control Systems*. In ESORICS, 2007.