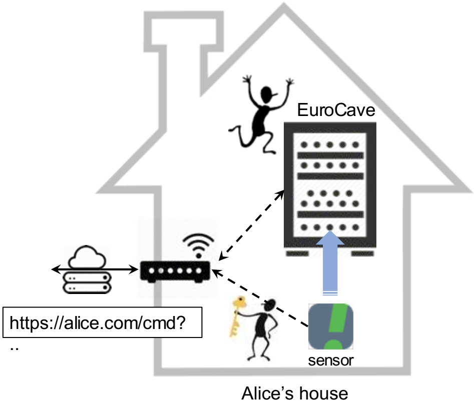
Decentralized Access Controls

Simon Foley
IMT Atlantique

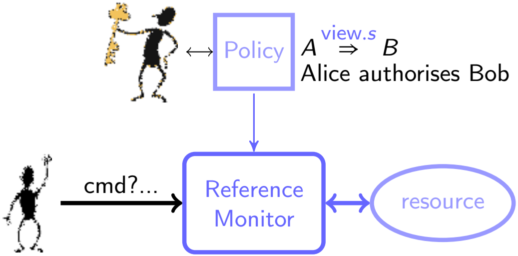
March 16, 2018

Motivation. Managing access control in Alice's smart house

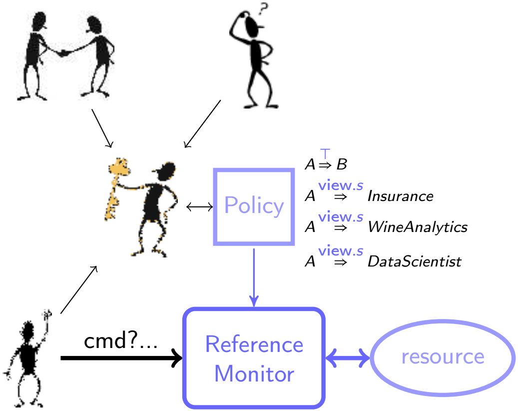
- Web-API for the things in Alice's house.
- Alice gives full access to things to her house-group containing Bob, and others.
- Alice grants EuroCave engineer access to a maintenance service.
- To insure his wine, bob installs an extra temperature/humidity sensor in EuroCave; grants access to insurance company.
- Insurance company outsources all wine monitoring to wine analytics company.
- Wine analytics company delegates access to Data Scientist.



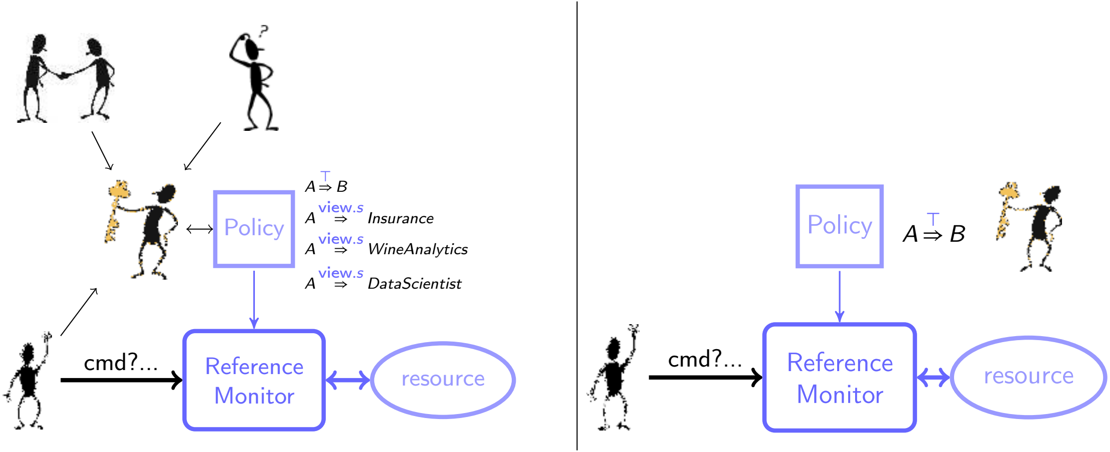
Centralised versus decentralised authorisation



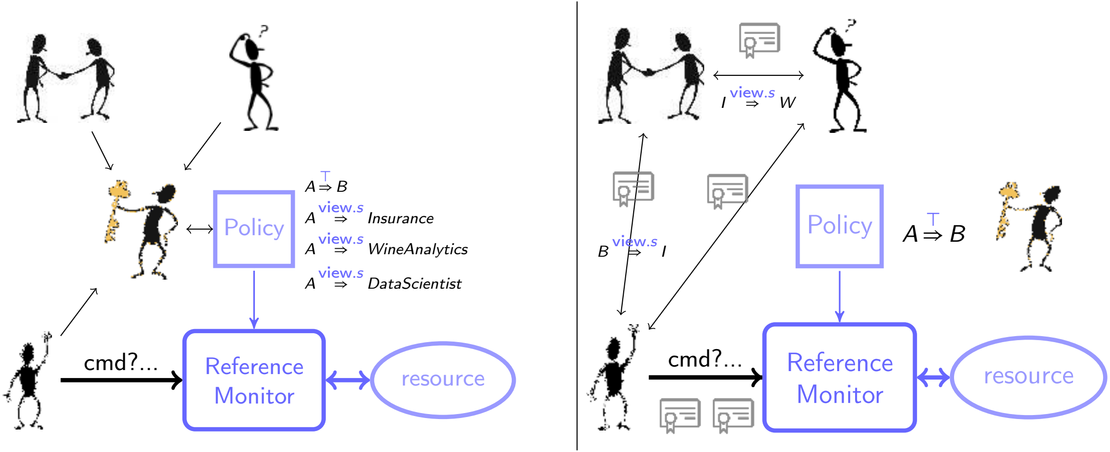
Centralised versus decentralised authorisation



Centralised versus decentralised authorisation



Centralised versus decentralised authorisation



Outline of Talk

Authorization Certificates

Permissions $(PERM, \sqsubseteq, \sqcap)$

Partially ordered set; $X \sqsubseteq Y$ means permission Y provides no less authorization than X and $X \sqcap Y$ is greatest lower bound of X, Y. For example, SPKI:

(tag (http alice.com/view?s)) $\sqsubseteq$ (tag (http (* prefix alice.com/)))

Delegation Statement

$P \overset{X}{\Longrightarrow} Q$ means that principal P delegates permission $X \in PERM$ to principal Q.

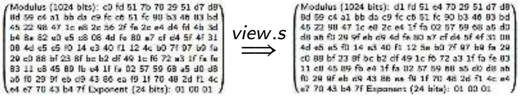
$$\frac{\{\!|P, X, D, V|\!\}_{sK}}{K \overset{X}{\Longrightarrow} P} \qquad \frac{P \overset{Y}{\Longrightarrow} Q; X \sqsubseteq Y}{P \overset{X}{\Longrightarrow} Q} \qquad \frac{P \overset{X}{\Longrightarrow} Q; Q \overset{Y}{\Longrightarrow} R;}{P \overset{X \sqcap Y}{\Longrightarrow} R}$$

D is delegation bit, and V lifetime: we ignore these in this presentation.

Naming principals

Principals as public keys

Using public keys to identify principals is awkward.



SDSI: use local name $(P\ \ N)$ to identify principal named as N in the namespace of principal P.

Speaks for statement

$P \rightarrow Q$ means that principal Q speaks for principal P.

$$\frac{\{\!|N, P, V|\!\}_{sK}}{(K\ \ N) \rightarrow P} \qquad \frac{P \rightarrow (Q\ \ N);\, Q \rightarrow R}{P \rightarrow (R\ \ N)} \qquad \frac{P \overset{X}{\Longrightarrow} Q;\, Q \rightarrow R}{P \overset{X}{\Longrightarrow} R}$$

Delegation Example

- Alice permits members in her group to access any device in her house

$$K_A \stackrel{\top}{\Longrightarrow} (K_A \text{ mbrs}); \quad \text{view}.s \sqsubseteq \top$$

- Bob and Clare are members

$$\begin{aligned} (K_A \text{ mbrs}) &\rightarrow (K_A \text{ Bob}); \\ (K_A \text{ Bob}) &\rightarrow (K_{CA} \text{ Robert}); \\ (K_{CA} \text{ Robert}) &\rightarrow (K_B); \\ (K_A \text{ mbrs}) &\rightarrow K_C; \end{aligned}$$

K_A/Alice's namespace

Name	Principal
mbrs	(K_A Bob)
mbrs	K_C
Bob	(K_{CA} Robert)

K_{CA} namespace

Name	Principal
Robert	K_B
...	...

Delegation Example

- Alice permits members in her group to access any device in her house

$$K_A \stackrel{\top}{\Longrightarrow} (K_A \text{ mbrs}); \quad \text{view}.s \sqsubseteq \top$$

- Bob and Clare are members

$$\begin{aligned} (K_A \text{ mbrs}) &\rightarrow (K_A \text{ Bob}); \\ (K_A \text{ Bob}) &\rightarrow (K_{CA} \text{ Robert}); \\ (K_{CA} \text{ Robert}) &\rightarrow (K_B); \\ (K_A \text{ mbrs}) &\rightarrow K_C; \end{aligned}$$

- Bob delegates access to wine sensor s to insurance company *Ivan*.

$$K_B \stackrel{\text{view}.s}{\Longrightarrow} (K_{CA}\ GFIA\ Ivan)$$

- Insurance company (K_I) fully trusts wine analytics company W,

$$K_I \stackrel{\text{view}.*}{\Longrightarrow} K_W$$

- grants authority to Data Scientist *Steve*

$$K_W \stackrel{\text{view}.*}{\Longrightarrow} (K_W\ Steve)$$

Delegation Example

- Alice permits members in her group to access any device in her house

$$K_A \stackrel{\top}{\Longrightarrow} (K_A \text{ mbrs}); \quad \text{view}.s \sqsubseteq \top$$

- Bob and Clare are members

$$\begin{aligned} (K_A \text{ mbrs}) &\rightarrow (K_A \text{ Bob}); \\ (K_A \text{ Bob}) &\rightarrow (K_{CA} \text{ Robert}); \\ (K_{CA} \text{ Robert}) &\rightarrow (K_B); \\ (K_A \text{ mbrs}) &\rightarrow K_C; \end{aligned}$$

- Bob delegates access to wine sensor s to insurance company *Ivan*.

$$K_B \stackrel{\text{view}.s}{\Longrightarrow} (K_{CA}\ GFIA\ Ivan)$$

- Insurance company (K_I) fully trusts wine analytics company W,

$$K_I \stackrel{\text{view}.*}{\Longrightarrow} K_W$$

- grants authority to Data Scientist *Steve*

$$K_W \stackrel{\text{view}.*}{\Longrightarrow} (K_W\ Steve)$$

Steve requests access; Alice deduces:

$$K_A \stackrel{\text{view}.s}{\Longrightarrow} (K_W\ Steve)$$

Subterfuge in Delegation Certificates

- Clare lives at Dishonest Dave's house

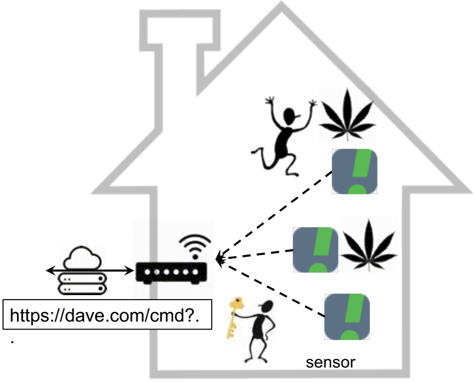
  $K_D \stackrel{\top}{\Longrightarrow} (K_D \text{ mbrs}); \quad (K_D \text{ mbrs}) \rightarrow K_C$

- Clare is also an occasional guest at Alice's house, but Dave intercepts and conceals membership $(K_A \text{ mbrs}) \rightarrow K_C$ from Clare.
- Clare grows plants, overseen by Evil Eve:

  $K_C \stackrel{\text{view}.s}{\Longrightarrow} K_E$

- Eve can access Alice's sensor s.

  $K_D \stackrel{\text{view}.s}{\Longrightarrow} K_E; \quad K_A \stackrel{\text{view}.s}{\Longrightarrow} K_E$



Dave's glasshouse

Subterfuge in Delegation Certificates

- Clare lives at Dishonest Dave's house

  $K_D \overset{\top}{\Longrightarrow} (K_D \text{ mbrs}); \quad (K_D \text{ mbrs}) \rightarrow K_C$

- Clare is also an occasional guest at Alice's house, but Dave intercepts and conceals membership $(K_A \text{ mbrs}) \rightarrow K_C$ from Clare.

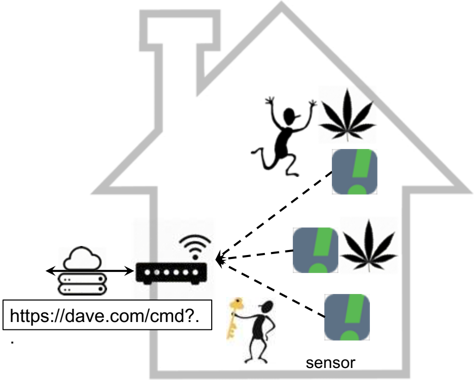
- Clare grows plants, overseen by Evil Eve:

  $K_C \overset{\text{view}.s}{\Longrightarrow} K_E$

- Eve can access Alice's sensor s.

  $K_D \overset{\text{view}.s}{\Longrightarrow} K_E; \quad K_A \overset{\text{view}.s}{\Longrightarrow} K_E$

- A ~~confused~~ hoodwinked deputy problem.



Dave's glasshouse

Subterfuge Intuition

Local delegation state: certificates seen by a principal

For example, Clare's current delegation state u:

$$[K_D \overset{\text{view}.s}{\Longrightarrow}_u K_C; K_C \overset{\text{view}.s}{\Longrightarrow}_u K_E; K_D \overset{\text{view}.s}{\Longrightarrow}_u K_E]$$

Delegation state equivalence $u \approx_P t$

P as sure of being in state u as being in state t. For example,

$$[K_D \overset{\text{view}.s}{\Longrightarrow}_u K_C; K_C \overset{\text{view}.s}{\Longrightarrow}_u K_E] \approx_{K_C} [K_A \overset{\text{view}.s}{\Longrightarrow}_u K_C; K_C \overset{\text{view}.s}{\Longrightarrow}_u K_E]$$

Avoiding Subterfuge

Every delegation state t, equivalent to a state s reachable by Clare, upholds Alice's policy.

$$\forall u \bullet \forall t \bullet policy(u) \wedge u \approx_{K_C} t \Rightarrow policy(t)$$

Avoiding Subterfuge

Globally distinct permissions?

Delegate a permission URI

$$K_A \overset{\texttt{http://www.alice.com/view?s}}{\Longrightarrow} (K_A \text{ mbrs})$$

Avoiding Subterfuge

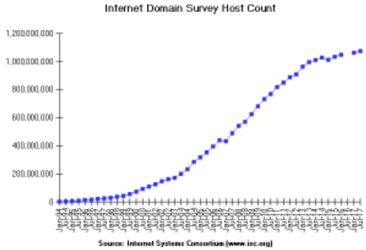
Globally distinct permissions?

Delegate a permission URI

$$K_A \stackrel{\texttt{http://www.alice.com/view?s}}{\Longrightarrow} (K_A \text{ mbrs})$$

Who decides the name?

- Register assignments with IANA/ICANN?
- Global security authority?



Avoiding Subterfuge

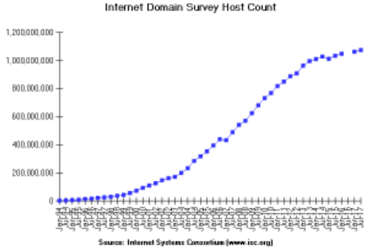
Globally distinct permissions?

Delegate a permission URI

$$K_A \overset{\texttt{http://www.alice.com/view?s}}{\Longrightarrow} (K_A \text{ mbrs})$$

Who decides the name?

- Register assignments with IANA/ICANN?
- Global security authority?



Dave can still forge the permission (signed or otherwise)

$$K_D \overset{\texttt{http://www.alice.com/view?s}}{\Longrightarrow} (K_D \text{ mbrs})$$

Avoiding Subterfuge

Globally distinct permissions?

Alice is owner/originator of her permissions

- Holds a CA domain certificate for alice.com
- Prior to delegation to Insurer, Clare uses Alice's domain certificate to confirm that Alice as owner of K_A is originator of permission alice.com/view.*

$$K_A \stackrel{\text{alice.com/view.}*}{\Longrightarrow} K_C; \quad (K_{ca} \text{ alice.com}) \rightarrow K_A$$

Who really owns the domain certificate?

- Requires reasoning outside of Authorization Model
- Why should one have to trust some global security authority?

Avoiding Subterfuge

Globally distinct permissions?

Alice is owner/originator of her permissions

- Holds a CA domain certificate for alice.com
- Prior to delegation to Insurer, Clare uses Alice's domain certificate to confirm that Alice as owner of K_A is originator of permission alice.com/view.*

$$K_A \stackrel{\text{alice.com/view.*}}{\Longrightarrow} K_C; \quad (K_{ca}$$

Who really owns the domain certificate?

- Requires reasoning outside of Authorization Model
- Why should one have to trust some global security

The New York Times

January 18, 2005

Purloined Domain Name Is an Unsolved Mystery

By TOM ZELLER Jr.

It was yet another reminder of how vulnerable a company's brand name can be in the world of electronic commerce.

In the space of about 48 hours over the weekend, Panix.com, New York City's oldest commercial Internet service provider, saw its name slip out of its control and become the center of an international cyberhunt to get it back. Whether maliciously or inadvertently, the company's main domain name - panix.com - had somehow been transfered to a company in Australia.

Mail to users with a panix.com address was suddenly being sent to a server computer in Canada that had no relation to the company. And in Vancouver, Wash., Panix's registrar - the broker responsible for securing rights to the domain name and administering its use - was completely unaware that the name had been pinched.

Avoiding Subterfuge

Globally distinct permissions?

Alice is owner/originator of her permissions

- Holds a CA domain certificate for alice.com
- Prior to delegation to Insurer, Clare uses Alice's domain certificate to confirm that Alice as owner of K_A is originator of permission alice.com/view.*

$$K_A \overset{\text{alice.com/view.}*}{\Longrightarrow} K_C; \quad (K_{ca}$$

Who really owns the domain certificate?

- Requires reasoning outside of Authorization Model
- Why should one have to trust some global security

Avoiding Subterfuge

Globally distinct permissions?

Alice is owner/originator of her permissions

- Holds a CA domain certificate for alice.com
- Prior to delegation to Insurer, Clare uses Alice's domain certificate to confirm that Alice as owner of K_A is originator of permission alice.com/view.*

$$K_A \overset{\text{alice.com/view.}*}{\Longrightarrow} K_C; \quad (K_{ca}$$

Who really owns the domain certificate?

- Requires reasoning outside of Authorization Model
- Why should one have to trust some global security

Local Permissions

A global/super security authority should not be have to be a requirement

- Services/devices decide local permission names
- A service may relate its local permissions to local permissions of other services
- Principals can delegate local permissions,
- and avoid subterfuge.

Local Permission Certificates

Signed permissions $\{view.s\}_{sA}$

Globally unique permission identifiers tied to their originator (these could be based on W3C Decentralized Identifiers).

Delegation reduction to permission originator only

Avoid ambiguity about origin of delegated authority.

$$\frac{P \overset{\{x\}_{sP}}{\Longrightarrow} Q;\, Q \overset{\{y\}_{sP}}{\Longrightarrow} R;}{P \overset{\{x \sqcap y\}_{sP}}{\Longrightarrow} R}$$

Local Permission Names

Identifying signed permissions is awkward.

$$(K_A\ Clare) \overset{:view.s}{\Longrightarrow} (K_A\ Insurer)$$

Use local permission name $\langle P\ N \rangle$ to identify permission named as N in the namespace of principal P.

$$(K_A\ Clare) \overset{\langle K_A\ \text{view.s}\rangle}{\Longrightarrow} (K_A\ Insurer)$$

with 20+ inference rules ...

Alice's house using local permissions

- Alice permits members in her group to access any device in her house

  $K_A \overset{\langle K_A\ \top\rangle}{\Longrightarrow} (K_A \text{ mbrs});$

  Alice asserts that $\top$ is top permission:

  $\langle K_A\ \mathit{view}.*\rangle \rightsquigarrow \langle K_A\ \top\rangle$

- Bob and Clare are members

$$\begin{array}{rcl} (K_A \text{ mbrs}) & \rightarrow & (K_A \text{ Bob}); \\ (K_A \text{ Bob}) & \rightarrow & (K_B); \\ (K_A \text{ mbrs}) & \rightarrow & K_C; \end{array}$$

- Bob delegates access to wine sensor s to insurance company *Ivan*.

  $K_B \overset{\langle K_A\ \text{view}.s\rangle}{\Longrightarrow} (K_{CA}\ \mathit{GFIA}\ \mathit{Ivan})$

  assuming Alice trusts GIFA views:

  $\langle K_A\ \mathit{view}.*\rangle \rightsquigarrow \langle K_{CA}\ \mathit{GFIA}\ \mathit{view}.*\rangle$

- Insurance company (K_I) fully trusts wine analytics company W,

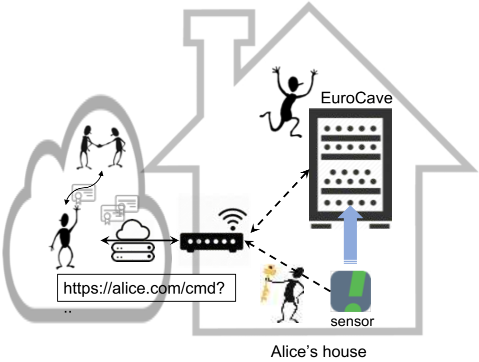
  $K_I \overset{\langle K_{CA}\ \mathit{GFIA}\ \text{view}.*\rangle}{\Longrightarrow} K_W;$

- grants authority to Data Scientist *Steve*

  $K_W \overset{\langle K_{CA}\ \mathit{GFIA}\ \text{view}.*\rangle}{\Longrightarrow} (K_W\ \mathit{Steve})$

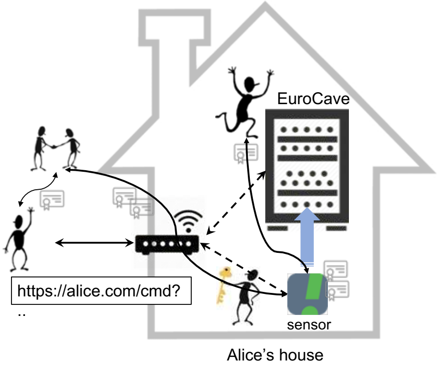
Access control decisions in practice

- Public key infrastructure to manage cryptographic credentials.
- Credential validation requires public key operations.
- Access decisions computationally OK.
- Feasible in cloud, or at Alice's perimeter.



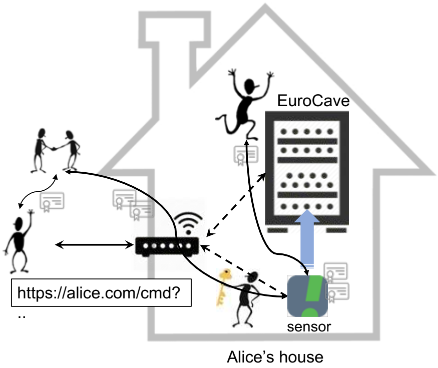
Access control decisions in practice

- Public key infrastructure to manage cryptographic credentials.
- Credential validation requires public key operations.
- Access decisions computationally OK.
- Feasible in cloud, or at Alice's perimeter.
- What if off-line, or we want IoT device to manage authorisation decisions/delegate?



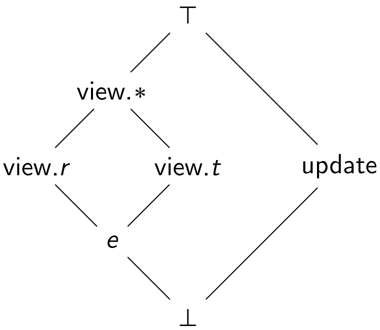
Access control decisions in practice

- Public key infrastructure to manage cryptographic credentials.
- Credential validation requires public key operations.
- Access decisions computationally OK.
- Feasible in cloud, or at Alice's perimeter.
- What if off-line, or we want IoT device to manage authorisation decisions/delegate?
- Want public key-free Access Control.



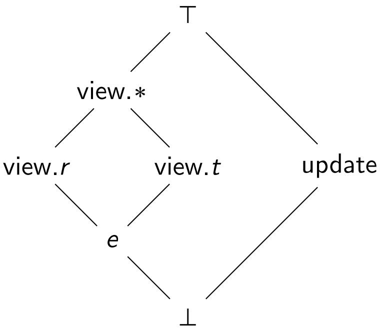
Lightweight Trust Management

Permission Ordering $(Perm, \sqsubseteq)$

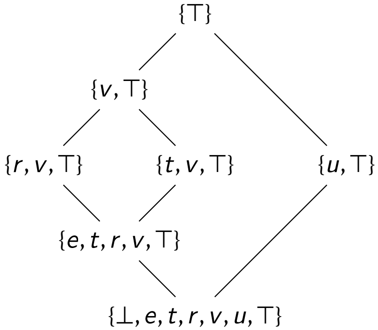


Lightweight Trust Management

Permission Ordering $(Perm, \sqsubseteq)$

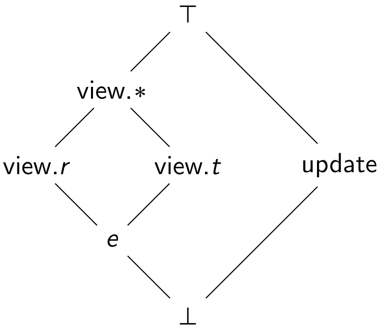


Isomorphism: $\lceil p \rceil = \{q : PERM | p \sqsubseteq q\}$

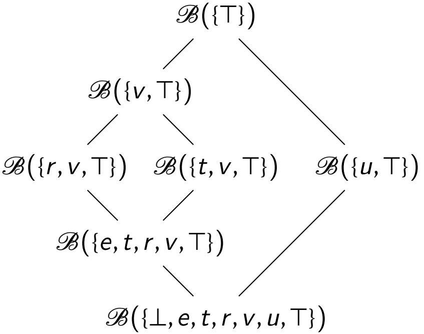


Lightweight Trust Management

Permission Ordering $(Perm, \sqsubseteq)$



Permissions in a Bloom filter $\mathscr{B}(\lceil p \rceil)$



Lightweight Trust Management

Properties of Bloom Filters

- Can check permission ordering

  $x \sqsubseteq y \approx \mathscr{B}(\lceil y \rceil) \subseteq \mathscr{B}(\lceil x \rceil)$

- Compute permission intersection

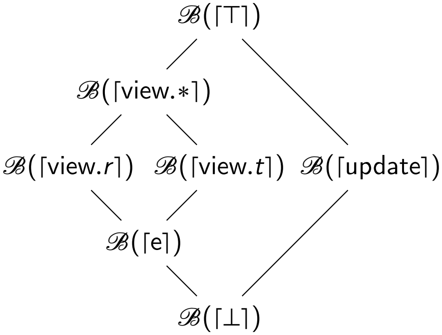
  $x \sqcap y \approx \mathscr{B}(\lceil x \rceil) \cup \mathscr{B}(\lceil y \rceil)$

with high probability assuming good Bloom filter configuration. Cannot with reasonable probability compute permission union

$x \sqcup y \not\approx \mathscr{B}(\lceil x \rceil) \cap \mathscr{B}(\lceil y \rceil)$

or given permission x, compute dominating permission $y \sqsupset x$, without knowing $\top$.

Permissions in a Bloom filter $\mathscr{B}(\lceil p \rceil)$



Using Bloom Permissions as access tokens

Access tokens can be delegated

Delegator holds permission $\mathscr{B}(\lceil y \rceil)$, grants:

$$X = \mathscr{B}(\lceil y \rceil) \sqcup \mathscr{B}(\lceil x \rceil \setminus \{\top\})$$

to recipient to delegate permission $x \sqsubseteq y$, since

$$x \leq y \Rightarrow \mathscr{B}(\lceil x \rceil) = \mathscr{B}(\lceil y \rceil) \sqcup \mathscr{B}(\lceil x \rceil \setminus \{\top\})$$

Access token check

If permission x is required to engage action and bit vector Y is presented, check:

$$\mathscr{B}(\lceil y \rceil) \sqcup \mathscr{B}(\lceil x \rceil \setminus \{\top\})$$

[Could use a lightweight based authentication protocol to prove possession of access token.]

Example

- Device has random secret seed $\top$.
- On first connection, gives $\mathscr{B}(\lceil \top \rceil)$ to its owner (resurrecting duckling).
- Owner, gives $\mathscr{B}(\lceil \text{view}.* \rceil)$ to Bob, who computes/gives

  $$\mathscr{B}(\lceil \text{view}.* \rceil) \sqcup \mathscr{B}(\lceil \text{view}.t \rceil \setminus \{\top\})$$

  to Clare, who presents it as an access token when requesting device access.

Using Bloom Permissions as access tokens

Access tokens can be delegated

Delegator holds permission $\mathscr{B}(\lceil y \rceil)$, grants:

$$X = \mathscr{B}(\lceil y \rceil) \sqcup \mathscr{B}(\lceil x \rceil \setminus \{\top\})$$

to recipient to delegate permission $x \sqsubseteq y$, since

$$x \leq y \Rightarrow \mathscr{B}(\lceil x \rceil) = \mathscr{B}(\lceil y \rceil) \sqcup \mathscr{B}(\lceil x \rceil \setminus \{\top\})$$

Access token check

If permission x is required to engage action and bit vector Y is presented, check:

$$\mathscr{B}(\lceil y \rceil) \sqcup \mathscr{B}(\lceil x \rceil \setminus \{\top\})$$

[Could use a lightweight based authentication protocol to prove possession of access token.]

Example

- Device has random secret seed $\top$.
- On first connection, gives $\mathscr{B}(\lceil \top \rceil)$ to its owner (resurrecting duckling).
- Owner, gives $\mathscr{B}(\lceil \text{view}.* \rceil)$ to Bob, who computes/gives

  $$\mathscr{B}(\lceil \text{view}.* \rceil) \sqcup \mathscr{B}(\lceil \text{view}.t \rceil \setminus \{\top\})$$

  to Clare, who presents it as an access token when requesting device access.

Implemented in HTTP/embedded web server with tokens as cookies. Use Bearer tokens & OAuth, or something else instead?

Related Work

Subterfuge

Trust Management/Decentralized Authorization

Global unsigned permission namespace with conventional reduction: X509 (X500 names), KeyNote (IANA names), RT (Application Domain Specification Documents), ...

Distributed Authorization Language [Zhou2006]

RT-style authorization logic, binds keys to permissions and restricted to originator reduction; subterfuge-freedom conjectured.

Local Permissions [Foley2011]

SPKI/SDSI with SDSI-like local naming scheme for permissions. 20+ deduction rules; subterfuge-freedom conjectured.

Blessings [Abadi 2015]

Uses SDSI to build CCN style permission naming (blessings) for IoT devices. Relies on widely witnessed global security authorities/CAs to provide root names.

Conclusion

Decentralised authorisation for IoT

- Public access credentials.
- Support a web of trust.
- Distributed, no global security authority.
- Revocation can be tricky.
- Public key operations expensive.

Lightweight Trust Management

- Secret access credentials.
- Based on cryptographic hash functions.
- Rely on probabilisitic data structures: useful for non security critical scenarios.
- Complement PK-based scheme, providing security-assurance between devices.

More information

1. Foley, S. N. (2014). *Noninterference Analysis of Delegation Subterfuge in Distributed Authorization Systems*. Journal of Trust Management, 1(11).
2. Foley, S. N., Navarro-Arribas, G. (2013). *A Bloom Filter Based Model for Decentralized Authorization*. Int. J. Intell. Syst., 28(6).
3. Foley, S. N., Abdi, S. (2011). *Avoiding Delegation Subterfuge Using Linked Local Permission Names*. Formal Aspects of Security and Trust, 2011.
4. Zhou, H., Foley, S. N. (2006). A Framework for Establishing Decentralized Secure Coalitions. In 19th IEEE Computer Security Foundations Workshop, (CSFW-19 2006), 2006.
5. Foley, S. N., Zhou, H. (2005). *Authorisation Subterfuge by Delegation in Decentralised Networks*. In Security Protocols Workshop, 2005