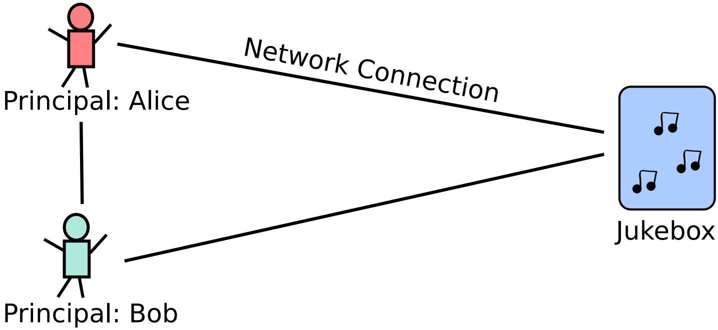
AURA: Programming with authorization and audit

Jeff Vaughan

Department of Computer and Information Science
University of Pennsylvania

Thesis Defense
September 28, 2009

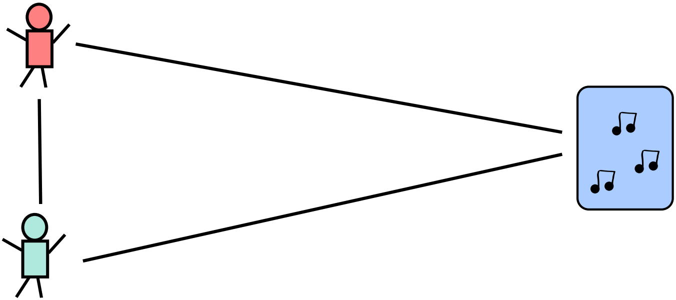
A distributed access control example



Jukebox's signature:

$$\text{playFor\_raw: (s: Song)} \rightarrow \text{(p: } \mathbf{prin}\text{)} \rightarrow \text{Mp3Of s}$$

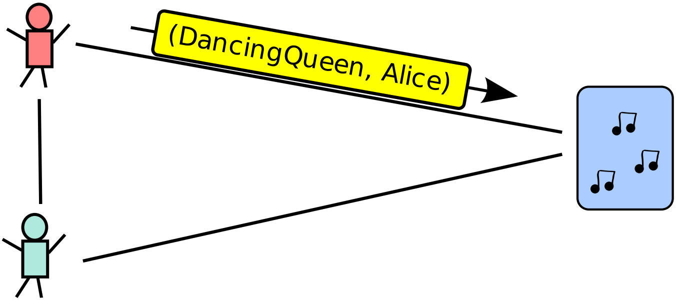
A distributed access control example



Jukebox's signature:

playFor_raw: (s: Song) → (p: **prin**) → Mp3Of s

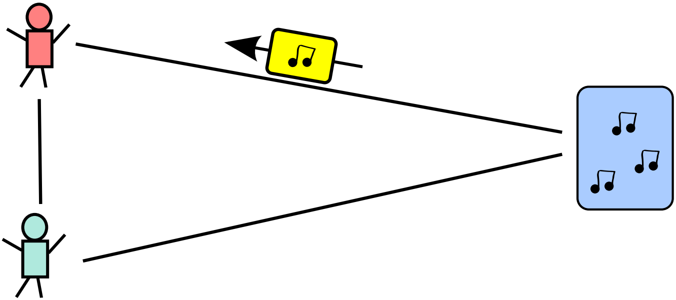
A distributed access control example



Jukebox's signature:

playFor_raw: (s: Song) $\rightarrow$ (p: **prin**) $\rightarrow$ Mp3Of s

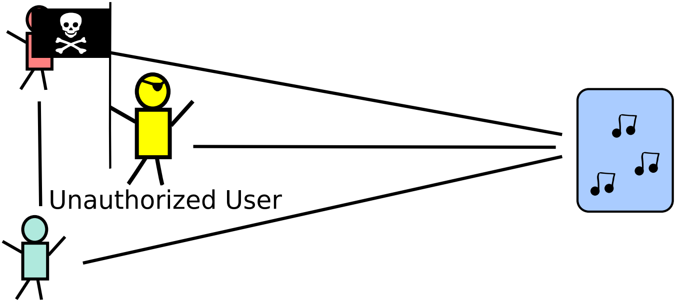
A distributed access control example



Jukebox's signature:

playFor_raw: (s : Song) $\to$ (p : **prin**) $\to$ Mp3Of s

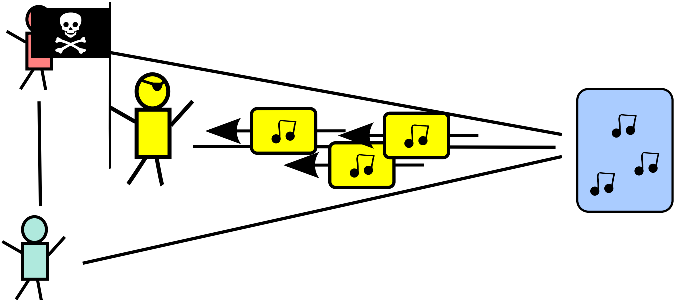
A distributed access control example



Jukebox's signature:

playFor_raw: (s: Song) $\rightarrow$ (p: **prin**) $\rightarrow$ Mp3Of s

A distributed access control example



Jukebox's signature:

$$\text{playFor\_raw}: (s: \text{Song}) \rightarrow (p: \textbf{prin}) \rightarrow \text{Mp3Of } s$$

International Cartel for Fonograph Players Policy

Policy Statement (Simple):

- Songs have one or more owners.
- An owner may authorize principals to play songs he owns.

International Cartel for Fonograph Players Policy

Policy Statement (Simple):

- Songs have one or more owners.
- An owner may authorize principals to play songs he owns.

Policy Enforcement Problems (Hard):

- distributed decision making
- mutual distrust
- prominent use of delegation

AURA: Enforce policy with proof carrying access control.

- Programs build *proofs* attesting to their access rights.
- Proof components
  - standard rules of inference
  - *evidence* capturing principal intent (e.g. signatures)
- AURA runtime:
  - checks proof structure (well-typedness)
  - logs appropriate proofs for later *audit*

Proof Carrying Code [Necula+ '98], Grey Project [Bauer+ '05], Protocol Analysis [Fournet+ '07], Aura [CSF '08, ICFP '08]

Encoding policy at the ICFP server

$$
\begin{aligned}
\mathsf{shareRule} \equiv \mathsf{ICFP}\ \textbf{says}\ (\\
&(\mathsf{o}\colon \textbf{prin}) \to (\mathsf{s}\colon \mathsf{Song}) \to (\mathsf{r}\colon \textbf{prin}) \to \\
&(\mathsf{Owns\ o\ s}) \to \\
&(\mathsf{o}\ \textbf{says}\ (\mathsf{MayPlay\ r\ s})) \to \\
&(\mathsf{MayPlay\ r\ s})))
\end{aligned}
$$

$$
\begin{aligned}
\mathsf{playFor}\colon (\mathsf{s}\colon \mathsf{Song}) \to (\mathsf{p}\colon \textbf{prin}) \to \\
&\textbf{pf}\ (\mathsf{ICFP}\ \textbf{says}\ (\mathsf{MayPlay\ p\ s})) \to \mathsf{Mp3Of\ s}
\end{aligned}
$$

Encoding policy at the ICFP server

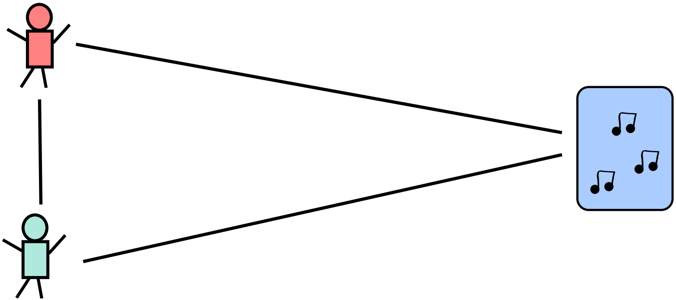
$$\begin{aligned}\text{shareRule} \equiv \text{ICFP } \textbf{says } (&\\ &(\text{o}: \textbf{prin}) \to (\text{s}: \text{Song}) \to (\text{r}: \textbf{prin}) \to \\ &(\text{Owns o s}) \to \\ &(\text{o } \textbf{says } (\text{MayPlay r s})) \to \\ &(\text{MayPlay r s})))\end{aligned}$$

$$\begin{aligned}\text{playFor}: (\text{s}: \text{Song}) \to (\text{p}: \textbf{prin}) \to &\\ \textbf{pf } (\text{ICFP } \textbf{says } (\text{MayPlay p s})) \to \text{Mp3Of s}&\end{aligned}$$

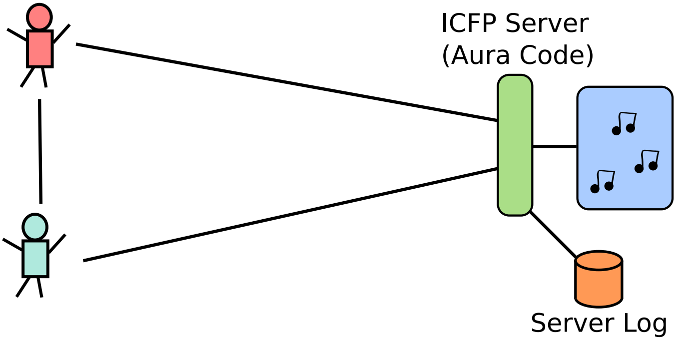
Key Property

A program can only call playFor when it has an appropriate access control proof.

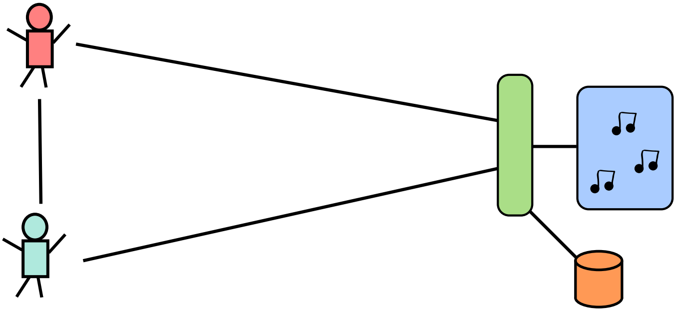
Using the ICFP policy.



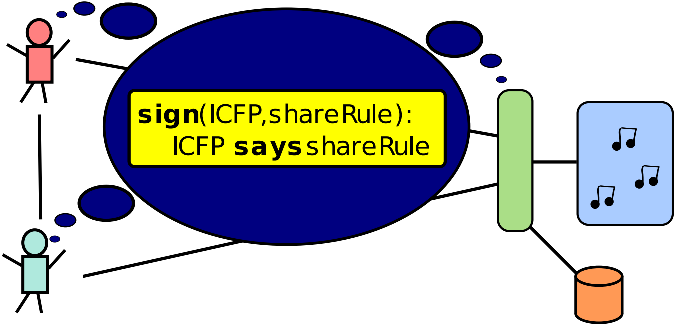
Using the ICFP policy.



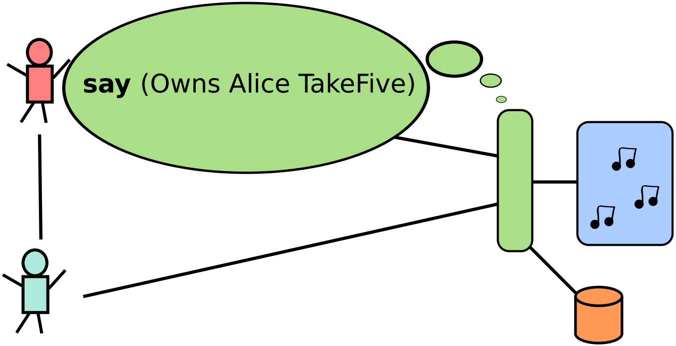
Using the ICFP policy.



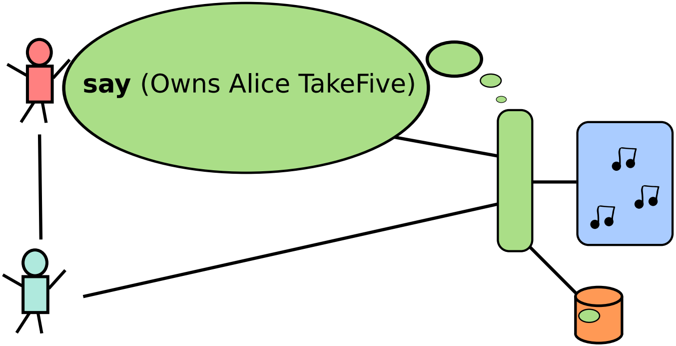
Using the ICFP policy.



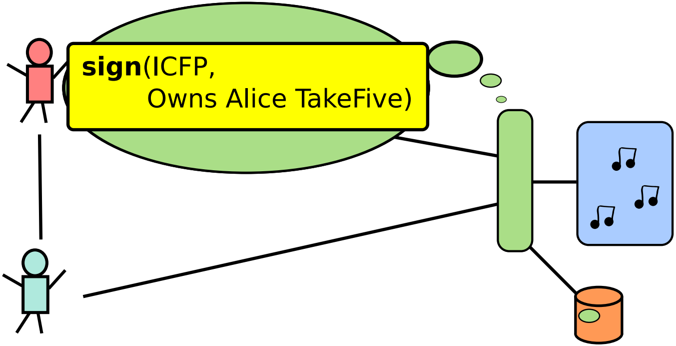
Using the ICFP policy.



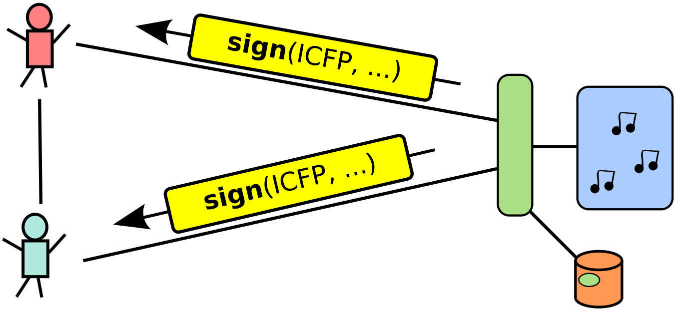
Using the ICFP policy.



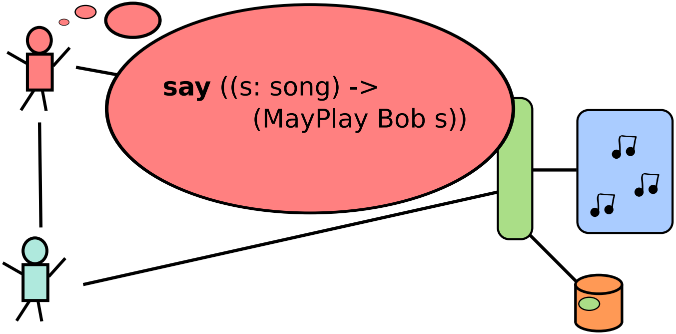
Using the ICFP policy.



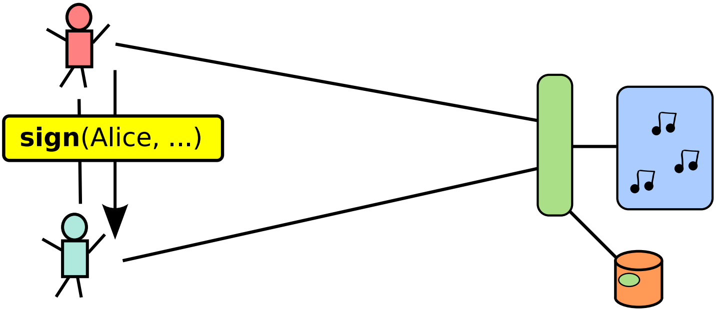
Using the ICFP policy.



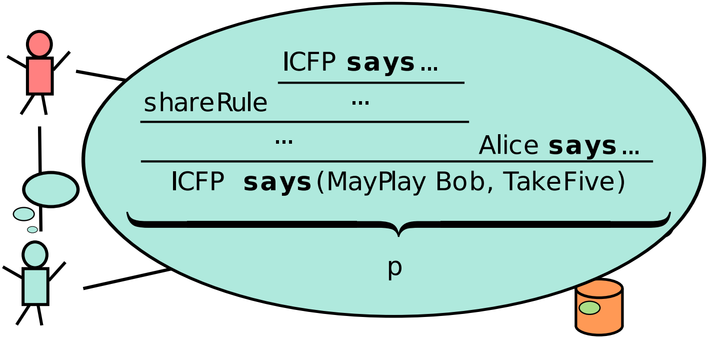
Using the ICFP policy.



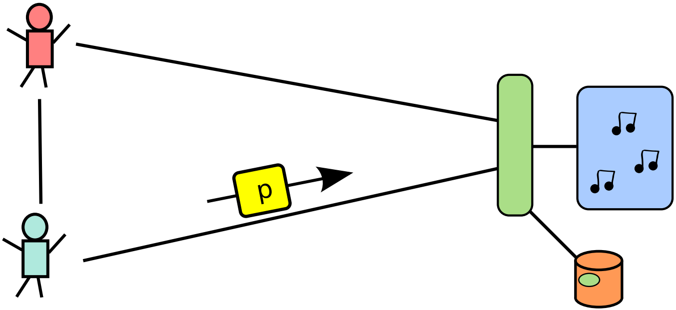
Using the ICFP policy.



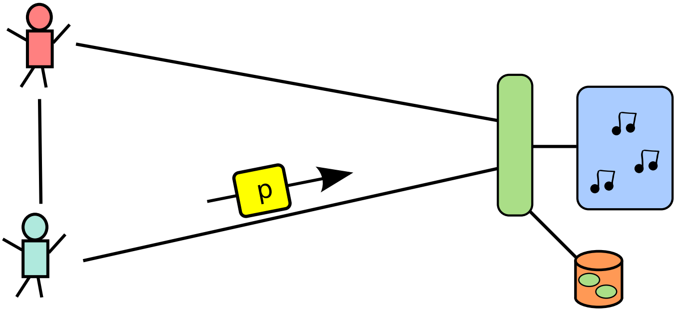
Using the ICFP policy.



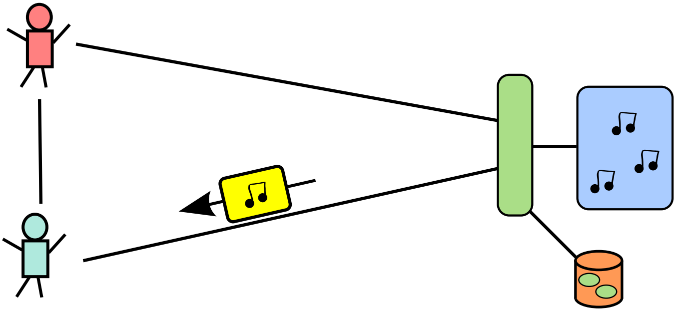
Using the ICFP policy.



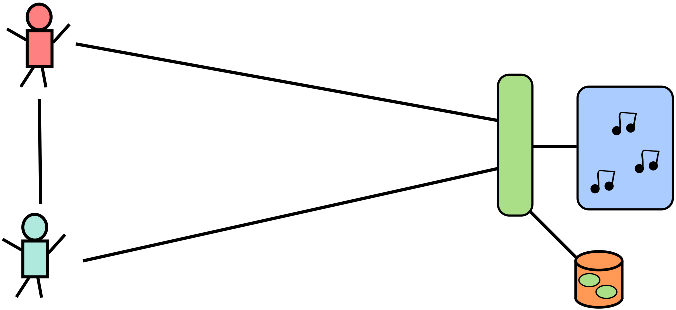
Using the ICFP policy.



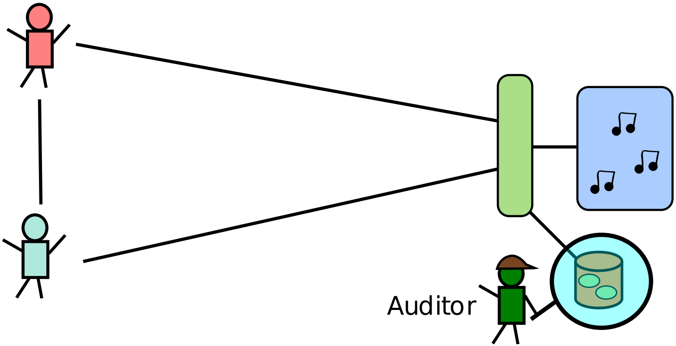
Using the ICFP policy.



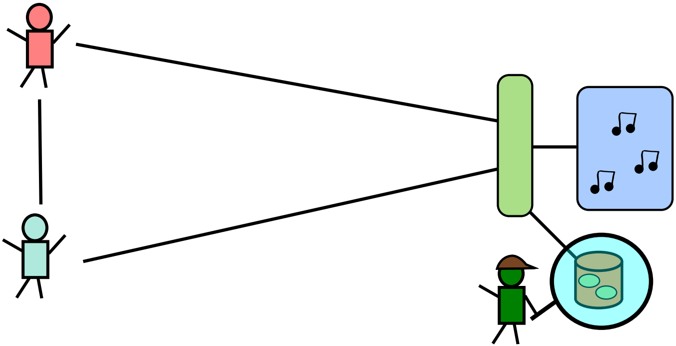
Using the ICFP policy.



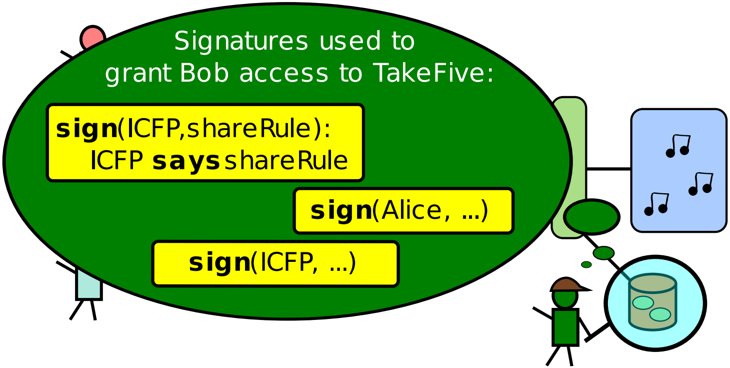
Using the ICFP policy.



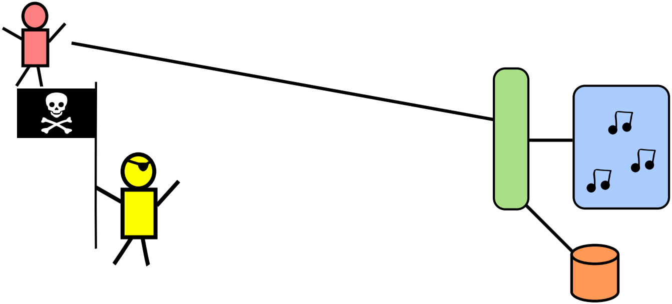
Using the ICFP policy.



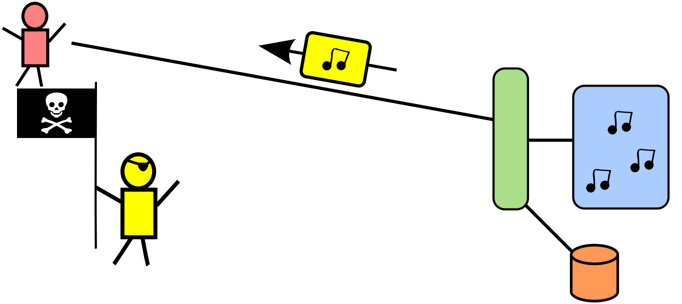
Using the ICFP policy.



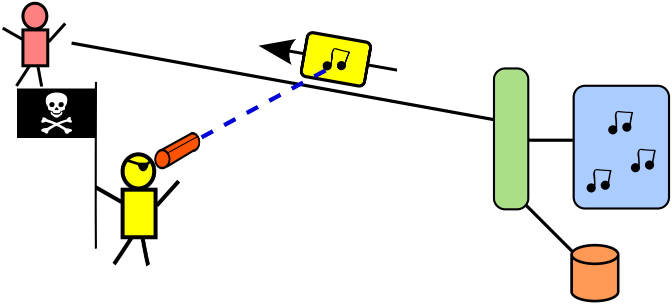
Access control alone can't ensure some properties.



Access control alone can't ensure some properties.



Access control alone can't ensure some properties.



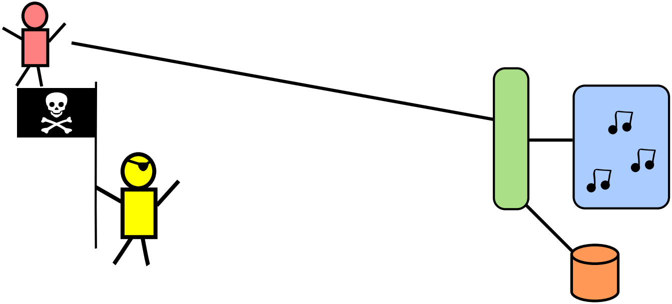
AURA$_{conf}$ protects confidential data.

- Types provide a formal description of confidentiality policy.
- Encryption provides an enforcement mechanism.
- Encryption works the level of (lazy) data values—not communication channels.

Design Motivation

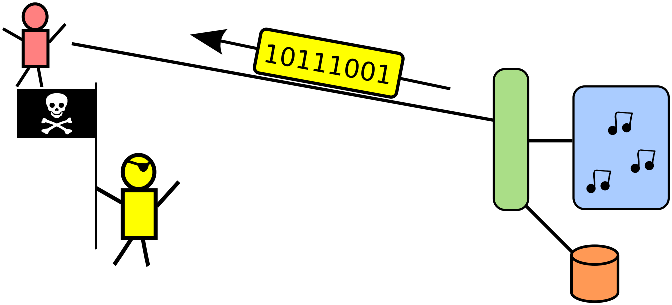
Secure sessions are transient.
Secure data is persistent.

for types described encrypted data.



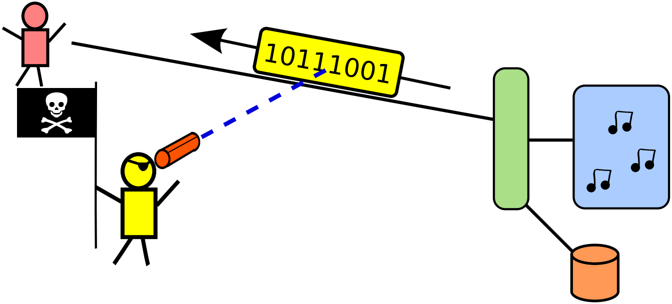
playForEnc: (s: Song) $\to$ (p: **prin**) $\to$
pf (ICFP **says** MayPlay p s) $\to$
(Mp3Of s) **for** p

for types described encrypted data.



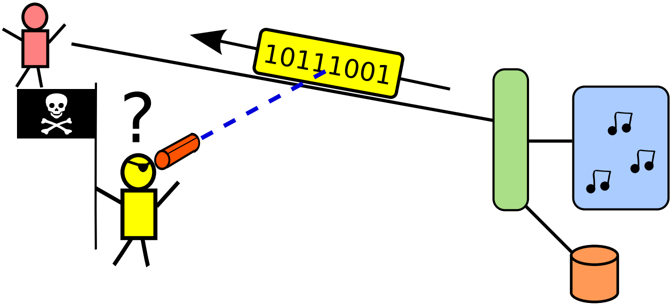
playForEnc: (s: Song) → (p: **prin**) →
pf (ICFP **says** MayPlay p s) →
(Mp3Of s) **for** p

for types described encrypted data.



playForEnc: (s: Song) $\rightarrow$ (p: **prin**) $\rightarrow$
pf (ICFP **says** MayPlay p s) $\rightarrow$
(Mp3Of s) **for** p

for types described encrypted data.



playForEnc: (s: Song) $\rightarrow$ (p: **prin**) $\rightarrow$
pf (ICFP **says** MayPlay p s) $\rightarrow$
(Mp3Of s) **for** p

Outline

Review of Core AURA

Aura's says modality represents affirmation.

- The proposition "principal Alice affirms proposition P."

  $$\text{Alice } \textbf{says}\ \text{P} : \textbf{Prop}$$

- Principals may actively affirm propositions with signatures.

  $$\textbf{sign}(\text{Alice},\ \text{P}) :\ \text{Alice } \textbf{says}\ \text{P}$$

- Principals affirm "true" propositions

  $$\textbf{return}\ \text{Alice}\ \ \text{p} :\ \text{Alice } \textbf{says}\ \text{P}$$

  when p: P.

DCC [Abadi+ '06], Logic with Explicit Time [DeYoung+ '08]

Assertions define access control predicates.

Example (Example: An assertion definition)

assert Owns: **prin** $\to$ Song $\to$ **Prop**

- Intuition: Assertions $\approx$ type variables.
- Assertions have no introduction form.
  - Owns is uninhabited
  - But A **says** Owns B S is inhabited by **sign**s.
- Assertions have no elimination form.
  - There are no "naive" proofs of

    ICFP **says** (Owns Bob Thriller) $\to$
    (P:**Prop**) $\to$ ICFP **says** P.

  - cf. Noninterference in DCC [Abadi '07]

Dependent types allow for expressive rules.

Example (Bob acts for Alice)

Alice **says** ((P: **Prop**) $\rightarrow$ Bob **says** P $\rightarrow$ P)

Dependent types allow for expressive rules.

Example (Bob acts for Alice)

$$\text{Alice } \textbf{says } ((\text{P}: \textbf{Prop}) \rightarrow \text{Bob } \textbf{says } \text{P} \rightarrow \text{P})$$

Example (Bob acts for Alice only regarding jazz)

$$\text{Alice } \textbf{says } ((\text{s}: \text{Song}) \rightarrow \text{isJazz s} \rightarrow \\ \text{Bob } \textbf{says } (\text{MayPlay Bob s}) \rightarrow \text{MayPlay Bob s})$$

Dependent types allow for expressive rules.

Example (Bob acts for Alice)

$$\text{Alice } \textbf{says } ((\text{P}: \textbf{Prop}) \to \text{Bob } \textbf{says } \text{P} \to \text{P})$$

Example (Bob acts for Alice only regarding jazz)

$$\begin{aligned}&\text{Alice } \textbf{says } ((\text{s}: \text{Song}) \to \text{isJazz s} \to \\ &\qquad \text{Bob } \textbf{says } (\text{MayPlay Bob s}) \to \text{MayPlay Bob s})\end{aligned}$$

Restricted formulation of dependent types:

- expressive enough for access control and confidentiality
- too weak for general correctness properties
- AURA feels more like ML than Coq

Programs build proofs explicitly.

- A baked-in proof search algorithm would either limit the logic's expressiveness (e.g. no quantifiers) or be incomplete.
- Expressive first-, and higher-, order predicates are useful.
- Applications can build specialized heuristics for proof search.

Design Principle

Don't let proof search mechanism constrain policy definitions.

Access control systems can be too restrictive.

The HypothetIcal Patient Privacy Act:

- A patient chooses who may read his chart.

$$(\text{patient} : \textbf{prin}) \to (\text{a} : \textbf{prin}) \to (\text{c} : \text{chart patient})$$
$$\to \text{patient } \textbf{says} \text{ (MayRead a c)}$$
$$\to \text{HIPPA } \textbf{says} \text{ (MayRead a c)}$$

- Doctors can read their patients' charts.

$$(\text{patient} : \textbf{prin}) \to (\text{d} : \textbf{prin}) \to (\text{DoctorOf patient d})$$
$$\to (\text{c} : \text{chart patient})$$
$$\to \text{HIPPA } \textbf{says} \text{ (MayRead d c)}$$

Access control systems can be too restrictive.

The HypothetIcal Patient Privacy Act:

- A patient chooses who may read his chart.

$$\begin{array}{l}(\text{patient}: \textbf{prin}) \rightarrow (\text{a}: \textbf{prin}) \rightarrow (\text{c}: \text{chart patient}) \\ \quad \rightarrow \text{patient } \textbf{says} \text{ (MayRead a c)} \\ \quad \rightarrow \text{HIPPA } \textbf{says} \text{ (MayRead a c)}\end{array}$$

- Doctors can read their patients' charts.

$$\begin{array}{l}(\text{patient}: \textbf{prin}) \rightarrow (\text{d}: \textbf{prin}) \rightarrow (\text{DoctorOf patient d}) \\ \quad \rightarrow (\text{c}: \text{chart patient}) \\ \quad \rightarrow \text{HIPPA } \textbf{says} \text{ (MayRead d c)}\end{array}$$

What happens in an emergency when the patient and designated doctors are not available?

Audit enables escape hatches in access control.

emergency: (patient: **prin**) $\rightarrow$ (a: **prin**)
$\rightarrow$ (c: chart patient)
$\rightarrow$ (reason: string)
$\rightarrow$ HIPPA **says** (MayRead a c)

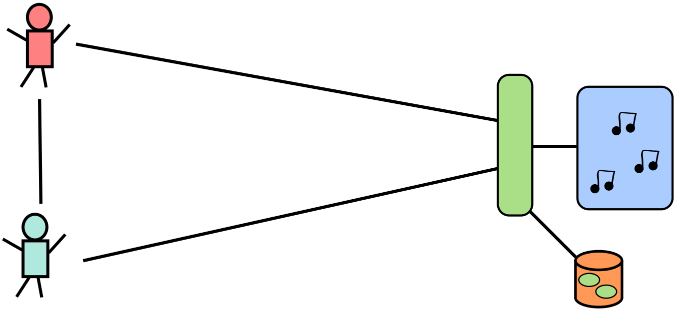
Audit enables escape hatches in access control.

emergency: (patient: **prin**) $\rightarrow$ (a: **prin**)
$\rightarrow$ (c: chart patient)
$\rightarrow$ (reason: string)
$\rightarrow$ HIPPA **says** (MayRead a c)

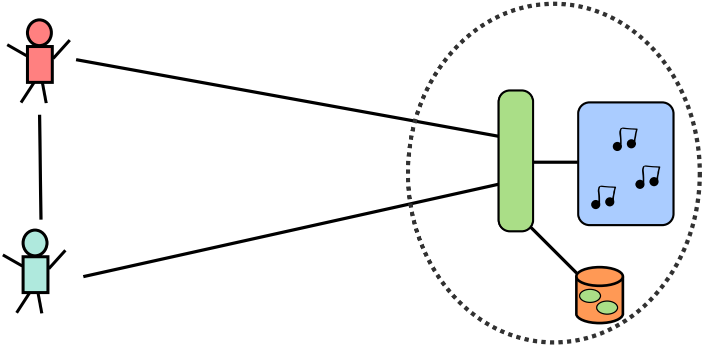
Justification

Logged actions can be evaluated after the fact by social, administrative or legal means—worthwhile when a false deny may be worse than a false allow.

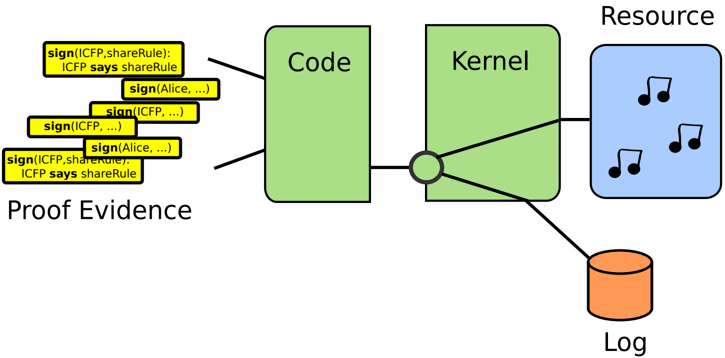
Using evidence minimizes the trusted computing base.



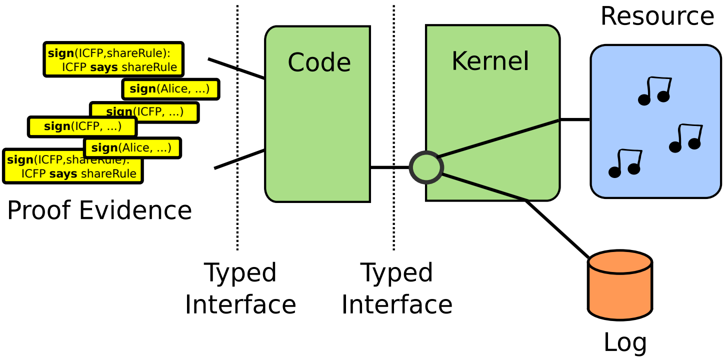
Using evidence minimizes the trusted computing base.



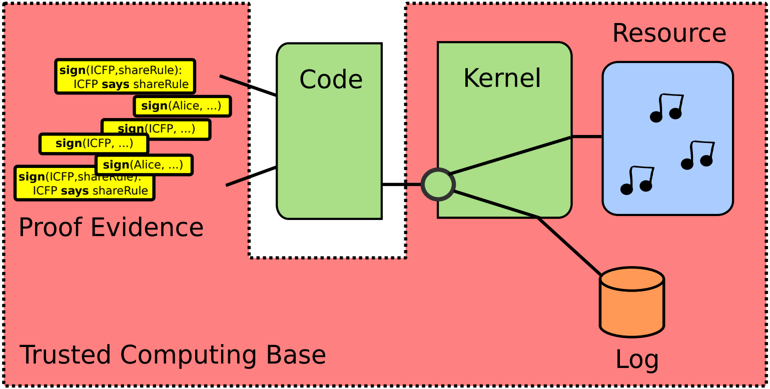
Using evidence minimizes the trusted computing base.



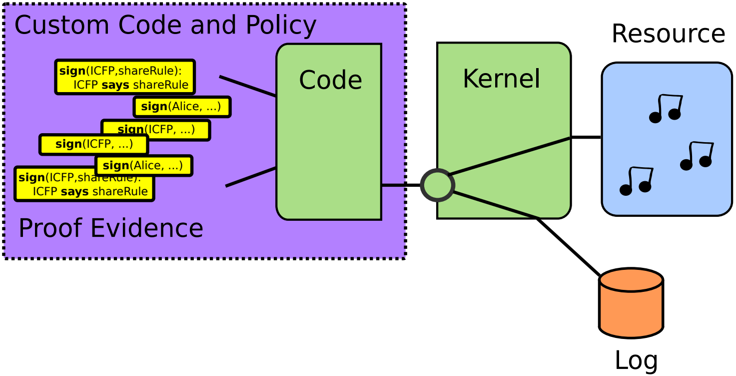
Using evidence minimizes the trusted computing base.



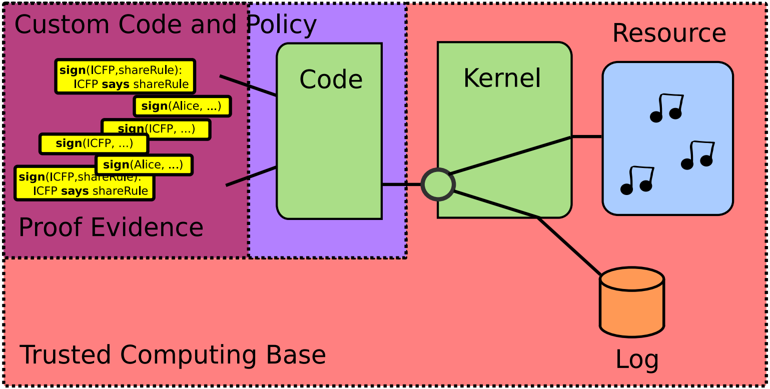
Using evidence minimizes the trusted computing base.



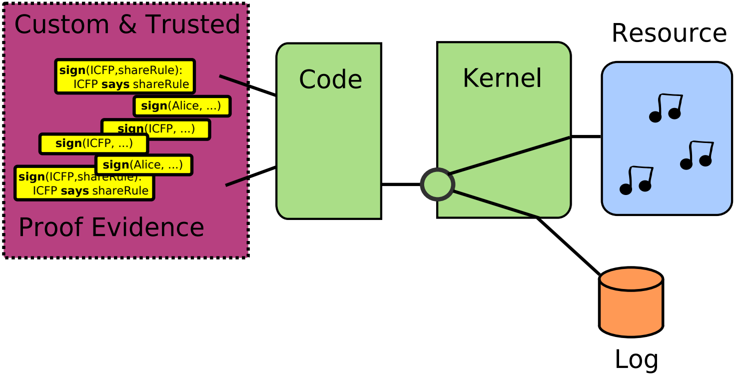
Using evidence minimizes the trusted computing base.



Using evidence minimizes the trusted computing base.



Using evidence minimizes the trusted computing base.



A Confidentiality Extension for AURA

AURA$_{\text{conf}}$ adds support for confidential data to AURA

- The real-world contains lots of confidential information.
  - Financial, medical, social data …
  - Data leaks have consequences: legal, financial….
- Goals of AURA$_{\text{conf}}$
  - Establish a natural connection between confidential expressions and cryptography.
  - Minimize disruptive changes to AURA's design.
    - Avoid straining the complexity budget for end-users.
    - (But Coq proofs help us manage meta-theoretic complexity.)
  - Provide for relevant auditing—decryption failures are interesting.

There is a large, partially explored, design space.

Notable approaches to confidentiality in distributed settings:

- Direct use of cryptography
  - Applied Crytpo. [Schneier '96]
- Language operations supporting cryptography
  - Spi Calculus [Abadi+ '98], λ_{seal} [Sumii+ '04]
- Information flow + explicit cryptography
  - Key-Based DLM [Chothia+ '03], [Askarov+ '06]
- Declarative policy enforcement by automatic encryption
  - SImp [Oakland '06]

None of these are good fits with AURA.

AURA$_{conf}$ represents confidentiality monadically: return.

return Alice 42: int **for** Alice

AURA$_{conf}$ represents confidentiality monadically: return.

return Alice 42: int **for** Alice

$\rightsquigarrow$

$\mathscr{E}$(Alice, 42, 0x32A3)
and some metadata

AURA$_{\text{conf}}$ represents confidentiality monadically: run.

run (**return** Alice 42): int

AURA$_{\text{conf}}$ represents confidentiality monadically: run.

run (**return** Alice 42): int

$\rightsquigarrow$

42

AURA$_{conf}$ represents confidentiality monadically: run.

$$\textbf{run}\ (\textbf{return}\ \text{Alice}\ \ 42) : \ \text{int}$$

$$\rightsquigarrow$$

$$42$$

- **run** can fail on “bad” ciphertext.
- **run** $e \rightsquigarrow e'$ where e' blames p.

$\text{AURA}_{\text{conf}}$ represents confidentiality monadically: return.

$$
\begin{array}{l}
\textbf{bind}\ (\text{int}\ \ \textbf{for}\ \ \text{Alice}) \\
\qquad (\textbf{return}\ \text{Alice}\ \ 21) \\
\qquad (\lambda\{\_\}\ \text{x}:\ \text{int}\,.\ \ \textbf{return}\ \text{Alice}\ \ (2{*}\text{x})) \\
\qquad\quad :\ \text{int}\ \ \textbf{for}\ \ \text{Alice}
\end{array}
$$

AURA$_{\text{conf}}$ represents confidentiality monadically: return.

bind (int **for** Alice)
(**return** Alice 21)
($\lambda\{\_\}$ x: int . **return** Alice (2$*$x))
: int **for** Alice

$\rightsquigarrow$

$\mathscr{E}$(Alice,
($\lambda\{\_\}$ x: int . **return** 2$*$x) (**run** $\mathscr{E}$(Alice, 21, 0x32A4))
0x32A3)
and some metadata

$\text{A}\textsc{ura}_{\text{conf}}$ represents confidentiality monadically: return.

$$
\begin{array}{l}
\textbf{bind}\ (\text{int}\ \ \textbf{for}\ \ \text{Alice}) \\
\qquad (\textbf{return}\ \text{Alice}\ \ 21) \\
\qquad (\lambda\{\_\}\ \text{x}:\ \text{int}\,.\ \ \textbf{return}\ \text{Alice}\ \ (2{*}\text{x})) \\
\qquad\quad :\ \text{int}\ \ \textbf{for}\ \ \text{Alice}
\end{array}
$$

$$\rightsquigarrow$$

$$
\begin{array}{l}
\mathscr{E}(\text{Alice}, \\
\quad (\lambda\{\_\}\ \text{x}:\ \text{int}\,.\ \ \textbf{return}\ 2{*}\text{x})\ (\textbf{run}\ \mathscr{E}(\text{Alice},\ 21,\ \text{0x32A4})) \\
\quad \text{0x32A3})
\end{array}
$$

and some metadata

$$\approx \mathscr{E}(\text{Alice},\ 42,\ \text{0x32A5})$$

and some metadata

The tension in $\text{A}\textsc{ura}_{\text{conf}}$'s design.

Expression e contains secrets. Clients analyzing e is:

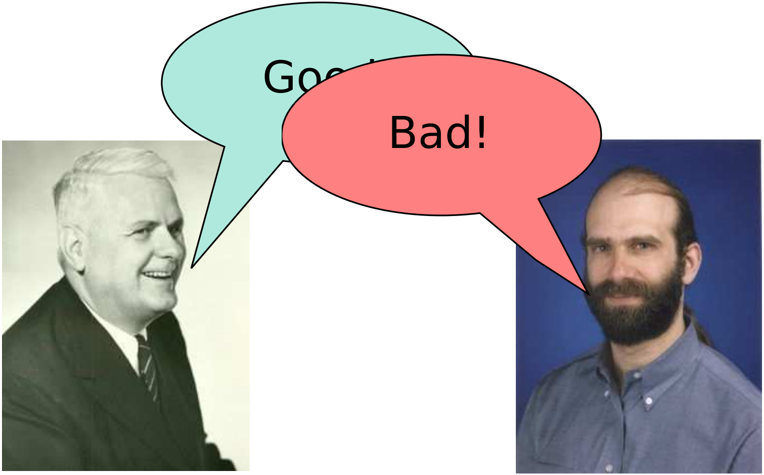
The tension in AURA$_{\text{conf}}$'s design.

Expression *e* contains secrets. Clients analyzing *e* is:

Type Theorist

The tension in AURA$_{\text{conf}}$'s design.

Expression *e* contains secrets. Clients analyzing *e* is:



Type Theorist

Cryptographer

Challenge 1: Typing is relative.

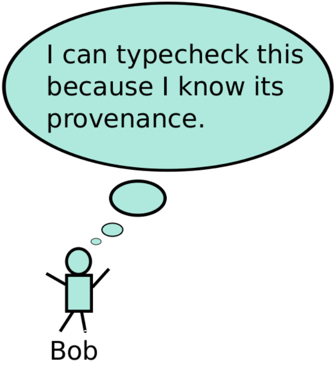
return Alice "toaster"

Challenge 1: Typing is relative.

$\mathscr{E}$(Alice, "toaster", 0x0312)

Challenge 1: Typing is relative.

$\mathscr{E}$(Alice, "toaster", 0x0312)



Challenge 1: Typing is relative.

$\mathscr{E}$(Alice, "toaster", 0x0312)

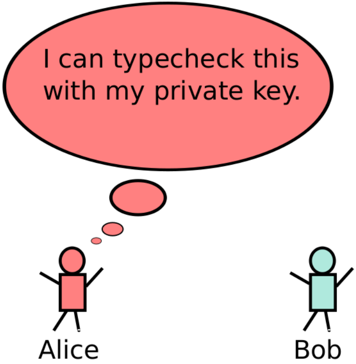
Bob

Challenge 1: Typing is relative.

$$\mathscr{E}(\text{Alice}, \text{"toaster"}, \text{0x0312})$$

Challenge 1: Typing is relative.

$\mathscr{E}$(Alice, "toaster", 0x0312)



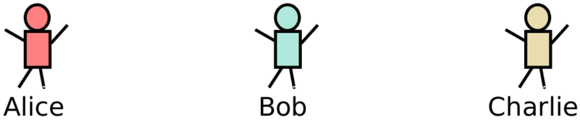
Challenge 1: Typing is relative.

$\mathscr{E}$(Alice, "toaster", 0x0312)

Alice

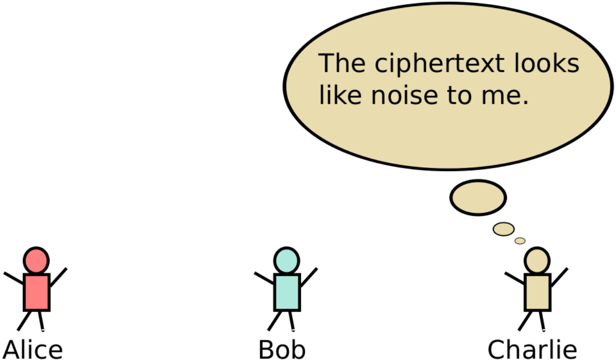
Bob

$\mathscr{E}$(Alice, "toaster", 0x0312)



Challenge 1: Typing is relative.

$\mathscr{E}$(Alice, "toaster", 0x0312)



Challenge 1: Typing is relative.

$\mathscr{E}$(Alice, "toaster", 0x0312)

Metadata guides typing of ciphertexts.

- $\mathscr{E}$(a, e, n): **bits**, always.

Metadata guides typing of ciphertexts.

- $\mathscr{E}$(a, e, n): **bits**, always.
- **cast** $\mathscr{E}$(a, e, n) **to** (int **for** Alice): int **for** Alice
  - A *true cast*
  - Possible if typechecker can statically decrypt $\mathscr{E}$(a,e,n).
  - Also possible if the typechecker has a prerecorded *fact*, attesting to the form of $\mathscr{E}$(a,e,n).

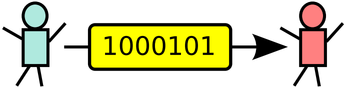
Metadata guides typing of ciphertexts.

- $\mathscr{E}$(a, e, n): **bits**, always.
- **cast** $\mathscr{E}$(a, e, n) **to** (int **for** Alice): int **for** Alice
  - A *true cast*
  - Possible if typechecker can statically decrypt $\mathscr{E}$(a,e,n).
  - Also possible if the typechecker has a prerecorded *fact*, attesting to the form of $\mathscr{E}$(a,e,n).
- **cast** $\mathscr{E}$(a, e, n) **to** (int **for** Alice) **blaming** p: int **for** Alice
  - A *justified cast*
  - Valid when p: c **says** ($\mathscr{E}$(a,e,n) **isa** (int **for** Alice)).

Challenge 2: Keys effect static & dynamic semantics.

- Dynamic semantics
  - Keys are required at runtime to implement **run** and **say**.
  - Type-and-effect analysis tracks these keys.
  - FX [Lucassen+ ’88], foundations [Talpin+ ’92]
- Static semantics
  - True casts need keys at *compile* time for typechecking.
  - Tracked using ideas from modal type systems.
  - Modal Proofs as Distributed Programs [Jia+ 04], ML5 [Murphy ’08]
- Combining these analyses is interesting!

Challenge 3: Typing exhibits hysteresis.



- Consider Bob preparing a confidential message for Alice

$$\textbf{return}\ \textit{Alice}\ 3 \quad \rightsquigarrow \quad \textbf{cast}\ \mathscr{E}(-)\ \textbf{to}\ \textit{int}\ \textbf{for}\ \textit{Alice}$$

- Naively: Bob lacks Alice's private key—he can't typecheck this.
- Evaluation creates new facts to guide the typechecker.
  - Ensures preservation holds.

Anatomy of the typing relation.

$$\Sigma; \mathscr{F}; W; \Gamma; U; V \vdash e : t$$

Anatomy of the typing relation.

$$\Sigma; \mathscr{F}; W; \Gamma; U; V \vdash e : t$$

- e has type t w.r.t. Γ's free variables and Σ's type definitions.

Anatomy of the typing relation.

$$\Sigma; \mathscr{F}; W; \Gamma; U; V \vdash e : t$$

- e has type t w.r.t. Γ's free variables and Σ's type definitions.
- Facts in $\mathscr{F}$ summarize knowledge about ciphertexts.

Anatomy of the typing relation.

$$\Sigma; \mathscr{F}; W; \Gamma; U; V \vdash e : t$$

- *e* has type *t* w.r.t. Γ's free variables and Σ's type definitions.
- Facts in $\mathscr{F}$ summarize knowledge about ciphertexts.
- *Statically available key* W indicates keys available for typechecking.

Anatomy of the typing relation.

$$\Sigma; \mathscr{F}; W; \Gamma; \color{red}{U}; V \vdash e : t$$

- e has type t w.r.t. Γ's free variables and Σ's type definitions.
- Facts in $\mathscr{F}$ summarize knowledge about ciphertexts.
- *Statically available key* W indicates keys available for typechecking.
- *Soft decryption limit* U specifies a subset of W safe to use currently.

Anatomy of the typing relation.

$$\Sigma; \mathscr{F}; W; \Gamma; U; {\color{red}V} \vdash e : t$$

- *e* has type *t* w.r.t. Γ's free variables and Σ's type definitions.
- Facts in $\mathscr{F}$ summarize knowledge about ciphertexts.
- *Statically available key* W indicates keys available for typechecking.
- *Soft decryption limit* U specifies a subset of *W* safe to use currently.
- *Effects label* V summarizes the keys needed to run *e*.

Anatomy of the typing relation.

$$\Sigma; \mathscr{F}; W; \Gamma; U; V \vdash e : t$$

- e has type t w.r.t. Γ's free variables and Σ's type definitions.
- Facts in $\mathscr{F}$ summarize knowledge about ciphertexts.
- *Statically available key* W indicates keys available for typechecking.
- *Soft decryption limit* U specifies a subset of W safe to use currently.
- *Effects label* V summarizes the keys needed to run e.

Sample typing judgments and non-judgments.

Example (Statically available keys)

$$\Sigma; \cdot; \mathsf{Bob}; \cdot; \mathsf{Bob}; \bot \vdash \textbf{cast}\ \mathscr{E}(\mathsf{Bob}, 7, -)\ \textbf{to}\ \mathsf{int}\ \textbf{for}\ \mathsf{Bob} : \mathsf{int}\ \textbf{for}\ \mathsf{Bob}$$

Sample typing judgments and non-judgments.

Example (Statically available keys)

$$\Sigma; \cdot; \mathsf{Bob}; \cdot; \mathsf{Bob}; \bot \vdash \textbf{cast}\, \mathscr{E}(\mathsf{Bob}, 7, -)\ \textbf{to}\ \mathsf{int}\ \textbf{for}\ \mathsf{Bob} : \mathsf{int}\ \textbf{for}\ \mathsf{Bob}$$

$$\Sigma; \cdot; \bot; \cdot; \mathsf{Bob}; \bot \nvdash \textbf{cast}\, \mathscr{E}(\mathsf{Bob}, 7, -)\ \textbf{to}\ \mathsf{int}\ \textbf{for}\ \mathsf{Bob} : \mathsf{int}\ \textbf{for}\ \mathsf{Bob}$$

Sample typing judgments and non-judgments.

Example (Statically available keys)

$$\Sigma; \cdot; \mathsf{Bob}; \cdot; \mathsf{Bob}; \bot \vdash \textbf{cast}\ \mathscr{E}(\mathsf{Bob}, 7, -)\ \textbf{to}\ \mathsf{int}\ \textbf{for}\ \mathsf{Bob} : \mathsf{int}\ \textbf{for}\ \mathsf{Bob}$$

$$\Sigma; \cdot; \bot; \cdot; \mathsf{Bob}; \bot \nvdash \textbf{cast}\ \mathscr{E}(\mathsf{Bob}, 7, -)\ \textbf{to}\ \mathsf{int}\ \textbf{for}\ \mathsf{Bob} : \mathsf{int}\ \textbf{for}\ \mathsf{Bob}$$

$$\Sigma; \cdot; \mathsf{Bob}; \cdot; \bot; \bot \nvdash \textbf{cast}\ \mathscr{E}(\mathsf{Bob}, 7, -)\ \textbf{to}\ \mathsf{int}\ \textbf{for}\ \mathsf{Bob} : \mathsf{int}\ \textbf{for}\ \mathsf{Bob}$$

Sample typing judgments and non-judgments.

Example (Statically available keys)

$$\Sigma; \cdot; \text{Bob}; \cdot; \text{Bob}; \bot \vdash \textbf{cast}\, \mathscr{E}(\text{Bob}, 7, -)\ \textbf{to}\ \text{int}\ \textbf{for}\ \text{Bob} : \text{int}\ \textbf{for}\ \text{Bob}$$

$$\Sigma; \cdot; \bot; \cdot; \text{Bob}; \bot \nvdash \textbf{cast}\, \mathscr{E}(\text{Bob}, 7, -)\ \textbf{to}\ \text{int}\ \textbf{for}\ \text{Bob} : \text{int}\ \textbf{for}\ \text{Bob}$$
$$\Sigma; \cdot; \text{Bob}; \cdot; \bot; \bot \nvdash \textbf{cast}\, \mathscr{E}(\text{Bob}, 7, -)\ \textbf{to}\ \text{int}\ \textbf{for}\ \text{Bob} : \text{int}\ \textbf{for}\ \text{Bob}$$

Example (Facts)

Suppose $\mathscr{E}(\text{Bob}, 7, -) : \text{int}\ \textbf{for}\ \text{Bob} \in \mathscr{F}$,

$$\Sigma; \mathscr{F}; \bot; \cdot; \text{Bob}; \bot \vdash \textbf{cast}\, \mathscr{E}(\text{Bob}, 7, -)\ \textbf{to}\ \text{int}\ \textbf{for}\ \text{Bob} : \text{int}\ \textbf{for}\ \text{Bob}$$

Sample typing judgments and non-judgments.

Example (Statically available keys)

$$\Sigma;\cdot;\mathsf{Bob};\cdot;\mathsf{Bob};\bot \vdash \textbf{cast}\ \mathscr{E}(\mathsf{Bob},7,-)\ \textbf{to}\ \mathsf{int}\ \textbf{for}\ \mathsf{Bob} : \mathsf{int}\ \textbf{for}\ \mathsf{Bob}$$

$$\Sigma;\cdot;\bot;\cdot;\mathsf{Bob};\bot \nvdash \textbf{cast}\ \mathscr{E}(\mathsf{Bob},7,-)\ \textbf{to}\ \mathsf{int}\ \textbf{for}\ \mathsf{Bob} : \mathsf{int}\ \textbf{for}\ \mathsf{Bob}$$
$$\Sigma;\cdot;\mathsf{Bob};\cdot;\bot;\bot \nvdash \textbf{cast}\ \mathscr{E}(\mathsf{Bob},7,-)\ \textbf{to}\ \mathsf{int}\ \textbf{for}\ \mathsf{Bob} : \mathsf{int}\ \textbf{for}\ \mathsf{Bob}$$

Example (Facts)

Suppose $\mathscr{E}(\mathsf{Bob},7,-) : \mathsf{int}\ \textbf{for}\ \mathsf{Bob} \in \mathscr{F}$,

$$\Sigma;\mathscr{F};\bot;\cdot;\mathsf{Bob};\bot \vdash \textbf{cast}\ \mathscr{E}(\mathsf{Bob},7,-)\ \textbf{to}\ \mathsf{int}\ \textbf{for}\ \mathsf{Bob} : \mathsf{int}\ \textbf{for}\ \mathsf{Bob}$$

$$\Sigma;\mathscr{F};\bot;\cdot;\bot;\bot \nvdash \textbf{cast}\ \mathscr{E}(\mathsf{Bob},7,-)\ \textbf{to}\ \mathsf{int}\ \textbf{for}\ \mathsf{Bob} : \mathsf{int}\ \textbf{for}\ \mathsf{Bob}$$

Evaluation tracks fact generation and authority.

$$\Sigma; \mathscr{F}_0; W \vdash \{\!|e, n|\!\} \;\rightarrow \{\!|e', n'|\!\} \text{ learning } \mathscr{F}$$

Evaluation tracks fact generation and authority.

$$\Sigma; \mathscr{F}_0; W \vdash \{\!|e, n|\!\} \rightarrow \{\!|e', n'|\!\} \text{ learning } \mathscr{F}$$

- e steps to e'.

Evaluation tracks fact generation and authority.

$$\Sigma; \mathscr{F}_0; W \vdash \{\!| e, n |\!\} \rightarrow \{\!| e', n' |\!\} \text{ learning } \mathscr{F}$$

- e steps to e'.
- Randomization seed n is updated to n'.

Evaluation tracks fact generation and authority.

$$\Sigma; \mathscr{F}_0; W \vdash \{\!|e, n|\!\} \rightarrow \{\!|e', n'|\!\} \text{ learning } \mathscr{F}$$

- e steps to e'.
- Randomization seed n is updated to n'.
- Key W is available for signing and decrypting.
  "The program is running with W's authority."

Evaluation tracks fact generation and authority.

$$\Sigma; \mathscr{F}_0; W \vdash \{\!|e, n|\!\} \rightarrow \{\!|e', n'|\!\} \text{ learning } \mathscr{F}$$

- e steps to e'.
- Randomization seed n is updated to n'.
- Key W is available for signing and decrypting.
  "The program is running with W's authority."
- Signature Σ, facts context $\mathscr{F}_0$, and key W are available for dynamic type-checking.

Evaluation tracks fact generation and authority.

$$\Sigma; \mathscr{F}_0; W \vdash \{\!|e, n|\!\} \rightarrow \{\!|e', n'|\!\} \text{ learning } \mathscr{F}$$

- e steps to e'.
- Randomization seed n is updated to n'.
- Key W is available for signing and decrypting.
  "The program is running with W's authority."
- Signature Σ, facts context $\mathscr{F}_0$, and key W are available for dynamic type-checking.
- New facts $\mathscr{F}$ are produced during encryptions.

Evaluation tracks fact generation and authority.

$$\Sigma; \mathscr{F}_0; W \vdash \{\!|e, n|\!\} \rightarrow \{\!|e', n'|\!\} \text{ learning } \mathscr{F}$$

- e steps to e'.
- Randomization seed n is updated to n'.
- Key W is available for signing and decrypting.
  "The program is running with W's authority."
- Signature Σ, facts context $\mathscr{F}_0$, and key W are available for dynamic type-checking.
- New facts $\mathscr{F}$ are produced during encryptions.

Fact contexts require special care.

Definition ($\mathrm{valid}_\Sigma\ \mathscr{F}$)

$\mathrm{valid}_\Sigma\ \mathscr{F}$ holds when

1. Σ is well formed: $\Sigma \vdash \diamond$.
2. Facts are true: $\mathscr{E}(a, e, n) : t$ **for** $b \in \mathscr{F}$ implies $a = b$ and $\Sigma; \cdot; b; \cdot; b; b \vdash e : t$.

Fact contexts require special care.

Definition ($\mathrm{valid}_\Sigma\, \mathscr{F}$)

$\mathrm{valid}_\Sigma\, \mathscr{F}$ holds when

1. Σ is well formed: $\Sigma \vdash \diamond$.
2. Facts are true: $\mathscr{E}(a, e, n) : t$ **for** $b \in \mathscr{F}$ implies $a = b$ and $\Sigma; \cdot; b; \cdot; b; b \vdash e : t$.

Lemma (New Fact Validity)

Assume $\mathrm{valid}_\Sigma\, \mathscr{F}_0$ *and* $\Sigma; \mathscr{F}_0; W; \Gamma; U; V \vdash e : t$. *Then* $\Sigma; \mathscr{F}_0; W \vdash \{|e, n|\} \rightarrow \{|e', n'|\}$ *learning* $\mathscr{F}$ *implies* $\mathrm{valid}_\Sigma\, \mathscr{F}$.

Fact contexts require special care.

Definition ($\mathrm{valid}_\Sigma\, \mathscr{F}$)

$\mathrm{valid}_\Sigma\, \mathscr{F}$ holds when

1. Σ is well formed: $\Sigma \vdash \diamond$.
2. Facts are true: $\mathscr{E}(a, e, n) : t$ **for** $b \in \mathscr{F}$ implies $a = b$ and $\Sigma; \cdot; b; \cdot; b; b \vdash e : t$.

Lemma (New Fact Validity)

Assume $\mathrm{valid}_\Sigma\, \mathscr{F}_0$ *and* $\Sigma; \mathscr{F}_0; W; \Gamma; U; V \vdash e : t$. *Then* $\Sigma; \mathscr{F}_0; W \vdash \{\!|e, n|\!\} \to \{\!|e', n'|\!\}$ *learning* $\mathscr{F}$ *implies* $\mathrm{valid}_\Sigma\, \mathscr{F}$.

Slogan

Preservation + Progress + New Fact Validity = Soundness

Noninterference: Secrets don't effect public outputs.

$\boxed{b \vdash}$ Aura Program

Noninterference: Secrets don't effect public outputs.

$\mathscr{E}$(Alice, "toaster", 0x0399)

: string **for** Alice

$b \vdash$ Aura Program

Noninterference: Secrets don't effect public outputs.

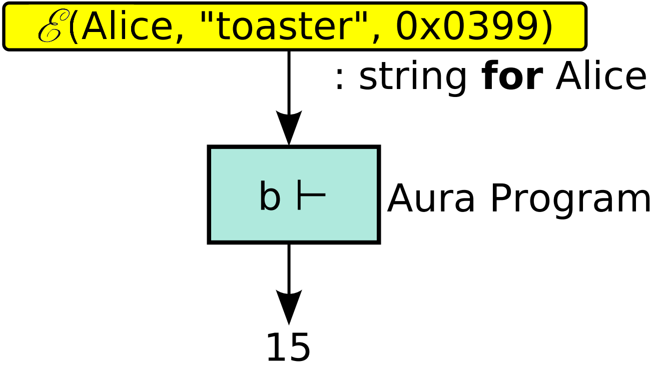
$\mathscr{E}$(Alice, "toaster", 0x0399)

: string **for** Alice

b $\vdash$

Aura Program

Noninterference: Secrets don’t effect public outputs.



Noninterference: Secrets don't effect public outputs.

$\boxed{\mathscr{E}(\text{Alice, "toaster", 0x0399})}$

: string **for** Alice

$\boxed{b \vdash}$ Aura Program

Noninterference: Secrets don't effect public outputs.

$\boxed{\mathscr{E}(\text{Alice, "lambda", 0x0312})}$

: string **for** Alice

$\boxed{b \vdash}$ Aura Program

Noninterference: Secrets don’t effect public outputs.

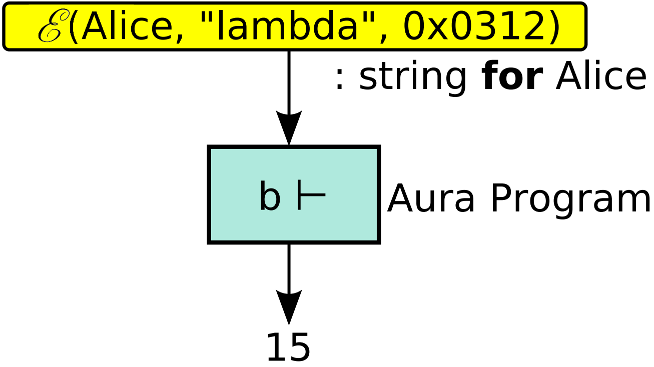
$\mathscr{E}$(Alice, "lambda", 0x0312)

: string **for** Alice

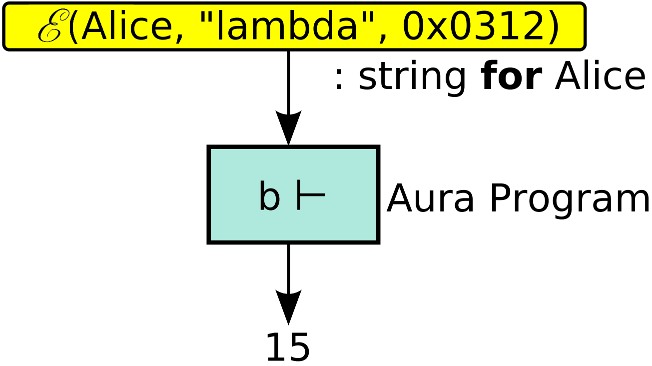
b $\vdash$

Aura Program

Noninterference: Secrets don't effect public outputs.



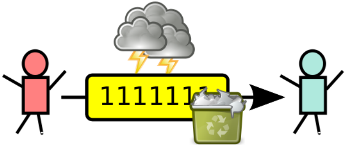
Noninterference: Secrets don't effect public outputs.



Noninterference [Denning+ '77],
Termination Insensitive Noninterference [Askarov+ '08]

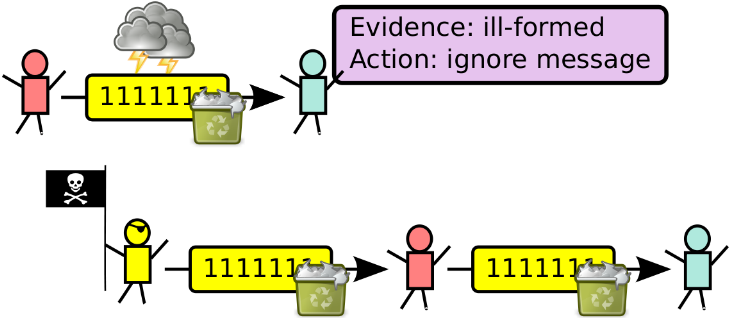
Decryption failures may be audited with justified casts.

Decryption failures may be audited with justified casts.

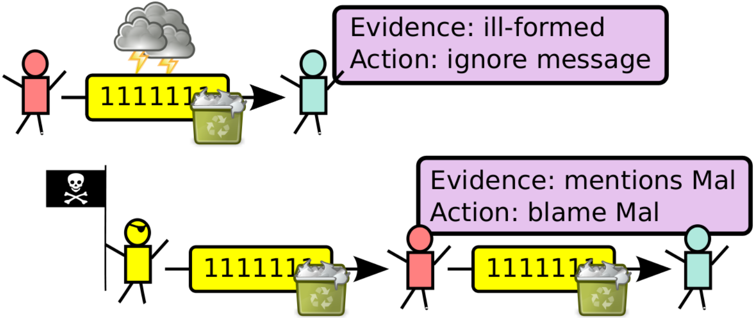


Decryption failures may be audited with justified casts.

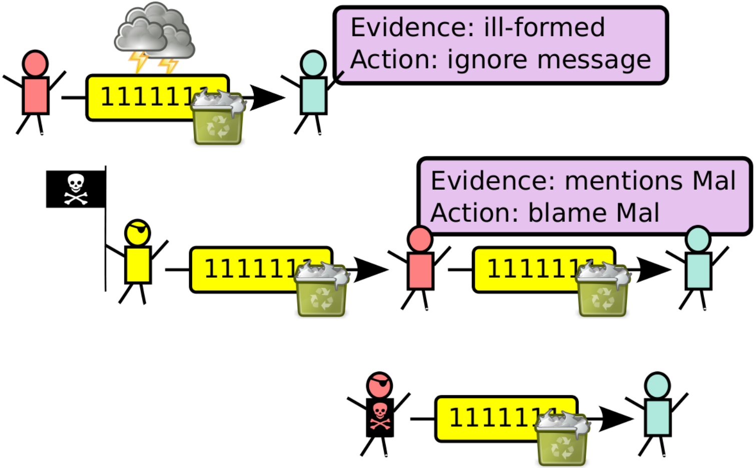
Decryption failures may be audited with justified casts.



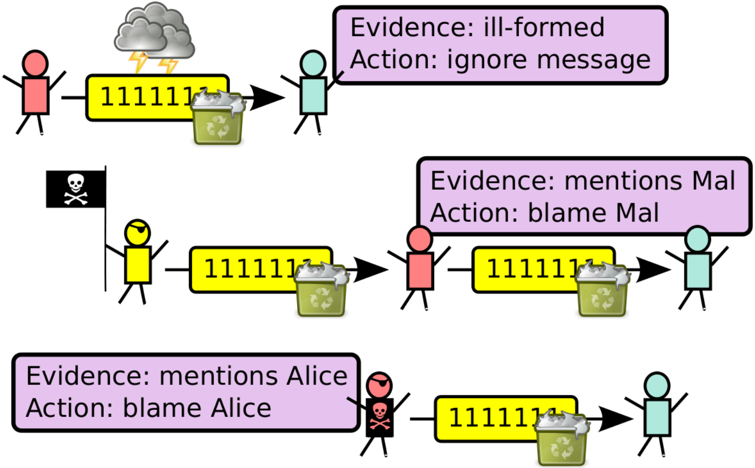
Decryption failures may be audited with justified casts.



Decryption failures may be audited with justified casts.



Decryption failures may be audited with justified casts.



Conclusion

Proposed and completed work.

Goals	Status
Define $\text{AURA}_{\text{conf}}$	✓
Syntactic soundness	✓
~~Dolev–Yao security~~	Noninterference
Submit a paper	ESOP '10 deadline Wednesday—almost ready to submit!

Summary

The AURA language family...

- unifies access control, computation, and confidentiality.
- supports arbitrary domain-specific authorization policies.
- mixes weak dependency, effects, and authorization logic in a compelling way.

Possible future directions

For AURA:

Build up surface syntax, tool support, communication model

Reach out refine FFI, build interoperable C# & Java libraries, write RFC for proof language

Look within type inference, simplify language spec., use type-and-effect analysis for termination, module abstraction via access control predicates

Possible future directions

For AURA:

Build up surface syntax, tool support, communication model

Reach out refine FFI, build interoperable C# & Java libraries, write RFC for proof language

Look within type inference, simplify language spec., use type-and-effect analysis for termination, module abstraction via access control predicates

For Jeff:

→ →

Acknowledgments

Thank you to all my collaborators on this work!

- Limin Jia
- Karl Mazurak
- Joseph Schorr
- Luke Zarko
- Steve Zdancewic
- Jianzhou Zhao