Automated Intruder Generation

Iliano Cervesato iliano@itd.nrl.navy.mil

ITT Industries, Inc @ NRL - Washington DC

http://www.cs.stanford.edu/~iliano/

Outline

I. MSR in brief
   - Access control
   - Dolev-Yao intruder

II. Access Control → DY Intruder

III. Protocol Spec. → Access Control

Part I

MSR

What's in MSR 2.0 ?

- Multiset rewriting with existentials
- Dependent types w/ subsorting 

- Memory predicates 

- Constraints 

Roles

Role state pred. var. declarations

- Generic roles

$$\left[\begin{array}{l} \exists L{:}\ \tau'_1(x_1) \times \ldots \times \tau'_n(x_n) \\ \ldots \\ \forall \mathbf{x}{:}\tau.\ \ lhs \rightarrow \exists \mathbf{y}{:}\tau'.\ \ rhs \\ \ldots \\ \forall \mathbf{x}{:}\tau.\ \ lhs \rightarrow \exists \mathbf{y}{:}\tau'.\ \ rhs \end{array}\right] \forall A$$

Role owner

- Anchored roles

$$\left[\begin{array}{l} \exists L{:}\ \tau'_1(x_1) \times \ldots \times \tau'_n(x_n) \\ \ldots \\ \forall \mathbf{x}{:}\tau.\ \ lhs \rightarrow \exists \mathbf{y}{:}\tau'.\ \ rhs \\ \ldots \\ \forall \mathbf{x}{:}\tau.\ \ lhs \rightarrow \exists \mathbf{y}{:}\tau'.\ \ rhs \end{array}\right] A$$

Rules

$\forall x_1 : \tau_1 .$

...

$\forall x_n : \tau_n .$

lhs

$\rightarrow$

$\exists y_1 : \tau'_1 .$

...

$\exists y_{n'} : \tau'_{n'} .$

rhs

- $N(t)$ Network
- $L(t, \ldots, t)$ Local state
- $M_A(t, \ldots, t)$ Memory
- χ Constraints

- $N(t)$ Network
- $L(t, \ldots, t)$ Local state
- $M_A(t, \ldots, t)$ Memory

MSR 2.0 – NS Initiator

$\forall A$

$\exists L$: princ × princ$^{(B)}$ × pubK B × nonce.

$\forall B$: princ
$\forall k_B$: pubK B

•

$\rightarrow$

$\exists n_A$:nonce.

$L(A,B,k_B,n_A)$
$N(\{n_A,A\}_{kB})$

$\forall$ ...
$\forall k_A$: pubK A
$\forall k'_A$: privK k_A
$\forall n_A, n_B$: nonce

$L(A,B,k_B,n_A)$
$N(\{n_A,n_B\}_{kA})$

$\rightarrow$

$N(\{n_B\}_{kB})$

MSR 2.0 – NS Responder

$\forall B$

$\exists \mathcal{L}$: princ$^{(B)}$ x pubK B$^{(kB)}$ x privK k_B x nonce.

$\forall k_B$: pubK B
$\forall k'_B$: privK k_B
$\forall A$: princ
$\forall n_A$: nonce
$\forall k_A$: pubK A

$N(\{n_A, A\}_{kB}) \rightarrow \exists n_B$:nonce. $\mathcal{L}(B, k_B, k'_B, n_B)$ $N(\{n_A, n_B\}_{kA})$

$\forall$ ...
$\forall n_B$: nonce

$\mathcal{L}(B, k_B, k'_B, n_B)$ $N(\{n_B\}_{kB}) \rightarrow$ •

Type Checking

$\Sigma \vdash P$

$\Gamma \vdash t : \tau$

t has type τ in Γ

P is well-typed in Σ

- Catches:
  - Encryption with a nonce
  - Transmission of a long term key
  - Circular key hierarchies, …

Access Control

New

$\Sigma \Vdash P$

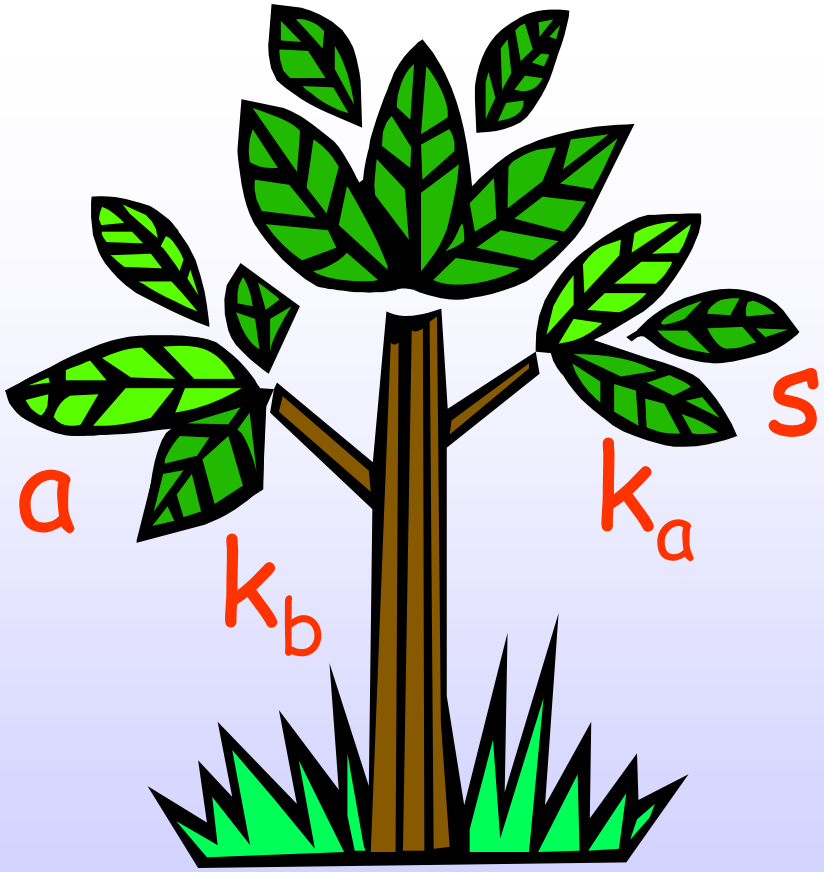
r is AC-valid for A in Γ

$\Gamma \Vdash_A r$

P is AC-valid in Σ

- Catches
  - A signing/encrypting with B's key
  - A accessing B's private data, …
- Static
- Decidable
- Gives meaning to Dolev-Yao intruder

... pictorially



An Overview of Access Control

- Interpret incoming information
  - Collect received data
  - Access unknown data
- Construct outgoing information
  - Generate data
  - Use known data
  - Access new data
- Verify access to data

Processing a Rule

$$\frac{\Gamma \Vdash_A lhs \gg \Delta \qquad\qquad \Gamma; \Delta \Vdash_A rhs}{\Gamma \Vdash_A lhs \rightarrow rhs}$$

Processing Predicates on the LHS

- Network messages

$$\frac{\Gamma ; \Delta \Vdash_A t \gg \Delta'}{\Gamma ; \Delta \Vdash_A N(t) \gg \Delta'}$$

- Memory predicates

$$\frac{\Gamma ; \Delta \Vdash_A t_1,\ldots,t_n \gg \Delta'}{\Gamma ; \Delta \Vdash_A M_A(t_1,\ldots,t_n) \gg \Delta'}$$

Interpreting Data on the LHS

- Pairs

$$\frac{\Gamma;\ \Delta \Vdash_A t_1,\ t_2 \gg \Delta'}{\Gamma;\ \Delta \Vdash_A (t_1,\ t_2) \gg \Delta'}$$

- Encrypted terms

$$\frac{\Gamma;\ \Delta \Vdash_A k \gg \Delta' \qquad \Gamma;\ \Delta' \Vdash_A t \gg \Delta''}{\Gamma;\ \Delta \Vdash_A \{t\}_k \gg \Delta''}$$

- Elementary terms

$$\left\{ \begin{array}{l} \dfrac{}{\Gamma;\ (\Delta,x) \Vdash_A x \gg (\Delta,x)} \\[2ex] \dfrac{}{(\Gamma,x{:}\tau);\ \Delta \Vdash_A x \gg (\Delta,x)} \end{array} \right.$$

Accessing Data on the LHS

- Shared keys

$$\frac{}{\Gamma;\ (\Delta,k) \Vdash_A k \gg (\Delta,k)}$$

$$\frac{}{(\Gamma,x{:}\mathit{shK}\ A\ B);\ \Delta \Vdash_A x \gg (\Delta,x)}$$

- Public keys

$$\frac{}{(\Gamma,k{:}\mathit{pubK}\ A,k'{:}\mathit{privK}\ k);\ (\Delta,k') \Vdash_A k \gg (\Delta,k')}$$

$$\frac{}{(\Gamma,k{:}\mathit{pubK}\ A,k'{:}\mathit{privK}\ k);\ \Delta \Vdash_A k \gg (\Delta,k')}$$

Generating Data on the RHS

- Nonces

$$\frac{(\Gamma, x{:}\mathit{nonce});\ (\Delta, x) \vdash_A \mathit{rhs}}{\Gamma;\ \Delta \vdash_A \exists x{:}\mathit{nonce}.\ \mathit{rhs}}$$

Constructing Terms on the RHS

- Pairs

$$\frac{\Gamma; \Delta \Vdash_A t_1 \qquad \Gamma; \Delta \Vdash_A t_2}{\Gamma; \Delta \Vdash_A (t_1, t_2)}$$

- Shared-key encryptions

$$\frac{\Gamma; \Delta \Vdash_A t \qquad \Gamma; \Delta \Vdash_A k}{\Gamma; \Delta \Vdash_A \{t\}_k}$$

Accessing Data on the RHS

- Principal

$$\frac{}{\Gamma, B{:}princ \Vdash_A B}$$

- Shared key

$$\frac{}{\Gamma, B{:}princ, k{:}shK\ A\ B \Vdash_A k}$$

- Public key

$$\frac{}{\Gamma, B{:}princ, k{:}pubK\ B \Vdash_A k}$$

- Private key

$$\frac{}{\Gamma, k{:}pubK\ A, k'{:}privK\ k \Vdash_A k'}$$

Part II

Access Control

⇓

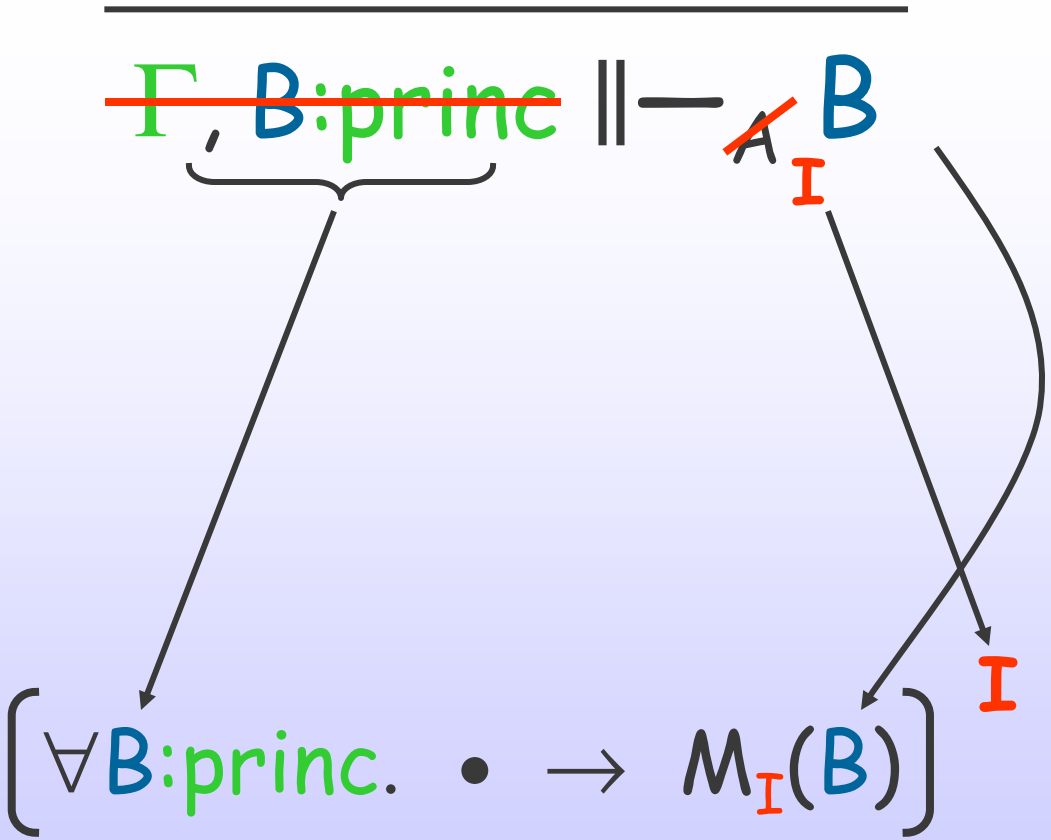
Dolev-Yao Intruder

The Dolev-Yao Intruder Model

- Interpret incoming information
  - Collect received data
  - Access unknown data
- Construct outgoing information
  - Generate data
  - Use known data
  - Access new data

- Same operations as AC!

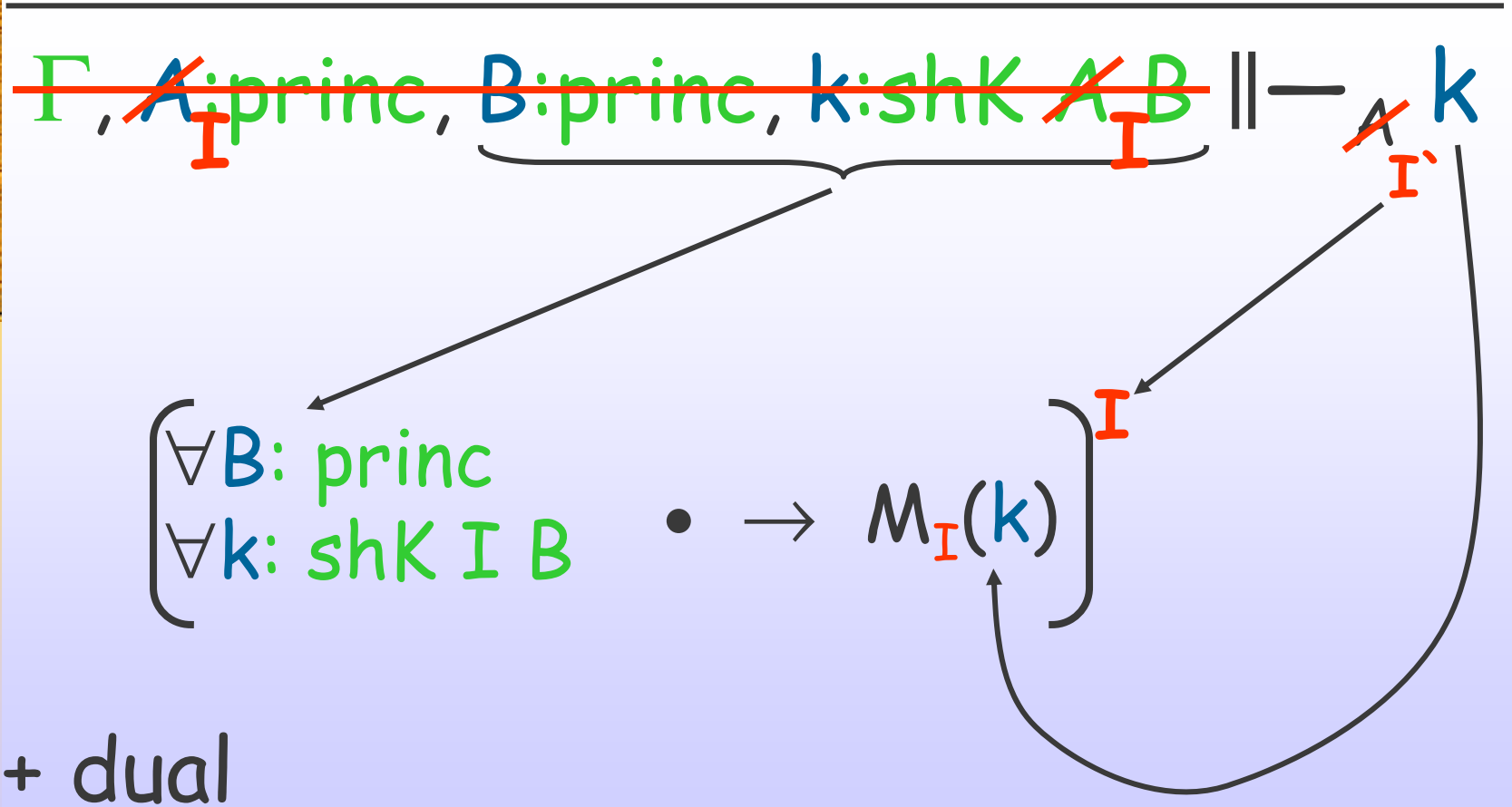
Accessing Principal Names



What did we do?

- Instantiate acting principal to **I**
- Accessed data $\rightarrow$ Intruder knowledge
- Meta-variables $\rightarrow$ Rule variables
- Ignore context Γ

Checking it out: Shared Keys



+ dual

Getting Confident: Pub./Priv. Keys

$$\frac{\Gamma, B{:}\mathrm{princ}, k{:}\mathrm{pubK}\ B \vdash_{\not A_I} k}{\left[\begin{array}{l}\forall B{:}\ \mathrm{princ} \\ \forall k{:}\ \mathrm{pubK}\ B\end{array} \bullet \rightarrow M_I(k)\right]^I}$$

$$\frac{\Gamma, \underset{I}{\not A}{:}\mathrm{princ}, k{:}\mathrm{pubK}\ \underset{I}{\not A}, k'{:}\mathrm{privK}\ k \vdash_{\not A_I} k'}{\left[\begin{array}{l}\forall k{:}\ \mathrm{pubK}\ I \\ \forall k'{:}\ \mathrm{privK}\ k\end{array} \bullet \rightarrow M_I(k')\right]^I}$$

Constructing Messages: Pairs

$$\frac{\Gamma, \Delta \Vdash_I t_1 \qquad \Gamma, \Delta \Vdash_I t_2}{\Gamma, \Delta \Vdash_I (t_1, t_2)}$$

$$\left[\forall t_1, t_2 : msg.\ M_I(t_1), M_I(t_2) \rightarrow M_I((t_1, t_2))\right]^I$$

Now, what did we do?

- Instantiate acting principal to **I**
- Accessed data → Intruder knowledge
- Meta-variables → Rule variables
- Ignore Γ and knowledge context Δ
- Premises → antecedent
- Conclusion → consequent
- Auxiliary typing derivation gives types

Carrying on: Shared-Key Encrypt.

$$\frac{\Gamma, \Delta \vdash_I t \qquad \Gamma, \Delta \vdash_I k}{\Gamma, \Delta \vdash_I \{t\}_k}$$

$$\left[\begin{array}{l} \forall A,B: \mathit{princ} \\ \forall k: \mathit{shK}\ A\ B \quad M_I(t), M_I(k) \rightarrow M_I(\{t\}_k) \\ \forall t: \mathit{msg} \end{array}\right]^I$$

Similar for public-key encryption

Generating Data: Nonces

$$\frac{(\Gamma, x{:}\mathsf{nonce});\ (\Delta, x) \vdash_I \mathsf{rhs}}{\Gamma;\ \Delta \vdash_I \exists x{:}\mathsf{nonce}.\ \mathsf{rhs}}$$

$$\left(\bullet \rightarrow \exists x{:}\mathsf{nonce}.\ M_I(x) \right)^I$$

Similarly for other generated data

Now, what did we do?

- Instantiate acting principal to **I**
- Accessed data → Intruder knowledge
- Meta-variables → Rule variables
- Ignore Γ and knowledge context Δ
- Premises → antecedent
- Conclusion → consequent
- Auxiliary typing derivation gives types
- One intruder rule for each AC rule
- Save generated object

Interpreting Shared-Key Encrypt.

$$\frac{\Gamma; \Delta \Vert\!-_I k >> \Delta' \qquad \Gamma; \Delta' \Vert\!-_I t >> \Delta''}{\Gamma; \Delta \Vert\!-_I \{t\}_k >> \Delta''}$$

$$\left[\begin{array}{l} \forall A,B\text{: princ} \\ \forall k\text{: shK } A\ B \\ \forall t\text{: msg} \end{array} \quad M_I(\{t\}_k), M_I(k) \rightarrow M_I(t)\right]^I$$

Similar for

- public-key encryption
- pairing

Now, what did we do?

- Instantiate acting principal to **I**
- Accessed data → Intruder knowledge
- Meta-variables → Rule variables
- Ignore Γ and knowledge context Δ
- Premises → antecedent
- Conclusion → consequent
- Auxiliary typing derivation gives types
- One intruder rule for each AC rule
- Save generated object

- Premises → consequent
- Conclusion → antecedant

Network Rules

$$\frac{\Gamma ; \Delta \Vdash_A t >> \Delta'}{\Gamma; \Delta \Vdash_A N(t) >> \Delta'}$$

$$\left(\forall t{:}msg.\ N(t) \rightarrow\ M_I(t)\right)^I$$

$$\frac{\Gamma ; \Delta \Vdash_A t}{\Gamma; \Delta \Vdash_A N(t)}$$

$$\left(\forall t{:}msg.\ M_I(t) \rightarrow\ N(t)\right)^I$$

... Other Rules?

Either

- redundant, or

$$\left[\forall t{:}msg.\ N(t) \rightarrow\ N(t)\right]^{I}$$

- or, innocuous (but sensible)

$$\left[\forall t_1,\ldots,t_n :msg.\ M'_I(t_1,\ldots,t_n) \rightarrow\ M_I(t_1),\ldots,M_I(t_n)\right]^{I}$$

Part III

Protocol Spec.

⇓

Access Control

Dissecting AC

Constructors atoms

5 activities:

- Interpret message components on LHS
- Access data (keys) on LHS
- Generate data on RHS
- Construct messages on RHS
- Access data on RHS

Accessing data

- Annotate the type of freely accessible data

```
princ: +type
```

- Make it conditional for dep. types

```
pubK: *princ -> +type

privK: ΠA:+princ. +pubK A -> +type
```

Generating data

- Again, annotate types

```
nonce: !type

shK: +princ -> +princ -> !type

shK: +princ -> +princ -> !type
```

Interpreting constructors

- Mark arguments as input or output

```
_,_: -msg -> -msg -> msg

{_}_: -msg -> ΠA:+princ. ΠB:+princ.+shK A B -> msg

{{_}}_: -msg -> ΠA:+princ. Πk:+pubK A.+privK k -> msg

hash: +msg -> msg

[_]_: +msg -> ΠA:*princ. Πk:*sigK A.*verK k -> msg
```

Annotating declarations

- Integrates semantics of types and constructors
- "Trimmed down" version of *AC*
- *Allows constructing AC rules*
- *Allows constructing the Dolev-Yao intruder*

... alternatively

Compute AC rules from protocol

- There are finitely many annotations
- Check protocol against each of them
- Keep the most restrictive ones that validate the protocol

Exponential!

More efficient algorithms?