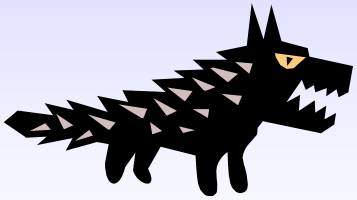
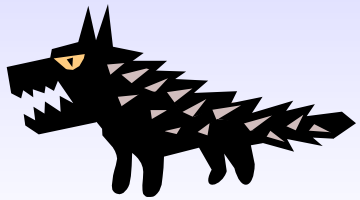


The Wolf Within



Iliano Cervesato iliano@itd.nrl.navy.mil

ITT Industries, Inc @ NRL - Washington DC

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

Outline

WORK IN PROGRESS

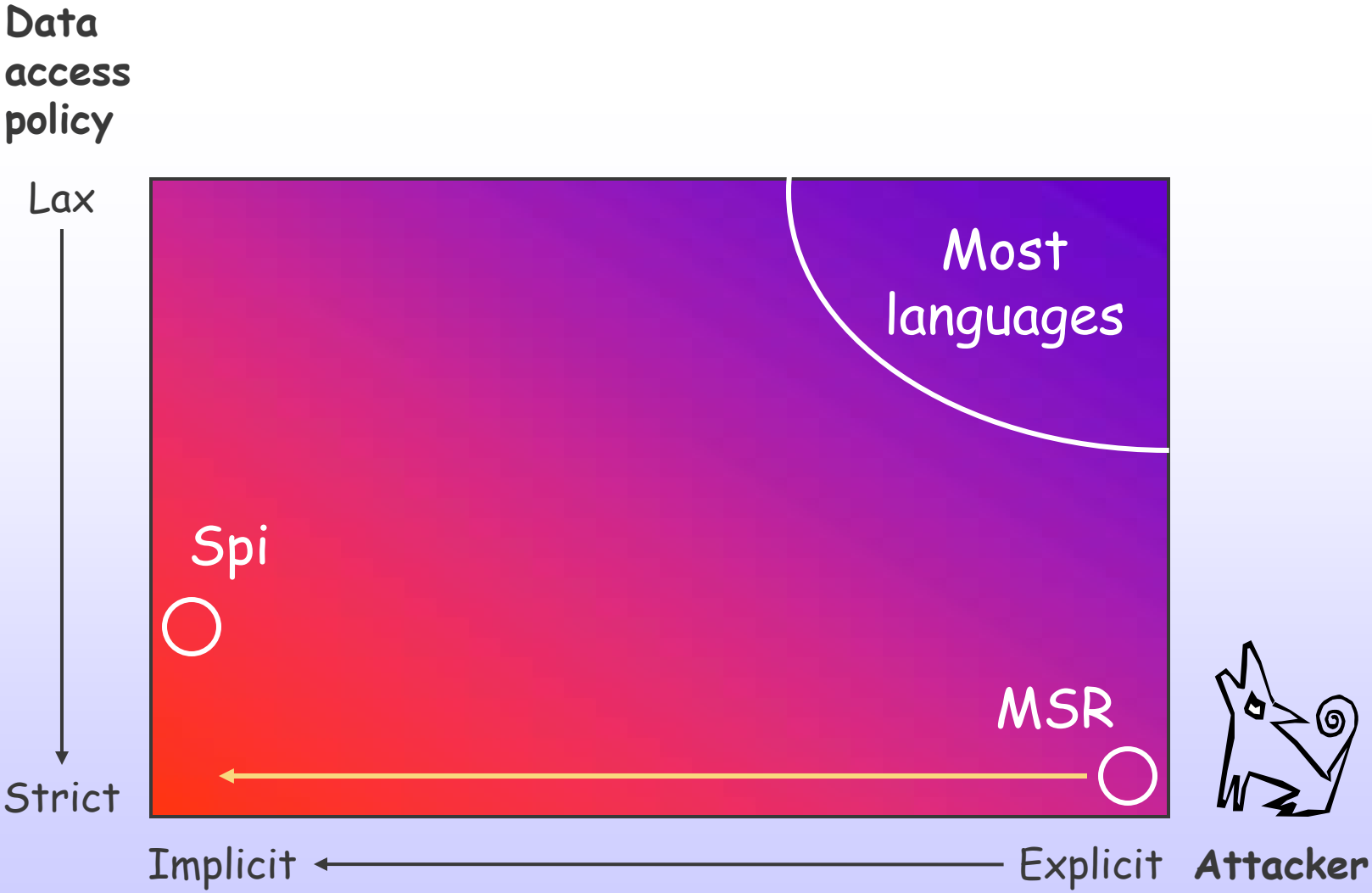
I. Protocol specification

II. MSR in brief
- Data Access Specification
- Dolev-Yao intruder

III. DAS → DY Intruder

IV. Protocol Spec. → DAS

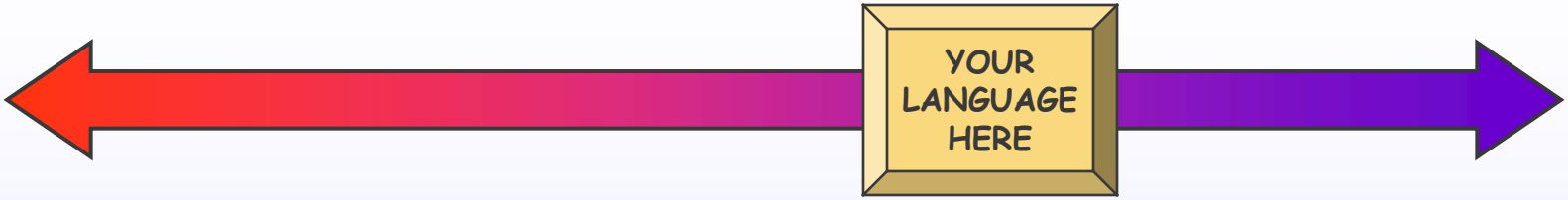
Data Access vs. Attacker

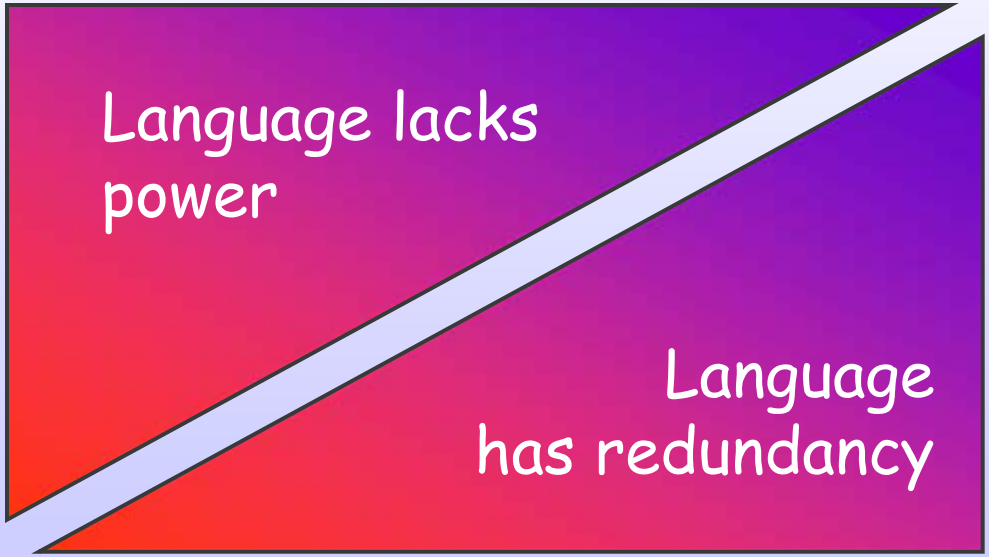


A 1D View

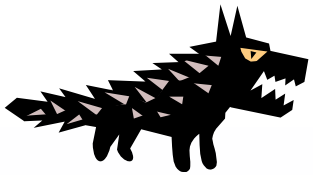
Strict language
Implicit attacker

Lax language
Explicit attacker



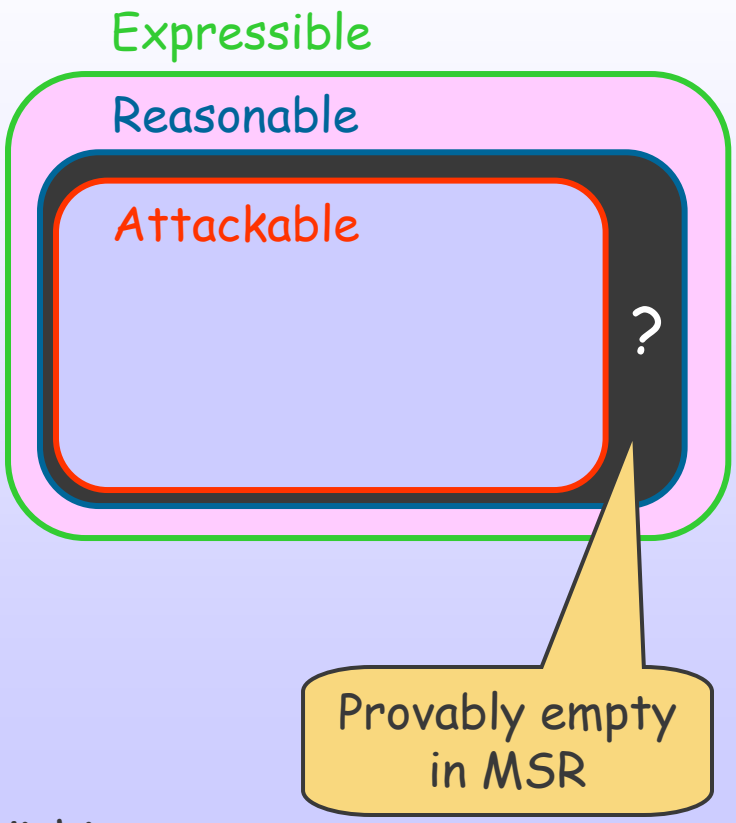


The Extremes



Lax and Explicit

Strict and Implicit



Expressible
= Reasonable
= Attackable

Summary

Strictness

+ Self-contained
+ Express what we want
– More complex

Explicitness

+ Accommodate weak attackers
– External

The option of explicitness is valuable

MSR

- Follows the Dolev-Yao abstraction
- Based on
  - Multiset rewriting, linear logic
  - Type theory
- Used to prove
  - Undecidability of protocol verification
  - Completeness of Dolev-Yao intruder
- Specifications
  - So many protocols … so little time …
- Related to CIL, strands, spi-calculus

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

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

Role owner

- Anchored roles

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

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

NS Initiator

$A \rightarrow B: \{n_A, A\}_{kB}$
$B \rightarrow A: \{n_A, n_B\}_{kA}$
$A \rightarrow B: \{n_B\}_{kB}$

$\forall A$

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

NS Responder

$A \rightarrow B: \{n_A, A\}_{kB}$
$B \rightarrow A: \{n_A, n_B\}_{kA}$
$A \rightarrow B: \{n_B\}_{kB}$

$\forall B$

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

$\forall k_B$: 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. $L(B, k_B, k'_B, n_B)$, $N(\{n_A, n_B\}_{kA})$

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

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

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, …

Data Access Specification

New

r is DAS-valid for A in Γ

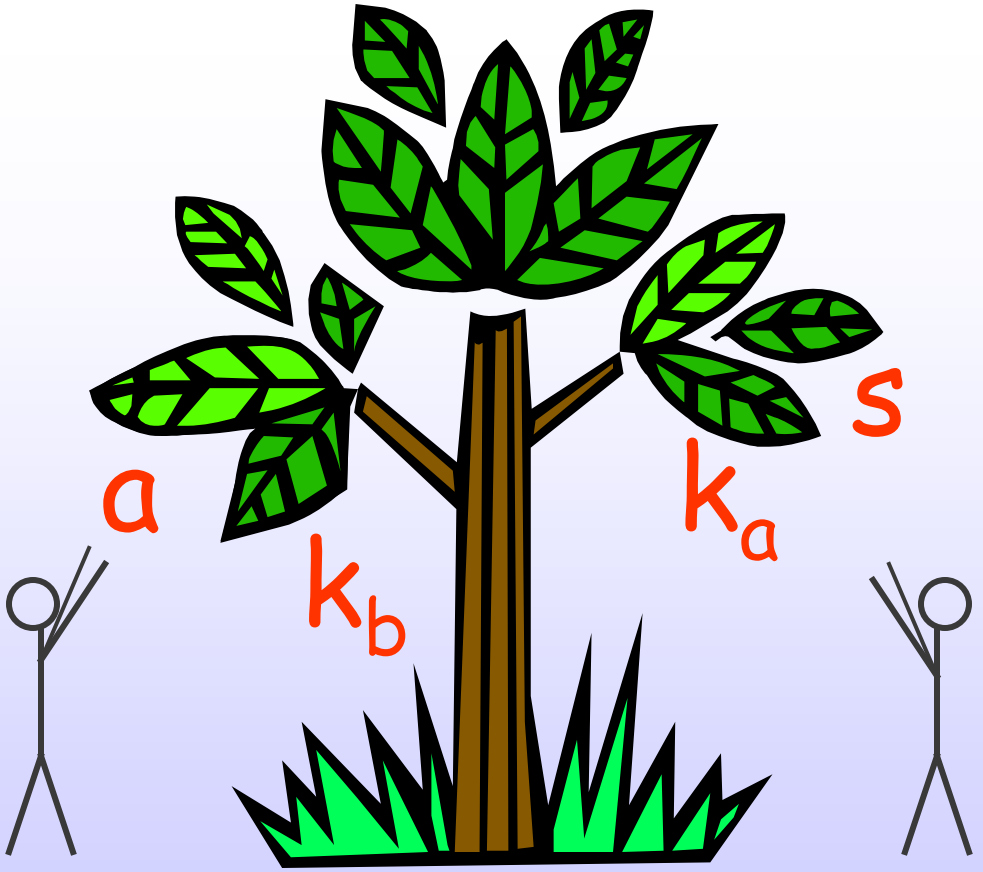
$\Gamma \Vdash_A r$

$\Sigma \Vdash P$

P is DAS-valid in Σ

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

... pictorially



An Overview of DAS

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

… all along, verify access to data

Verifying a Rule

Knowledge set:

Collects what A knows

Context

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

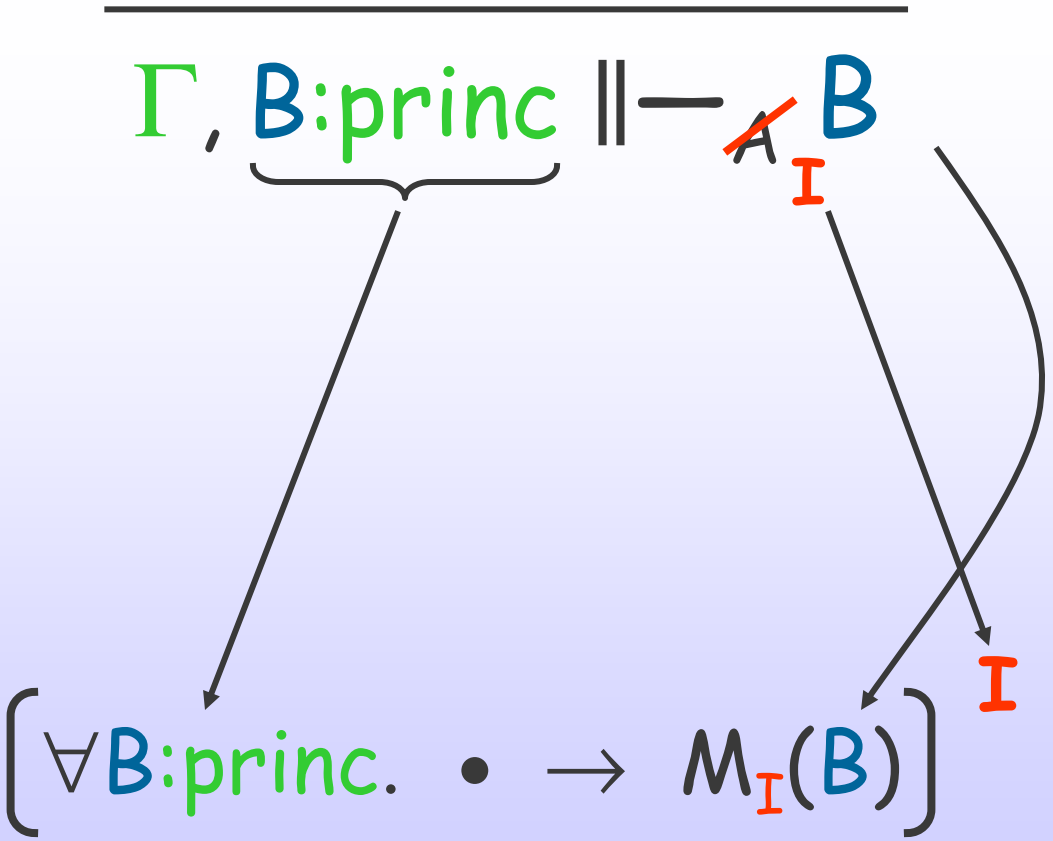
Role owner

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 DAS!

Accessing Principal Names

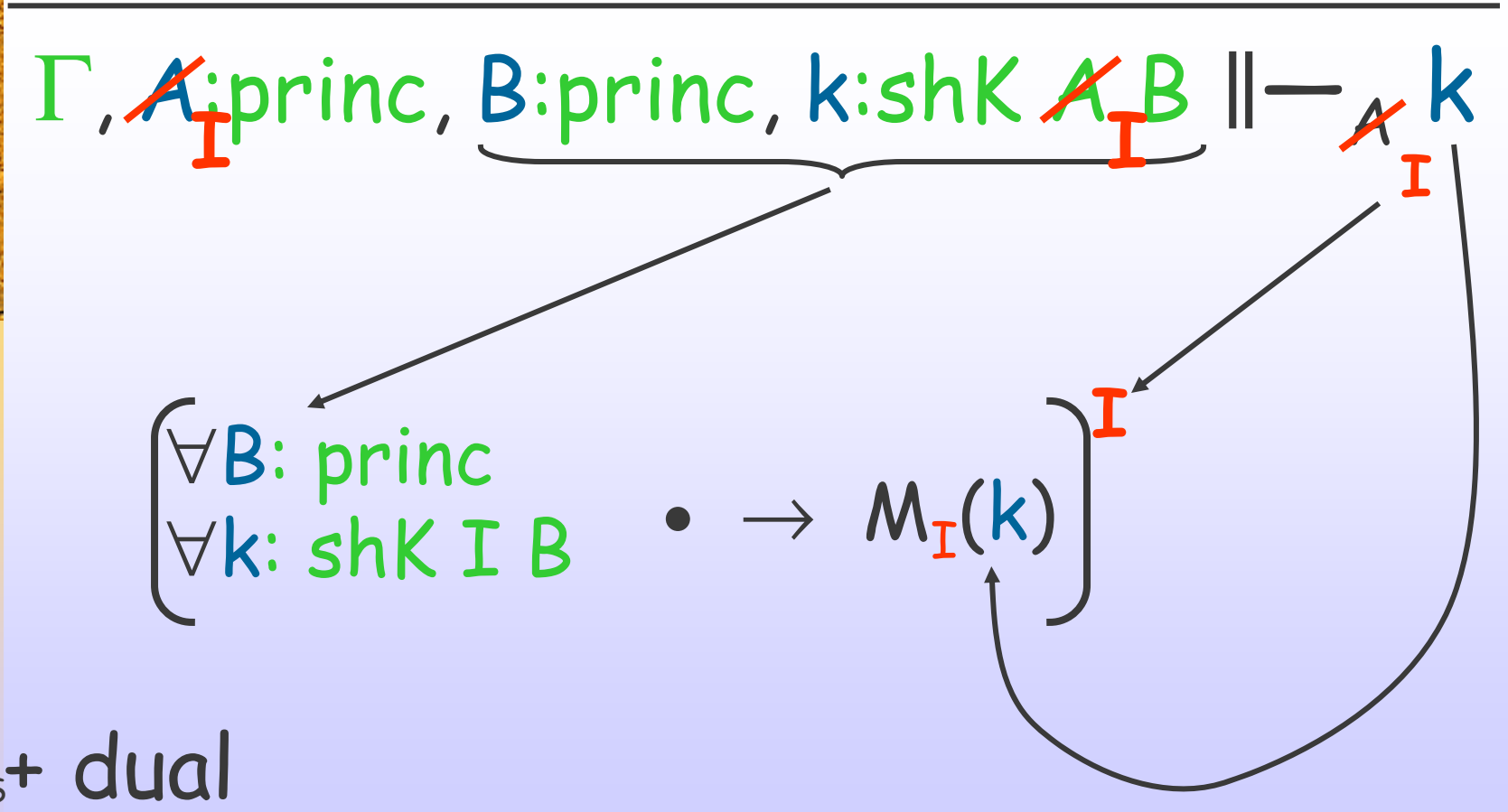


What did we do?

RHS data access:

- Instantiate acting principal to **I**
- Accessed data → Intruder knowledge
- Meta-variables → Rule variables
- Context provides types

Checking it out: Shared Keys



Intro.

MSR

DAS → DY

RHS data
RHS msg
New data
LHS data
LHS msg

Spec. → DAS + dual

Getting Confident: Pub./Priv. Keys

$$\Gamma, B{:}\mathsf{princ}, k{:}\mathsf{pubK}\ B \vdash_{\cancel{A}_I} k$$

$$\left[\begin{array}{l}\forall B{:}\ \mathsf{princ} \\ \forall k{:}\ \mathsf{pubK}\ B\end{array} \bullet \rightarrow M_I(k)\right]^I$$

$$\Gamma, \underset{I}{\cancel{A}}{:}\mathsf{princ}, k{:}\mathsf{pubK}\ \underset{I}{\cancel{A}}, k'{:}\mathsf{privK}\ k \vdash_{\cancel{A}_I} k'$$

$$\left[\begin{array}{l}\forall k{:}\ \mathsf{pubK}\ I \\ \forall k'{:}\ \mathsf{privK}\ k\end{array} \bullet \rightarrow M_I(k')\right]^I$$

Constructing Messages: Pairs

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

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

$$\frac{\Gamma \vdash t_1 : msg \qquad \Gamma \vdash t_2 : msg}{\Gamma \vdash (t_1, t_2) : msg}$$

Now, what did we do?

RHS message construction:

- Instantiate acting principal to **I**
- Meta-variables → Rule variables
- Premises → antecedent
- Conclusion → consequent
- Types from auxiliary typing derivation

Carrying on: Shared-Key Encrypt.

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

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

Similar for public-key encryption

Generating Nonces

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

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

Similarly for other generated data

Now, what did we do?

Data generation on the RHS:

- Instantiate acting principal to **I**
- Auxiliary typing derivation gives types
- Remember generated object
- Follow knowledge acquisition flow

Accessing Shared Keys on the LHS

$$\frac{}{(\Gamma,\ k{:}shK\ I\ B);\ \Delta \Vdash_I k \gg (\Delta, k)}$$

$$\left(\begin{array}{l}\forall B{:}\ princ \\ \forall k{:}\ shK\ I\ B\end{array} \quad \bullet \rightarrow M_I(k)\right)^I$$

Similarly for other keys

Now, what did we do?

LHS data access:

- Instantiate acting principal to **I**
- Meta-variables → Rule variables
- Types from auxiliary typing derivation
- Follow knowledge acquisition flow
- Remember generated object

Same target rules as for RHS data access

Interpreting Shared-Key Encrypt.

$$\frac{\Gamma;\ \Delta \Vdash_I k \gg\!> \Delta' \qquad \Gamma;\ \Delta' \Vdash_I t \gg\!> \Delta''}{\Gamma;\ \Delta \Vdash_I \{t\}_k \gg\!> \Delta''}$$

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

Similar for public-key encryption and pairing

Now, what did we do?

LHS message interpretation

- Instantiate acting principal to **I**
- Meta-variables → Rule variables
- Types from auxiliary typing derivation
- Follow knowledge acquisition flow
- Conclusion ⟶ antecedant
- "Last" premises ⟶ consequent

Network Rules

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

LHS

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

RHS

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

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

... Other Rules?

Either

- redundant, or

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

- or, innocuous (but sensible)

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

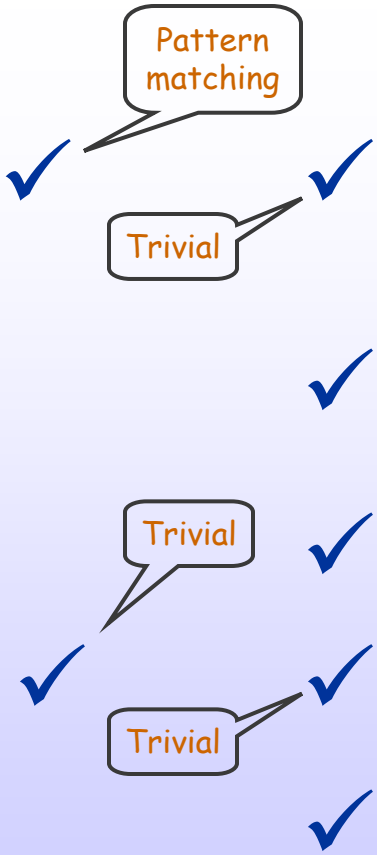
Automating DAS Rule Design?

- One size does not fit all
- Look at protocol
  - Typed MSR spec.
  - Usage of constructs
- Involve construct declarations
  - Not sufficient
  - Use annotations

Generating DAS rules from use

Constructors atoms

- 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

anybody can access

```
princ: +type
```

- Make it conditional for dep. types

```
pubK:  *princ -> +type
```

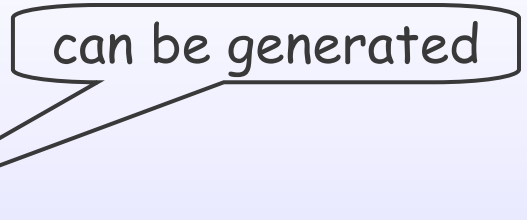
doesn't matter

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

owner

Generating data

- Again, annotate types




```
nonce: !type

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

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

Pattern-matching constructors

- Mark *arguments* as input or output

output

input

```
_,_: -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
```

+ or -

Annotating Declarations

- Integrates semantics of types and constructors
- "Trimmed down" version of DAS
- Allows constructing DAS rules
  - … and Dolev-Yao intruder

... alternatively

Compute DAS 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?