Abstract Specification of Crypto-Protocols and their Attack Models in MSR

Iliano Cervesato iliano@itd.nrl.navy.mil

ITT Industries, Inc @ NRL - Washington DC

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

Outline

I. Dolev-Yao specifications

II. Specifying Protocols in MSR

III. Access Control

IV. Specifying Attacker Models

V. Inferring the Dolev-Yao Attacker

Part I

Dolev-Yao Specification of Security Protocols

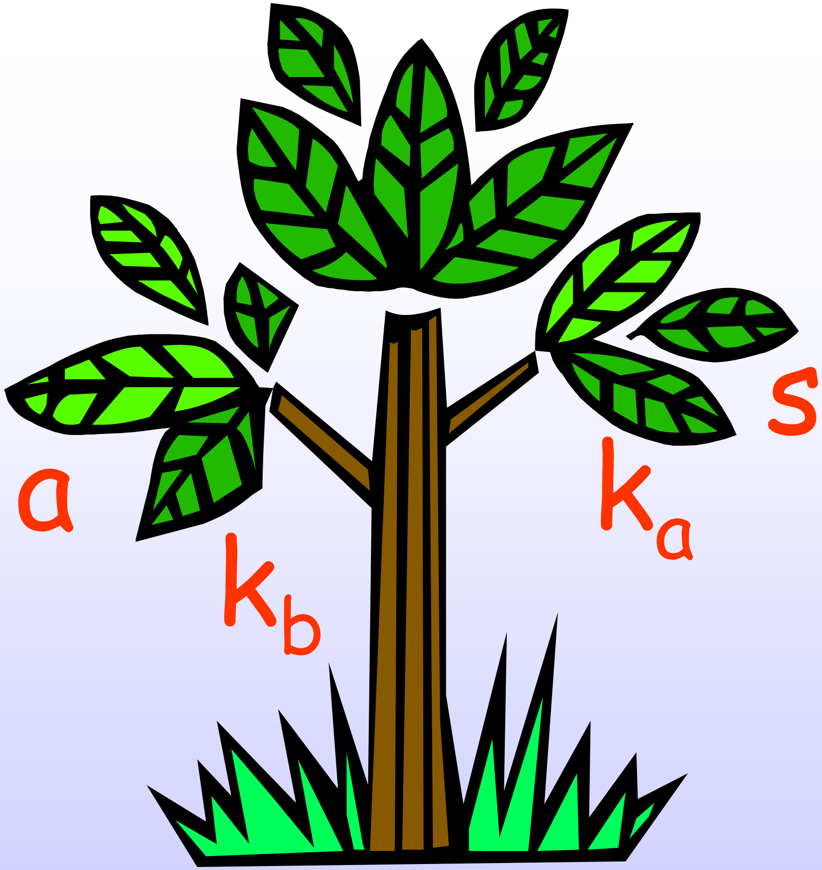
Why is Protocol Analysis Difficult?

- Subtle cryptographic primitives
  - Dolev-Yao abstraction
- Distributed hostile environment
  - “Prudent engineering practice“
- Inadequate specification languages
  - *… the devil is in details …*

Dolev-Yao Abstraction

- Symbolic data
  - No bit-strings
- Perfect cryptography
  - No guessing of keys
- Public knowledge soup
  - Magic access to data

… pictorially



Languages to Specify What?

- Message flow
- Message constituents
- ~~Operating environment~~
- ~~Protocol goals~~

Desirable Properties

- Unambiguous
- Simple
- Flexible
  - Applies to a wide class of protocols
- Insightful
  - Gives insight about protocols

Part II

Specifying Protocols in MSR

What's in MSR 2.0 ?

- Multiset rewriting with existentials
- Dependent types w/ subsorting New
- Memory predicates New
- Constraints New

Terms

- Atomic terms
  - Principal names A
  - Keys k
  - Nonces n
  - …
- Term constructors
  - (_ _)
  - {_}_   {{_}}_
  - [_]_
  - …

Definable

Rules

$\forall x_1 : \tau_1 . \; \ldots \; \forall x_n : \tau_n . \quad \text{lhs} \quad \rightarrow \quad \exists y_1 : \tau'_1 . \; \ldots \; \exists y_{n'} : \tau'_{n'} . \quad \text{rhs}$

lhs:

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

rhs:

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

Types of Terms

- A: princ
- n: nonce
- k: shK A B
- k: pubK A
- k': privK k
- … (definable)

Types can <u>depend</u> on term

- Captures relations between objects

Subtyping

$$\tau :: msg$$

- Allows atomic terms in messages
- Definable
  - Non-transmittable terms
  - Sub-hierarchies

Role state predicates

$L_l(A,t, \ldots, t)$

- Hold data local to a role instance
  - Lifespan = role
- Invoke next rule
  - L_l = control
  - $(A,t, \ldots, t)$ = data

Memory Predicates

New

$$M_A(t, \ldots, t)$$

- Hold private info. across role exec.
- Support for subprotocols
  - Communicate data
  - Pass control
- Interface to outside system
- Implements intruder

Constraints

χ

- Guards over interpreted domain
  - Abstract
  - Modular
- Invoke constraint handler
- E.g.: timestamps
  - $(T_E = T_N + T_d)$
  - $(T_N < T_E)$

Type of predicates

- Dependent sums

$$\Sigma x{:}\,\tau.\ \underline{\tau}$$

$$\tau^{(x)} \times \underline{\tau}$$

x

- Forces associations among arguments
  - E.g.: $\mathrm{princ}^{(A)} \times \mathrm{pubK}\ A^{(k_A)} \times \mathrm{privK}\ k_A$

Roles

Role state pred. var. declarations

- Generic roles

$$\left(\begin{array}{lllll} \multicolumn{5}{l}{\exists \mathcal{L}: \tau'^{(x_1)}_1 \times \ldots \times \tau'^{(x_n)}_n} \\ \multicolumn{5}{l}{\ldots} \\ \forall \mathbf{x}:\tau. & lhs & \to & \exists \mathbf{y}:\tau'. & rhs \\ \ldots & & \ldots & & \ldots \\ \forall \mathbf{x}:\tau. & lhs & \to & \exists \mathbf{y}:\tau'. & rhs \end{array}\right)^{\forall A}$$

Role owner

- Anchored roles

$$\left[\begin{array}{lllll} \multicolumn{5}{l}{\exists \mathcal{L}: \tau'^{(x_1)}_1 \times \ldots \times \tau'^{(x_n)}_n} \\ \multicolumn{5}{l}{\ldots} \\ \forall \mathbf{x}:\tau. & lhs & \to & \exists \mathbf{y}:\tau'. & rhs \\ \ldots & & \ldots & & \ldots \\ \forall \mathbf{x}:\tau. & lhs & \to & \exists \mathbf{y}:\tau'. & rhs \end{array}\right]^{A}$$

Needham-Schroeder Public-Key Protocol (fragment)

$A \to B: \{n_A, A\}_{kB}$

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

$A \to B: \{n_B\}_{kB}$

("Usual Notation")

MSR 2.0 – NS Initiator

$$\left(\begin{array}{l} \exists L : \text{princ} \times \text{princ}^{(B)} \times \text{pubK } B \times \text{nonce}. \\[2em] \begin{array}{l} \forall B: \text{princ} \\ \forall k_B : \text{pubK } B \end{array} \quad \bullet \quad \rightarrow \quad \exists n_A : \text{nonce}. \quad \begin{array}{l} L(A,B,k_B,n_A) \\ N(\{n_A, A\}_{kB}) \end{array} \\[2em] \begin{array}{l} \forall \ \ldots \\ \forall k_A : \text{pubK } A \\ \forall k'_A : \text{privK } k_A \\ \forall n_A, n_B : \text{nonce} \end{array} \quad \begin{array}{l} L(A,B,k_B,n_A) \\ N(\{n_A, n_B\}_{kA}) \end{array} \quad \rightarrow \quad N(\{n_B\}_{kB}) \end{array} \right)^{\forall A}$$

MSR 2.0 – NS Responder

$$\left(\begin{array}{l} \exists L: \text{princ}^{(B)} \times \text{princ}^{(A)} \times \text{pubK } B^{(kB)} \times \text{privK } k_B \\ \qquad \times\ \text{nonce} \times \text{pubK } A \times \text{nonce}. \\ \\ \begin{array}{l} \forall k_B : \text{pubK } B \\ \forall k'_B : \text{privK } k_B \\ \forall A : \text{princ} \\ \forall n_A : \text{nonce} \\ \forall k_A : \text{pubK } A \end{array} \quad N(\{n_A, A\}_{kB}) \rightarrow \exists n_B : \text{nonce}. \begin{array}{l} L(\ldots) \\ N(\{n_A, n_B\}_{kA}) \end{array} \\ \\ \begin{array}{l} \forall \ \ldots \\ \forall n_B : \text{nonce} \end{array} \quad \begin{array}{l} L(B, k_B, k'_B, A, n_A, k_A, n_B) \\ N(\{n_B\}_{kB}) \end{array} \rightarrow \bullet \end{array} \right)^{\forall B}$$

Type Checking

New

$\Sigma \vdash P$

$\Gamma \vdash t : \tau$

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, …
- Static and dynamic uses
- Decidable

Execution Model

1-step firing

$$P \triangleright C \rightarrow C'$$

- Activate roles
- Generates new role state pred. names
- Instantiate variables
- Apply rules
- Skips rules

Snapshots

Active role set

$$C = [S]^{R}_{\Sigma}$$

State

- $N(t)$
- $L_l(t, \ldots, t)$
- $M_A(t, \ldots, t)$

Signature

- $a : \tau$
- $L_l : \underline{\tau}$
- $M\_ : \underline{\tau}$

Rule application

$$F, \chi \rightarrow \exists \underline{n}:\underline{\tau}.\ G(\underline{n})$$

- Constraint check

$$\Sigma \models \chi \qquad \text{(constraint handler)}$$

- Firing

$$\underbrace{[S_1]^R_{\Sigma}}_{S,\ F} \rightarrow \underbrace{[S_2]^R_{\Sigma,\ \underline{c}:\underline{\tau}}}_{S,\ G(\underline{c})} \qquad \underline{c} \text{ not in } S_1$$

Properties

- Admissibility of parallel firing
- Type preservation
- Access control preservation
- Completeness of Dolev-Yao intruder

Part III

"Access Control"

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

Overview

- 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

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

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

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

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

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

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

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

- Shared key

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

- Public key

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

- Private key

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

Part IV

Specifying Attacker Models

Execution with an Attacker

$$P,\ P_I \triangleright C \rightarrow C'$$

- Selected principal(s): I
- Generic capabilities: P_I
  - Well-typed
  - AC-valid
- Modeled completely within MSR

The Dolev-Yao Intruder

- Specific protocol suite P_{DY}
- Underlies every protocol analysis tool
- Completeness still unproved !!!

Capabilities of the D-Y Intruder

- Intercept / emit messages
- Split / form pairs
- Decrypt / encrypt with known key
- Look up public information
- Generate fresh data

DY Intruder – Data access

- $M_I(t)$: Intruder knowledge

$$\left[\forall A: \text{princ}. \;\bullet \rightarrow M_I(A)\right]^I$$

$$\left(\begin{array}{l}\forall A: \text{princ}\\ \forall k: \text{shK } I\ A\end{array} \;\bullet \rightarrow M_I(k)\right)^I \quad + \textit{dual}$$

$$\left(\begin{array}{l}\forall A: \text{princ}\\ \forall k: \text{pubK } A\end{array} \;\bullet \rightarrow M_I(k)\right)^I \quad \left(\begin{array}{l}\forall k: \text{pubK } I\\ \forall k': \text{privK } k\end{array} \;\bullet \rightarrow M_I(k')\right)^I$$

- No nonces, no other keys, …

DY Intruder – Data Generation

- Safe data

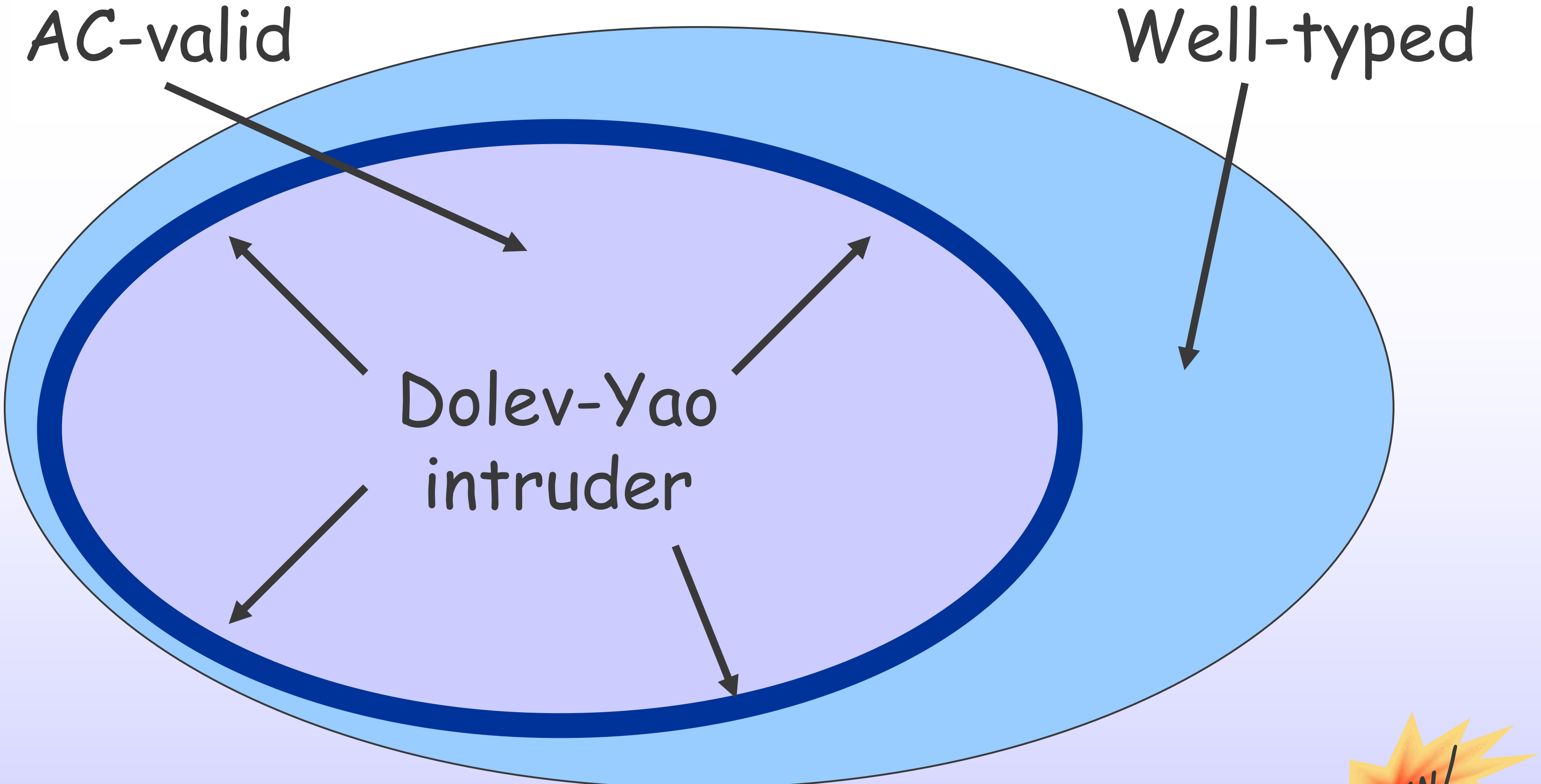
$$\left[\bullet \rightarrow \exists n{:}\mathsf{nonce}.\ M_I(n)\right]^I \qquad \left[\bullet \rightarrow \exists m{:}\mathsf{msg}.\ M_I(m)\right]^I$$

- Anything else ?

$$\left[\forall A,B{:}\mathsf{princ}.\ \bullet \rightarrow \exists k{:}\mathsf{shK}\ A\ B.\ M_I(k)\right]^I \qquad ???$$

- It depends on the protocol !!!
  - Automated generation

DY Intruder vs. AC



- D-Y Intruder inferred from AC rules

- AC rules inferred from prot. spec. (with help)

Completeness of D-Y Intruder

- If $$P \triangleright [S]^{R}_{\Sigma} \rightarrow [S']^{R'}_{\Sigma'}$$

  with all well-typed and *AC*-valid

- Then

$$\underline{P},\ P_{DY} \triangleright [\underline{S}]^{\underline{R}}_{\underline{\Sigma}} \rightarrow [\underline{S'}]^{\underline{R'}}_{\underline{\Sigma'}}$$

Consequences

- Justifies design of current tools
- Support optimizations
  - D-Y intr. often too general/inefficient
    - Generic optimizations
    - *Per protocol* optimizations
    - Restrictive environments
- Caps multi-intruder situations

Part V

New

Inferring the 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

$$\frac{}{\cancel{\Gamma, B{:}princ} \Vdash_{\cancel{A}\,I} B}$$

$$\left[\forall B{:}princ.\ \bullet \to M_I(B)\right]^I$$

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

$$\cancel{\Gamma, \cancel{A}_{I}\text{:princ}, \underbrace{B\text{:princ}, k\text{:shK}\ \cancel{A}_{I}\ B}} \Vdash_{\cancel{A}_{I}} k$$

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

+ dual

Getting Confident: Pub./Priv. Keys

$$\frac{}{\cancel{\Gamma, \underbrace{B{:}\mathrm{princ}, k{:}\mathrm{pubK}\,B}} \Vdash_{\cancel{A}\ \mathrm{I}} k}$$

$$\left[\begin{matrix}\forall B: \mathrm{princ} \\ \forall k: \mathrm{pubK}\,B\end{matrix}\quad \bullet \to M_{\mathrm{I}}(k)\right]^{\mathrm{I}}$$

$$\frac{}{\cancel{\Gamma, \underset{\mathrm{I}}{\cancel{A}}{:}\mathrm{princ}, \underbrace{k{:}\mathrm{pubK}\,\underset{\mathrm{I}}{\cancel{A}}, k'{:}\mathrm{privK}\,k}} \Vdash_{\cancel{A}\ \mathrm{I}} k'}$$

$$\left[\begin{matrix}\forall k: \mathrm{pubK}\,\mathrm{I} \\ \forall k': \mathrm{privK}\,k\end{matrix}\quad \bullet \to M_{\mathrm{I}}(k')\right]^{\mathrm{I}}$$

Constructing Messages: Pairs

$$\frac{\cancel{\Gamma ; \Delta} \Vdash_{\boxed{I}} t_1 \qquad \cancel{\Gamma ; \Delta} \Vdash_{\boxed{I}} t_2}{\cancel{\Gamma ; \Delta} \Vdash_{\boxed{I}} (t_1, t_2)}$$

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

Now, what did we do?

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

Carrying on: Shared-Key Encrypt.

$$\frac{\cancel{\Gamma; \Delta} \Vdash_{I} t \qquad\qquad \cancel{\Gamma; \Delta} \Vdash_{I} k}{\cancel{\Gamma; \Delta} \Vdash_{I} \{t\}_k}$$

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

Similar for public-key encryption

Generating Data: Nonces

$$\frac{\cancel{(\Gamma, x{:}\mathrm{nonce}); (\Delta, x)} \Vdash_{I} rhs}{\cancel{\Gamma; \Delta} \Vdash_{I} \exists x{:}\mathrm{nonce}.\ rhs}$$

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

Similarly for other generated data

Now, what did we do?

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

Interpreting Shared-Key Encrypt.

$$\frac{\cancel{\Gamma;\Delta} \Vdash_{I} k \gg \Delta' \qquad \cancel{\Gamma;\Delta'} \Vdash_{I} t \gg \Delta''}{\cancel{\Gamma;\Delta} \Vdash_{I} \{t\}_k \gg \Delta''}$$

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

Similar for

- public-key encryption
- pairing

Now, what did we do?

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

- **Premises $\rightarrow$ consequent**
- **Conclusion $\rightarrow$ antecedant**

Network Rules

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

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

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

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

... Other Rules?

Either

- redundant, or

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

- or, innocuous (but sensible)

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

Dissecting AC

5 activities:

	Constructors	atoms
• Interpret message components on LHS	✓ Pattern matching	✓ Trivial
• Access data (keys) on LHS		✓
• Generate data on RHS		✓
• Construct messages on RHS	✓ Trivial	✓ Trivial
• 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?

Wrap-up

Case-Studies

- Full Needham-Schroeder public-key
- Otway-Rees
- Neuman-Stubblebine repeated auth.
- OFT group key management

Conclusions

- Framework for specifying protocols
  - Precise
  - Flexible
  - Powerful
- Provides
  - Type /AC checking
  - Sequential / parallel execution model
  - Insights about Dolev-Yao intruder

Future work

- Experimentation
  - Clark-Jacob library
  - Fair-exchange protocols
  - More multicast
- Pragmatics
  - Type-reconstruction
  - Operational execution model(s)
  - Implementation
- Automated specification techniques