MSR

A Framework for Security Protocols and their Meta-Theory

Iliano Cervesato iliano@itd.nrl.navy.mil

ITT Industries, Inc @ NRL - Washington DC

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

Outline

I. Mis-specification languages

II. MSR

- Overview
- Typing
- Access control
- Execution
- Properties
- Example

III. The most powerful attacker

- Dolev-Yao intruder

Part I

Mis-Specification Languages

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

Languages to Specify What?

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

Desirable Properties

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

"Usual Notation"

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

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

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

How does it do?

- Flow
  - Expected run
- Constituents
  - Side remarks
- Environment
  - Side remarks
- Goals
  - Side remarks

- Unambiguous ☹
- Simple ☺
- Flexible ☺
- Powerful ☺
- Insightful 😐

Strands

$$
\begin{tikzcd}
\{n_A, A\}_{kB} \arrow[r] \arrow[d, Rightarrow] & {} & {} \arrow[r] & \{n_A, A\}_{kB} \arrow[d, Rightarrow] \\
\{n_A, n_B\}_{kA} \arrow[d, Rightarrow] & {} \arrow[l] & {} & \{n_A, n_B\}_{kA} \arrow[l] \arrow[d, Rightarrow] \\
\{n_B\}_{kB} \arrow[r] & {} & {} \arrow[r] & \{n_B\}_{kB}
\end{tikzcd}
$$

How do they do?

- Flow
  - Role-based
- Constituents
  - Informal math.
- Environment
  - Side remarks
- Goals
  - Side remarks

- Unambiguous 😐
- Simple ☺
- Flexible ☹
- Powerful ☹
- Insightful ☺

MSR 1.x - Initiator

Nonce generation

Message transmission

$$\pi_{A0}(A) \rightarrow L_0(A), \pi_{A0}(A)$$

$$L_0(A), \pi_{A1}(B) \rightarrow \exists n_A.\ L_1(A,B,n_A), N(\{n_A,A\}_{kB}), \pi_{A1}(B)$$

$$L_1(A,B,n_A), N(\{n_A,n_B\}_{kA}) \rightarrow L_2(A,B,n_A,n_B)$$

$$L_2(A,B,n_A,n_B) \rightarrow L_3(A,B,n_A,n_B), N(\{n_B\}_{kB})$$

where $\pi_{A0}(A) = Pr(A), PrvK(A,k_A^{-1})$

$\pi_{A1}(B) = Pr(B), PubK(B,k_B)$

MSR 1.x - Responder

Role state predicate

$$\pi_{B0}(B) \rightarrow L_0(B), \pi_{B0}(B)$$

$$L_0(A), \pi_{B1}(A), N(\{n_A, A\}_{kB}) \rightarrow L_1(A,B,n_A), \pi_{B1}(A)$$

$$L_1(A,B,n_A) \rightarrow \exists n_B.\ L_2(A,B,n_A,n_B), N(\{n_A,n_B\}_{kA})$$

$$L_2(A,B,n_A,n_B), N(\{n_B\}_{kB}) \rightarrow L_3(A,B,n_A,n_B)$$

Persistent Info.

where $\pi_{B0}(B) = Pr(B), PrvK(B,k_B^{-1})$

$\pi_{B1}(A) = Pr(A), PubK(A,k_A)$

How did we do?

- Flow
  - Role-based
- Constituents
  - Persistent info.
- Environment
  - *In part*
- ~~Goals~~

- Unambiguous ☺
- Simple 😐
- Flexible ☹
- Powerful ☹
- Insightful ☺

How *will* we do?

- Flow
  - Role-based
- Constituents
  - Strong typing
- Environment
  - *In part*
- ~~Goals~~

- Unambiguous ☺
- Simple ☺
- Flexible ☺
- Powerful ☺
- Insightful ☺

Part II

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
- Subsumes persistent information
  - Static
  - Local
  - Mandatory

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 <u>subprotocols</u>
  - Communicate data
  - Pass control
- Interface to outside system
- Implements intruder

Constraints

New

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

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

Role owner

- Anchored roles

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

MSR 2.0 – NS Initiator

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

MSR 2.0 – NS Responder

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

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

Access Control

New

$\Sigma \Vdash P$

P is AC-valid in Σ

r is AC-valid for A in Γ

$\Gamma \Vdash_A r$

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

Snapshots

Active role set

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

State

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

Signature

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

Execution Model

1-step firing

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

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

Rule application

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

- Constraint check

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

- Firing

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

Properties

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

New

Completed Case-Studies

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

Part III

The Most Powerful Attacker

Execution with an Attacker

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

- 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

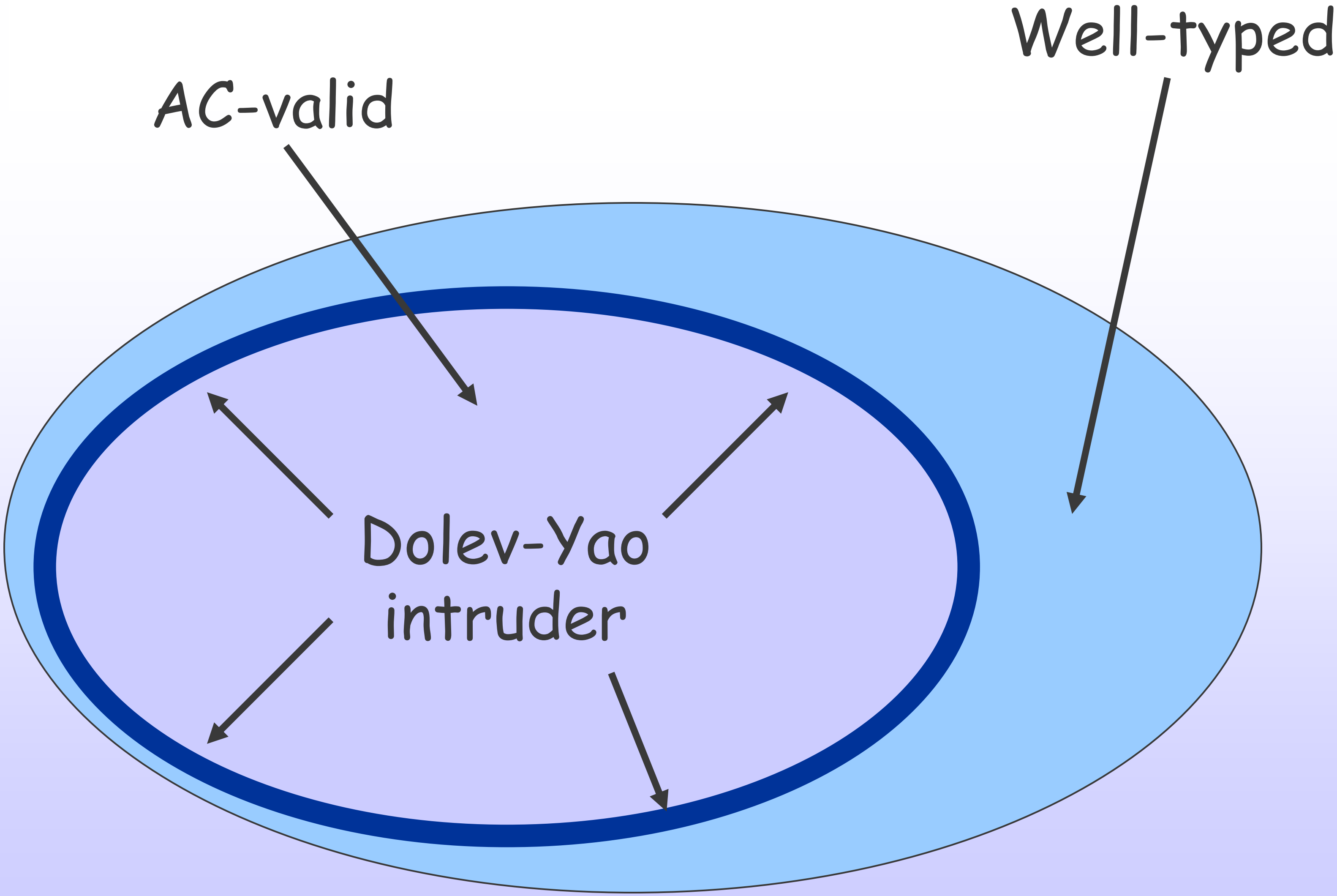
$$\left[\bullet \rightarrow \exists n{:}\mathrm{nonce}.\ M_I(n)\right]^I \qquad \left[\bullet \rightarrow \exists m{:}\mathrm{msg}.\ M_I(m)\right]^I$$

- Anything else ?

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

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

DY Intruder Stretches AC to Limit



Completeness of D-Y Intruder

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

  with all well-typed and *AC*-valid

- Then

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

Encoding of **P**, S, Σ

P Remove roles anchored on **I**

S Map **I**'s state / mem. pred. using M_I

Σ Remove **I**'s role state pred.; add M_I

Encoding of R

- No encoding on structure of R
  - Lacks context!

- Encoding on *AC*-derivation for R

$$A :: \Sigma \Vdash R$$

  - Associate roles from P_{DY} to each *AC* rule

Completeness proof

- Induction on execution sequence
- Simulate every step with P_{DY}
  - Rule application
    - Induction on *AC*-derivation for R
    - Every *AC*-derivation maps to execution sequence relative to P_{DY}
  - Rule instantiation
    - *AC*-derivations preserved
    - Encoding unchanged

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

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