



# 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”



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

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

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

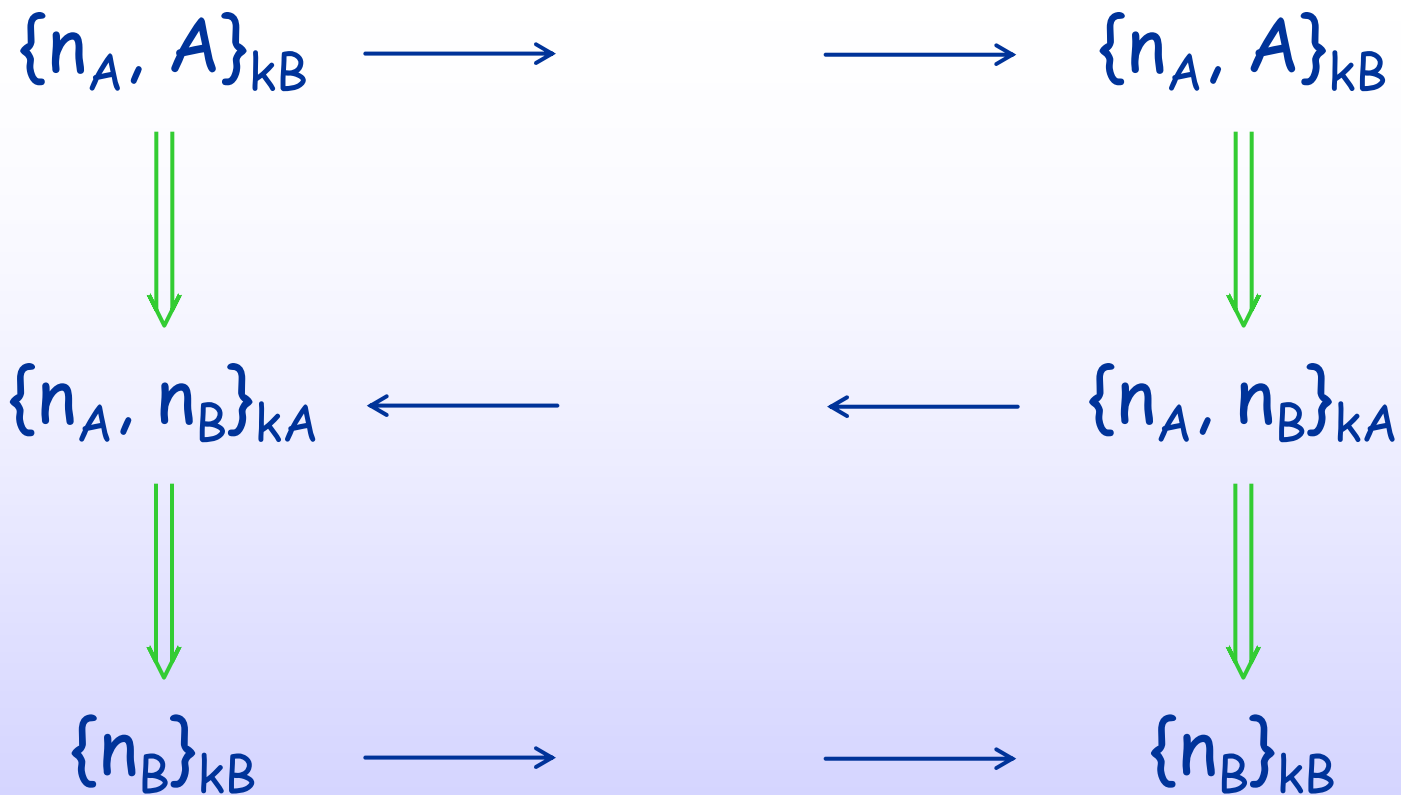
# How does it do?

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

- Unambiguous 😞
- Simple 😊
- Flexible 😊
- Powerful 😊
- Insightful 😐



# Strands



# 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

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

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

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

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

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

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

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

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

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

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 
- Memory predicates 
- Constraints 

# Terms

- Atomic terms

- Principal names  $A$
- Keys  $k$
- Nonces  $n$
- ...

- Term constructors

- $(\_ \_)$
- $\{ \_ \} \_ \quad \{ \{ \_ \} \} \_$
- $[ \_ ] \_$
- ...

**D**  
**e**  
**f**  
**i**  
**n**  
**a**  
**b**  
**i**  
**e**



# 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
- $L(t, \dots, t)$  Local state
- $M_A(t, \dots, t)$  Memory
- $\chi$  Constraints

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

# Types of Terms

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

Types can depend on term

- Captures relations between objects
- Subsumes persistent information
  - Static
  - Local
  - Mandatory

# Subtyping

$\tau :: \text{msg}$

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



# Role state predicates

$$L_i(A, t, \dots, t)$$

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



# Memory Predicates



$$M_A(t, \dots, t)$$

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



# Constraints



$\chi$

- 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

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

A thought bubble containing the expression  $\sum x. \tau. \underline{\tau}$  is connected by a line to a callout bubble pointing to the  $\underline{\tau}$  term in the expression  $\tau(x) \times \underline{\tau}$ . The callout bubble contains the variable  $x$ .

- Forces associations among arguments

➤ E.g.:  $\text{princ}^{(A)} \times \text{pubK } A^{(k_A)} \times \text{privK } k_A$



# Roles

Role state pred.  
var. declarations

- Generic roles

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

Role  
owner

- Anchored roles

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



# MSR 2.0 – NS Initiator



$$\left( \begin{array}{l}
 \exists \textcolor{red}{L}_B: \text{princ } x \times \text{princ}^{(B)} x \times \text{pubK } B \times \text{nonce}. \\
 \\
 \forall B: \text{princ} \\
 \forall k_B: \text{pubK } B \quad \bullet \quad \rightarrow \quad \exists \textcolor{red}{n}_A: \text{nonce}. \quad \begin{array}{l} \textcolor{green}{L}(A, B, k_B, n_A) \\ \textcolor{blue}{N}(\{n_A, A\}_{k_B}) \end{array} \\
 \\
 \forall \dots \\
 \forall k_A: \text{pubK } A \quad \textcolor{green}{L}(A, B, k_B, n_A) \\
 \forall k'_A: \text{privK } k_A \quad \textcolor{blue}{N}(\{n_A, n_B\}_{k_A}) \quad \rightarrow \quad \textcolor{blue}{N}(\{n_B\}_{k_B}) \\
 \forall n_A, n_B: \text{nonce}
 \end{array} \right)^{\forall A}$$

# MSR 2.0 – NS Responder



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

$$N(\{n_A, A\}_{k_B}) \rightarrow \exists n_B: \text{nonce}. \mathcal{L}(\dots) N(\{n_A, n_B\}_{k_A})$$

# Type Checking



$\Sigma \vdash P$

$t$  has type  
 $\tau$  in  $\Gamma$

$\Gamma \vdash t : \tau$

$P$  is well-  
typed in  $\Sigma$

- Catches:
  - Encryption with a nonce
  - Transmission of a long term key
  - Circular key hierarchies, ...
- Static and dynamic uses
- Decidable



# Access Control



$r$  is AC-valid  
for  $A$  in  $\Gamma$

$\Sigma \Vdash P$

$P$  is AC-  
valid in  $\Sigma$

$\Gamma \Vdash_A r$

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



# Snapshots



$$C = [S]^R_{\Sigma}$$

Active role  
set

## State

- $N(t)$
- $L_I(t, \dots, t)$
- $M_A(t, \dots, t)$

## Signature

- $a : \tau$
- $L_I : \underline{\tau}$
- $M_{\underline{\quad}} : \underline{\tau}$

# Execution Model



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

1-step  
firing

- 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

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

- Firing

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



# Properties

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





# 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

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

- 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

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

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

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

- No nonces, no other keys, ...



# DY Intruder – Data Generation

- Safe data

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

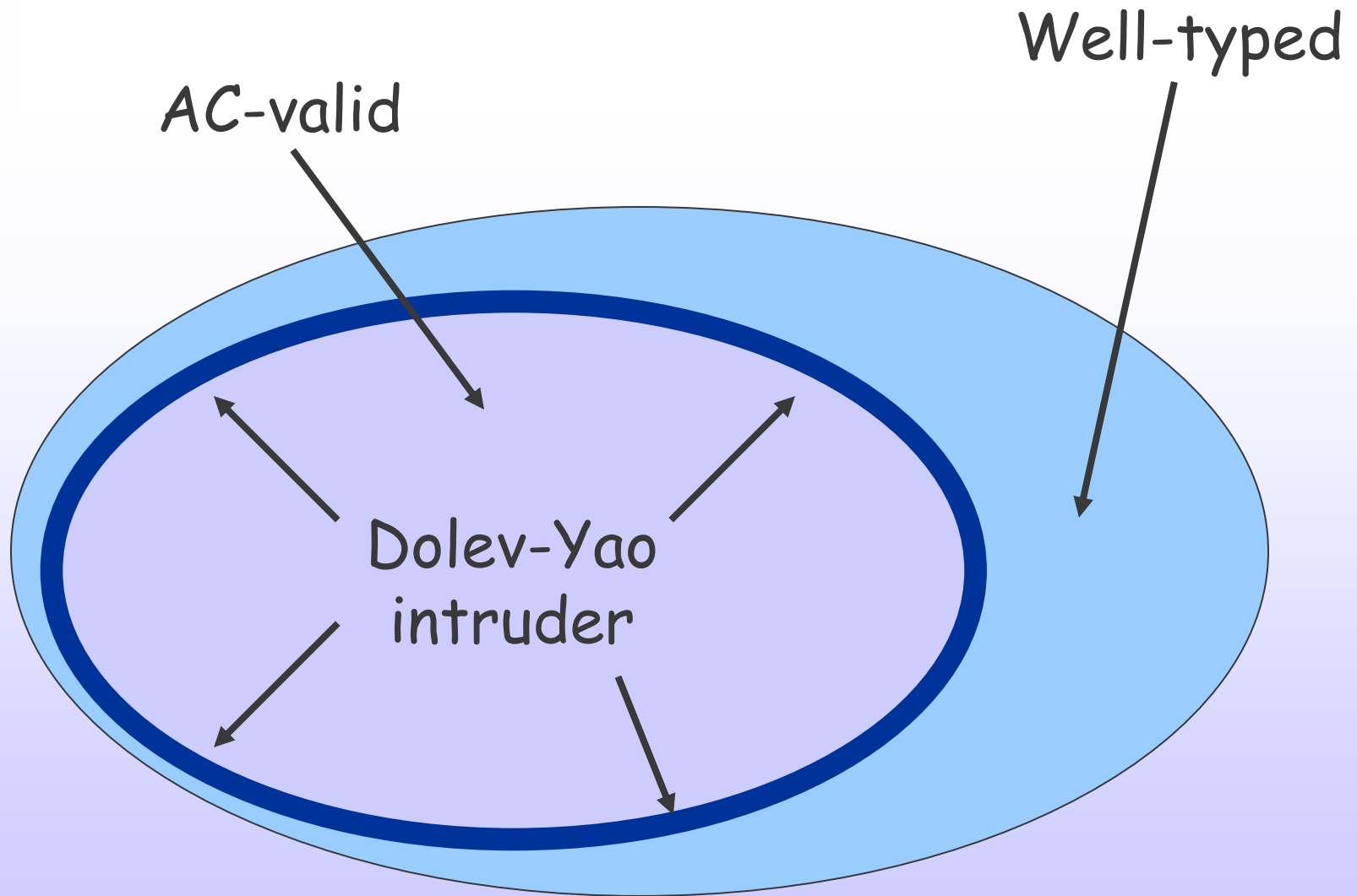
- Anything else ?

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

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



# DY Intruder Stretches AC to Limit



# Completeness of D-Y Intruder

- If  $P \triangleright [S]_{\Sigma}^R \rightarrow [S']_{\Sigma'}^{R'}$   
with all well-typed and AC-valid

- Then

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





# Encoding of $\mathbf{P}$ , $\mathbf{S}$ , $\Sigma$

$\mathbf{P}$  Remove roles anchored on  $\mathbf{I}$

$\mathbf{S}$  Map  $\mathbf{I}$ 's state / mem. pred. using  $\mathbf{M}_{\mathbf{I}}$

$\Sigma$  Remove  $\mathbf{I}$ 's role state pred.; add  $\mathbf{M}_{\mathbf{I}}$



# Encoding of $R$

- No encoding on structure of  $R$ 
  - Lacks context!
- Encoding on AC-derivation for  $R$

$$A :: \Sigma \parallel - R$$

- 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