COMx501: Computer Security and Forensics

Achim D. Brucker

a.brucker@sheffield.ac.uk **https://www.brucker.ch/**

Software Assurance & Security Research
Department of Computer Science, The University of Sheffield, Sheffield, UK
https://logicalhacking.com/

March 21, 2018

{* LogicaλHacking *}.com

The University Of Sheffield.

COMx501: Computer Security and Forensics

Part 8a: Formal Analysis of Security Protocols

Achim D. Brucker

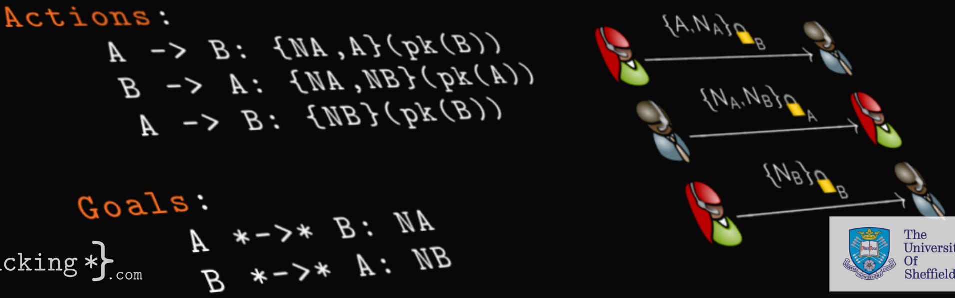
a.brucker@sheffield.ac.uk https://www.brucker.ch/

Software Assurance & Security Research

Department of Computer Science, The University of Sheffield, Sheffield, UK

https://logicalhacking.com/

March 6, 2018



{* LogicaλHacking *}.com

The University Of Sheffield.

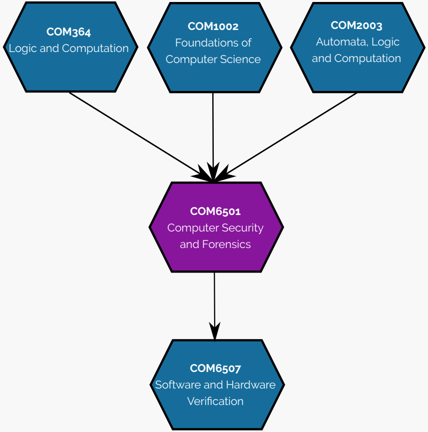
Outline

What Are Formal Methods?

What we have seen so far

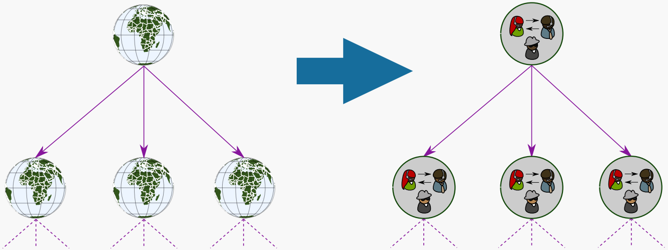
Designing correct security protocols is difficult!

- A language is formal when it has a well-defined syntax and semantics. Additionally there is often a deductive system for determining the truth of statements.
  Examples: propositional logic, first-order logic.
- A model (or construction) is formal when it is specified in a formal language.
- Standard protocol notation is not formal.
- We will see today how to formalize such notations.

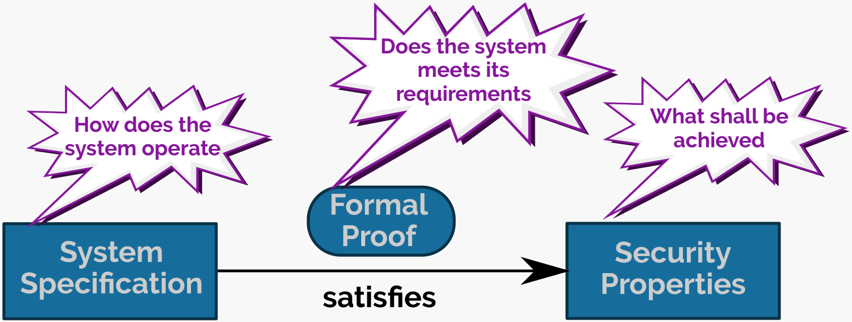


Goal: formally model protocols and their properties and provide a mathematically sound means for reasoning about these models

Basis: suitable abstraction of protocols

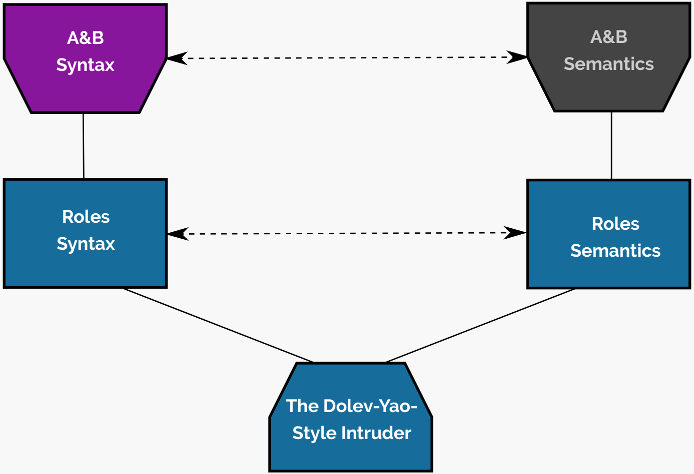


Analysis: with formal methods based on mathematics and logic

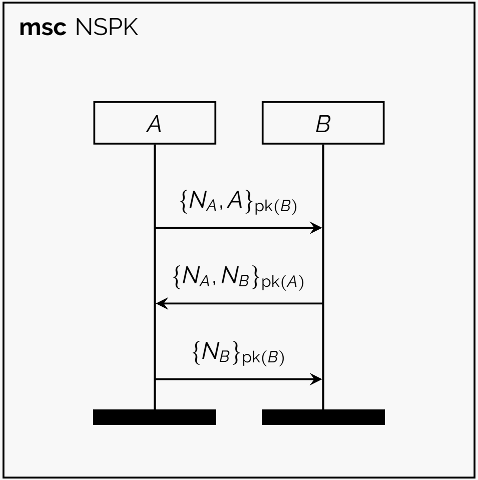

How does the system operate
Does the system meets its requirements
What shall be achieved
Formal Proof
System Specification
satisfies
Security Properties

Outline

Overview


A&B
Syntax
A&B
Semantics
Roles
Syntax
Roles
Semantics
The Dolev-Yao-
Style Intruder

Alice & Bob (AnB) Notation: Message Sequence Charts



A Formal Alice & Bob Language: Syntax

Protocol: NSPK

Types:

Agent: A, B;
Number: N_A, N_B;
Function: pk;

Knowledge:

A: A, B, pk, inv(pk(A));
B: B, pk, inv(pk(B));

Actions:

$$A \longrightarrow B: \ \{N_A, A\}_{\mathrm{pk}(B)}$$
$$B \longrightarrow A: \ \{N_A, N_B\}_{\mathrm{pk}(A)}$$
$$A \longrightarrow B: \ \{N_B\}_{\mathrm{pk}(B)}$$

Goals:

$$A \bullet\!\!\rightarrow\!\!\bullet \ B : N_A$$
$$B \bullet\!\!\rightarrow\!\!\bullet \ A : N_B$$

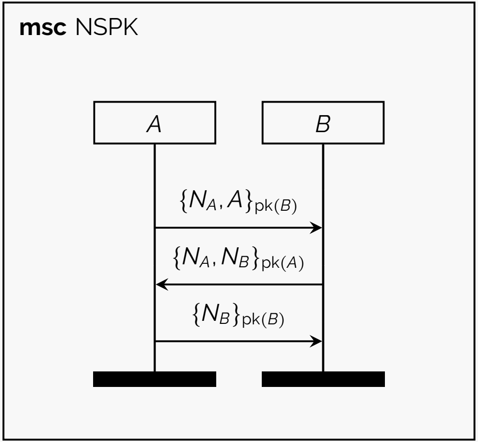
- Protocol:
  Name of the protocol
- Types:
  Types of all identifiers
  (we might not consider them in the analysis)
- Knowledge:
  Initial knowledge of each role
- Actions:
  The exchanged messages
- Goals:
  The goals that we want to achieve:
  - $A \bullet\!\!\rightarrow\!\!\bullet B : N_A$ secure transmission of nonce N_A from A to B
  - $A \bullet\!\!\rightarrow B : N_A$ authentic transmission of nonce N_A from A to B
  - $A \rightarrow\!\!\bullet B : N_A$ confidential transmission of nonce N_A from A to B
- Finally, a plain text notation for tools

A Formal Alice & Bob Language: Syntax

```
Protocol: NSPK

Types:
     Agent A,B;
     Number NA,NB;
     Function pk;

Knowledge:
     A: A, B, pk, inv(pk(A));
     B: B, pk, inv(pk(B));

Actions:
     A -> B: {NA,A}(pk(B))
     B -> A: {NA,NB}(pk(A))
     A -> B: {NB}(pk(B))

Goals:
     A *->* B: NA
     B *->* A: NB
```

- Protocol:
  Name of the protocol
- Types:
  Types of all identifiers
  (we might not consider them in the analysis)
- Knowledge:
  Initial knowledge of each role
- Actions:
  The exchanged messages
- Goals:
  The goals that we want to achieve:
  - $A \bullet\!\!\rightarrow\!\!\bullet\ B : N_A$ secure transmission of nonce N_A from A to B
  - $A \bullet\!\!\rightarrow\ B : N_A$ authentic transmission of nonce N_A from A to B
  - $A \rightarrow\!\!\bullet\ B : N_A$ confidential transmission of nonce N_A from A to B
- Finally, a plain text notation for tools

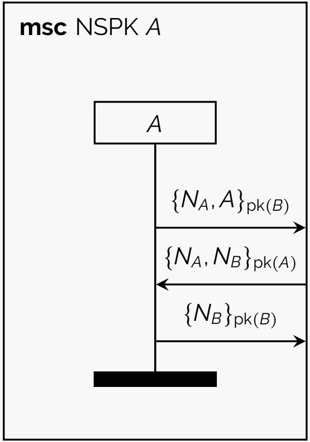
Idea:
Split a message sequence chart into single roles (aka chords, symbolic strands, role scripts):

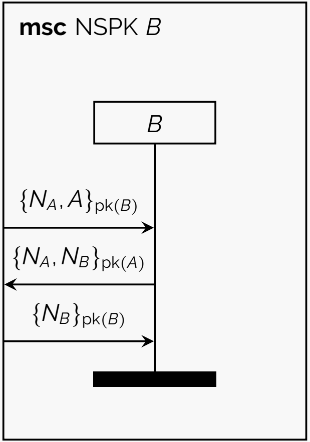


Towards a (Formal) Meaning of AnB Specifications (1/2)

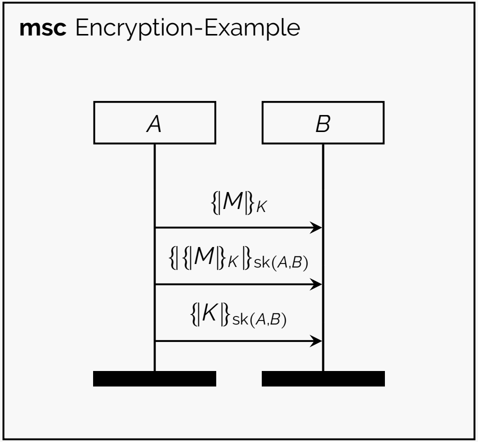
Idea:

Split a message sequence chart into single roles (aka chords, symbolic strands, role scripts):





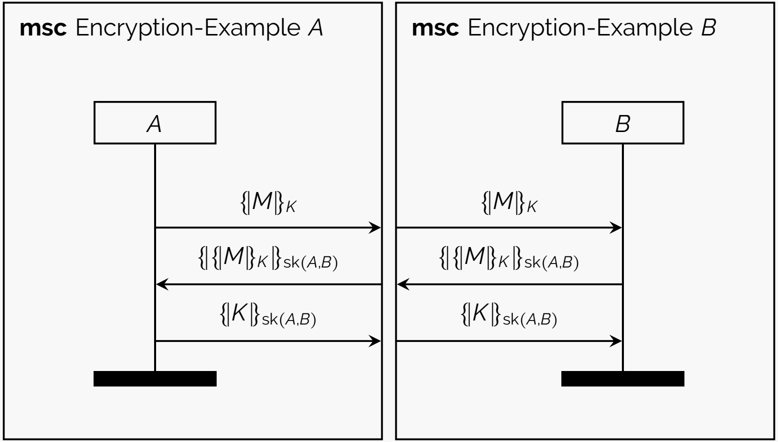
Note: Non-trivial for some protocols:



Where $sk(A, B)$ is a shared key of A and B, K is fresh

Towards a (Formal) Meaning of AnB Specifications (2/2)

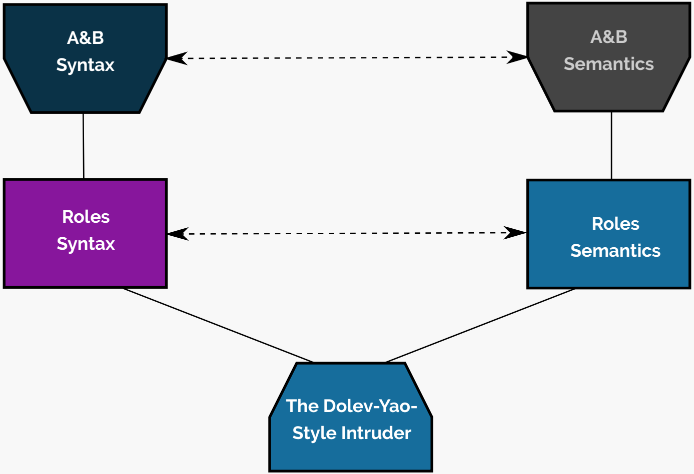
Note: Non-trivial for some protocols:



Where sk(A, B) is a shared key of A and B, K is fresh

Outline

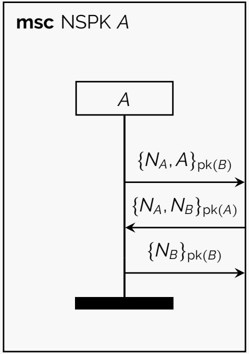
Overview



A&B Syntax
A&B Semantics
Roles Syntax
Roles Semantics
The Dolev-Yao-Style Intruder

Roles Syntax

- Graphical:



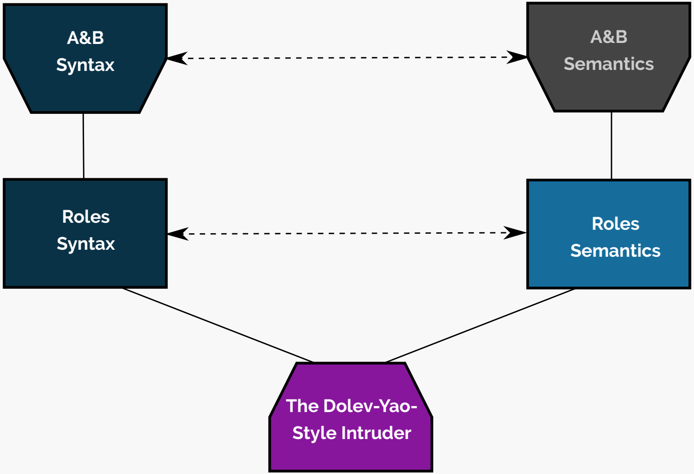
- Textual:

$$\mathsf{snd}(\{N_A, A\}_{\mathsf{pk}(B)}) \cdot \mathsf{rcv}(\{N_A, N_B\}_{\mathsf{pk}(A)}) \cdot \mathsf{snd}(\{N_B\}_{\mathsf{pk}(B)})$$

  - $\mathit{Event} = \mathsf{snd}(\mathit{Term}) \mid \mathsf{rcv}(\mathit{Term}) \mid \mathsf{sig}(\mathit{SID}, \mathit{Term})$
  - $\mathit{Roles} = \mathit{Event}^*$
  - $\mathit{Protocols} = \mathit{Rolename} \rightharpoonup \mathit{Role}$
- Role scripts
  - A protocol is described by a role script for each role name
  - Role names are variables of type agent
  - Signal (sig) events are used for property specification and verification

Outline

Overview



A&B Syntax
A&B Semantics
Roles Syntax
Roles Semantics
The Dolev-Yao-Style Intruder

On the Security of Public Key Protocols (IEEE Trans. Inf. Th., 1983)

- Consider a public key system in which for every user X
  - there is a public encryption function E_X
    - every user can apply this function
  - and a private decryption function D_X
    - only X can apply this function
  - These functions have the property that $E_X D_X = D_X E_X = 1$
- The **Dolev-Yao intruder**:
  - Controls the network (read, intercept, send)
  - Is also a user, called Z
  - Can apply E_X for any X
  - Can apply D_Z

On the Security of Public Key Protocols

DANNY DOLEV AND ANDREW C. YAO, MEMBER, IEEE

I. INTRODUCTION

0018-9448/83/0300-0198$01.00 ©1983 IEEE

Message Term Algebra (1/4)

Signature Σ:
set of function symbols for cryptographic and non-cryptographic operations, for instance:

Symbol	Arity	Meaning	Public
i	0	name of the intruder	yes
$\mathrm{inv}(\cdot)$	1	private-key of a given public-key	no
$\{\cdot\}_{\cdot}$	2	asymmetric encryption	yes
$\{\!\vert \cdot \vert\!\}_{\cdot}$	2	symmetric encryption	yes
$\langle \cdot, \cdot \rangle$	2	pairing / concatenation	yes
$\exp(\cdot, \cdot)$	2	exponentiation module fixed prime p	yes
$\cdot \oplus \cdot$	2	bitwise exclusive or	yes
$f(\cdot)$	1	user-defined function symbol f	user-defined

$\Sigma_P \subseteq \Sigma$ to denote the public functions:
every agent (including the intruder) can apply the function to known messages

Let $\Sigma_0 \subseteq \Sigma$ be a countable set of constants

- convention: constants start with a lower-case letter, e.g., *alice*, *na17*, *k2*, ...

Let $\mathcal{V}$ be a countable set of variables with $\mathcal{V} \cap \Sigma = \emptyset$

- convention: variables start with a upper-case letter, e.g., A, NA, K, ...

Definition (Set of Message Terms)

The set $\mathcal{T}_\Sigma(\mathcal{V})$ of all message terms is the least set such that

- $\mathcal{V} \subseteq \mathcal{T}_\Sigma(\mathcal{V})$
- If $t_1, \ldots, t_n \in \mathcal{T}_\Sigma(\mathcal{V})$ and $f \in \Sigma$ and f has arity n, then also $f(t_1, \ldots, t_n) \in \mathcal{T}_\Sigma(\mathcal{V})$

For $\mathcal{V} = \emptyset$, we write $\mathcal{T}_\Sigma$, the set of all ground message terms.

Message Term Algebra (3/4)

Definition (Σ-Algebra)

For a signature Σ, a Σ-Algebra $\mathcal{A}$ consists of a universe A and interprets every symbol $f \in \Sigma$ of arity n as a function $f^{\mathcal{A}} : A^n \to A$.
We extend this interpretation to terms as follows:

$$
\begin{aligned}
v^{\mathcal{A}} &= v && \text{for } v \in \mathcal{V} \\
(f(t_1, \dots, t_n))^{\mathcal{A}} &= f^{\mathcal{A}}(t_1^{\mathcal{A}}, \dots, t_n^{\mathcal{A}})
\end{aligned}
$$

We write $t_1 \approx_{\mathcal{A}} t_2$ for $t_1^{\mathcal{A}} = t_n^{\mathcal{A}}$, and simply $t_1 \approx t_2$ when $\mathcal{A}$ is clear.

Example: $A = \{0, 1\}^{2048}$, $\{\!|\cdot|\!\}_{\cdot}^{\mathcal{A}}$ = description-of-AES, …

Message Term Algebra (4/4)

The most simple and most widely used interpretation is the free message term algebra.

Definition (Free Message Term Algebra)

In the free message term algebra, $A = \mathcal{T}_\Sigma(\mathcal{V})$ and

$$f^{\mathcal{A}}(t_1, \dots, t_n) = f(t_1, \dots, t_n)$$

In the free algebra

- We interpret every term by itself
- $t_1 \approx t_2$ iff $t_1 = t_2$
- $a \not\approx b$ for any distinct constants a and b
- If $m_1 \not\approx m_2$ then also $h(m_1) \not\approx h(m_2)$
  (Thus, hash functions are collision free)
- $Dec_x(Enc_x(M)) \not\approx M$ for any functions Dec and Enc ...
- $\exp(\exp(g, X), Y) \not\approx \exp(\exp(g, Y), X)$

Algebraic Properties

Definition

A set of equations E induces a congruence relation $\approx_E$ on terms and thus the equivalence class $[t]_E$ of a term E. The quotient algebra $\mathcal{T}_\Sigma(\mathcal{V})_{/\approx_E}$ interprets each term by its equivalence class.

Example

$$\{\{M\}_K\}_{\mathsf{inv}(K)} \approx M \qquad \mathsf{inv}(\mathsf{inv}(K)) \approx K$$
$$\{\!|\{\!|M|\!\}_K|\!\}_K \approx M \qquad \exp(\exp(B, X), Y) \approx \exp(\exp(B, Y), X)$$

- Two terms are semantically equal iff that is a consequence of E
- For the above example equations:
  - $a \not\approx b$ for any distinct constants a and b
  - If $m_!\not\approx m_2$ then also $h(m_1) \not\approx h(m_2)$
  - $\{\{M\}_{\mathsf{inv}(K)}\}_K \approx M$
  - $\{\!|\{\!|M|\!\}_{\exp(\exp(g,Y),X)}|\!\}_{\exp(\exp(g,X),Y)} \approx M$

Substitution (1/2)

Definition (Substitution)

A substitution $\sigma = [x_1 \mapsto t_1, \dots, x_n \mapsto t_n]$ is the following function from $\mathcal{V}$ to $\mathcal{T}_\Sigma(\mathcal{V})$:

$$\sigma(x) = \begin{cases} t_1 & \text{if } x = x_i \\ x & \text{otherwise} \end{cases}$$

where $\{x_1, \dots, x_n\}$ is called the domain $\mathrm{dom}(\sigma)$ of σ.
We use postfix notation (writing $x\sigma$ instead of $\sigma(x)$) and extend σ to a homomorphism over $\mathcal{T}_\Sigma(\mathcal{V})$:

$$f(t_1, \dots, t_n)\sigma = f(t_1\sigma, \dots, t_n\sigma)$$

Example

For $\sigma = [x \mapsto f(y), y \mapsto z]$ we have $g(z, y, f(x))\sigma = g(z, z, f(f(y)))$

Definition

We denote with $\sigma\tau$ the composition of substitutions σ and τ, i.e., $\tau \circ \sigma$
The substitution σ is more general than the substitution τ on the set of variables V, written $\sigma \preceq^{V} \tau$, iff there is a substitution ρ such that for all $x \in V$ holds $x\sigma\rho = x\tau$.
For $V = \mathcal{V}$, we simply write $\sigma \preceq \tau$.

Example

Let

- $\sigma_1 = [x \mapsto f(y)]$,
- $\sigma_2 = [x \mapsto f(c)]$,
- $\sigma_3 = [x \mapsto f(c), y \mapsto c]$,
- $\rho = [y \mapsto c]$

then we have

- $\sigma_1 \preceq^{\{x\}} \sigma_2$ since $x\sigma_1\rho = x\sigma_2$
- $\sigma_1 \not\preceq \sigma_2$: for any ρ such that $\sigma_1 \preceq \sigma_2$ we have that $y\rho = c$, thus $y\sigma_1\rho \neq y\sigma_2$
- $\sigma_1 \preceq \sigma_3$ since $\sigma_1\rho = \sigma_3$
- $\sigma_2 \preceq \sigma_3$ since $\sigma_2\rho = \sigma_3$

Definition (Unification)

Given a set $\{(s_1, t_1), \ldots, (s_n, t_n)\}$ of pairs of terms. A unifier σ for this unification problem is a substitution such that

$$s_1\sigma \approx t_1\sigma \text{ and } \cdots \text{ and } s_n\sigma \approx t_n\sigma$$

Matching is a variant of unification where

$$s_1\sigma \approx t_1 \text{ and } \cdots \text{ and } s_n\sigma \approx t_n$$

(Special case of unification if the t_i are ground)

Example

- $\{(f(x, c), f(g(y), y))\}$ has the unifier $[x \mapsto g(c), y \mapsto c]$
- $\{(f(x, c), g(y))\}$ has no unifier—in the free algebra

Definition (Unification Algorithm)

Input: A unification problem U and a substitution σ, initially empty

Output: A substitution or failure

If $U = \emptyset$ then return σ. Otherwise pick any pair (s, t) in U and

- if $s = t$, remove this pair and continue
- if s is a variable
  - if $s \in vars(t)$: return with failure
  - otherwise: remove the pair from U and continue with $\sigma[s \mapsto t]$
- if t is a variable: analogous to previous case
- otherwise, i.e., $s = f(s_1, \ldots, s_n)$ and $t = g(t_1, \ldots, t_m)$:
  - if $f = g$ (and thus $n = m$): replace in U the pair (s, t) with the pairs $\{(s_1, t_1), \ldots, (s_n, t_n)\}$ and continue
  - if $f \neq g$: return with failure

Unification in the Free Algebra

Definition (Unification Algorithm)

Input: A unification problem U and a substitution σ, initially empty

Output: A substitution or failure

$$\vdots$$

Theorem

- *If the algorithm returns failure then, the unification problem has no unifier (in the free algebra)*
- *If the algorithm returns a substitution σ, then σ is the most general unifier (in the free algebra), i.e., for all unifiers τ it holds that $\sigma \preceq \tau$*

Unification in Other Algebras

- When considering other algebras, unifiability is in general undecidable, e.g. associativity and distributivity
- Similarly, matchability and ground equality of terms are in general undecidable, e.g. encoding computational logic
- Even when decidable, there is in general no unique most general unifier, e.g., $\{\exp(X, Y), \exp(X', c)\} \ldots$
- Some unification problems are decidable but *infinitary*: in general, there is an infinite set of most general unifiers, e.g. associativity

Definition (Dolev-Yao Closure)

Given a set of terms M, we define $\mathcal{DY}(M)$ as the least closure of M under the following rules:

$$\frac{}{m \in \mathcal{DY}(M)}\ \text{Axiom}\ (m \in M) \qquad \frac{s \in \mathcal{DY}(M)}{t \in \mathcal{DY}(M)}\ \text{Algebra}\ (s \approx t)$$

$$\frac{t_1 \in \mathcal{DY}(M) \quad \cdots \quad t_n \in \mathcal{DY}(M)}{f(t_1, \ldots, f_n) \in \mathcal{DY}(M)}\ \text{Composition}\ (f \in \Sigma_p) \qquad \frac{\langle m_1, m_2 \rangle \in \mathcal{DY}(M)}{m_i \in \mathcal{DY}(M)}\ \text{Proj}_i$$

$$\frac{\{\!|m|\!\}_k \in \mathcal{DY}(M) \quad k \in \mathcal{DY}(M)}{m \in \mathcal{DY}(M)}\ \text{DecSym}$$

$$\frac{\{m\}_k \in \mathcal{DY}(M) \quad \text{inv}(k) \in \mathcal{DY}(M)}{m \in \mathcal{DY}(M)}\ \text{DecAsym} \qquad \frac{\{m\}_{\text{inv}(k)} \in \mathcal{DY}(M)}{m \in \mathcal{DY}(M)}\ \text{OpenSig}$$

Example

Let $M = \{x, \{\!|b, \exp(g,y)|\!\}_k, k, m\}$, can we prove

$$\{\!|m|\!\}_{\exp(\exp(g,x),y)} \overset{?}{\in} \mathcal{DY}(M)$$

Proof:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\{\!|b, \exp(g,y)|\!\}_k \in \mathcal{DY}(M)} \qquad \dfrac{}{k \in \mathcal{DY}(M)}}{\langle b, \exp(g,y)\rangle \in \mathcal{DY}(M)}\,\text{DecAsym}}{\exp(g,y) \in \mathcal{DY}(M)}\,\text{Proj}_2 \qquad \dfrac{}{x \in \mathcal{DY}(M)}}{\exp(\exp(g,y),x) \in \mathcal{DY}(M)}\,\text{Comp.}}{\exp(\exp(g,x),y) \in \mathcal{DY}(M)}\,\text{Alg.} \qquad \dfrac{}{m \in \mathcal{DY}(M)}}{\{\!|m|\!\}_{\exp(\exp(g,x),y)} \in \mathcal{DY}(M)}\,\text{Comp.}$$

Example

Let $M = \{x, \{|b, \exp(g, y)|\}_k, k, m\}$, can we prove

$$\{|m|\}_{\exp(\exp(g,x),y)} \overset{?}{\in} \mathcal{DY}(M)$$

Proof (using notation from Huth&Ryan/COM2003/364):

1	$\{\|b, \exp(g, y)\|\}_k \in \mathcal{DY}(M)$	premise
2	$k \in \mathcal{DY}(M)$	premise
3	$x \in \mathcal{DY}(M)$	premise
4	$m \in \mathcal{DY}(M)$	premise
5	$\langle b, \exp(g, y)\rangle \in \mathcal{DY}(M)$	DecAsym 1,2
6	$\exp(g, y) \in \mathcal{DY}(M)$	Proj_2
7	$\exp(\exp(g, y), x) \in \mathcal{DY}(M)$	Composition 6,3
8	$\exp(\exp(g, x), y) \in \mathcal{DY}(M)$	Alg. Prop.
9	$\{\|m\|\}_{\exp(\exp(g,x),y)} \in \mathcal{DY}(M)$	Composition 8,4

Explicit vs. Implicit Destructors

Implicit Destructor Rules (no destruction operation)

$$\frac{\langle m_1, m_2 \rangle \in \mathcal{DY}(M)}{m_i \in \mathcal{DY}(M)}\ \mathrm{Proj}_i \qquad \frac{\{|m|\}_k \in \mathcal{DY}(M) \quad k \in \mathcal{DY}(M)}{m \in \mathcal{DY}(M)}\ \mathrm{DecSym}$$

$$\frac{\{m\}_k \in \mathcal{DY}(M) \quad \mathrm{inv}(k) \in \mathcal{DY}(M)}{m \in \mathcal{DY}(M)}\ \mathrm{DecAsym} \qquad \frac{\{m\}_{\mathrm{inv}(k)} \in \mathcal{DY}(M)}{m \in \mathcal{DY}(M)}\ \mathrm{OpenSig}$$

versus

Explicit Destructors with algebraic properties

$$\pi_1(\langle m_1, m_2 \rangle) \approx m_1 \qquad \{\{m\}_k\}_{\mathrm{inv}(k)} \approx m \qquad \mathrm{open}(\{m\}_{\mathrm{inv}(k)}) \approx m$$

$$\pi_2(\langle m_1, m_2 \rangle) \approx m_2 \qquad \{|\{|m|\}_k|\}_k \approx m$$

- Implicit destructor rules are redundant with these properties
- Explicit has strictly more derivable messages
- Considerably more difficult to handle

What We Learned Today

- Security protocols are difficult to design
- Formal methods help to define protocols and security goals precisely
- Formal methods helps

Note:

- Today's lecture was a "deep dive", not all details are equally relevant for the exam
- In the exam you should be able to
  - work with the (graphical and textual) AnB notation (slides 207 - 209)
  - work with the core of the Dolev-Yao Intruder (slides 221 and 228)
  - be able to prove simple properties (slides 229, 230)
  - solve problems similar to the ones discussed in the homework (and mock exam)
- We will practice this in the first lab using OFMC

Thank you for your attention!
Any questions or remarks?

Contact:

Dr. Achim D. Brucker
Department of Computer Science
University of Sheffield
Regent Court
211 Portobello St.
Sheffield S1 4DP, UK

a.brucker@sheffield.ac.uk
@adbrucker
https://de.linkedin.com/in/adbrucker/
https://www.brucker.ch/
https://logicalhacking.com/blog/

Bibliography I

Achim D. Brucker and Sebastian A. Mödersheim.

Integrating automated and interactive protocol verification.

In Pierpaolo Degano and Joshua Guttman, editors, *Workshop on Formal Aspects in Security and Trust (FAST 2009)*, number 5983 in Lecture Notes in Computer Science, pages 248–262. Springer-Verlag, 2009.

An extended version of this paper is available as IBM Research Technical Report, RZ3750.

D. Dolev and A. C. Yao.

On the security of public key protocols.

Symposium on Foundations of Computer Science, 0:350–357, 1981.

Michael Huth and Mark Ryan.

Logic in Computer Science: Modelling and Reasoning About Systems.

Cambridge University Press, New York, NY, USA, 2004.

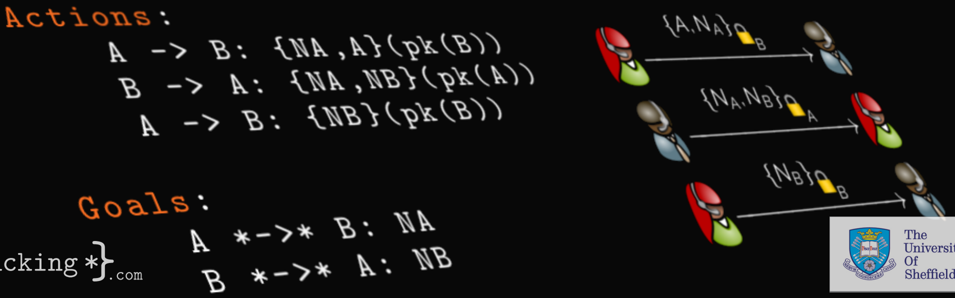
COMx501: Computer Security and Forensics
Part 8b: Formal Analysis of Security Protocols

Achim D. Brucker

a.brucker@sheffield.ac.uk **https://www.brucker.ch/**

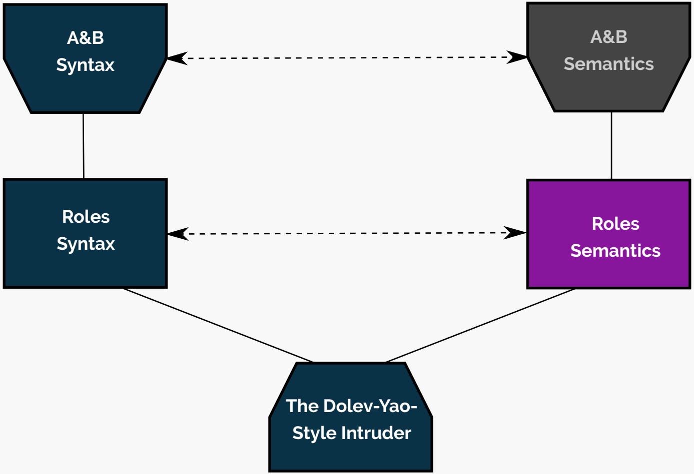
Software Assurance & Security Research
Department of Computer Science, The University of Sheffield, Sheffield, UK
https://logicalhacking.com/

March 12, 2018 (This part is not relevant for the exam or quizzes)



Outline

Overview



A&B Syntax
A&B Semantics
Roles Syntax
Roles Semantics
The Dolev-Yao-Style Intruder

Free Variables

Those variables of a chord are called free for which the first occurrence is not in a receive event:

Definition

$$f_v(protocol, role) = \{role\} \cup f'_{v_{\{role\}}}(protocol(role))$$

$$f'_{v_M}(role) = \begin{cases} f'_{v_{M \cup vars(m)}}(role') & \text{if } role = \mathrm{rcv}(m) \cdot role' \\ (vars(m) \setminus M) \cup f'_{v_{M \cup vars(m)}}(role') & \text{if } role = \mathrm{snd}(m) \cdot role' \text{ or } role = \mathrm{sig}(sig, m) \cdot role' \\ \emptyset & \text{if } role = [] \end{cases}$$

Example (NSPK)

$$f_v(NSPK, A) = \{A, B, N_A\}$$
$$f_v(NSPK, B) = \{B, N_B\}$$

Role-based Protocol Specs (aka cords, strands, scripts, etc.)

- To execute a role, we first instantiate all free variables:
  - The name of the agent playing the role
  - The name of other agents that are free in the role
  - All the freshly generated values in the role
- This yields a closed role description, i.e., one without free variables
- The remaining bound variables are placeholders for parts of messages that are going to be received and for which the value is not yet determined

Definition (State)

- $State = Trace \times IntruderKnowledge \times Threads$
- $Trace = (TID \times Event)^*$
- $IntruderKnowledge = \mathcal{P}(Term)$
- $Threads = TID \rightharpoonup Role$

where the trace and the intruder knowledge are ground and the threads are closed

The *TID* are Thread IDs for each honest agent playing a role

Initial State

- We start with an initial state $([], IK_0, th_0)$ where
  - IK_0 is the initial intruder knowledge and
  - th_0 are the threads of honest agents
- IK_0 and th_0 are parameters of the verification problem
- Many analysis methods work only for the case that $\mathrm{dom}(th_0)$ is finite
- For infinitely many threads, we need an appropriate finite representation
- For a given protocol P, all initial threads instantiate a role in P:
  - For each $tid \in \mathrm{dom}(th_0)$, $th_0(tid) = P(R)\sigma$ for some role R and such that $P(R)\sigma$ is closed.
  - Usually, σ substitutes all variables except role names with fresh constants (disjoint in all threads)
  - We denote with $role(tid)$ the protocol role R that is instantiated
  - and with $player(tid)$ we denote the player $R\sigma$ of that role

Initial State: Example

Example (An initial state that is sufficient to find the attack against NSPK)

$$
\begin{aligned}
tr_0 &= [\,] \\
IK_0 &= \{a, b, i, \mathrm{pk}(a), \mathrm{pk}(b), \mathrm{pk}(i), \mathrm{inv}(\mathrm{pk}(i))\} \\
th_0(0) &= NSPK(A)[A \mapsto a, B \mapsto i, N_A \mapsto na_0] \\
th_0(1) &= NSPK(B)[A \mapsto B, N_B \mapsto nb_0]\, \mathrm{rcv}\big(\{N_A, A\}_{\mathrm{pk}(b)}\big) \cdot \mathrm{snd}\big(\{N_A, nb_1\}_{\mathrm{pk}(A)}\big) \cdot \mathrm{rcv}\big(\{nb_1\}_{\mathrm{pk}(b)}\big)
\end{aligned}
$$

Note bound variables here

Operational Semantics

Rules

$$\frac{th(tid) = \mathsf{snd}(t) \cdot tl}{(tr, IK, th) \rightarrow \Big(tr \cdot (tid, \mathsf{snd}(t)), IK \cup \{t\}, th[tid \mapsto tl]\Big)}\ \mathsf{snd}$$

$$\frac{th(tid) = \mathsf{rcv}(t) \cdot tl \qquad \mathsf{dom}(\sigma) = var(t) \qquad t\sigma \in \mathcal{DY}(IK))}{(tr, IK, th) \rightarrow (tr \cdot (tid, \mathsf{rcv}(t\sigma)), IK, th[tid \mapsto tl\sigma])}\ \mathsf{rcv}$$

$$\frac{th(tid) = \mathsf{sig}(sig, t) \cdot tl}{(tr, IK, th) \rightarrow (tr \cdot (tid, \mathsf{sig}(sig, t)), IK, th[tid \mapsto tl])}\ \mathsf{sig}$$

Operational Semantics: Example

Example (NSPK Attack)

Trace	*th*(0)	*th*(1)
$(0, \mathsf{snd}(\{na_0, i\}_{\mathsf{pk}(i)}))$	$\mathsf{snd}(\{na_0, i\}_{\mathsf{pk}(i)})$	$\mathsf{rcv}(\{N_A, A\}_{\mathsf{pk}(b)})$
$(1, \mathsf{rcv}(\{na_0, a\}_{\mathsf{pk}(b)}))$	$\mathsf{rcv}(\{na_0, N_B\}_{\mathsf{pk}(a)})$	$\mathsf{snd}(\{N_A, nb_1\}_{\mathsf{pk}(A)})$
$(1, \mathsf{snd}(\{na_0, nb_1\}_{\mathsf{pk}(a)}))$	$\mathsf{snd}(\{N_B\}_{\mathsf{pk}(i)})$	$\mathsf{rcv}(\{nb_1\}_{\mathsf{pk}(b)})$
$(0, \mathsf{rcv}(\{na_0, nb_1\}_{\mathsf{pk}(a)}))$		
$(0, \mathsf{snd}(\{nb_1\}_{\mathsf{pk}(i)}))$		
$(1, \mathsf{rcv}(\{nb_1\}_{\mathsf{pk}(b)}))$		

Attack!

Outline

Our Goal: An Automated Verification Tool

We would like to have a program V with

- Input:
  - some description of a program (protocol) P
  - some description of a functional specification (security goals) S
- Output: “Yes” if P satisfies S, and “No” otherwise
- Bonus: n the No case, give a counter-example, i.e. an input on which P violates the specification

Sadly:

Theorem (Rice)

Let S be any non-empty, proper subset of the computable functions. Then the verification problem for S (the set of programs P that compute a function in S) is undecidable.

Our Goal: An Automated Verification Tool (Pragmatics)

There are many reasons for making the state space infinite,e .g.,

- Messages: The intruder can compose arbitrarily complex messages, $i, h(i), h(h(i)), \ldots$
- Sessions: No bound on the number of executions of the protocol. (In our model: infinitely many threads in the initial state).
- Nonces: In an unbounded number of sessions, honest agents create an infinite number of fresh nonces.

For building a useful tool, we

- may bound (a subset) of the sets to make the state space finite
- are satisfied with a tool that finds problems (semi-decision-procedure)

Today: many formal analysis tools for security protocols exist, e.g,

- OFMC, ProVerif, SATMC, ...
- Inductive protocol analysis, e.g., using Isabelle/HOL (also in combination with OFMC)
- ...

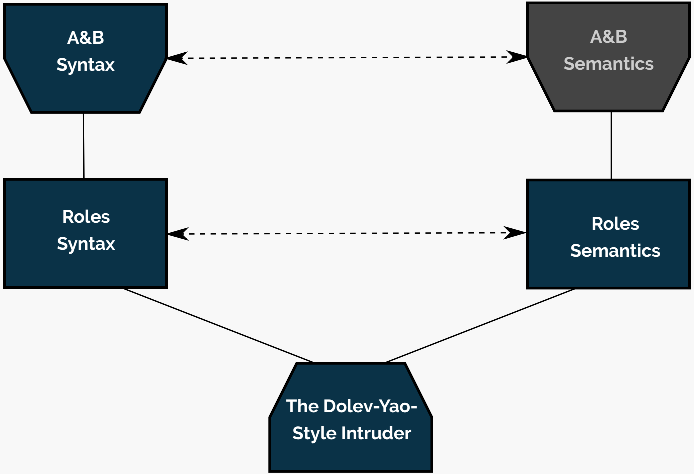
OFMC: A Symbolic Model-Checker for Security Protocols

```
Protocol: NSPK

Types:
    Agent A,B;
    Number NA,NB;
    Function pk;

Knowledge:
    A: A, B, pk, inv(pk(A));
    B: B, pk, inv(pk(B));

Actions:
    A -> B: {NA,A}(pk(B))
    B -> A: {NA,NB}(pk(A))
    A -> B: {NB}(pk(B))

Goals:
    A *->* B: NA
    B *->* A: NB
```

```
> ofmc nspk.AnB

INPUT:
   nspk.AnB
SUMMARY:
  ATTACK_FOUND
GOAL:
  secrecy
BACKEND:
  Open-Source Fixedpoint Model-Checker version 2014
STATISTICS:
  ...

ATTACK TRACE:
(x502,1) -> i: {NA(1),x502}_(pk(i))
i -> (x25,1): {NA(1),x502}_(pk(x25))
(x25,1) -> i: {NA(1),NB(2)}_(pk(x502))
i -> (x502,1): {NA(1),NB(2)}_(pk(x502))
(x502,1) -> i: {NB(2)}_(pk(i))
i -> (i,17): NB(2)
i -> (i,17): NB(2)
```

Outline

Overview



A&B Syntax
A&B Semantics
Roles Syntax
Roles Semantics
The Dolev-Yao-Style Intruder

What We Learned Today

- Security protocols are difficult to design
- Formal methods help to define protocols and security goals precisely
- Formal methods helps

Note:

- Today's lecture was a "deep dive", not all details are equally relevant for the exam
- In the exam you should be able to
  - work with the (graphical and textual) AnB notation (slides 207 - 209)
  - work with the core of the Dolev-Yao Intruder (slides 221 and 228)
  - be able to prove simple properties (slides 229, 230)
  - solve problems similar to the ones discussed in the homework (and mock exam)
- We will practice this in the first lab using OFMC

Outlook

- **Focus of the lecture until today:**

  security systems
  and
  how to build them correctly

- **Focus of the next weeks:**

  how to build secure systems
  or
  how to build systems securely

- Or in other words: let's address the following problem

UK Code is Least Secure, Report Finds

https://www.infosecurity-magazine.com/news/uk-code-is-least-secure-report/

10 MAR 2017 NEWS

UK Code is Least Secure, Report Finds

Steve Evans

Freelance journalist, copywriter and editorial consultant

Follow @Evans_Steve

The UK ranks bottom of the league for the security of its code, according to a new report.

The research from software analytics firm Cast

Thank you for your attention!
Any questions or remarks?

Contact:

Dr. Achim D. Brucker
Department of Computer Science
University of Sheffield
Regent Court
211 Portobello St.
Sheffield S1 4DP, UK

a.brucker@sheffield.ac.uk
@adbrucker
https://de.linkedin.com/in/adbrucker/
https://www.brucker.ch/
https://logicalhacking.com/blog/

Bibliography I

Achim D. Brucker and Sebastian A. Mödersheim.

Integrating automated and interactive protocol verification.

In Pierpaolo Degano and Joshua Guttman, editors, *Workshop on Formal Aspects in Security and Trust (FAST 2009)*, number 5983 in Lecture Notes in Computer Science, pages 248–262. Springer-Verlag, 2009.

An extended version of this paper is available as IBM Research Technical Report, RZ3750.

D. Dolev and A. C. Yao.

On the security of public key protocols.

Symposium on Foundations of Computer Science, 0:350–357, 1981.

Michael Huth and Mark Ryan.

Logic in Computer Science: Modelling and Reasoning About Systems.

Cambridge University Press, New York, NY, USA, 2004.

Document Classification and License Information