Protocol Verification Techniques - Theorem Provers

Design and Verification of Security Protocols and Security Ceremonies

Programa de Pós-Graduacão em Ciências da Computacão
Dr. Jean Everson Martina

March-June 2018

Programa de Pós-Graduação em Ciência da Computação UFSC

UNIVERSIDADE FEDERAL DE SANTA CATARINA

Attention!

Attention!

This topic will be divided into two lectures. One will deal with automatic theorem provers using FOL and the second will deal with theorem provers using HOL

Security Protocol Analysis using Theorem Proving

A Small Review on Logics

Before we get our hand on theorem proving we need to talk a bit about logics:

A Small Review on Logics

Before we get our hand on theorem proving we need to talk a bit about logics:

- Propositional Logic
- First-Order Logic (FOL)

Propositional Logic

- It is basic logical system;

Propositional Logic

- It is basic logical system;
- It studies its arguments and their structure;

Propositional Logic

- It is basic logical system;
- It studies its arguments and their structure;
  - An argument is a declarative sentence in natural language like English;

Propositional Logic

- It is basic logical system;
- It studies its arguments and their structure;
  - An argument is a declarative sentence in natural language like English;
  - Example: *"The bus is late"*

Propositional Logic

- It is basic logical system;
- It studies its arguments and their structure;
  - An argument is a declarative sentence in natural language like English;
  - Example: *"The bus is late"*
- It was discovered by Aristóteles in ancient Greece;

Propositional Logic

- It is basic logical system;
- It studies its arguments and their structure;
  - An argument is a declarative sentence in natural language like English;
  - Example: *"The bus is late"*
- It was discovered by Aristóteles in ancient Greece;
- Each sentence receive a truth value being T (True) or F (False).

Propositional Logic

- There are well defined rules to extract meaning from complex arguments:

Propositional Logic

- There are well defined rules to extract meaning from complex arguments:
  - Modus Ponens;

Propositional Logic

- There are well defined rules to extract meaning from complex arguments:
  - Modus Ponens;
  - Modus Tolens;

Propositional Logic

- There are well defined rules to extract meaning from complex arguments:
  - Modus Ponens;
  - Modus Tolens;
  - Negation of the implication, etc;

Propositional Logic

- There are well defined rules to extract meaning from complex arguments:
  - Modus Ponens;
  - Modus Tolens;
  - Negation of the implication, etc;
- It is a classical logic that is easy to understand.

An Example Formula in Propositional Logic

It is Raining. : P

An Example Formula in Propositional Logic

It is Raining.	:	P
Jane carries her umbrella.	:	Q

An Example Formula in Propositional Logic

It is Raining.	:	P
Jane carries her umbrella.	:	Q
Jane gets wet.	:	R

An Example Formula in Propositional Logic

It is Raining.	:	P
Jane carries her umbrella.	:	Q
Jane gets wet.	:	R

$$(P \wedge \neg Q) \rightarrow R, \neg R, P \vdash Q$$

An Example Formula in Propositional Logic

It is Raining.	:	P
Jane carries her umbrella.	:	Q
Jane gets wet.	:	R

$$(P \wedge \neg Q) \rightarrow R, \neg R, P \vdash Q$$

- **Symbols**: $\wedge$ – "and"; $\vee$ – "or"; $\not-$ "not"; $\rightarrow$ – "implies"; $\leftrightarrow$ – "equivalent"; $\vdash$ – "proves"; and $\nvdash$ – "do not prove".

First-Order Logic (FOL)

- It is also known as *Predicate's Logic* ou *Quantificational Logic*;

First-Order Logic (FOL)

- It is also known as *Predicate's Logic* ou *Quantificational Logic*;
- Extends the expressiveness of propositional logic;

First-Order Logic (FOL)

- It is also known as *Predicate's Logic* ou *Quantificational Logic*;
- Extends the expressiveness of propositional logic;
  - It is hard to express sentence like *"Somethins **is** or **has** ..."* in propositional logic;

First-Order Logic (FOL)

- It is also known as *Predicate's Logic* ou *Quantificational Logic*;
- Extends the expressiveness of propositional logic;
  - It is hard to express sentence like *"Somethins **is** or **has** ..."* in propositional logic;
- The main difference of First-Order Logic is the existence of quantifiers:

First-Order Logic (FOL)

- It is also known as *Predicate's Logic* ou *Quantificational Logic*;
- Extends the expressiveness of propositional logic;
  - It is hard to express sentence like *"Somethins **is** or **has** ..."* in propositional logic;
- The main difference of First-Order Logic is the existence of quantifiers:
  - $\exists$ (there exists), and $\forall$ (for all);

First-Order Logic (FOL)

- It is also known as *Predicate's Logic* ou *Quantificational Logic*;
- Extends the expressiveness of propositional logic;
  - It is hard to express sentence like *"Somethins* ***is*** *or* ***has*** *..."* in propositional logic;
- The main difference of First-Order Logic is the existence of quantifiers:
  - $\exists$ (there exists), and $\forall$ (for all);
- Other concepts are: predicates, variables, functions and constants;

First-Order Logic (FOL)

- It is also known as *Predicate's Logic* ou *Quantificational Logic*;
- Extends the expressiveness of propositional logic;
  - It is hard to express sentence like *"Somethins **is** or **has** ..."* in propositional logic;
- The main difference of First-Order Logic is the existence of quantifiers:
  - $\exists$ (there exists), and $\forall$ (for all);
- Other concepts are: predicates, variables, functions and constants;
- This logic is expressive enough to verify security protocols.

An Example Formula in FOL

$S(x, y)$: x is Son of y (S is a predicate)

An Example Formula in FOL

$$
\begin{aligned}
S(x, y) &\quad : \quad x \text{ is Son of } y \ (S \text{ is a predicate}) \\
B(x, y) &\quad : \quad x \text{ is Brother of } y \ (B \text{ is another predicate})
\end{aligned}
$$

An Example Formula in FOL

$$
\begin{aligned}
S(x,y) &\quad : \quad x \text{ is Son of } y \ (S \text{ is a predicate}) \\
B(x,y) &\quad : \quad x \text{ is Brother of } y \ (B \text{ is another predicate}) \\
f(x) &\quad : \quad \text{returns the father of } x \ (f \text{ is a function})
\end{aligned}
$$

An Example Formula in FOL

$$
\begin{array}{lcl}
S(x, y) & : & x \text{ is Son of } y \text{ (}S\text{ is a predicate)} \\
B(x, y) & : & x \text{ is Brother of } y \text{ (}B\text{ is another predicate)} \\
f(x) & : & \text{returns the father of } x \text{ (}f\text{ is a function)}
\end{array}
$$

$$\forall x[S(x, f(m) \rightarrow B(x, m))]$$

(m is a constant, and x is a variable)

An Example Formula in FOL

$$\begin{array}{lcl} S(x,y) & : & x \text{ is Son of } y \text{ (}S\text{ is a predicate)} \\ B(x,y) & : & x \text{ is Brother of } y \text{ (}B\text{ is another predicate)} \\ f(x) & : & \text{returns the father of } x \text{ (}f\text{ is a function)} \end{array}$$

$$\forall x[S(x, f(m) \rightarrow B(x, m))]$$

(m is a constant, and x is a variable)

Our protocols will be modelled this way!

Defining Predicates for Protocol Verification

- $E(x)$: x is an entity (agent) in the protocols;

Defining Predicates for Protocol Verification

- $E(x)$: x is an entity (agent) in the protocols;
- $Stores(x, y)$: x is stored by entity y;

Defining Predicates for Protocol Verification

- $E(x)$: x is an entity (agent) in the protocols;
- $Stores(x, y)$: x is stored by entity y;
- $Knows(x, y)$: x is known by entity y;

Defining Predicates for Protocol Verification

- $E(x)$: x is an entity (agent) in the protocols;
- $Stores(x, y)$: x is stored by entity y;
- $Knows(x, y)$: x is known by entity y;
- $M(x)$: a message x is sent in the protocol.

Defining Functions for Protocol Verification

- Grouping message components:
  - $pair(x, y)$; $triple(x, y, z)$;

Defining Functions for Protocol Verification

- Grouping message components:
  - $pair(x, y)$; $triple(x, y, z)$;
- Message exchange:
  - $sent(x, y, z)$: agent x sends to agent y message z;

Defining Functions for Protocol Verification

- Grouping message components:
  - $pair(x, y)$; $triple(x, y, z)$;
- Message exchange:
  - $sent(x, y, z)$: agent x sends to agent y message z;
- Key related functions:
  - $krkey(x, y)$: private key x belongs to agent y;

Defining Functions for Protocol Verification

- Grouping message components:
  - $pair(x, y)$; $triple(x, y, z)$;
- Message exchange:
  - $sent(x, y, z)$: agent x sends to agent y message z;
- Key related functions:
  - $krkey(x, y)$: private key x belongs to agent y;
  - $kukey(x, y)$: x belongs to agent y; and

Defining Functions for Protocol Verification

- Grouping message components:
  - $pair(x, y)$; $triple(x, y, z)$;
- Message exchange:
  - $sent(x, y, z)$: agent x sends to agent y message z;
- Key related functions:
  - $krkey(x, y)$: private key x belongs to agent y;
  - $kukey(x, y)$: x belongs to agent y; and
  - $kp(x, y)$: private key x and public key y make a key pair.

Defining Functions for Protocol Verification

- *Nonce* functions:
  - $nonce(x, y)$: *nonce* x is generated by entity y;

Defining Functions for Protocol Verification

- *Nonce* functions:
  - $nonce(x, y)$: *nonce* x is generated by entity y;
- Cryptographic primitives:
  - $encr(x, y)$: x is encrypted using key y; and

Defining Functions for Protocol Verification

- *Nonce* functions:
  - $nonce(x, y)$: *nonce* x is generated by entity y;
- Cryptographic primitives:
  - $encr(x, y)$: x is encrypted using key y; and
  - $sign(x, y)$: x is signed using key y.

Defining Constants for Protocol Verification

- Agents participating in the protocols:
  - a (Alice); b (Bob); c (Charlie).

Defining Constants for Protocol Verification

- Agents participating in the protocols:
  - *a* (Alice); *b* (Bob); *c* (Charlie).
- Private keys and public keys:
  - *kra*; *kua*: private key and public key belonging to Alice;
  - *krb*; *kub*: private key and public key belonging to Bob;
  - *krc*; *kuc*: private key and public key belonging to Charlie;

Defining Constants for Protocol Verification

- Agents participating in the protocols:
  - *a* (Alice); *b* (Bob); *c* (Charlie).
- Private keys and public keys:
  - *kra*; *kua*: private key and public key belonging to Alice;
  - *krb*; *kub*: private key and public key belonging to Bob;
  - *krc*; *kuc*: private key and public key belonging to Charlie;
- *Nonces*:
  - *na*; *nb*; *nc*.

Needham-Schroeder Public Key Protocol

1. A $\rightarrow$ B: $\{|N_a, A|\}_{K_b}$
2. B $\rightarrow$ A: $\{|N_a, N_b|\}_{K_a}$
3. A $\rightarrow$ B: $\{|N_b|\}_{K_b}$

NSPKP Goals

- The goal of the protocol is to establish mutual authentication between two parties A and B in the presence of adversary;
- A and B obtain a secret shared key though direct communication using public key cryptography;
- This adversary can intercept messages, delay messages, read and copy messages and generate messages;
- This adversary can not learn the privates keys of principals.

Defining Initial Knowledge

- A first step is to define the initial knowledge of each agent;

Defining Initial Knowledge

- A first step is to define the initial knowledge of each agent;
- For example Alice's is:

Defining Initial Knowledge

- A first step is to define the initial knowledge of each agent;
- For example Alice's is:

Example

$E(a)$
$Knows(kp(krkey(kra, a), kukey(kua, a)), a)$
$Knows(kukey(kub, b), a)$
$Knows(kukey(kuc, c), a)$
$Knows(nonce(na, a), a)$

Defining Initial Knowledge

- A first step is to define the initial knowledge of each agent;
- For example Alice's is:

Example

$E(a)$
$Knows(kp(krkey(kra, a), kukey(kua, a)), a)$
$Knows(kukey(kub, b), a)$
$Knows(kukey(kuc, c), a)$
$Knows(nonce(na, a), a)$

We do the same thing to Bob and Charlie changing the constants only.

Describing NSPKP

- We model the message exchange;

Describing NSPKP

- We model the message exchange;
- The first massage is modelled to:

Describing NSPKP

- We model the message exchange;
- The first massage is modelled to:

Example

$$
\begin{aligned}
&Knows(kukey(kua, a), a) \wedge \\
&Knows(kp(krkey(kra, a), kukey(kua, a)), a) \wedge \\
&Knows(kukey(kub, b), a) \wedge \\
&Knows(nonce(na, a), a) \\
&\rightarrow \\
&M(sent(a, b, encr(pair(na, a), kub)) \wedge \\
&Stores(pair(na, b)a)
\end{aligned}
$$

Describing NSPKP

- The second massage is modelled to:

Describing NSPKP

- The second massage is modelled to:

Example

$$\forall x[\mathit{Knows}(\mathit{kukey}(\mathit{kub}, b), b) \wedge$$
$$\mathit{Knows}(\mathit{kp}(\mathit{krkey}(\mathit{krb}, b), \mathit{kukey}(\mathit{kub}, b)), b) \wedge$$
$$\mathit{Knows}(\mathit{kukey}(\mathit{kua}, a), b) \wedge$$
$$\mathit{Knows}(\mathit{nonce}(\mathit{nb}, b), b) \wedge$$
$$M(\mathit{sent}(x, b, \mathit{encr}(\mathit{pair}(\mathit{na}, a), \mathit{kub}))$$
$$\rightarrow$$
$$M(\mathit{sent}(b, a, \mathit{encr}(\mathit{pair}(\mathit{na}, \mathit{nb}), \mathit{kua})) \wedge$$
$$\mathit{Stores}(\mathit{pair}(\mathit{nb}, a), b)]$$

Describing NSPKP

- The third massage is modelled to:

Describing NSPKP

- The third massage is modelled to:

Example

$\forall x[$
$Stores(pair(na, b), a) \wedge$
$M(sent(x, a, encr(pair(na, nb), kua))$
$\rightarrow$
$M(sent(a, b, encr(nb), kub))]$

Logical Model for the Attacker

- The attacker is a Dolev-Yao one;

Logical Model for the Attacker

- The attacker is a Dolev-Yao one;
- The attacker model add some logical elements:

Logical Model for the Attacker

- The attacker is a Dolev-Yao one;
- The attacker model add some logical elements:
  - The constant c which represents the attacker himself;

Logical Model for the Attacker

- The attacker is a Dolev-Yao one;
- The attacker model add some logical elements:
  - The constant *c* which represents the attacker himself;
  - The attacker data when impersonating another valid user int he protocol;

Logical Model for the Attacker

- The attacker is a Dolev-Yao one;
- The attacker model add some logical elements:
  - The constant c which represents the attacker himself;
  - The attacker data when impersonating another valid user int he protocol;
  - The predicate $Im(x)$ which holds the knowledge acquired by the attacker by the manipulation of the exchange messages;

Logical Model for the Attacker

- The attacker is a Dolev-Yao one;
- The attacker model add some logical elements:
  - The constant c which represents the attacker himself;
  - The attacker data when impersonating another valid user int he protocol;
  - The predicate $Im(x)$ which holds the knowledge acquired by the attacker by the manipulation of the exchange messages;
  - This predicate work is a similar way to $M(x)$ predicate.

Attacker's Initial Knowledge

1. The attacker is an entity of the protocols and has its own lawful data set:
   - $E(c)$

Attacker's Initial Knowledge

1. The attacker is an entity of the protocols and has its own lawful data set:
   - $E(c)$
   - He also has his key pair and nonce which are omitted here;

Attacker's Initial Knowledge

1. The attacker is an entity of the protocols and has its own lawful data set:
   - $E(c)$
   - He also has his key pair and nonce which are omitted here;
2. He knows the public data of other agents:
   - $Knows(kukey(kua, a), c)$
   - $Knows(kukey(kub, b), c)$

Attacker's Initial Knowledge

1. The attacker is an entity of the protocols and has its own lawful data set:
   - $E(c)$
   - He also has his key pair and nonce which are omitted here;
2. He knows the public data of other agents:
   - $Knows(kukey(kua, a), c)$
   - $Knows(kukey(kub, b), c)$
3. He can record all the messages:
   - $\forall x, y, w[M(sent(x, y, w)) \rightarrow Im(w)]$

Attacker's Messages Manipulation Capabilities

1. He can decompose the message into smaller pieces:
   - $\forall u, v[Im(pair(u, v)) \rightarrow Im(u) \wedge Im(v)]$
   - $\forall u, v, w[Im(triple(u, v, w)) \rightarrow Im(u) \wedge Im(v) \wedge Im(w)]$

Attacker's Messages Manipulation Capabilities

1. He can decompose the message into smaller pieces:
   - $\forall u, v[Im(pair(u, v)) \rightarrow Im(u) \wedge Im(v)]$
   - $\forall u, v, w[Im(triple(u, v, w)) \rightarrow Im(u) \wedge Im(v) \wedge Im(w)]$
2. He can fabricate message form the knowledge he acquired:
   - $\forall u, v[Im(u) \wedge Im(v) \rightarrow Im(pair(u, v))]$
   - $\forall u, v, w[Im(u) \wedge Im(v) \wedge Im(w) \rightarrow Im(triple(u, v, w))]$

Attacker's Messages Manipulation Capabilities

1. He can decompose the message into smaller pieces:
   - $\forall u, v[Im(pair(u, v)) \rightarrow Im(u) \wedge Im(v)]$
   - $\forall u, v, w[Im(triple(u, v, w)) \rightarrow Im(u) \wedge Im(v) \wedge Im(w)]$
2. He can fabricate message form the knowledge he acquired:
   - $\forall u, v[Im(u) \wedge Im(v) \rightarrow Im(pair(u, v))]$
   - $\forall u, v, w[Im(u) \wedge Im(v) \wedge Im(w) \rightarrow Im(triple(u, v, w))]$
3. He can send fake messages:
   - $\forall u, x, y[Im(u) \wedge E(x) \wedge E(y) \rightarrow M(sent(x, y, u))]$

Attacker's Cryptographic Capabilities

1. Anything can potentially be a key:
   - $\forall u, v[Im(u) \wedge E(v) \rightarrow Knows(krkey(u, v), c)]$
   - $\forall u, v[Im(u) \wedge E(v) \rightarrow Knows(kukey(u, v), c)]$

Attacker's Cryptographic Capabilities

1. Anything can potentially be a key:
   - $\forall u, v[Im(u) \land E(v) \rightarrow Knows(krkey(u, v), c)]$
   - $\forall u, v[Im(u) \land E(v) \rightarrow Knows(kukey(u, v), c)]$
2. Anything can potentially be a *nonce*:
   - $\forall u, v[Im(u) \land E(v) \rightarrow Knows(nonce(u, v), c)]$

Attacker's Cryptographic Capabilities

1. Anything can potentially be a key:
   - $\forall u, v[Im(u) \wedge E(v) \rightarrow Knows(krkey(u, v), c)]$
   - $\forall u, v[Im(u) \wedge E(v) \rightarrow Knows(kukey(u, v), c)]$
2. Anything can potentially be a *nonce*:
   - $\forall u, v[Im(u) \wedge E(v) \rightarrow Knows(nonce(u, v), c)]$
3. He can encrypt and sing with known keys:
   - $\forall u, v, x[Im(u) \wedge Knows(kukey(v, x), c) \wedge E(x) \rightarrow Im(encr(u, v))]$
   - $\forall u, v, x[Im(u) \wedge Knows(krkey(v, x), c) \wedge E(x) \rightarrow Im(sign(u, v))]$

More Attacker's Cryptographic Capabilities

1. Decrypt messages with known keys:
   - $\forall u, v, w, x[Im(encr(u, v)) \wedge Knows(kp(krkey(w, x), kukey(v, x)), c) \wedge E(x) \rightarrow Im(u))]$

More Attacker's Cryptographic Capabilities

1. Decrypt messages with known keys:
   - $\forall u, v, w, x[Im(encr(u, v)) \wedge Knows(kp(krkey(w, x), kukey(v, x)), c) \wedge E(x) \rightarrow Im(u))]$
2. Decrypt messages with known *nonces*:
   - $\forall u, v, w[Im(encr(u, v)) \wedge Knows(nonce(v, w), c) \wedge E(w) \rightarrow Im(u))]$

More Attacker's Cryptographic Capabilities

1. Decrypt messages with known keys:
   - $\forall u, v, w, x[Im(encr(u, v)) \wedge Knows(kp(krkey(w, x), kukey(v, x)), c) \wedge E(x) \rightarrow Im(u))]$
2. Decrypt messages with known *nonces*:
   - $\forall u, v, w[Im(encr(u, v)) \wedge Knows(nonce(v, w), c) \wedge E(w) \rightarrow Im(u))]$
3. Learn signed messages:
   - $\forall u, v[Im(sign(u, v)) \rightarrow Im(u)]$

All that can be Mechanised

- The proofs can be made manually with pen and paper;

All that can be Mechanised

- The proofs can be made manually with pen and paper;
- However it is more convenient to use theorem prover to do the hard work;

All that can be Mechanised

- The proofs can be made manually with pen and paper;
- However it is more convenient to use theorem prover to do the hard work;
- Any FOL capable theorem prober can to the job;

All that can be Mechanised

- The proofs can be made manually with pen and paper;
- However it is more convenient to use theorem prover to do the hard work;
- Any FOL capable theorem prober can to the job;
- We like a lot SPASS:
  - Deals with FOL;
  - Makes proofs by contradiction;

All that can be Mechanised

- The proofs can be made manually with pen and paper;
- However it is more convenient to use theorem prover to do the hard work;
- Any FOL capable theorem prober can to the job;
- We like a lot SPASS:
  - Deals with FOL;
  - Makes proofs by contradiction;
  - General use.

Testing the Logical Model

- We need to write conjectures;

Testing the Logical Model

- We need to write conjectures;
- the theorem prover will tell us if it is true or not:

Testing the Logical Model

- We need to write conjectures;
- the theorem prover will tell us if it is true or not:
  - Conjectures are statements that we don know if they are true or not from the axioms;

Testing the Logical Model

- We need to write conjectures;
- the theorem prover will tell us if it is true or not:
    - Conjectures are statements that we don know if they are true or not from the axioms;
    - We can then extract knowledge from the test of conjectures;

Testing the Logical Model

- We need to write conjectures;
- the theorem prover will tell us if it is true or not:
    - Conjectures are statements that we don know if they are true or not from the axioms;
    - We can then extract knowledge from the test of conjectures;
- Lowe's attack can be easily reproduced with this setting we just saw;

Problems with Protocol Verification with FOL

- By definition FOL is non decidable;

Problems with Protocol Verification with FOL

- By definition FOL is non decidable;
- We need to use a subset that are the Monadic Horn Clauses;

Problems with Protocol Verification with FOL

- By definition FOL is non decidable;
- We need to use a subset that are the Monadic Horn Clauses;
- The proofs are as good as the conjectures created;

Problems with Protocol Verification with FOL

- By definition FOL is non decidable;
- We need to use a subset that are the Monadic Horn Clauses;
- The proofs are as good as the conjectures created;
- If you are not clever it will not work.

Discussion

- What else can you foresee modelled using this strategy?

Discussion

- What else can you foresee modelled using this strategy?
- Can this be extended?

Discussion

- What else can you foresee modelled using this strategy?
- Can this be extended?
- What this strategy can not do?

Questions????

UNIVERSIDADE FEDERAL
DE SANTA CATARINA

UNIVERSIDADE FEDERAL
DE SANTA CATARINA