Security protocol analysis using the Tamarin Prover

Jannik Dreier, Lucca Hirshi

Based on slides by Cas Cremers
Overview & Structure

- **Overview**
  1) Introduction
  2) Foundations: Security Protocols Modeling
     - What are they? How to mathematically model them?
     - Modeling protocols in Tamarin
  3) Foundations: Properties
     - Typical properties
     - Modeling properties in Tamarin
  4) Tamarin Demo
  5) Where do I go from here?
     - Next steps
Overview & Structure

- **Mode of operation**
  - No need to use tools during the talk, exercises are for afterwards!
  - There will be time for questions during the talk and at the end
  - Please raise your hand in zoom or ask questions offline in the discord channel #security-protocols
  - We’ll have a short break in the middle
Security Protocols

- **Distributed programs using** **cryptographic primitives**
  *e.g., digital signature, encryption*

- **Protocol participants exchange messages over an insecure network**
  *e.g., internet*

- **Achieve security goals**
  *e.g., authentication, confidentiality*

- **Real-world examples:**
  - **TLS**
  - **EMV**
  - **5G AKA** ...
Security Protocols

- Distributed programs using cryptographic primitives e.g., digital signature, encryption

- Protocol participants exchange messages over an insecure network e.g., internet

- Achieve security goals e.g., authentication, confidentiality

- Real-world examples:
  - TLS
  - EMV
  - 5G AKA
Problem

• **How do we know if a protocol is secure?**
  - Traditionally: Smart people stare at it
Problem

• **How do we know if a protocol is secure?**
  - Traditionally: Smart people stare at it

• **More structured approach:**
  - Specify threat model & intended property
  - Stare at the protocol, try to find attack
  - Write the proof
Problem

• **How do we know if a protocol is secure?**
  – Traditionally: Smart people stare at it

• **More structured approach:**
  – Specify threat model & intended property
  – Stare at the protocol, try to find attack
  – Write the proof

• **Can formal methods help?**
  – Model checking, verification
Formal methods?

- **Goal:** reason about programs, protocols, hardware...

- **Approach:**
  1) **Build precise model** (e.g., using automata, rewrite systems, transition systems, ...)
  2) **Specify desired properties** (using logic) \( \forall \text{tr. } P(\text{tr}) \)
  3) **Prove** that properties hold in **all possible executions** (e.g., using model checking, constraint solving, ...)

- Allows to **find and correct** bugs/flaws in specifications and implementations
In Tamarin: Trace properties

- For now: trace properties (but more later!):
  - \( \forall \ tr \in \text{traces(System)} . \ P(tr) \)
The Tamarin Prover

• Symbolic analysis tool for systems in presence of a Dolev-Yao style network adversary

• Some highlights:
  – TLS 1.3
  – 5G-AKA
  – EMV (Chip and pin)
Simon Meier
Benedikt Schmidt
Cas Cremers
David Basin
Resources & documentation

- Sources on github:
  https://github.com/tamarin-prover/

- 100+ page manual:
  https://tamarin-prover.github.io/manual/

- Plenty of examples/case studies

- Algorithm details in theses, papers, see
  https://tamarin-prover.github.io/
Symbolic security analysis

- Idea: make transition system
  - with *protocol participants*
  - with *adversary* controlling network

- Encode security property
  - **Secrecy**
    There is no trace in which the *adversary* learns $k$
  - **Authentication**
    In all traces, if an initiator completes, there exists a responder with...

- And check!

- Unfortunately, this turns out to be undecidable :-(

Tamarin workflow

Property P

System S
Tamarin workflow

Property P

System S

Tamarin prover

Tamarin’s algorithm
Tamarin workflow

Property P

System S

Tamarin prover

Tamarin’s algorithm

Solution exists: ATTACK
Tamarin workflow

- System S
- Property P

Tamarin prover

Tamarin’s algorithm

Solution exists: ATTACK

No solution exists: PROOF
Tamarin workflow

- Property P
- System S

Tamarin prover

Tamarin’s algorithm

Solution exists: ATTACK

No solution exists: PROOF

Run out of time or memory
Tamarin workflow

Tamarin prover

Property P

System S

Provide hints for the prover (e.g. invariants)

Tamarin’s algorithm

Interactive mode
Inspect partial proof

Solution exists: ATTACK

No solution exists: PROOF

Run out of time or memory
• What we saw:
  – We all use security protocols on a daily basis
  – What formal methods are and why they can help
  – What Tamarin was built for
  – Who is behind it
  – The main workflow of Tamarin

Next up: Tamarin’s foundations
Outline

Example: Securing remote keyless
Example: Securing remote keyless
Example: Securing remote keyless

Car #41

generate random $X$

$X, Y$

Car #41

generate random $Y$
Example: Securing remote keyless

\[ senc(<X,Y>, k) \]

- Car
  - Key \( k \)
  - generate random \( X \)
- Car
  - Key \( k \)
  - generate random \( Y \)
Protocol Specifications

- Often written like this *(message sequence chart)*:

\[
\text{Initiator (car)} \quad k \quad \text{Responder (key)} \quad k
\]

\[
X \quad \text{senc}(<X,Y>,k) \quad \text{‘OK’, doors open}
\]
Protocol Specifications

- Often written like this (message sequence chart): 5G AKA
Protocol Specifications

- Often written like this (message sequence chart):

![Message Sequence Chart]

**TLS 1.3**

- How to mathematically model security protocols?
Symbolic Model

• Messages + cryptographic primitives $\rightarrow$ term algebra
  – Cryptography is idealized
    
    e.g., impossible to decrypt without the full key

  – Primitives modelled as function symbols
    
    e.g., $\text{senc}(:, :)$, $\text{sdec}(:, :)$

  – Known properties about them modelled as algebraic relations in an equational theory ($=_{E}$)
    
    e.g., $\text{sdec}(\text{senc}(m,k), k) =_{E} m$
Symbolic Model

• Messages + cryptographic primitives → term algebra
  - Crypto is idealized:
    e.g., impossible to decrypt without the full key
    e.g., impossible to forge a signature without the full key

  - Primitives modelled as function symbols
    e.g., senc(\cdot,\cdot), sdec(\cdot,\cdot)
    e.g., signature → sign(\cdot,\cdot), verify(\cdot,\cdot), sk(\cdot)

  - Known properties about them modelled as algebraic relations in an equational theory =_{E}
    e.g., sdec(senc(m,k),k) =_{E} m
    e.g., verify(sign(m,sk(vk)),vk,m) =_{E} ‘legit’
Symbolic Model

- Protocol agents modelled as programs in a formal language that can output and input messages
  
  *e.g.*, Tamarin offers such a language, other example: applied π-calculus

- Attacker 🧟 = network

  He can:
  - **Eavesdrop**: learns all protocol outputs
  - **Deduce**: derive new terms using primitives
  - **Inject**: choose all protocol inputs

- **Benefits**: high-level of automation!
  
  *e.g.*, with techniques such as rewriting theory, resolution, model-checking, …
Modeling Workflow

Protocol’s specification $\xrightarrow{X} \text{Protocol’s model} 

\[ P_{\text{in}} = \text{in}(x). \]
\[ \text{new } Y. \]
\[ \text{out(enc((x, Y), k))} \]
\[ P_{\text{out}} = \ldots \]

Undecidable

Reachability in a transition system

Security goal $\xrightarrow{\text{Unreachability of bad states}}$

\[ \text{e.g. } \text{cannot steal} \]
\[ \text{car} \]
\[ \text{e.g. States(}\text{poster knows } k) \]

Your job: modeling

Tamarin’s job

$\infty$ choices

$\infty$ # sessions
Intermission

Time for some questions
Modeling in Tamarin

- **Multiset rewriting**
- **Basic ingredients:**
  - **Terms:** messages (term algebra)
  - **Facts:** available actions and data (think “sticky notes on the fridge”)
    - Special facts: Out(t) (for output), In(t) (for input), K(t) (for adversary knowledge)
- **State** of system is a multiset of facts
  - **Initial state** is the empty multiset
  - **Rules** specify the transition rules (“moves”)
- **Rules** are of the form: \( l \rightarrow\left[ a \right] \rightarrow r \) where:
  - \( l \) is a multiset of consumed facts
  - \( r \) is a multiset of produced facts
  - \( a \) is a multiset of action facts acting as labels (think automata)
Example 1: basic

- **Rules**
  - rule 1: \([ ] \) \[ \text{Init()} \] \[ A('5') \]
  - rule 2: \[ A(x) \] \[ \text{Step}(x) \] \[ B(x) \] \(// x: \text{free variable}\)

- **Execution example**
  - [ ]
    - [ Init() ]\[ A('5') \]
    - [ Init() ]\[ A('5'), A('5') \]
    - [ Step('5') ]\[ A('5'), B('5') \]

- **Corresponding trace**
  - [ Init(), Init(), Step('5') ]
Example 2: fresh & public

• **Rules**
  - rule 1: \([ \text{Fr}(\neg k) ] \rightarrow [ \text{GenKey}(\$A) ] \rightarrow [ \text{Key}(\$A, \neg k) ]\)

• **Execution example**
  - \([ ]\)
    - \([ \text{GenKey}(\text{‘alex’}) ] \rightarrow [ \text{Key}(\text{‘alex’, k.1}) ]\)
    - \([ \text{GenKey}(\text{‘alex’}) ] \rightarrow [ \text{Key}(\text{‘alex’, k.1}), \text{Key}(\text{‘alex’, k.2}) ]\)
    - \([ \text{GenKey}(\text{‘blake’}) ] \rightarrow [ \text{Key}(\text{‘alex’, k.1}), \text{Key}(\text{‘alex’, k.2}), \text{Key}(\text{‘blake’, k.3}) ]\)

• **Corresponding trace**
  - \([ \text{GenKey}(\text{‘alex’}), \text{GenKey}(\text{‘alex’}), \text{GenKey}(\text{‘blake’}) ]\)
Example 3: persistent facts

• Rules
  - rule1: [ ] → [ Init() ] → [ !C('ok'), D('1') ]
  - rule2: [ !C(x), D(y) ] → [ Step(x,y) ] → [ D(h(y)) ]

• Execution example
  • [ ]
    • [ Init() ] → [ !C('ok'), D('1') ]
    • [ Step('ok','1') ] → [ !C('ok'), D(h('1')) ]
    • [ Step('ok',h('1')) ] → [ !C('ok'), D(h(h('1'))) ]

• Corresponding trace
  • [ Init(), Step('ok', '1'), Step('ok', h('1')) ]
Example 4: Secure Remote Keyless

\[ \text{Initiator (car)} \quad A, k \]

\[ \text{Responder (key)} \quad B, k \]

senc(<X,Y>,k)
Example 4: Secure Remote Keyless

A, k

\[ senc(<X, varY>, k) \]

\[ 'OK' \]
Example 4: Secure Remote Keyless Initiator (car)

Key Generation:

```
rule genKey: [ Fr(\~k) ] \rightarrow [ !Ltk( $A, \$B, \~k ) ]
```

Diagram:
- Initiator (car)
- \( A, k \)
- \( X \)
- \( senc(<X,\text{var}Y>, k) \)
- ‘OK’
Example 4: Secure Remote Keyless

- **Key Generation:**
  
  rule genKey: \[ Fr(\sim k) \] \[ \rightarrow [ !Lt(k( A, B, \sim k) \] \]

- **First output:**
  
  rule Init_1: \[ Fr( \sim X ), !Lt(k( A, B, \sim k) \] \[ \rightarrow [ Out(\sim X), Init_1( \sim X, A, B, \sim k) \] \]
Example 4: Secure Remote Keyless

- **Key Generation:**
  
  rule genKey: \([ \text{Fr}(\sim k) ] \rightarrow [ !\text{Ltk}( A, B, \sim k ) ]\)

- **First output:**
  
  rule Init_1: \([ \text{Fr}( \sim X ), !\text{Ltk}( A, B, \sim k ) ] \rightarrow [ \text{Out}(\sim X), \text{Init}_1( \sim X, A, B, \sim k ) ]\)

- **First input and second output:**
  
  rule Init_2: \([ \text{Init}_1( \sim X, A, B, \sim k ), \text{In}( \text{senc}(<\sim X, \text{var}Y>, \sim k) ) ] \rightarrow [ \text{Out}(\text{'OK'}), \text{Init}_2( \sim X, \text{var}Y, A, B, \sim k ) ]\)
Tamarin tackles complex interaction with

Your protocol modeled with rewrite rules.....

adversary controlling the network

Out(t)

In(t)
Tamarin tackles complex interaction with 

Your protocol modeled with rewrite rules

adversary controlling the network
• What we saw:
  - Symbolic modeling
  - A small example protocol (will be studied in the tutorial)
  - The underlying model of Tamarin: multiset rewriting
  - Basic elements: fresh, public, constant, persistent

Next up: modeling properties
Intermission

Time for some questions
Outline

Security properties

Most common:

- Authentication
- Secrecy

Depending on the protocol:

- Perfect forward secrecy
- Anonymity
- Unlinkability
- ...

...
Property specification

- **first order logic** interpreted over a trace
  - False
  - Equality \( t_1 =_E t_2 \)
  - Timepoint ordering \( #i < #j \)
  - Timepoint equality \( #i = #j \)
  - Action at timepoint \( #i \)
  - Standard logical operators \( \implies, \& \), |, not( )
Property specification

• $l \rightarrow [a] \rightarrow r$

• Actions stored as (action) trace

Additionally:
- adversary knows facts: $K()$

\textbf{rule Init\_1:}
\[
[ \text{Fr}( \neg X ), !\text{Ltks}( A, B, \neg k ) ]
\rightarrow
[ \text{NonceI}( A, X, \neg k ) ]
\rightarrow
[ \text{Out}( \neg X ), \text{Init\_1}( \neg X, A, B, \neg k ) ]
\]

\textbf{Lemma trivialSecrecyNonceI:}
\[
(\text{All } #i A X k. \text{NonceI}(A, X, k)@i \Rightarrow \text{Not } (\text{Ex } #j. K(X)@j))
\]
Secrecy

**rule Ltk_reveal:**
\[
\begin{align*}
[ !\text{Ltk}($A$, $B$, ltk) ] & \land [ \text{RevLTK}($A$, ltk), \text{RevLTK}($B$, ltk) ] \rightarrow [ \text{Out}(ltk) ]
\end{align*}
\]

**lemma secrecyNonceI:**
/* It cannot be that */
"not( 
  Ex A X k #i #j.
  /* an inititor claims to have created a nonce with a secrecy claim, */
  \text{NonceI}(A, X, k) @ i
  /* but the adversary knows this nonce */
  \& K(X) @ j
  /* without having performed a long-term key reveal for a key associated to this session. */
  \& not (Ex X #r. RevLtk(X, k) @ r)
)"
Authentication ?
Authentication ?
A authenticates B iff:

- Whenever A finishes a session supposedly with B (commit),
- then B started a session with A using the same data (running).
Authentication in Tamarin

rule Init_2:
[ Init_1( ~X, $I, $R, ~k), In( senc(<~X, varY>, ~k) ) ]
--->[ RunningI($I, $R, ~k, <~X, varY>) ]-->
[ Out('OK'), Init_2( ~X, varY, $I, $R, ~k) ]

rule Resp_2:
[ Resp_1( ~Y, $R, $I, ~k, varX, ~Y ), In( 'OK' ) ]
--->[ CommitR($R, $I, ~k, <varX, ~Y>) ]-->
[]

lemma agree_R:
" /* Whenever a responder commits to running a session, then*/
   All actor peer k params #i.
       CommitR(actor, peer, k, params) @ i
   ==>  
   /* there is an initiator running a session with the same parameters */
       (Ex #j. RunningI(peer, actor, k, params) @ j & j < i)
   /* or the adversary perform a long-term key reveal on k */
       | (Ex A #r. RevLtk(A, k) @ r)
"

'c' constant
~t t has type fresh
$t t has type public
!F F is persistent
Injective Authentication?
Injective Authentication?
• What we saw:
  – How we can write security properties in Tamarin
  – How to formalize authentication and secrecy

Next up: Demo
Outline

Part 4:
Demo
• What we saw:
  – How to run Tamarin
  – How prove lemmas
  – How to interpret the results
  – How to read Tamarin’s graphs

Next up: Tips, tricks & exercises
Part 5:
Conclusion
How do I know my model is correct?

- Many ways to model incorrectly
- **Check warnings** when loading the model in Tamarin
- **Executability:** write a lemma to check that the protocol can at least be executed
- Check whether attacks found are realistic (within the model)
- **Break** the protocol on purpose and check whether Tamarin finds the expected attack(s)
- Look at the chains...
  - (requires an understanding of the algorithm)
- Much easier to check these things than in manual proofs!
Practical Exercises

- **See**
  
  https://gitlab.inria.fr/jdreier/cyber-in-saclay-tamarin/-/tree/master/exercises
  (Link also on the website!)

- There is a file containing the running example from the talk, plus instructions, and a link to a VM with Tamarin.

- **Do:**
  
  - Don’t forget to do a `git pull` if you use the VM
  - Read the file README.md (go to the link above)
  - Play with ex1.spthy
  - Check further security properties
  - Improve protocol
Practical Exercises

- Don’t forget the manual
  
  https://tamarin-prover.github.io/manual/

- Meet us in the gather.town in exercise room

- You can also use the discord channels
  
  #tamarin-monday or #tamarin-tuesday for questions (also before and after the session)