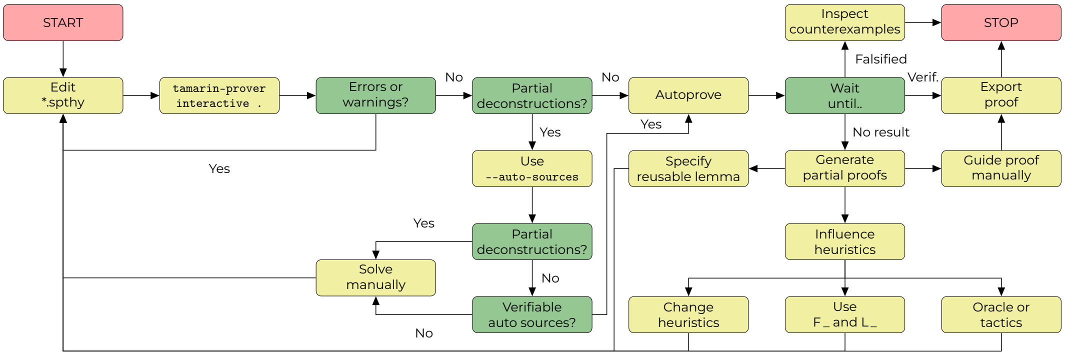
CISPA
HELMHOLTZ CENTER FOR
INFORMATION SECURITY

Formal Analysis of Real-World Security Protocols

Lecture 8: Advanced Features (Part 1)

Aleksi Peltonen | 2025-01-06

Recap: Tamarin workflow



This lecture

Pre-Computation and Partial Deconstructions

Injective Facts

Observational Equivalence

User-Specified Equational Theories

Pre-Computation and Partial Deconstructions

Pre-computing sources

- Tamarin performs optimizations to accelerate proof construction, one of which is **pre-computation of sources**
- For all facts in the model: Backwards search with the constraint solving algorithm to determine their sources
  - Sources: **Partial executions** that yield a fact
  - The search is incomplete; we do not check for non-termination
- For **attacker knowledge**, Tamarin considers three cases:
  1. Fresh values: `KU(~x)`
  2. Public values: `KU($x)`
  3. Function applications for all equations in the theory: `KU(f(`$x_1, \ldots, x_n$`))`

Saturation

- Once the pre-computation of sources is completed, Tamarin applies a **saturation process**
  - If there is an open premise inside a source corresponding to another source, the second source is applied to the open premise
  - This is applied repeatedly until a fixpoint is reached or a bound is hit
- Often fast, but can be limited **if necessary**
  - Limit max number of saturations: `--saturation=`
  - Limit chain goals to solve: `--open-chains=`
  - Exclude facts from the pre-computations: `[no_precomp]`
- Use with caution; worth trying if Tamarin seems to be stuck in a loop

Partial deconstructions

- To avoid non-termination, Tamarin stops when it encounters a chain goal that it cannot entirely resolve
  - This is called a **partial construction** (or an **open chain**)
  - Unresolved partial deconstructions often lead to **non-termination when proving lemmas**
- You can see a list of partial deconstructions in the GUI

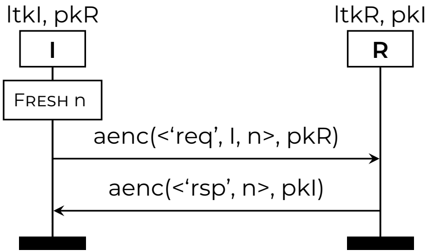
```
Raw sources (9 cases, 6 partial deconstructions left)
Refined sources (9 cases, deconstructions complete)
```

- The command-line interface will **not show you these**

Sources lemmas

- Tamarin's internal pre-computations can be inspected under **refined sources**
  - If all partial deconstructions can be solved, Tamarin will show you the message *deconstructions complete*
  - Otherwise, it will list the number of partial deconstructions left
- If Tamarin is unable to solve all partial deconstructions, you need to do so **before trying to prove any lemmas**
- We do this by creating and **proving** a special lemma with the annotation `[sources]`

Example



Simple challenge-responder protocol with two messages

- One partial deconstruction (out of six) is caused by Tamarin failing to find the sources of encryptions
- Specifically, Tamarin cannot determine whether the the **responder receives a value from the initiator, or from the attacker**
- See Chapter 8.4 in the course book for a detailed explanation

Option 1: Auto-generating sources lemmas

- Tamarin can automatically try to construct sources lemmas with the command-line argument `--auto-sources`
- This will tell Tamarin to look for inputs that cause partial deconstructs and try to solve them
  - The algorithm adds action facts `AUTO_IN_` or `AUTO_OUT_` to the rules and creates a sources lemma using them
- Works okay(ish), but is **not guaranteed to solve the problem**
- Might produce a lemma that solves the open chains **but cannot be proven**, or one that does not even solve the problem
- Use this as a initial option to give you hints for how to solve manually

Option 2: Writing sources lemmas manually

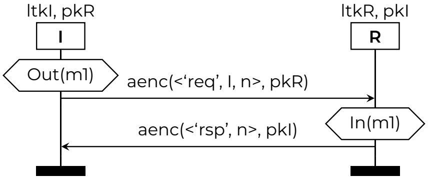
- **Approach**
  1. Identify the rules, messages, and variables causing partial deconstructions by inspecting the raw sources
  2. For each message input, look for matching outputs
  3. Add additional actions to the rules if needed
  4. Write a lemma proving the potential sources
- If there are multiple partial deconstructions, you have to write lemmas for all of them
  - Can also be one large lemma with logical AND (&) clauses
  - This is usually easier for Tamarin to prove

Example

```
// Initiator
rule Rule_I_1:
  let
    m1 = aenc{'req',$I,~n}pkR
  in
  [ Fr(~n)
  , !Ltk($I, ltkI)
  , !Pk(R, pkR) ]
--[ SecretI($I,R,~n)
  , Out_I(m1) ]->
  [ Out(m1)
  , State_I($I,R,~n,pkR,ltkI) ]

rule Rule_I_2:
  let
    m2 = aenc{'rsp',~n}pk(ltkI)
  in
  [ State_I($I,R,~n,pkR,ltkI)
  , In(m2) ]
--[  ]->
  [  ]
```

```
// Responder
rule Rule_R:
  let
    m1 = aenc{'req',I,x}pk(ltkR)
    m2 = aenc{'rsp',x}pkI
  in
  [ !Ltk(R, ltkR)
  , !Pk(I, pkI)
  , In(m1) ]
--[ In_R(m1, x) ]->
  [ Out(m2) ]
```



Solving paritial deconstructions

- Intuition: When `Rule_R` receives a message `m` with a variable `x`, then either
  - it was the output from `Rule_I_1`, or
  - the adversary already knew `x` before the message was received
- We can write a lemma stating that
  1. either the input was the expected protocol message coming from the initiator, or
  2. the adversary crafted a message of a corresponding format
- Adding this lemma **solves the partial deconstruction**

Example

Auto-generated sources lemma:

```
lemma AUTO_typing [sources]:
  all-traces
  "(⊤) ∧
  (∀ x m #i.
  (AUTO_IN_TERM_2_0_0_1_1__Rule_R( m, x ) @ #i) ⇒
  ((∃ #j. (!KU( x ) @ #j) ∧ (#j < #i)) ∨
   (∃ #j. (AUTO_OUT_TERM_2_0_0_1_1__Rule_R( m ) @ #j)
   ∧ (#j<#i)))))"
```

Manually written sources lemma:

```
lemma sources [sources]:
    " All m x #i . In_R(m, x)@i
        ==> ((Ex #j. Out_I(m)@j & #j < #i)
             | (Ex #k. KU(x)@k & #k < #i))
    "
```

Injective Facts

Injective facts

- Tamarin has built-in support for reasoning about injective fact symbols (or **injective facts** for short), i.e., facts whose instances are always **unique**
- An injective fact is one where all instances of the fact `Fact(~x, ...)` come from either an initialization rule that creates it with an actual fresh fact `Fr(~x)` in its premise, or from a rule that has just consumed and produced it again
- Identifying whether a fact symbol is injective is, in general, **undecidable**

Injective facts in Tamarin

- You can check if Tamarin detected that your theory contains injective facts in the GUI
  - To do this, click on *Multiset rewriting rules* on the left-hand side
- Tamarin can **optimize its reasoning** for facts that it determines to be injective
  - When writing models, try to use facts of the form

```
Fact(~id, term_1, term_2, ...)
```

    when e.g., storing state information

Observational Equivalence

Observational equivalence

- Until now, all properties have been evaluated over individual traces
  - These are called **trace properties** and are ideal for analyzing security
- Observational equivalence describes a hyperproperty that **compares two traces**
  - Used to analyze **privacy properties** like anonymity and unlinkability
- Support in Tamarin was added in 2015 [BDS15]
- Concretely: Enter the two systems as a bi-system where one input gives two versions of the same system
- The systems are identical, except for terms wrapped under `diff(x,y)`
  - `x` = left instance
  - `y` = right instance

Differences to trace properties

- Main difference to trace properties: **No user-defined lemmas**
    - Instead: Automatically created equivalence lemma used to compare two systems
- Analyzing the model is similar to the workflow presented in Lecture 6, with some minor changes
    1. Fine-tuning heuristics is often less effective
        - Instead: Fine-tune the model
    2. Equivalence proofs are often much larger
        - Longer verification times
    3. Restrictions cause issues

Use in Tamarin

- Argument: `--diff`
  - This adds the diff operation to allowed function symbols and generates extra lemmas
- In the **interactive mode**, Tamarin shows two versions of the message theory, rules, and precomputed sources
- See Chapter 13 in the course book for examples


```
Diff-Lemmas

lemma Observational_equivalence:
rule-equivalence
  case Rule_Alice_and_Bob_pairing
  by sorry
next
  case Rule_Alice_first
  by sorry
next
  case Rule_Alice_second
  by sorry
next
  case Rule_Bob
  by sorry
next
  case Rule_Destrd_0_fst
  by sorry
next

    ⋮

end
```

User-Specified Equational Theories

Recap: Equational theories

- An **equational theory** is a set of rules that determine which terms are considered equivalent
- **Motivation:**
  - Some messages (such as exponentiation) can be constructed in more than one way
  - Convenient for modeling cryptographic primitives
  - Allows us to model degenerate cases of cryptographic primitives

User-specified equational theories

- You can define your own equational theories in Tamarin:

```
equations: EXPR1 = EXPR2
```

- For example, we can define symmetric encryption as:

```
functions: senc/2, sdec/2
equations: sdec(senc(m,k),k) = m
```

- User-defined equational theories **cannot overlap with built-in ones**
- Due to fundamental theoretical limitations, **Tamarin cannot handle arbitrary equational theories**

Subterm convergence

- An equation is **subterm-convergent** when its right-hand side is either a strict subterm of its left-hand side or a constant
- An equational theory is subterm-convergent when all of its individual equations have this property

  `h(f(X),Y,Z) = f(X)` is subterm-convergent

  `h(f(X),Y,Z) = f(Y)` is not

- An equational theory is supported by Tamarin if:
  1. It is subterm-convergent, and
  2. It is syntactically disjoint from the built-in equational theories

Subterm convergence

- Equational theories that are **not subterm-convergent** must at least (1) be convergent, and (2) have the finite variant property (FVP)
- This is not trivial to check by users (but still necessary)
    - There are tools to help with this
- If either condition is not met, Tamarin will almost certainly **not terminate**
- Correctness can also not be guaranteed, since it relies on the correctness of equational theories

Reading material

Recommended reading: [Bas+25, Ch. 8, 13–14], [BDS15]

[Bas+25] D. Basin, C. Cremers, J. Dreier, and R. Sasse. **Modeling and Analyzing Security Protocols with Tamarin: A Comprehensive Guide.** Draft v0.9.5. May 2025.

[BDS15] D. Basin, J. Dreier, and R. Sasse. **Automated Symbolic Proofs of Observational Equivalence.** In: Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security. 2015.

Reading material

Additional reading: [CJL22], [Dre+17]

[CJL22] C. Cremers, C. Jacomme, and P. Lukert. **Subterm-based proof techniques for improving the automation and scope of security protocol analysis.** Cryptology ePrint Archive, Paper 2022/1130. 2022.

[Dre+17] J. Dreier, C. Duménil, S. Kremer, and R. Sasse. **Beyond Subterm-Convergent Equational Theories in Automated Verification of Stateful Protocols.** In: Proceedings of the 6th International Conference on Principles of Security and Trust. 2017.