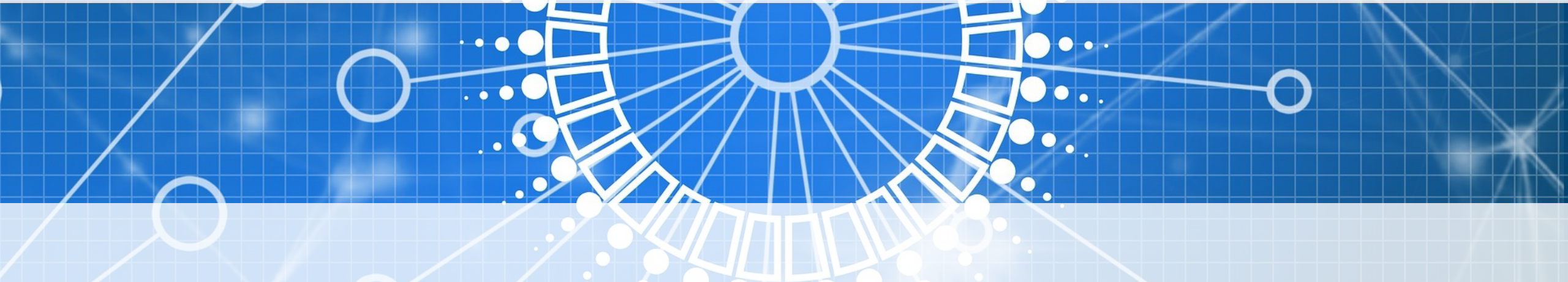
An introduction to analyzing cryptographic protocols using Tamarin prover



Douglas Stebila

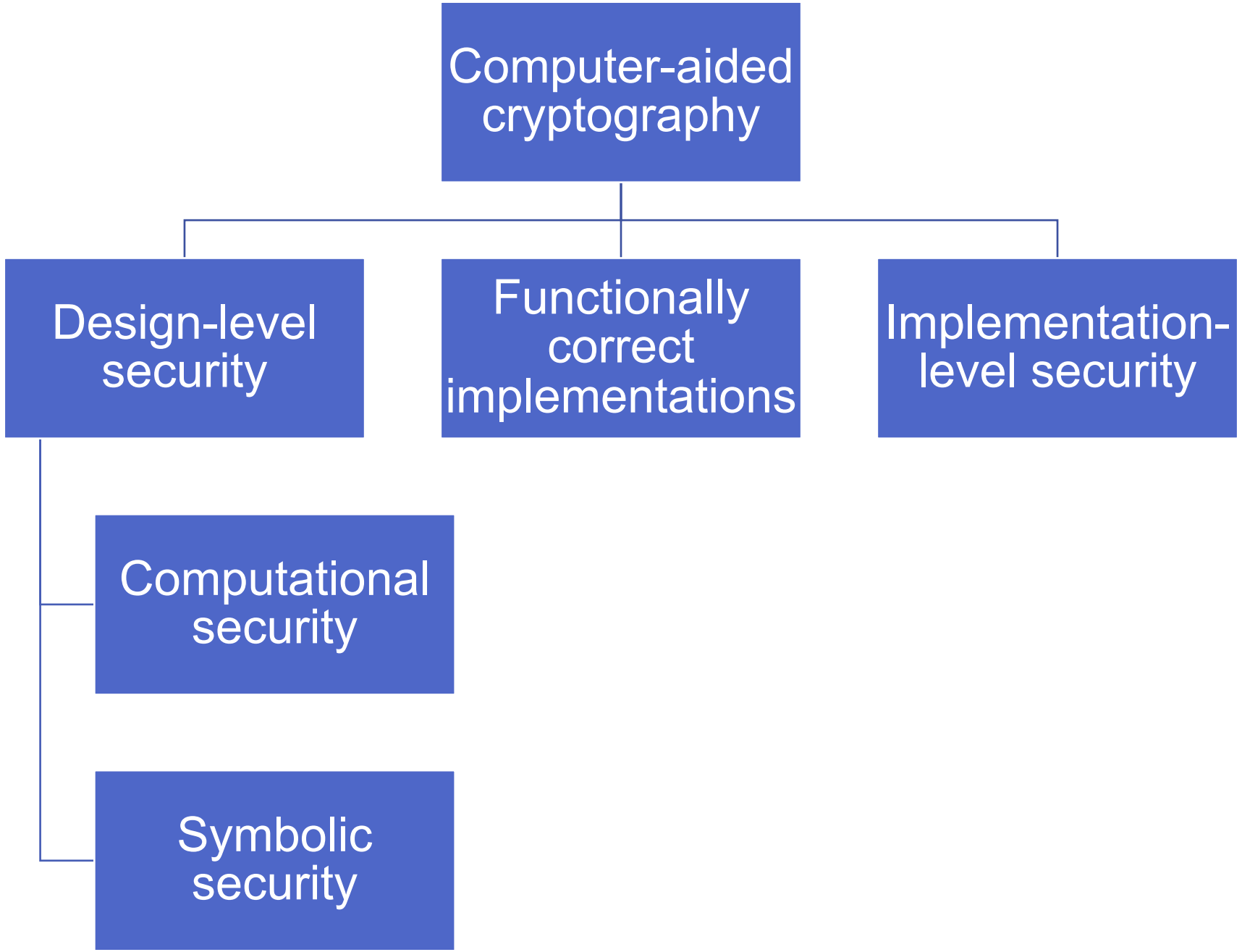
UNIVERSITY OF WATERLOO

Outline

1. Overview of computer-aided cryptography
2. Introduction to Tamarin
3. Example: KEM-based key exchange
4. Advanced example: authenticated key exchange
5. Hands-on exercises

Overview of computer-aided cryptography

Key resource:
Barbosa, Barthe, Bhargavan, Blanchet, Cremers, Liao, Parno. SoK: Computer-Aided Cryptography. IEEE S&P 2021. https://eprint.iacr.org/2019/1393



https://eprint.iacr.org/2019/1393

Design-level security

- Use of mathematical and logical arguments to analyze the security of a cryptographic design (protocol, primitive) against a class of attacks

Components of design-level security

1. Definition of the type of object being analyzed
   - e.g.: key encapsulation mechanism
2. Security definition for that type of object
   - e.g.: indistinguishability under chosen ciphertext attack
3. A specific scheme being analyzed
   - e.g.: Kyber / ML-KEM
4. A formal argument that the scheme satisfies the desired security property, possibly under some cryptographic assumptions
   - e.g. Theorem and proof that Kyber is IND-CCA under the module-LWE assumption in the random oracle model.

Motivation for computer-aided analysis

- Arguments can have errors
  - Hand-written proofs often have errors
- Security modelling might be inappropriate
  - Model and protocol might be simplified to a "cryptographic core" but an attack might exist outside that core

Symbolic model

- Abstract model for modeling and analyzing protocols
- Objects (messages, keys, nonces) are represented as opaque / atomic terms
- Cryptographic primitives are black-box over terms with an equational theory
  - e.g. Dec(Enc(m, k), k) = m
- Adversaries can only compute new terms based on terms they already know and the equational theory
  - e.g. Given c = Enc(m, k), can only learn m if one knows all of k and applies Dec(c, k), otherwise know nothing about m

Security in the symbolic model

- Symbolic model is a simplified model of protocols
- Enables analysis using symbolic logic
- Suitable for automatically search for and verifying logical flaws in a complex system
- Two types of security properties:
  - Trace properties: a specific bad event can never occur on any trace of execution of the system
    - e.g. Confidentiality: data meant to be secret is never learned by the adversary
  - Equivalence properties: the adversary cannot distinguish between two protocols

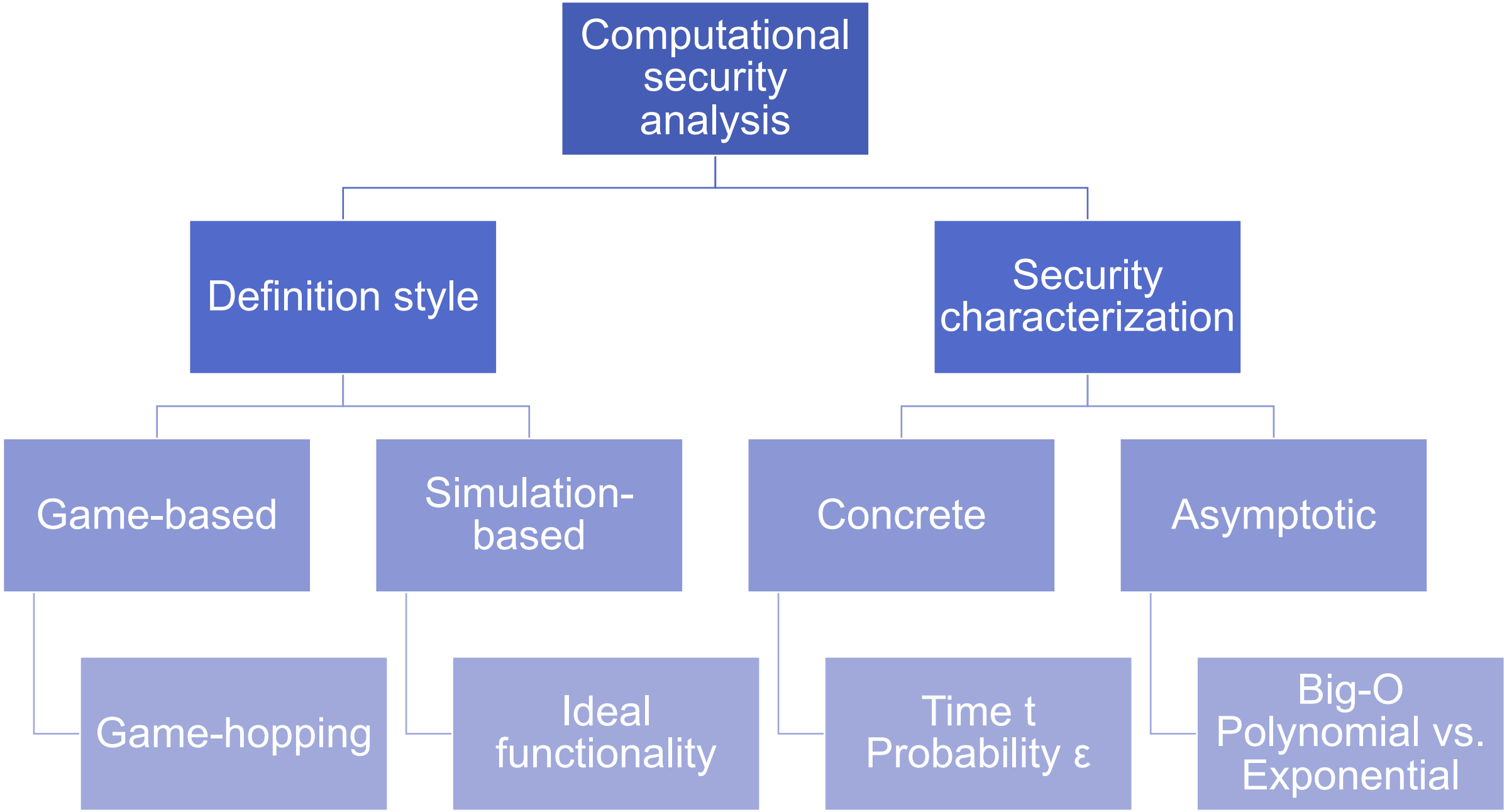
Computational model

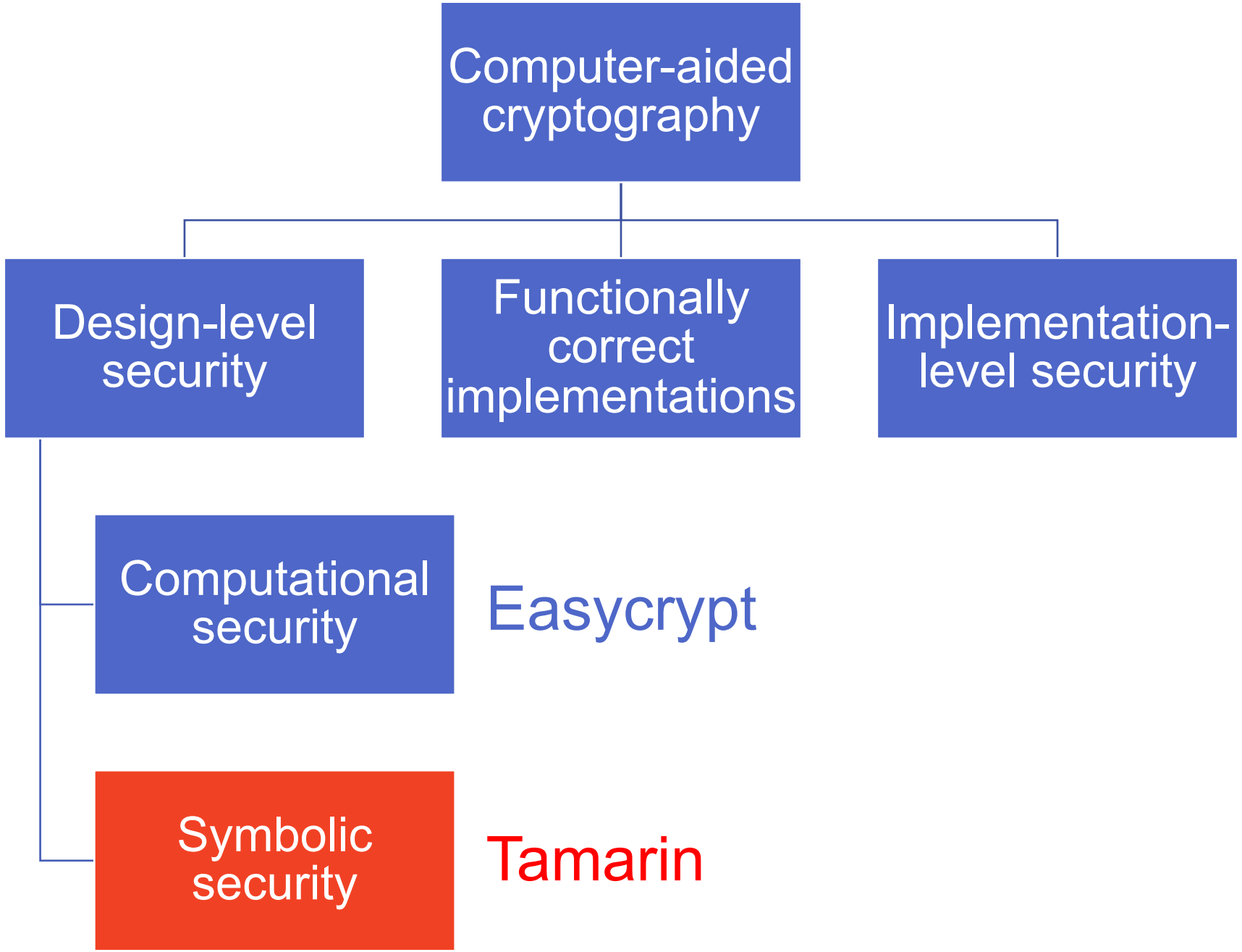
- Objects (messages, keys, nonces) are bitstrings or mathematical elements with internal structure
- Cryptographic primitives are probabilistic algorithmics
- Adversaries are probabilistic Turing machines operating on bitstrings / mathematical elements
  - Adversaries can try to compute partial information by employing computational resources

Security in the computational model

Basic structure of a security statement:

- If there exists an adversary A that can break property X of scheme S in runtime t with probability ε, then there exists an adversary A', which uses A, that can break property X' of scheme S' in runtime t' with probability ε'.


Computational security analysis
Definition style
Security characterization
Game-based
Simulation-based
Concrete
Asymptotic
Game-hopping
Ideal functionality
Time t
Probability ε
Big-O
Polynomial vs. Exponential



https://eprint.iacr.org/2019/1393

Introduction to Tamarin

Formal verification using Tamarin

- Tamarin prover is a model checker for security protocols in the symbolic model
- Protocol and adversary powers are specified as a set of state machine transitions (“multiset rewriting rules”)
- Security property is specified as a predicate over actions recorded during state machine transitions
- Tamarin prover explores (infinite) state space of all possible executions to find an execution trace that violates the security property or verifies that none exists (or fails to terminate)

https://tamarin-prover.github.io/

Formal verification using Tamarin

- Tamarin successfully used on many academic and real-world cryptographic protocols
- Especially effective on key exchange protocols
  - Note Tamarin models key exchange security based on *learning* session key, not *indistinguishability*
- Tamarin model of TLS 1.3 drafts [CHSV,CHHSV] found several flaws
  - Especially in interactions between different protocols modes
    - e.g. in TLS 1.3 pre-shared key resumption
  - Expensive: months of person-effort, 1 week of computation time, 100 GB RAM

[CHSV] Cremers, Horvat, Scott, van der Merwe, IEEE S&P 2016.
[CHHSV] Cremers, Horvat, Hoyland, Scott, van der Merwe, ACM CCS 2017.

Tamarin: high-level

- **Modeling** protocol & adversary done using multiset rewriting
  - Specifies transition system; induces set of traces
- **Property** specification using fragment of first-order logic
  - Specifies “good” traces
- Tamarin tries to
  - provide proof that all system traces are good, or
  - construct a counterexample trace of the system (attack)

Slide from https://github.com/tamarin-prover/teaching/blob/master/Tamarin-Tutorial-morning.pdf

Modeling in Tamarin

- **Multiset rewriting**; surprisingly similar to “oracles”
- Basic ingredients:
  - **Terms** (think “messages”)
  - **Facts** (think “sticky notes on the fridge”)
  - Special facts: **Fr(t)**, **In(t)**, **Out(t)**, **K(t)**
- 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 --> r`
  - `l --[ a ]-> r`

Slide from https://github.com/tamarin-prover/teaching/blob/master/Tamarin-Tutorial-morning.pdf

The model

- **Term algebra**
  - enc(_,_), dec(_,_), h(_,_), _^_, _$^{-1}$, _*_, 1, …
- **Equational theory**
  - dec(enc(m,k),k) $=_E$ m,
  - (x^y)^z $=_E$ x^(y*z),
  - $(x^{-1})^{-1} =_E$ x, …
- **Facts**
  - F(t1,...,tn)
- **Transition system**
  - State: multiset of facts
  - Rules: l –[a]→ r
- **Tamarin-specific**
  - Built-in Dolev-Yao attacker rules
    - In( ), Out( ), K( )
  - Special **Fresh** rule:
    - [] --[]--> [Fr(**x**)]
      - With additional constraints on systems such that **x** unique

Slide from https://github.com/tamarin-prover/teaching/blob/master/Tamarin-Tutorial-morning.pdf

Semantics

- **Transition relation**

  $S \text{ –}[a]\rightarrow_R ((S \setminus^{\#} I) \cup^{\#} r)$

  where $I \text{ –}[a]\rightarrow r$ is a ground instance of a rule and $I \subseteq^{\#} S$

- **Executions**

  $Exec(R) = \{ [\,] \text{ –}[a_1]\rightarrow \ldots \text{ –}[a_n]\rightarrow S_n$
  $\mid \forall n . Fr(n) \text{ appears only once on rhs } \}$

- **Traces**

  $Traces(R) = \{ [a_1,\ldots,a_n]$
  $\mid [\,] \text{ –}[a_1]\rightarrow \ldots \text{ –}[a_n]\rightarrow S_n \in Exec(R) \}$

Slide from https://github.com/tamarin-prover/teaching/blob/master/Tamarin-Tutorial-morning.pdf

‘c’ constant

Semantics: example 1

- **Rules**
  - rule 1: [ ] –[Init()]→ [A('5')]
  - rule 2: [A(x)] –[Step(x)]→ [B(x)]
- **Execution example**
  - [ ]
  - –[Init()]→ [A('5')]
  - –[Init()]→ [A('5'), A('5')]
  - –[Step('5')]→ [A('5'), B('5')]
- **Corresponding trace**
  - [Init(), Init(), Step('5')]

Slide from https://github.com/tamarin-prover/teaching/blob/master/Tamarin-Tutorial-morning.pdf

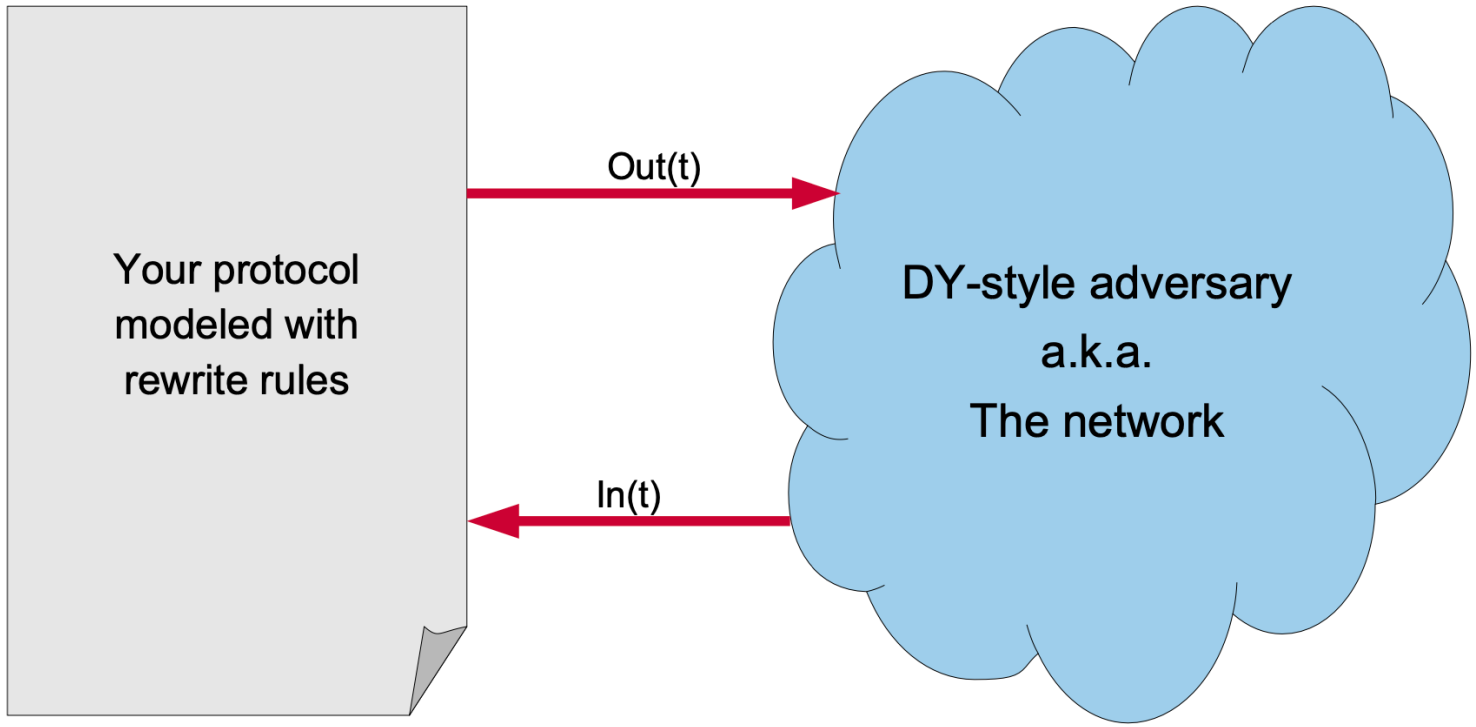
‘c’ constant

Semantics: example 2 (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'))]

Slide from https://github.com/tamarin-prover/teaching/blob/master/Tamarin-Tutorial-morning.pdf

Tamarin tackles complex interaction with adversary



Slide from https://github.com/tamarin-prover/teaching/blob/master/Tamarin-Tutorial-morning.pdf

Property specification

- first order logic interpreted over a trace
  - False — False
  - Equality — $t_1 =_E t_2$
  - Timepoint ordering — #i < #j
  - Timepoint equality — #i = #j
  - Action at timepoint #i — A@#i
- `l --[ a ]-> r`
- Actions stored as (action) trace

  Additionally:
  adversary knows facts: K()

Slide from https://github.com/tamarin-prover/teaching/blob/master/Tamarin-Tutorial-morning.pdf

Algorithm intuition

- **Constraint solving algorithm**
- Main ingredients:
  - Dependency graphs
  - Deconstruction (decryption) chains
  - Finite variant property (more this afternoon)
- **Invariant**: if adversary knows M then either
  - M was sent in plain
  - Adversary can construct M by knowing subterms
  - Adversary can deconstruct M …. from message sent by protocol rule

Slide from https://github.com/tamarin-prover/teaching/blob/master/Tamarin-Tutorial-morning.pdf

Basic principles

- Backwards search using **constraint reduction rules** (27!)
- Turn negation of formula into set of constraints
- Case distinctions
  - E.g.: Possible sources of a message or fact
- Try to establish:
  - no solutions exist for constraint system, or
  - there exists a „realizable“ execution (trace)
- If multiple rules can be applied: use heuristics

Slide from https://github.com/tamarin-prover/teaching/blob/master/Tamarin-Tutorial-morning.pdf

Heuristics?

- If Tamarin terminates, one of two options:
  - **Proof**, or
  - **counterexample** (in this context: attack)
- At each stage in proof, multiple constraint solving rules might be applicable
  - Similar to “how shall I try to prove this?”
  - Choice influences speed & termination, but not the outcome after termination
- Complex **heuristics choose rule**
  - user can give hints or override

Slide from https://github.com/tamarin-prover/teaching/blob/master/Tamarin-Tutorial-morning.pdf

Lemmas

- When it doesn’t terminate…
- Guide the proof manually; export
- Write **lemmas**
  - “**Hints**” for the prover
    - They don't change the proof obligation, only help finding a proof
  - Specify lemma that can be used to prune proof trees at multiple points
  - ... more this afternoon and at TLS:DIV

Slide from https://github.com/tamarin-prover/teaching/blob/master/Tamarin-Tutorial-morning.pdf

Example: KEM-based key exchange

Key encapsulation mechanisms

$\mathsf{KGen}() \xrightarrow{\$} (\mathsf{pk}, \mathsf{sk})$

$\mathsf{Encaps}(\mathsf{pk}) \xrightarrow{\$} (\mathsf{ct}, \mathsf{ss})$

$\mathsf{Decaps}(\mathsf{sk}, \mathsf{ct}) \rightarrow \mathsf{ss}$

$\mathsf{KGen}(\mathsf{sk}) \rightarrow \mathsf{pk}$

$\mathsf{Encaps}(\mathsf{pk}, \mathsf{coins}) \rightarrow (\mathsf{ct}, \mathsf{ss})$

$\mathsf{Decaps}(\mathsf{sk}, \mathsf{ct}) \rightarrow \mathsf{ss}$

Correctness:
$\mathsf{Decaps}(\mathsf{sk}, \mathsf{Encaps}(\mathsf{KGen}(\mathsf{sk}), \mathsf{coins})[1])$
$= \mathsf{Encaps}(\mathsf{KGen}(\mathsf{sk}), \mathsf{coins})[2]$

Key encapsulation mechanisms

$\mathsf{KGen}() \xrightarrow{\$} (\mathsf{pk}, \mathsf{sk})$

$\mathsf{Encaps}(\mathsf{pk}) \xrightarrow{\$} (\mathsf{ct}, \mathsf{ss})$

$\mathsf{Decaps}(\mathsf{sk}, \mathsf{ct}) \to \mathsf{ss}$

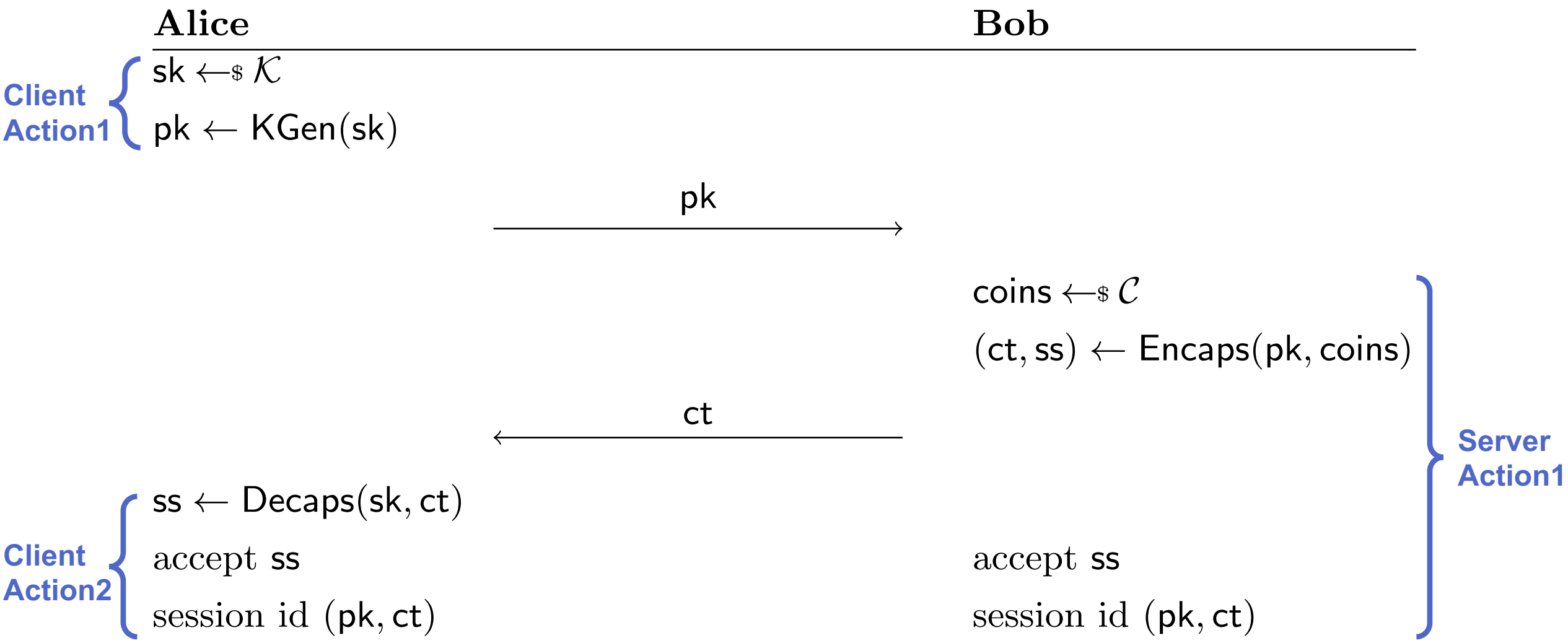
$\mathsf{KGen}(\mathsf{sk}) \to \mathsf{pk}$

$\mathsf{Encaps\_ct}(\mathsf{pk}, \mathsf{coins}) \to \mathsf{ct}$
$\mathsf{Encaps\_ss}(\mathsf{pk}, \mathsf{coins}) \to \mathsf{ss}$

$\mathsf{Decaps}(\mathsf{sk}, \mathsf{ct}) \to \mathsf{ss}$

Correctness:
$\mathsf{Decaps}(\mathsf{sk}, \mathsf{Encaps\_ct}(\mathsf{KGen}(\mathsf{sk}), \mathsf{coins}))$
$= \mathsf{Encaps\_ss}(\mathsf{KGen}(\mathsf{sk}), \mathsf{coins})$

A basic KEM-based key exchange

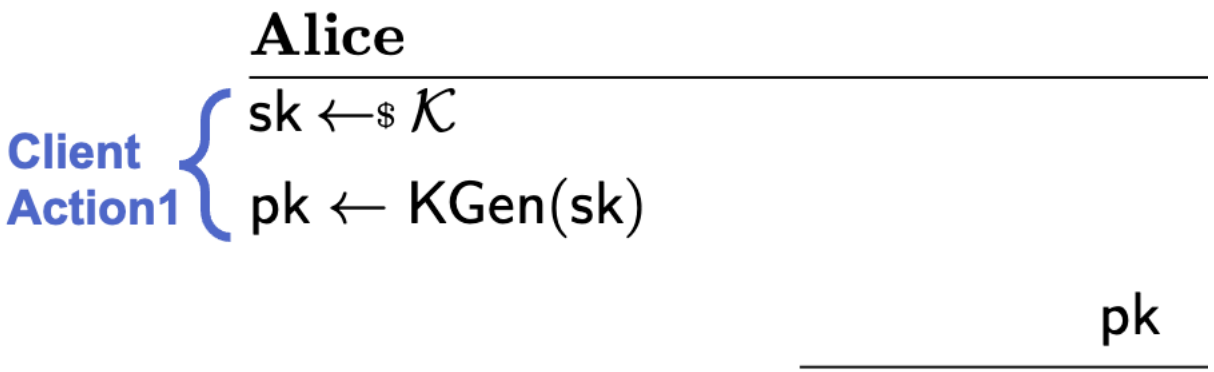


Tamarin rules

1. (Shorthand declarations)
2. Requirements
3. Produces
4. Action facts recorded

```
rule RULENAME:
    let
        SHORTHAND_DECLARATION_1
        SHORTHAND_DECLARATION_2
    in
        [ REQUIREMENTS ]
      --[ ACTION_FACTS_RECORDED ]->
        [ PRODUCES ]
```

Tamarin rule




```
rule ClientAction1:
  let
    pk_e = KEM_PK(~sk_e)
  in
    [ Fr(~sk_e), Fr(~tid) ]
  --[ ]->
    [ Out(pk_e),
      ClientState(~tid, pk_e, ~sk_e)
    ]
```

Rule name

Shorthand declaration

Requirements

Produces

Correctness

If an honest client thread accepts a session key ss_c with session id sid_c,

and an honest server thread accepts a session key ss_s with session id sid_s,

and they have the same session ids (sid_c = sid_s),

then they have the same shared secret (ss_c = ss_s)

Security

If an honest client thread accepts a session key ss_c,

then the adversary does not know ss_c.

For all threads tid_c and shared secrets ss_c,

if Role(tid_c, ‘client’)

and SessionKey(tid_c, ss_c),

then the adversary does not know ss_c.

Security

Lemmas are predicates over “action facts” that we recorded in our rules

For all threads tid_c and shared secrets ss_c,

if Role(tid_c, ‘client’)

and SessionKey(tid_c, ss_c),

then the adversary does not know ss_c.

```
lemma ss_secure:
  "
    All tid_c ss_c
    .
    Role(tid_c, 'client')
    & SessionKey(tid_c, ss_c)
    ==>
    not(K(ss_c))
  "
```

Security

For all threads tid_c and shared secrets ss_c,

if Role(tid_c, ‘client’)

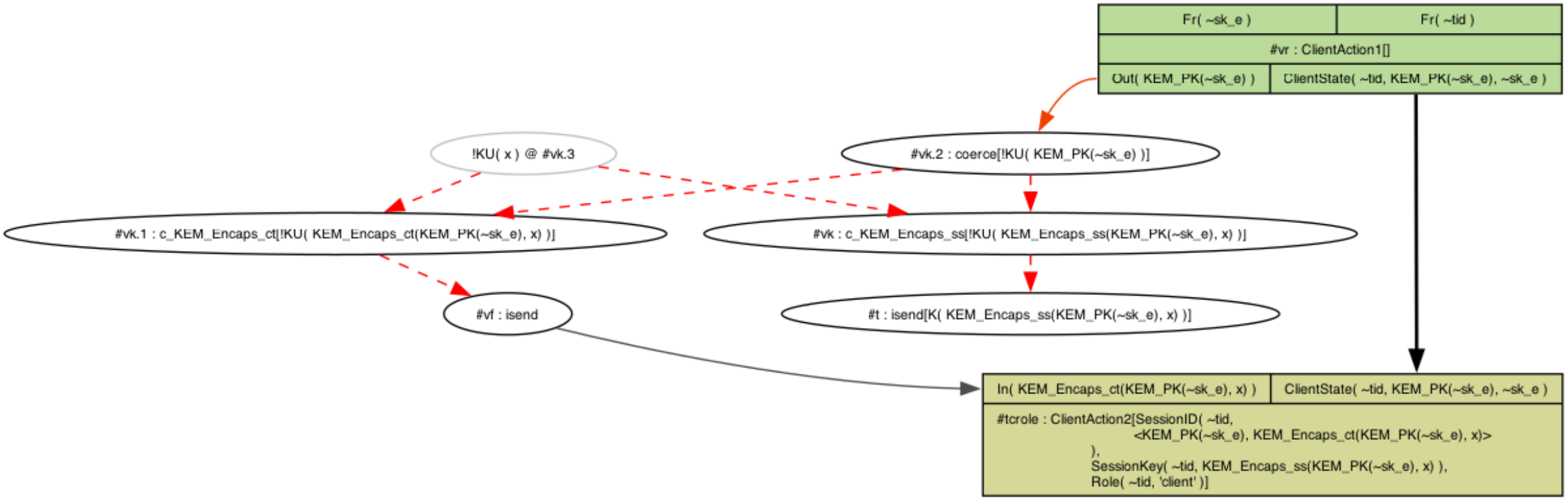
and SessionKey(tid_c, ss_c),

then the adversary does not know ss_c.

```
lemma ss_secure_take1:
  "
    All tid_c ss_c #t1 #t2
    .
    Role(tid_c, 'client') @ #t1
    & SessionKey(tid_c, ss_c) @ #t2
    ==>
    not(Ex #t . K(ss_c) @ #t)
  "
```

All action facts are timestamped
(Can use this to order events if needed)

Attack found!



Start interactive Tamarin session: tamarin-prover interactive .
Open example1_complete.spthy
For lemma ss_secure_take1, click "Sorry" then "autoprove"

Security against a passive adversary

If an honest client thread accepts a session key ss_c,

and the adversary was passive,

then the adversary does not know ss_c.

Security against a passive adversary

If an honest client thread accepts a session key ss_c,

and there exists a server thread with a matching session identifier,

then the adversary does not know ss_c.

Advanced example: authenticated key exchange

Authenticated key exchange

Alice		**Bob**
		$\mathsf{sk}_B \leftarrow\!\!\$\ \mathsf{SIG}.\mathcal{K}$
$(\mathsf{pk}_B, \text{“Bob”})$		$\mathsf{pk}_B \leftarrow\!\!\$\ \mathsf{SIG.KGen}(\mathsf{sk}_B)$
$\mathsf{sk}_e \leftarrow\!\!\$\ \mathsf{KEM}.\mathcal{K}$		
$\mathsf{pk}_e \leftarrow \mathsf{KEM.KGen}(\mathsf{sk})$		
	$\mathsf{pk}_e \longrightarrow$	
		$\mathsf{coins} \leftarrow\!\!\$\ \mathsf{KEM}.\mathcal{C}$
		$(\mathsf{ct}, \mathsf{ss}) \leftarrow \mathsf{KEM.Encaps}(\mathsf{pk}_e, \mathsf{coins})$
		$\sigma \leftarrow\!\!\$\ \mathsf{SIG.Sign}(\mathsf{sk}_B, \mathsf{ct})$
	$\mathsf{ct}, \sigma \longleftarrow$	
reject if $\neg\mathsf{SIG.Vf}(\mathsf{pk}_B, \mathsf{ct}, \sigma)$		
$\mathsf{ss} \leftarrow \mathsf{KEM.Decaps}(\mathsf{sk}, \mathsf{ct})$		
accept $\mathsf{H}(\mathsf{ss}, \mathsf{pk}_B \Vert \mathsf{pk}_e \Vert \mathsf{ct})$		accept $\mathsf{H}(\mathsf{ss}, \mathsf{pk}_B \Vert \mathsf{pk}_e \Vert \mathsf{ct})$
session id $(\mathsf{pk}_B, \mathsf{pk}_e, \mathsf{ct})$		session id $(\mathsf{pk}_B, \mathsf{pk}_e, \mathsf{ct})$
owner $\perp$		owner “Bob”
peer “Bob”		peer $\perp$

Threat model

- Attacker can compromise long-term keys of parties
- Attacker can compromise session keys of parties

Security properties

- Client session key: secure if no session key reveal at owner or partner, and no long-term key reveal of peer before client accepted

Hands-on exercise

Getting started

- Tamarin website:
  - https://tamarin-prover.github.io/
- Fairly detailed manual
- Lots of examples in the Github repository

- Fairly easy to install on Linux or macOS using brew package manager

Exercises

- Download from https://github.com/dstebila/tamarin-examples
- Example 3: PRF security
- Example 4: stream cipher built from a PRF

Example 3: PRF security

- Fully commented file with all the answers
- You should carefully read through the file
- Then run the proofs
- Examine the attack traces found and try to understand them

- Scenario:
  - Let f(k, m) be a function
  - Imagine honest user generates key k
  - Adversary can submit m and get f(k, m)
  - Adversary shouldn't be able to learn f(k, m') for any m' not previously queried
- Note: keys have associated keyid's; the adversary isn't given the key but is given the keyid as a handle

Example 4: stream cipher built from a PRF

- You need to fill in the details
- Stream cipher: Enc(k, m) = f(k, nonce) XOR m for randomly chosen nonce
- Part 1: security against ciphertext-only attacks
- Part 2: correctness
- Part 3: insecurity against chosen plaintext + ciphertext attacks