CISPA
HELMHOLTZ CENTER FOR
INFORMATION SECURITY

Formal Analysis of Real-World Security Protocols

Lecture 2: Protocols in the Symbolic Model

Aleksi Peltonen | 04.11.2024

Model components

What **components** do we need to model protocols?

1.	All possible sent and received messages	Lecture 1
2.	All possible protocol behaviors	Lecture 2
3.	The attacker	Lecture 3
4.	Security properties that we want to verify	

This lecture

Unification

Term Deduction

Protocol Modeling

Protocols as Rules and Facts

Multiset Rewriting

Unification

Unification modulo E

Recall from last week: **Unification** determines if two terms with variables can be made equal. Two terms s and t are said *unifiable* if there exists a substitution σ, called a *unifier*, such that $s\sigma = t\sigma$.

In practice, we often perform unification **modulo** E, i.e., taking into account some equational theory. We write this as $=_E$.

Example

The equation $x \times 1 = y \times 2$ has no solution if we only consider only syntactic equality, since we have not defined anything about the multiplication function. However, if we define *commutativity* for the operator, we can solve the equation using e.g., $\sigma = \{x \mapsto 2, y \mapsto 1\}$.

Term Deduction

Inference rules

- An **inference rule** is of the form:

$$\frac{t_1 \quad t_2 \quad \cdots \quad t_n}{t}$$

  where t and $t_1, \ldots, t_n$ are terms
- Defines how we can use a set of terms to learn a new term
- An **inference system** is a set of inference rules
- We can use inference rules to represent **attacker capabilities** in our model!
  - More in lecture 3, when we talk about attacker models

Term deduction

- What can we deduce from the knowledge we have?
- **Deduction rules:**
  1. Construction:

$$\frac{k \quad m}{\mathrm{senc}(m, k)}$$

  2. Deconstruction:

$$\frac{k \quad \mathrm{senc}(m, k)}{m}$$

- More in lecture 4, when we talk about constraint systems

Term deduction example

Let $\mathcal{T}$ be a set of terms as follows:

$$\mathcal{T} = \{\mathrm{senc}(a, b), \mathrm{senc}(b, c), \mathrm{senc}(c, d), \langle d, e\rangle\}$$

Can we deduce a? **Yes.**

$$\dfrac{\dfrac{\dfrac{\dfrac{\langle d, e\rangle}{d} \qquad \mathrm{senc}(c, d)}{c} \qquad \mathrm{senc}(b, c)}{b} \qquad \mathrm{senc}(a, b)}{a}$$

Automatic term deduction

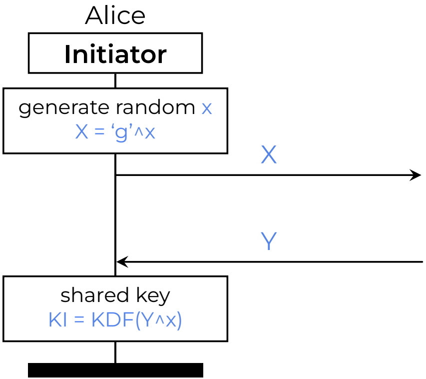
- **Intruder deduction problem**: given a state of the protocol execution, can the intruder derive a given message m?
- Is manual message deduction possible? **Yes.**
- Is it easy an convenient? **No.**
- Can we automate it? **Yes!**
- More in lecture 5, when we talk about Tamarin's constraint solving algorithm

Protocol Modeling

Modeling protocol execution

Protocol descriptions are “blueprints”:

- Protocols describe multiple **roles**
  - e.g., client - server, initiator - responder
- Parties **execute** these roles
  - e.g., Alice as the initiator, Bob as the responder, Charlie as the client, etc.
  - Parties can execute multiple roles
  - Each role execution at a party is a separate **thread**

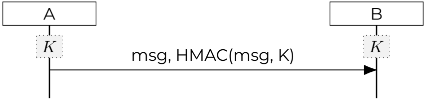


Modeling protocols

- By now you have seen how to model messages as terms:
  - We represent cryptographic functions with **equational theories**
  - We can deduce terms from other terms
- How do we combine all this to model protocols?
  - Multiple options!

1. Modeling protocols as **processes** (e.g., ProVerif)

```
let A(K:bitstring) =
  new msg:bitstring;
  out(c, (msg, HMAC(msg, K)));
  event SendMessage(msg)
.

let B(K:bitstring) =
  in(c, (msg:bitstring,
         MAC:bitstring));
  if MAC = HMAC(msg, K) then
  event ReceiveMessage(msg)
.
```

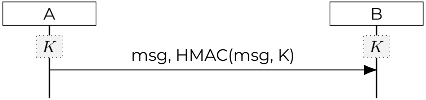


Modeling protocols

- By now you have seen how to model messages as terms:
  - We represent cryptographic functions with **equational theories**
  - We can deduce terms from other terms
- How do we combine all this to model protocols?
  - Multiple options!

2. Modeling protocols as **multiset rewriting rules** (e.g., Tamarin)

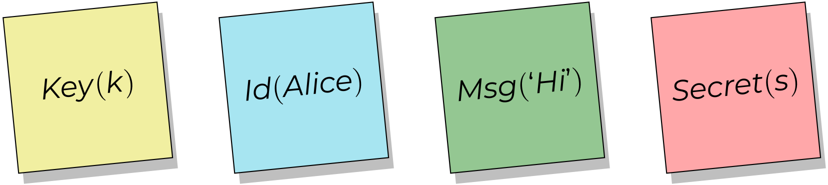
```
rule a_snd_msg:
   [ !A(K)
   , Fr(~msg) ]
 --[ SendMessage(msg) ]->
   [ Out(<~msg, HMAC(~msg,K)>) ]

rule b_rcv_msg:
   [ !B(K)
   , In(<msg, HMAC(msg, K)>) ]
 --[ ReceiveMessage(msg) ]->
   [ ]
```



Protocols as Rules and Facts

Facts

- **Facts** are used to store information about
  1. the transition system's current **state**
  2. the performed actions that are relevant for property specification
- Informally: *sticky notes on a fridge*



Facts

- Formally: We assume an unsorted signature Σ_{Fact} and define a **fact** as $\mathsf{F}(t_1, \ldots, t_n)$ for $\mathsf{F} \in \Sigma_{Fact}$ and $t_1, \ldots, t_n \in \mathcal{T}_\Sigma(\mathcal{V} \cup \mathcal{C})$
- We define the *state* of the global transition system as a **multiset of facts**
  - A multiset (sometimes also called a “bag”) is a set, in which members can occur multiple times
    - e.g., $\{1, 1, 1, 1, \ldots\}$, $\{Alice, Alice, Bob\}$, $\{a, a, b, c, d, d, e\}$
  - We write $\subseteq^\sharp$ for multiset inclusion, $\cup^\sharp$ for multiset union, and $\setminus^\sharp$ for multiset difference
  - $X^\sharp$ denotes the finite multisets with elements from X

Facts

- Facts can be either **linear** or **persistent**
  - Linear facts are consumed when we use them in a transition system
  - Persistent fact do not change
- Tamarin has several built-in fact symbols:

`K/1`	:	`K(t)`	–	check if the adversary can derive the term `t`
`In/1`	:	`In(t)`	–	`t` was received from the network
`Out/1`	:	`Out(t)`	–	`t` was sent to the network
`Fr/1`	:	`Fr(t)`	–	`t` was freshly generated

Rules

Rules model the possible *transitions* in a protocol.

- Syntax: $L \rightarrow R$

Intuitively, rules specify transitions as follows: If there is an instantiation of the facts in L in the current state of the system, we can make a transition to replace the facts in L by the facts in R with the same instantiation.

Example

Consider a system state $S_n = [Msg(\text{'}hello\text{'})]$ and a rule $Msg(X) \rightarrow Msg(Y)$. Using the substitution $\sigma = \{X \mapsto \text{'}hello\text{'}, Y \mapsto \text{'}bye\text{'}\}$ we can apply the rule to get $S_{n+1} = [Msg(\text{'}bye\text{'})]$.

Rules

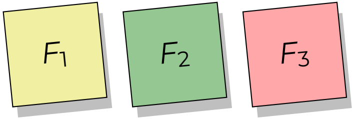
We use rule in several different ways:

- **Adversary rules** determine which messages the adversary can deduct from its knowledge set
- **Protocol** rules formalize the behavior of the model we are analyzing
- **Initialization rules** define the generation of cryptographic keys and other values
- **The FRESH rule** is a special built-in rule that generates a unique (fresh) value
  - Syntax: $[\,] \rightarrow [\,Fr(x)\,]$

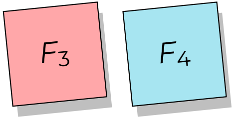
Multiset Rewriting

Multiset rewriting (informally)

- Multiset rewriting
  - **Terms** (think "messages")
  - **Facts** (think "sticky notes on the fridge")
- The state of a system is a multiset of facts
  - Initial state is the empty multiset
  - Rules specify the transition rules ("moves")
- Rules are of the form:
  - $[\,l\,] \rightarrow [\,r\,]$ (or $[\,l\,] \;\text{--}[\;a\;]\!\rightarrow [\,r\,]$)



$$[F_1, F_2] \rightarrow [F_4]$$



Multiset rewriting (formally)

- We model the states of our transition system as finite **multisets of facts**
  - We use a fixed set of fact symbols to encode the *adversary's knowledge*, *freshness information*, and the *messages* on the network
  - The remaining fact symbols are used to represent the protocol state
- We assume an unsorted signature Σ_{Fact} partitioned into **linear** and **persistent** fact symbols
- We define the set of facts as the set $\mathcal{F}$ consisting of all facts $F(t_1, .., t_k)$ such that $t_i \in \mathcal{T}$ and $\mathcal{F} \in \Sigma^k_{Fact}$

Multiset rewriting (formally)

- A labeled **multiset rewriting rule** is a triple (l, a, r) with $l, a, r \in \mathcal{F}$, denoted $l \dashv[\ a\ ]\!\rightarrow r$
- For $r_i = l \dashv[\ a\ ]\!\rightarrow r$, we define the premises as $prems(r_i) = l$, the actions as $acts(r_i) = a$, and the conclusions as $concs(r_i) = r$
- A **protocol** is a *finite set of protocol rules*
  - Our formal notion of a protocol encompasses both the rules executed by the honest participants and the adversary's capabilities, like revealing long-term keys

Transition relation

- Let R be a set of rules constructed over a given signature
- Let $\mathcal{G}^\sharp$ denote the multiset of all **ground facts**, i.e., facts built from the signature that *do not contain variables*
- Let *gri* be the function that, given a set of rules, yields the set of all ground instances of those rules
- We specify a labeled operational semantics for R (including the FRESH rule) using a labeled **transition relation** steps of the type

$$steps(R) \subseteq \mathcal{G}^\sharp \times (gri(R \cup \textsc{Fresh})) \times \mathcal{G}^\sharp$$

Transition relation

We define steps using the inference rule notation: For each instance for which the premises (above the line) hold, the conclusion (below the line) can be drawn:

$$\frac{l \;\text{--}[\; a \;]\!\rightarrow r \in_E gri(R \cup \{\textsc{Fresh}\}) \qquad lfacts(l) \subseteq^\sharp S \qquad pfacts(l) \subseteq set(S)}{S \xrightarrow{\text{set(a)}}_R ((S \setminus^\sharp lfacts(l)) \cup^\sharp r)}$$

where $lfacts(l)$ is the multiset of all linear facts in l and $pfacts(l)$ is the set of all persistent facts in l.

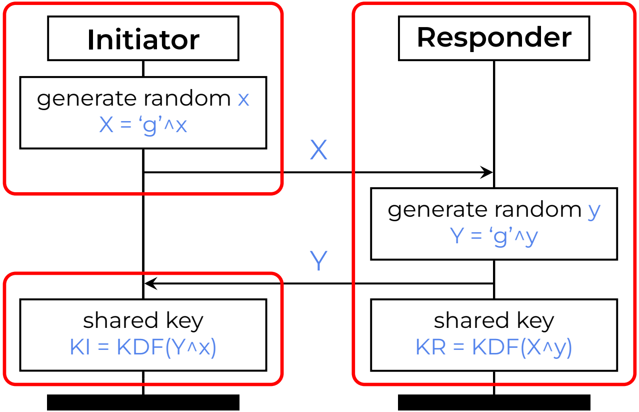
Transition relation

Informally: using a rule instance $l \;-[\; a \;]\!\!\rightarrow r$, we can make a step S to S', if

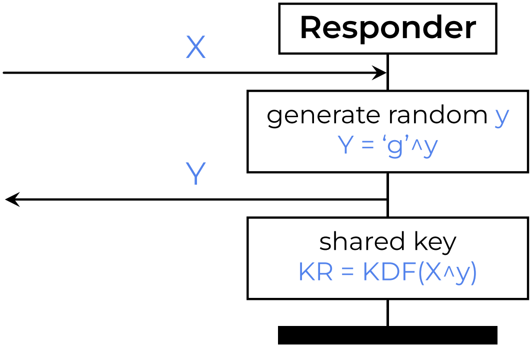
1. $l \;-[\; a \;]\!\!\rightarrow r$ is a ground instance of a rule in R or the Fresh rule,
2. S' is the result of removing the linear facts in l from S, and adding the facts in r,
3. the multiset of linear facts in l occurs in S, and
4. the set of persistent facts in l occurs in S.

Examples

Example 1: Diffie-Hellman key exchange

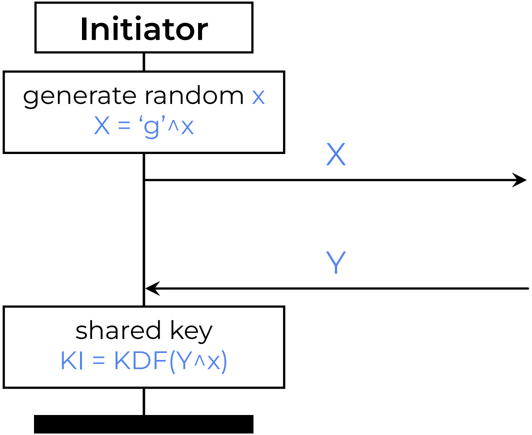


Responder model



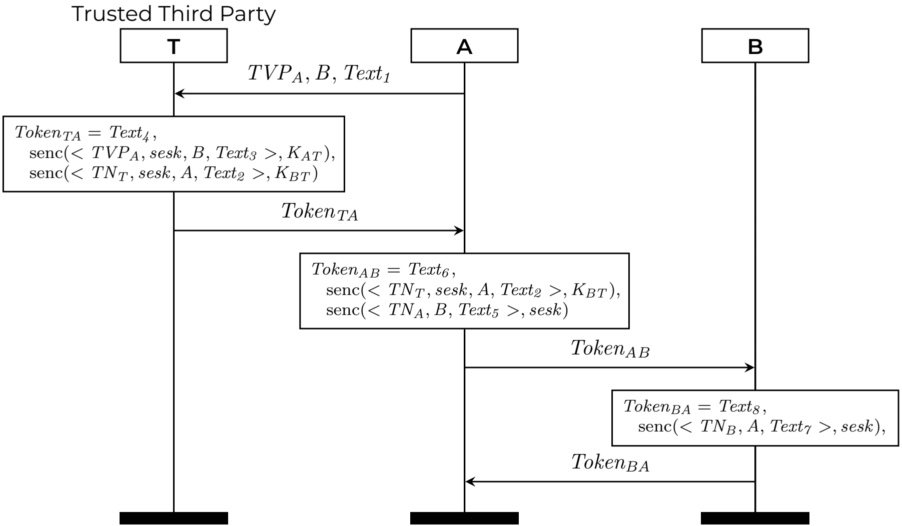

```
builtins: diffie-hellman

functions: KDF/1

/* 1. Receive X
   2. Generate random y
   3. Send Y = 'g'^y
   4. Calculate KR */
rule responder:
    let
        Y = 'g'^~y        // 3
        KR = KDF(X^y)     // 4
    in
    [ In(X)               // 1
    , Fr(~y) ]            // 2
    -->
    [ Out(Y) ]            // 3
```

Initiator model

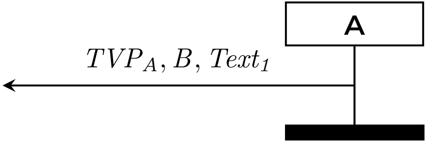



```
/* 1. Generate random x
   2. Send X = 'g'^x */
rule initiator_1:
    let
      X = 'g'^~x        // 2
    in
    [ Fr(~x)            // 1
    , Fr(~tid) ]
    -->
    [ Out(X)            // 2
    , St_Init_1(~tid, ~x) ]


/* 3. Receive Y
   4. Calculate KI */
rule initiator_2:
    let
      KI = KDF(Y^x)     // 4
    in
    [ In(Y)             // 3
    , St_Init_1(~tid, ~x) ]
    -->
    [ ]
```

Example 2: ISO/IEC

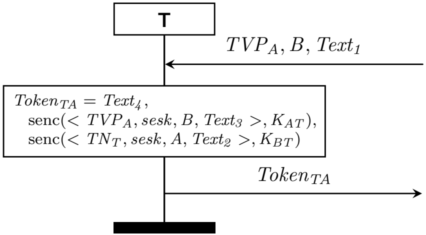


Model



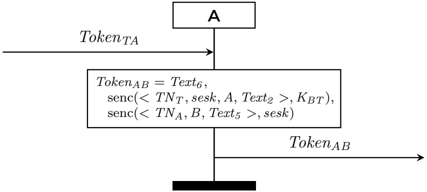

```
/* Setup shared keys between $X (variable)
   and 'T' (fixed trusted server) */
rule Setup:
    [ Fr(~kXT) ]
    -->
    [ !SharedKey($X,'T',~kXT) ]

/* A initiates the protocol with T */
rule A1:
    [ Fr(~tvpA), Fr(~text1)  ]
    -->
    [ Out(<~tvpA,$B,~text1>)
    , StA1($B,~tvpA) ]
```

Model



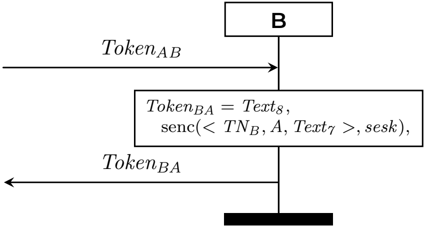

```
/* T receives message from A
   and responds to A */
rule T:
    let
      m1 = ~text4
      m2 = senc(<tvpa,~sesK,B,~text3>,kat)
      m3 = senc(<~tnT,~sesK,A,~text2>,kbt)
      tokenTA = <m1,m2,m3>
    in
    [ In(<tvpa,B,txt1>)
    , !SharedKey(A,T,kat)
    , !SharedKey(B,T,kbt)
    , Fr(~text2), Fr(~text3), Fr(~text4)
    , Fr(~sesK), Fr(~tnT) ]
    -->
    [ Out(tokenTA) ]
```

Model




```
/* A receives message from T
   and responds to B */
rule A2:
    let
      t2 = senc(<tvpA,sesk,B,text3>,kat)
      tokenTA  = <t1,t2,t3>
      m1 = ~text6
      m2 = t3
      m3 = senc(<~tnA,B,~text5>,sesk)
      tokenAB = <m1,m2,m3>
    in
    [ In(tokenTA)
    , !SharedKey(A,T,kat)
    , StA1(B,tvpA)
    , Fr(~text5), Fr(~text6), Fr(~tnA) ]
    -->
    [ Out(tokenAB)
    , StA2(A,B,~tnA,sesk) ]
```

Model




```
/* B receives message from A
   and responds to A */
rule B:
    let
      t2 = senc(<tnt,sesk,A,text2>,kbt)
      t3 = senc(<tna,B,text5>,sesk)
      tokenAB = <t1,t2,t3>
      m1 = ~text8
      m2 = senc(<~tnB,A,~text7>,sesk)
      tokenBA = <m1,m2>
    in
    [ In(tokenAB)
    , !SharedKey(B,'T',kbt)
    , Fr(~text7),Fr(~text8),Fr(~tnB) ]
    -->
    [ Out(tokenBA) ]
```

Model


```
/* A receives response from B */
rule A3:
     let
       t2 = senc(<tnb,A,text7>,sesk)
       tokenBA = <t1,t2>
     in
     [ In(tokenBA)
     , StA2(A,B,tna,sesk) ]
     -->
     [ ]
```

Summary

Next lecture

- We now know how to model..
  - ..messages as **terms**
  - ..cryptographic primitives as **equational theories**
  - ..protocol states as **facts**
  - ..protocol behavior as **multiset rewriting rules**
- We can now model protocols in a way that Tamarin understands!
- In the next lecture, we will learn about modeling **attacker behavior** and express **protocol properties**

Reading material

Recommended reading: [Bas+24, Ch. 3, 4.1.5–4.2.1], [CK14, Ch. 3]

[Bas+24] D. Basin, C. Cremers, J. Dreier, and R. Sasse. **Modeling and Analyzing Security Protocols with Tamarin: A Comprehensive Guide.** Draft v0.5. Sept. 2024.

[CK14] V. Cortier and S. Kremer. **Formal Models and Techniques for Analyzing Security Protocols: A Tutorial.** In: Found. Trends Program. Lang. 1.3 (Nov. 2014).