A gentle introduction to the Tamarin Prover

A tool for automated analysis of cryptographic protocols

D'Ambrosi Denis[1]

[1]DMIF
University of Udine

June 2023

INSTITUTO DE INFORMÁTICA UFG

UFG UNIVERSIDADE FEDERAL DE GOIÁS

Table of Contents

Introduction

> Two main "ingredients" for formal protocol verification:
>
> - A well-defined threat model
> - A (possibly automated) tool to verify that a protocol is actually safe against said model

In this presentation I will introduce the **Tamarin Prover** for the **Symbolic Model**

The Dolev Yao Model

Substitutes real world cryptographic operations with **term-algebras** supported by **equational theories**

Example 1
Symmetric encryption and decryption are modelled as

$$\mathrm{sdec}_k\ \mathrm{senc}_k = 1$$

Example 2
Asymmetric encryption and decryption are modelled as

$$\mathrm{adec}_{pr}\ \mathrm{aenc}_{pub} = \mathrm{adec}_{pub}\ \mathrm{aenc}_{pr} = 1$$

The Dolev Yao Model: perfect cryptography assumption

Only the entities that are in possess of k are able to encrypt and decrypt any message with it $\rightarrow$ **perfect cryptography assumption**
Given a message M and its image $\mathrm{senc}_k\ M$ (or, $\mathrm{aenc}_{pub}\ M$), we assume that it is impossible for an attacker who does not know k (or pr) to:

- guess or bruteforce k (or pr);
- manipulate $\mathrm{senc}_k\ M$ (or $\mathrm{aenc}_{pub}\ M$)
- infer any information about M from $\mathrm{senc}_k\ M$ (or $\mathrm{aenc}_{pub}\ M$).

The Dolev Yao Model: the attacker

We assume that an attacker can:

1. Eavesdrop any outbuond message to learn some terms
2. Modify any message substituting some terms
3. Forge new messages creating new terms
4. Drop any message

Always following the perfect cryptography assumption

The Dolev Yao Model: properties to prove

After formalizing a protocol, its security goals can be expressed as:

Trace properties
Invariants that should hold for each possible execution of the protocol

Observational equivalence properties
Properties that an attacker should not be able to distinguish between two different runs of the protocol

The Dolev Yao Model: computational limits

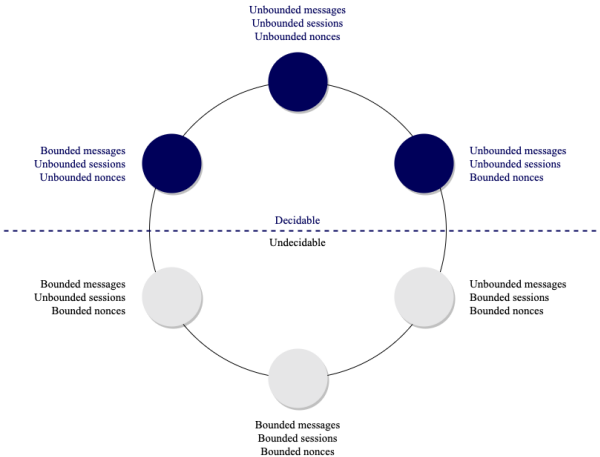


Figure 1: Decidability in symbolic verification

Term-algebra

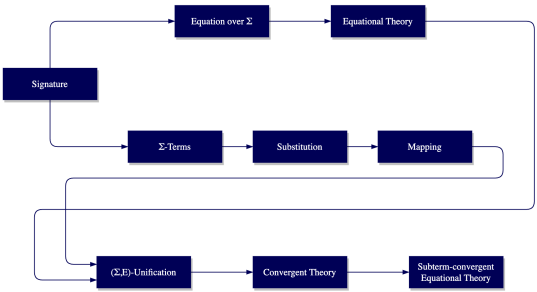


Figure 2: Graph of the definitions within this section

Subterm-convergent Equational Theories

A set of equations in the form $lhs = rhs$ such that

1. each term in lhs and rhs has a normal form
2. if a term t can be rewritten as t_1 and t_2, then it is possible to reach a fourth term t' from t_1 and t_2 in a finite amount of steps
3. rhs is either ground and in normal form or a proper subterm of lhs

Equational theories in Tamarin

AC-equational theory

Function symbols: $*/2, pair/2, fst/1, snd/1$

Equations:

1. $x * (y * z) \simeq (x * y) * z$ (associativity)
2. $x * y \simeq y * z$ (commutativity)
3. $fst(pair(x, y)) \simeq x$ (projection onto the first component)
4. $snd(pair(x, y)) \simeq y$ (projection onto the second component)

Hashing

Function symbol: $h/1$

No equations

Equational theories in Tamarin, continued

Symmetric encryption

Function symbols: $senc/2, sdec/2$

Equations:

1. $sdec(senc(m, k), k) = m$

Asymmetric encryption

Function symbols: $pk/1, aenc/2, adec/2$

Equations:

1. $adec(aenc(m, pk(k)), k) = m$

Signing

Function symbols: $sign/2, verify/3, pk/1, true/0$

Equations:

1. $verify(sign(m, sk), m, pk(sk)) = true$

Equational theories in Tamarin, continued

Diffie-Hellman

Function symbols: $inv/1, 1/0, \hat{}/2, */2$

Equations:

1. $(x^y)^z \simeq x^{(y*z)}$
2. $x^1 \simeq x$
3. $x * y \simeq y * x$
4. $(x * y) * z \simeq x * (y * z)$
5. $x * 1 \simeq x$
6. $x * inv(x) \simeq 1$

Equational theories in Tamarin, continued

Additional built-in equational theories include:

1. Revealing Signing
2. Bilinear Pairing
3. Xor
4. Multiset
5. Reliable Channel

Also possible to define custom theories:

Homomorphic (RSA) encryption

$$aenc(m1, k) * aenc(m2, k) \simeq aenc(m1 * m2, k)$$

ECDH + ECDSA

$$g^x \simeq pk(x)$$

Protocols as sets of rewriting rules

In Tamarin, the state is made up of a multiset of **Facts** (intuitively, *true predicates*)
Two types of facts:

1. **Linear facts**, which are consumed just once $\rightarrow$ useful for modeling ephemeral information:
   - messages
   - one-time-keys
   - mutable state
2. **Persistent facts**, which are persistent $\rightarrow$ useful for modeling enduring knowledge:
   - long term keys
   - identities
   - associations

Protocols as sets of rewriting rules, continued

The evolution of the state is defined through **multiset rewriting rules**:

Multiset Rewriting rule

Given a multiset $\Gamma_t = \{F_0, ..., F_n\}$ and a sequence of multisets $trace_t = \langle a_0, ..., a_{t-1} \rangle$ at a time t, we can define a rewrite rule as a triple of multisets $RR = \langle L, A, R \rangle$ (written as $RR = L - [A] \to R$) such that:

- we can apply RR to Γ_t if there is at least one ground instance (i.e. an instance with no variables) $rr = l - [a] \to r$ of RR so that $l \subseteq^{\#} \Gamma_t$
- applying rr to Γ_t yields to a new state Γ_{t+1} and an increased trace $trace_{t+1}$ obtained as
  - $\Gamma_{t+1} = \Gamma_t \setminus^{\#} lin(l) \cup^{\#} r$
  - $trace_{t+1} = \langle a_0, ..., a_{t-1}, a \rangle$

Note that each rule is generally labelled by a name N, thus can be seen as a pair (N, RR)

Protocols as sets of rewriting rules, continued

Set of possible traces

Given a set of labelled rewriting rules $P = \{(N_1, RR_1), ..., (N_m, RR_m)\}$, we define the set of possible traces generated by P as

$$traces(P) = \{\langle A_1, ..., A_n\rangle | \exists S_1, ..., S_n . \varnothing^{\#} \xrightarrow{A_1} S_1 \xrightarrow{A_2} ... \xrightarrow{A_n} S_n\}$$

where A_i is RR_i's action facts multiset. We also require that no instance of $Fresh()$ is used more than once ($\rightarrow$ no collisions)

Observable trace

Given a trace tr, we can compute its relative observable trace tr_{obs} by removing all the empty multisets from it:

$$tr_{obs} = \langle A_i | A_i \in tr \wedge A_i \neq \varnothing^{\#}\rangle$$

Dolev Yao Rules

The following set of rules formalizes the Symbolic model:

1. Term generation: `[] --[]-> [ Fr(~msg) ]`
2. Term generation by the attacker: `[ Fr(~msg) ] --[]-> [ K(~msg) ]`
3. Sending to the network: `[ Out(msg) ] --[]-> [ K(msg) ]`
4. Receiving from the network: `[ K(msg) ] --[ K(msg) ]-> [ In(msg) ]`
5. Knowledge of public names: `[] --[]-> [ K($x) ]`
6. Use of non-private functions: `[ K(x1,...,xn) ] --[]-> [ K(f(x1,...,xn)) ]`

Modelling special channels

Confidential channel

```
rule Send_Over_Confidential_Channel :
    [ ConfOut(msg) ]
    --[]->
    [ ConfIn(msg) ]

rule Attacker_Sends_Over_ConfChannel :
    [ K(msg) ]
    --[]->
    [ ConfIn(msg) ]
```

Confidential channel (with id)

```
rule Send_Over_Confidential_Channel :
    [ ConfOut(msg, channel) ]
    --[]->
    [ ConfIn(msg, channel) ]

rule Attacker_Sends_Over_ConfChannel :
    [ K(msg), K(channel) ]
    --[]->
    [ ConfIn(msg, channel) ]
```

Authenticated channel

```
rule Send_Over_Authentic_Channel :
    [ AuthOut(msg) ]
    --[ K(msg) ]->
    [ AuthIn(msg), K(msg) ]
```

Secure channel

```
rule Send_Over_Secure_Channel :
    [ SecOut(msg) ]
    --[]->
    [ SecIn(msg) ]
```

Trace Properties

Properties can be formalized as **guarded fragments of first order logic**
Properties are built upon the following atoms:

- false $\bot$
- logical operators $\neg \ \wedge \ \vee \ \implies$
- quantifiers and variables $\forall \ \exists \ a \ b \ c$
- term equality $t_1 \approx t_2$
- time point ordering and equality $i < j$ and $i = j$
- action facts at time points $F@i$

Trace Properties, continued

Correctness

Given a set of protocol rules P and a property to prove ϕ, P is correct in regards to ϕ if and only if the set of traces generated by P is a subset of the one generated by ϕ:

$$P \vDash \phi \iff traces(\phi) \subseteq traces(P)$$

Note that if $P \nvDash \phi$, all traces belonging to $traces(P)/traces(\phi)$ represent valid attacks.

Observational equivalence properties

Trace equivalence
Two different protocols P_1, P_2 are trace equivalent if and only if for each trace of P_1 exists a trace of P_2 so that the messages exchanged during the two executions are indistinguishable.

To aid termination, Tamarin allows only for the specification of *diff equivalence properties*:

Diff equivalence
Two protocols P_1, P_2 are diff-equivalent if and only if they have the same structure and differ only by the messages exchanged.
Thus, the two protocol have the same structure during execution.

Aiding termination

Tools provided to aid termination:

- source lemmas
- oracles
- interactive mode
- restrictions
- re-use lemmas

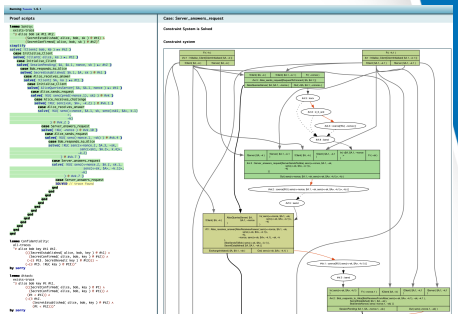


Figure 3: Tamarin's interactive mode

The Needham-Schroeder Protocol

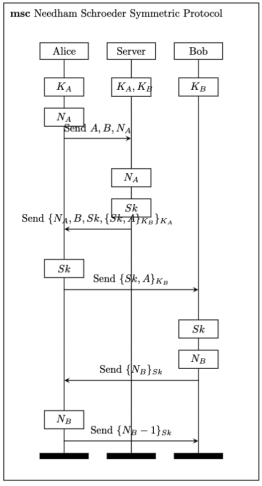
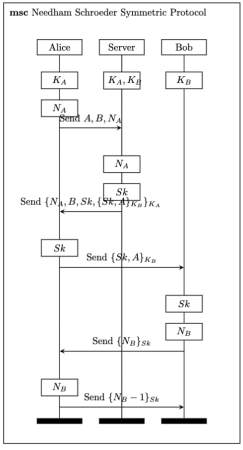


Figure 4: The Needham-Schroeder Protocol

The Needham-Schroeder Protocol, continued

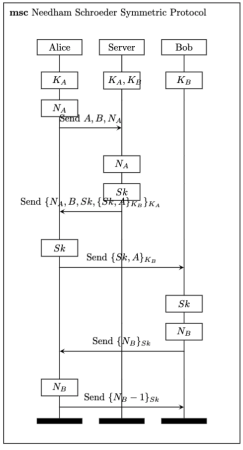


Initialisation

```
rule Initialise_Client :
    [
        Fr(~k)
    ]
    --[ ClientInitialised($A, ~k) ]->
    [
        !Client($A, ~k),
        !Server($A, ~k$)
    ]
```

```
restriction EachClientCanBeInitialisedOnce :
    "All client key1 key2 #t1 #t2 .
        ClientInitialised(client, key1) @ #t1 &
        ClientInitialised(client, key2) @ #t2
            ==> #t1 = #t2"
```

The Needham-Schroeder Protocol, continued

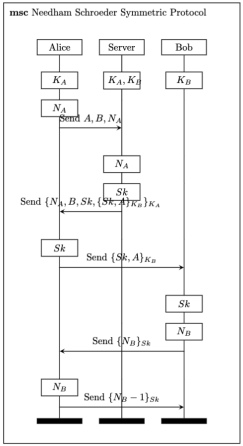


$A \rightarrow S : A, B, N_A$

```
rule Alice_sends_request :
    [
        !Client(alice, ka),
        !Client(bob, kb),
        Fr(~nonce)
    ]
    --[ RequestToConnect(alice, bob) ]->
    [
        AliceQueriesServer(alice, bob, ~nonce),
        Out(<alice, bob, ~nonce>)
    ]
```

```
restriction NoSelfCommunication:
    "All client #t .
        RequestToConnect(client, client) @ #t ==> F"
```

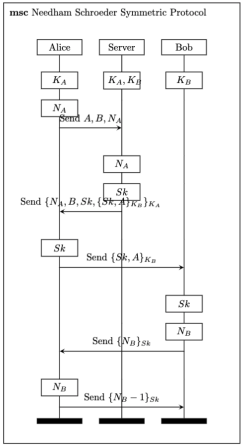
The Needham-Schroeder Protocol, continued



$$A \rightarrow B : \{Sk, A\}_{K_B}$$

```
1  rule Alice_receives_answer :
2      let
3          message_to_bob = senc(<sk1, alice>, k)
4          message_to_alice = senc(<nonce, bob, sk,
       ↪ message_to_bob>, ka)
5      in
6      [
7          !Client(alice, ka),
8          AliceQueriesServer(alice, bob, nonce),
9          In(message_to_alice)
10     ]
11     --[ AliceReceivesAnswer(message_to_alice, nonce,
       ↪ message_to_bob, sk, ka),
12         AliceSendsToBob(message_to_bob),
13         SecretEstablished(alice, bob, sk) ]->
14     [
15         ExchangeInitiated(alice, bob, sk),
16         Out(message_to_bob)
17     ]
```

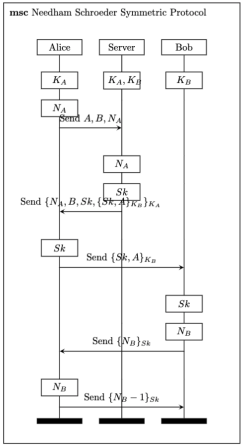
The Needham-Schroeder Protocol, continued



$B \rightarrow A : \{N_B\}_{Sk}$

```
1  rule Bob_responds_to_Alice :
2      let
3          message_from_alice = senc(<sk, alice>, kb)
4          encrypted_nonce = senc(~nonce, sk)
5      in
6      [
7          In(message_from_alice),
8          Fr(~nonce),
9          !Client(alice, ka),
10         !Client(bob, kb),
11         !Server(bob, kb)
12     ]
13     --[ BobReceivesFromAlice(message_from_alice, sk, kb),
14         SecretEstablished(bob, alice, sk),
15         BobSendsNonce(encrypted_nonce) ]->
16     [
17         SessionPending(bob, alice, ~nonce, sk),
18         Out(encrypted_nonce)
19     ]
```

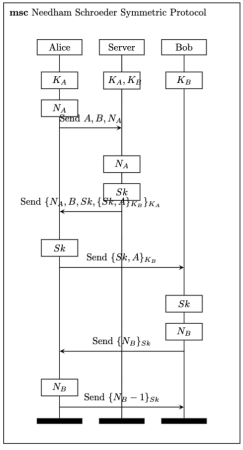
The Needham-Schroeder Protocol, continued



$A \rightarrow B : \{N_B - 1\}$

```
1  function: pred/1
2
3  rule Alice_receives_challenge :
4      let
5          nonce_received = senc(nonce, sk)
6          nonce_sent = senc(pred(nonce), sk)
7      in
8      [
9          !Client(alice, ka),
10         ExchangeInitiated(alice, bob, sk),
11         !Client(bob, kb),
12         In(nonce_received)
13     ]
14     --[ AliceReceivesNonce(nonce_received, nonce, sk) ,
15         AliceSendsNonce(nonce_sent)]->
16     [
17         !Session(alice, bob, sk),
18         Out(nonce_sent)
19     ]
```

The Needham-Schroeder Protocol, continued



Conclusion

```
1  rule Bob_receives_challenge_answer :
2      let
3          received_nonce = senc(pred(nonce), sk)
4      in
5      [
6          !Client(bob, kb),
7          !Client(alice, ka),
8          SessionPending(bob, alice, nonce, sk),
9          In(received_nonce)
10     ]
11     --[ BobReceivesNonce(received_nonce, nonce, sk),
12         SecretConfirmed(alice, bob, sk) ]->
13     [
14         !Session(bob, alice, sk)
15     ]
```

The Needham-Schroeder Protocol, continued

Aiding termination:

```
lemma types [sources] :
    "(All #t sk kb message .
        BobReceivesFromAlice(message, sk,
    ↪  kb) @ #t ==>
        (Ex #t1 . AliceSendsToBob(message
    ↪ ) @ #t1)
        | (Ex #t1 . KU(sk) @ #t1 & KU(kb)
    ↪  @ #t1))
    &
    (All #t sk ka message nonce
    ↪ message_bob .
        AliceReceivesAnswer(message,
    ↪ nonce, message_bob, sk, ka) @ #t
    ↪ ==>
        (Ex #t1 . ServerSendsToAlice(
    ↪ message) @ #t1)
        | (Ex #t1 . KU(sk) @ #t1 & KU(ka)
```

```
    &
    (All #t sk encnonce nonce .
        AliceReceivesNonce(encnonce,
    ↪ nonce, sk) @ #t ==>
        (Ex #t1 . BobSendsNonce(encnonce)
    ↪  @ #t1)
        | (Ex #t1 . KU(sk) @ #t1 & KU(
    ↪ nonce) @ #t1)
        | (Ex #t1 . KU(encnonce) @ #t1))
    &
    (All #t sk encnonce nonce .
        BobReceivesNonce(encnonce, nonce,
    ↪  sk) @ #t ==>
        (Ex #t1 . AliceSendsNonce(
    ↪ encnonce) @ #t1)
        | (Ex #t1 . KU(sk) @ #t1 & KU(
    ↪ nonce) @ #t1)
```

The Needham-Schroeder Protocol, continued

Aiding termination:

```
restriction NoSendingPrivateKeys :
    "All client key #t .
        ClientInitialised(client, key) @
    ↪ #t
        ==> not(Ex #t1 . KU(key) @ #t1)"

```

```
rule Initialise_Client :
    [
        Fr(~k)
    ]
    --[ ClientInitialised($A, ~k) ]->
    [
        !Client($A, ~k),
        !Server($A, ~k$)
    ]
```

The Needham-Schroeder Protocol, continued

Properties to prove:

Sanity checking

```
1  lemma Sanity :
2      exists-trace
3      "Ex alice bob sk #t1 #t2 .
4          SecretEstablished(alice, bob, sk)
       ↪  @ #t1 &
5          SecretConfirmed(alice, bob, sk) @
       ↪  #t2"
```

Confidentiality

```
1  lemma Confidentiality :
2      "(All alice bob key #t1 #t2 .
3          SecretEstablished(alice, bob, key
       ↪ ) @ #t1 &
4          SecretConfirmed(alice, bob, key)
       ↪ @ #t2 &
5          not(Ex #t3 . SecretReveal(key) @
       ↪ #t3)
6          ==>
7              not(Ex #t3 . KU(key) @ #t3))"
```

The Needham-Schroeder Protocol, continued

Modelling a long term key reveal:

```
rule Secret_reveal :
    [ !Session(client1, client2, sk) ]
    --[ SecretReveal(sk) ]->
    [ Out(sk) ]
```

The Denning and Sacco attack:

The attack

```
lemma Attack :
    exists-trace
    "Ex alice bob key #t #t1 .
        SecretConfirmed(alice, bob, key)
     ↪ @ #t &
        SecretConfirmed(alice, bob, key)
     ↪ @ #t1 &
        #t < #t1 &
        not(Ex #t2 .
            SecretEstablished(bob, alice,
     ↪  key) @ #t2 & #t < #t2)"
```

Example on using box

Example 3

Em uma versão da linguagem BASIC, o nome de uma variável é uma sequência de um ou dois caracteres alfanuméricos, em que letras maiúsculas e minúsculas não são distinguidas. Além disso, um nome de variável deve começar com uma letra e deve ser diferente das cinco sequências de dois caracteres reservadas para o uso de comandos. Quantos nomes diferentes de variáveis são possíveis nesta versão do BASIC?

Solução

Pela regra da soma, $V = V_1 + V_2$. Como as variáveis só podem começar com letras, temos que $V_1 = 26$. Pela regra do produto, há $26 \cdot 36 = 936$ sequências de tamanho 2 que comecem com uma letra e terminam com um caracter alfanumérico. Porém, não se deve usar 5 variáveis reservadas. Assim, $V_2 = 26 \cdot 36 - 5 = 931$. Logo, há $V = V_1 + V_2 = 26 + 931 = 957$ nomes diferentes para variáveis nesta versão do BASIC.

Example on using table

Table 1: Countries and their codes

Country Name	Code 2	Code 3
Afghanistan	AF	AFG
Aland Islands	AX	ALA
Albania	AL	ALB
Algeria	DZ	DZA

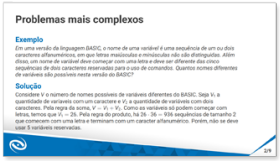
Example on using image

titlepage

mainpoint

blank

vertical



horizontal

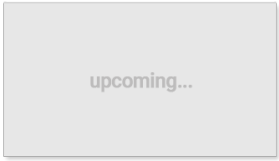


Figure 5: Template's Layouts.

Clean layout and two-column text

This is a text in first column.

$$E = mc^2$$

$$1 + 2 + \cdots + k = \frac{k \cdot (k+1)}{2}.$$

- First item
- Second item

This text will be in the second column and on a second tought this is a nice looking layout in some cases.

1. First
2. Second

Sample frame title

In this slide, some important text will be **highlighted** because it’s important. Please, don’t abuse it.

Remark
Sample text

Important theorem
Sample text in alert box

Examples 1
Sample text in green box. The title of the block is “Examples".

Preliminary Empirical Study

Sample frame title

This is a text in second frame. For the sake of showing an example.

- Text visible on slide 1

Sample frame title

This is a text in second frame. For the sake of showing an example.

- Text visible on slide 1
- Text visible on slide 2
  - text subitem

Sample frame title

This is a text in second frame. For the sake of showing an example.

- Text visible on slide 1
- Text visible on slide 2
  - text subitem
- Text visible on slides 3

Sample frame title

This is a text in second frame. For the sake of showing an example.

- Text visible on slide 1
- Text visible on slide 2
  - text subitem
- Text visible on slide 4

Thanks

Doubts and Suggestions

altinodantas@gmail.com or deuslirio.junior@gmail.com

A gentle introduction to the Tamarin Prover

A tool for automated analysis of cryptographic protocols

D'Ambrosi Denis[1]

[1]DMIF
University of Udine

June 2023

INSTITUTO DE INFORMÁTICA
UFG

UFG
UNIVERSIDADE FEDERAL DE GOIÁS