Exercise: Protocol Verification Using FDR

Software Security

Steffen Helke
Chair of Software Engineering

21st January 2019

Brandenburgische Technische Universität Cottbus - Senftenberg

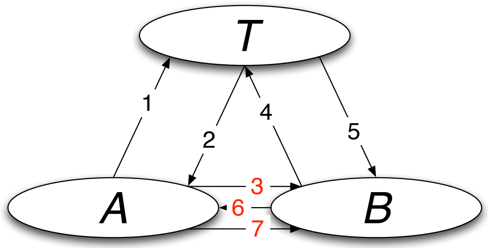
Objectives of today's lecture

- ➜ Understanding a *CSP formalizations for the Needham Schroeder protocol*
- ➜ Repetition: Getting to know how to *model messages* of a security protocol using CSP
- ➜ Being able to *prove important security properties* on a given protocol using the model checker FDR

Formal Verification of the Needham-Schroeder Protocol

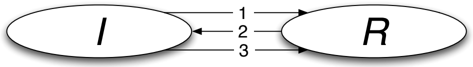
Repetition: Protocol Steps of the NSP

1 $A \rightarrow T : \{A, B\}$
2 $T \rightarrow A : \{B, PK_B\}_{SK_T}$
3 $A \rightarrow B : \{A, N_A\}_{PK_B}$
4 $B \rightarrow T : \{B, A\}$
5 $T \rightarrow B : \{A, PK_A\}_{SK_T}$
6 $B \rightarrow A : \{N_A, N_B\}_{PK_A}$
7 $A \rightarrow B : \{N_B\}_{PK_B}$



Attack for the Simplified Protocol Variant

Simplified NSP Version without using T



Attack Scenario

1.1 $A \rightarrow C : \{N_A, A\}_{PK(C)}$

2.1 $C(A) \rightarrow B : \{N_A, A\}_{PK(B)}$

2.2 $B \rightarrow C(A) : \{N_A, N_B\}_{PK(A)}$

1.2 $C \rightarrow A : \{N_A, N_B\}_{PK(A)}$

1.3 $A \rightarrow C : \{N_B\}_{PK(C)}$

2.3 $C(A) \rightarrow B : \{N_B\}_{PK(B)}$

How to code NSP using CSP?

1. Model the roles *Initiator* and *Responder* using generic CSP processes and run these processes in parallel
2. Define concrete participants, e.g. A, B and C who can play any of these roles
3. Describe each protocol step using a CSP event
4. Communicate all messages via appropriate CSP channels

Enrichment of Protocol Messages

Which participants are related to a message?

Extend protocol messages in such a way that information about the sender and receiver is also transferred

1.1 $A \rightarrow C : A.C.\{N_A, A\}_{PK(C)}$

2.1 $C(A) \rightarrow B : A.B.\{N_A, A\}_{PK(B)}$

2.2 $B \rightarrow C(A) : B.A.\{N_A, N_B\}_{PK(A)}$

1.2 $C \rightarrow A : C.A.\{N_A, N_B\}_{PK(A)}$

1.3 $A \rightarrow C : A.C.\{N_B\}_{PK(C)}$

2.3 $C(A) \rightarrow B : A.B.\{N_B\}_{PK(B)}$

How to formalize the three different message types for NSP?

$$
\begin{aligned}
MSG1 = {} & \{Msg_1.a.b.Encrypt_1.k.n_a.a' \mid \\
& \quad a, a' \in Initiator, b \in Responder, k \in Key, n_a \in Nonces\} \\
MSG2 = {} & \{Msg_2.b.a.Encrypt_2.k.n_a.n_b \mid \\
& \quad a \in Initiator, b \in Responder, k \in Key, n_a, n_b \in Nonces\} \\
MSG3 = {} & \{Msg_3.a.b.Encrypt_3.k.n_b \mid \\
& \quad a \in Initiator, b \in Responder, k \in Key, n_b \in Nonces\}
\end{aligned}
$$

$$MSGs = MSG1 \cup MSG2 \cup MSG3$$

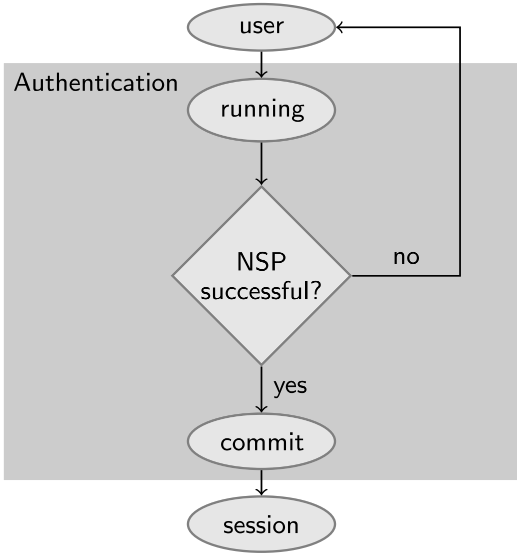
How to code the messages using CSPm?

```
datatype KEY     = ka     | kb     | kc
datatype AKTEUR  = A      |  B     | C
datatype NONCE   = NonceA | NonceB | NonceC
datatype TICKET1 = Encrypt1.KEY.NONCE.AKTEUR
datatype TICKET2 = Encrypt2.KEY.NONCE.NONCE
datatype TICKET3 = Encrypt3.KEY.NONCE

datatype MSG = Msg1.AKTEUR.AKTEUR.TICKET1
  | Msg2.AKTEUR.AKTEUR.TICKET2
  | Msg3.AKTEUR.AKTEUR.TICKET3

channel comm : MSG
```

Question: How many different events can be communicated via the channel *comm*?

➜ This channel accepts $3^5 + 3^5 + 3^4 = 567$ different events

How to observe the current state of the Protocol?

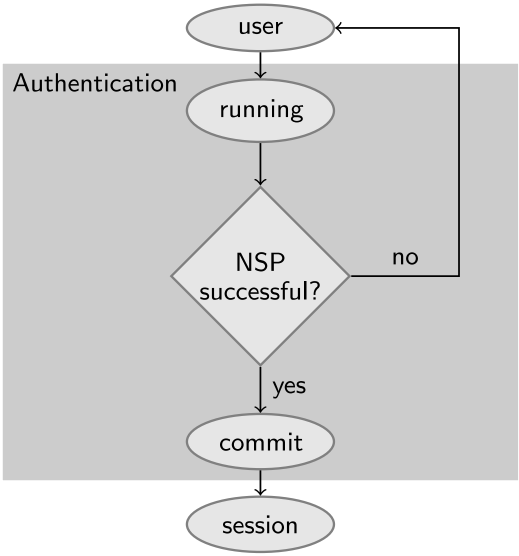


channel
user,
session,
I_running,
R_running,
I_commit,
R_commit :
Initiator.Responder

Example 1

Event *I_running.A.B* indicates that participant *A* starts a NSP as initiator with participant *B* as responder.

How to observe the current state of the Protocol?



Example 2

Event *R_running.A.B* indicates that participant *B* starts a NSP as responder with participant *A* as initiator.

Example 3

Event *I_commit.A.B* indicates that participant *A* as initiator is convinced that he/she communicates with participant *B* as a responder.

Initiator Process

$$
\begin{aligned}
& INITIATOR(a, n_a) = \\
& \quad user!a?b \rightarrow I\_running.a.b \rightarrow \\
& \quad comm.Msg_1.a.b.Encrypt_1.key(b)!n_a.a \rightarrow \\
& \quad comm.Msg_2.b.a.Encrypt_2.key(a)?n'_a.n_b \rightarrow \\
& \quad \textbf{if } n_a = n'_a \\
& \quad \textbf{then } comm.Msg_3.a.b.Encrypt_3.key(b)!n_b \rightarrow \\
& \quad \quad I\_commit.a.b \rightarrow session.a.b \rightarrow Skip \\
& \quad \textbf{else } Stop
\end{aligned}
$$

Responder Process

$$RESPONDER(b, n_b) =$$

$$\begin{array}{l} user?a!b \rightarrow R\_running.a.b \rightarrow \\ comm.Msg_1.a.b.Encrypt_1.key(b)?n_a.a \rightarrow \\ comm.Msg_2.b.a.Encrypt_2.key(a)!n_a.n_b \rightarrow \\ comm.Msg_3.a.b.Encrypt_3.key(b)?n'_b \rightarrow \\ \textbf{if } n_b = n'_b \\ \textbf{then } R\_commit.a.b \rightarrow session.a.b \rightarrow Skip \\ \textbf{else } Stop \end{array}$$

Initiator and responder synchronization is based on the event set S

$$S = \{| \; comm, session.A.B \; |\}$$

How to synchronize the communication between initiator and responder?

Parallel composition of initiator and responder based on

$$\boxed{Initiator(A, N_a) \; \{| \; comm, session.A.B \; |\} \; Responder(B, N_b)}$$

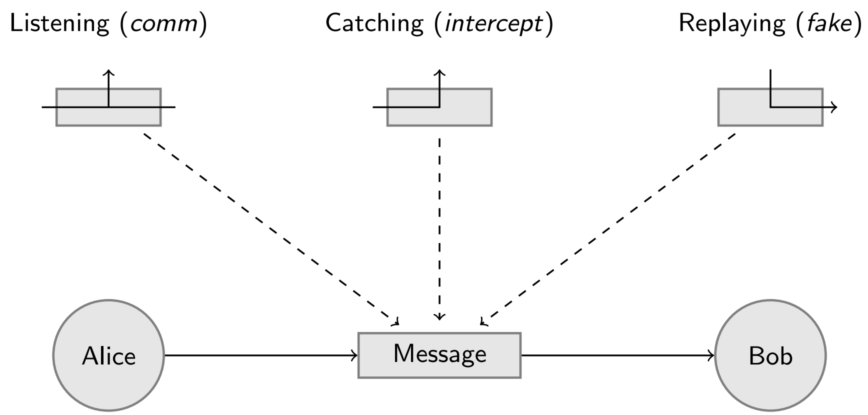
$$INITIATOR(a, n_a) =$$

$$\begin{array}{l} user!a?b \rightarrow \\ I\_running.a.b \rightarrow \\ comm.Msg_1.a.b.Encrypt_1.key(b)!n_a.a \rightarrow \\ comm.Msg_2.b.a.Encrypt_2.key(a)?n'_a.n_b \rightarrow \\ \textbf{if } n_a = n'_a \\ \textbf{then } comm.Msg_3.a.b.Encrypt_3.key(b)!n_b \rightarrow \\ \quad I\_commit.a.b \rightarrow \\ session.a.b \rightarrow Skip \\ \textbf{else } Stop \end{array}$$

$$RESPONDER(b, n_b) =$$

$$\begin{array}{l} user?a!b \rightarrow \\ R\_running.a.b \rightarrow \\ comm.Msg_1.a.b.Encrypt_1.key(b)?n_a.a \rightarrow \\ comm.Msg_2.b.a.Encrypt_2.key(a)!n_a.n_b \rightarrow \\ \\ comm.Msg_3.a.b.Encrypt_3.key(b)?n'_b \rightarrow \\ \textbf{if } n_b = n'_b \\ \textbf{then } R\_commit.a.b \rightarrow \\ session.a.b \rightarrow Skip \\ \textbf{else } Stop \end{array}$$

How to model attacker channels?



channel *comm*, *fake*, *intercept* : *MSGs*

How do I rename the channels of the initiator process to obtain a suitable attacker interface?

$$\begin{array}{l} INITIATOR1 = \\ \quad INITIATOR(A, N_a) \\ \qquad [[\; comm.Msg_1 \leftarrow comm.Msg_1, \\ \qquad\;\; comm.Msg_1 \leftarrow intercept.Msg_1, \\ \qquad\;\; comm.Msg_2 \leftarrow comm.Msg_2, \\ \qquad\;\; comm.Msg_2 \leftarrow fake.Msg_2, \\ \qquad\;\; comm.Msg_3 \leftarrow comm.Msg_3, \\ \qquad\;\; comm.Msg_3 \leftarrow intercept.Msg_3 \;]] \end{array}$$

How do I rename the channels of the responder process to obtain a suitable attacker interface?

$$
\begin{aligned}
& RESPONDER1 = \\
& \quad RESPONDER(B, N_b) \\
& \qquad [[\ comm.Msg_1 \leftarrow comm.Msg_1, \\
& \qquad\quad comm.Msg_1 \leftarrow fake.Msg_1, \\
& \qquad\quad comm.Msg_2 \leftarrow comm.Msg_2, \\
& \qquad\quad comm.Msg_2 \leftarrow intercept.Msg_2, \\
& \qquad\quad comm.Msg_3 \leftarrow comm.Msg_3, \\
& \qquad\quad comm.Msg_3 \leftarrow fake.Msg_3\ ]]
\end{aligned}
$$

What could an attacker do in principle?

1. He/she is able to listen to and/or intercept messages
2. He/she is able to learn nonces
3. He/she is able to send new messages using the learned nonces
4. He/she is able to replay old messages (possibly modified)
5. It is also possible to replay old encrypted messages that the attacker cannot decrypt

Note, the formalization of such an attacker behaviour is also called *Dolev-Yao model* based on a research paper from 1983[1]

[1] *D. Dolev and A. Yao: On the security of public key protocols, IEEE Journal Transactions on Information Theory, 29/2, 1983.*

Attacker Process (1)

$$
\begin{aligned}
& INTRUDER(m1s, m2s, m3s, ns) = \\
& \quad comm.Msg_1?a.b.Encrypt_1.k.n.a' \rightarrow \\
& \qquad \textbf{if } k = K_I \textbf{ then } INTRUDER(m1s, m2s, m3s, ns \cup \{n\}) \\
& \qquad \textbf{else } INTRUDER(m1s \cup \{Encrypt_1.k.n.a'\}, m2s, m3s, ns) \\
& \square\ intercept.Msg_1?a.b.Encrypt_1.k.n.a' \rightarrow \\
& \qquad \textbf{if } k = K_I \textbf{ then } INTRUDER(m1s, m2s, m3s, ns \cup \{n\}) \\
& \qquad \textbf{else } INTRUDER(m1s \cup \{Encrypt_1.k.n.a'\}, m2s, m3s, ns) \\
& \square\ comm.Msg_2?b.a.Encrypt_2.k.n.n' \rightarrow \\
& \qquad \textbf{if } k = K_I \textbf{ then } INTRUDER(m1s, m2s, m3s, ns \cup \{n, n'\}) \\
& \qquad \textbf{else } INTRUDER(m1s, m2s \cup \{Encrypt_2.k.n.n'\}, m3s, ns) \\
& \square\ intercept.Msg_2?b.a.Encrypt_2.k.n.n' \rightarrow \\
& \qquad \textbf{if } k = K_I \textbf{ then } INTRUDER(m1s, m2s, m3s, ns \cup \{n, n'\}) \\
& \qquad \textbf{else } INTRUDER(m1s, m2s \cup \{Encrypt_2.k.n.n'\}, m3s, ns)
\end{aligned}
$$

Attacker Process (2)

$$
\begin{aligned}
& INTRUDER(m1s, m2s, m3s, ns) = \\
& \square\ comm.Msg_3?a.b.Encrypt_3.k.n \rightarrow \\
& \qquad \textbf{if } k = K_I \textbf{ then } I(m1s, m2s, m3s, ns \cup \{n\}) \\
& \qquad \textbf{else } I(m1s, m2s, ms3 \cup \{Encrypt_3.k.n\}, ns) \\
& \square\ intercept.Msg_3?a.b.Encrypt_3.k.n \rightarrow \\
& \qquad \textbf{if } k = K_I \textbf{ then } I(m1s, m2s, m3s, ns \cup \{n\}) \\
& \qquad \textbf{else } I(m1s, m2s, ms3 \cup \{Encrypt_3.k.n\}, ns) \\
& \square\ fake.Msg_1?a.b?m{:}m1s \rightarrow I(m1s, m2s, m3s, ns) \\
& \square\ fake.Msg_2?b.a?m{:}m2s \rightarrow I(m1s, m2s, m3s, ns) \\
& \square\ fake.Msg_3?a.b?m{:}m3s \rightarrow I(m1s, m2s, m3s, ns) \\
& \square\ fake.Msg_1?a.b!Encrypt_1?k?n{:}ns?a' \rightarrow I(m1s, m2s, m3s, ns) \\
& \square\ fake.Msg_2?b.a!Encrypt_2?k?n{:}ns?n'{:}ns \rightarrow I(m1s, m2s, m3s, ns) \\
& \square\ fake.Msg_3?a.b!Encrypt_3?k?n{:}ns \rightarrow I(m1s, m2s, m3s, ns)
\end{aligned}
$$

Note: The identifier *INTRUDER* is abbreviated here in the recursive call by *I*

How to construct a complete system process including the capabilities of an attacker?

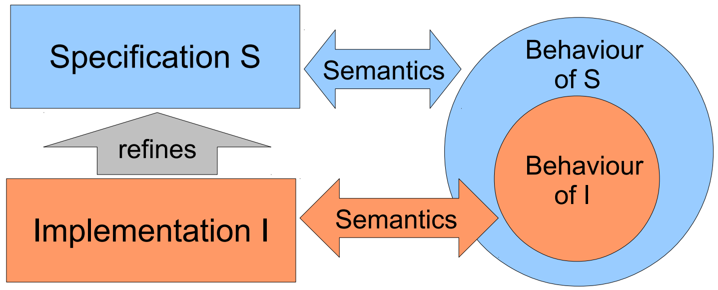
$$\mathit{AGENTS} = \\ \quad \mathit{INITIATOR1} \,[\![\, \{\!|\, \mathit{comm}, \mathit{session.A.B} \,|\!\} \,]\!]\, \mathit{RESPONDER1}$$

$$\mathit{INTRUDER1} = \mathit{INTRUDER}(\varnothing, \varnothing, \varnothing, \{N_C\})$$

$$\mathit{SYSTEM} = \\ \quad \mathit{AGENTS} \,[\![\, \{\!|\, \mathit{fake}, \mathit{comm}, \mathit{intercept} \,|\!\} \,]\!]\, \mathit{INTRUDER1}$$

Conformance by Refinement

- *Abstract specification* represents an important security property of our protocol, e.g. correct authentication
  ➜ CSP processes *AI* & *AR* (next slides)
- *Concrete implementation* represents both the behavior of the NSP and also the possible behavior of the attacker
  ➜ CSP process *SYSTEM* (last slide)



Specification for a Correct Authentication of the Initiator

$$AI_0 = \quad I\_running.A.B \to R\_commit.A.B \to AI_0$$

$$AI = \quad AI_0 \,|||\, RUN(\Sigma \setminus A_2)$$

where $A_2 = \{\!|\, I\_running.A.B, R\_commit.A.B \,|\!\}$,
$\Sigma \mathrel{\widehat{=}}$ complete communication alphabet

and $RUN(M) \mathrel{\widehat{=}}$ infinite process that communicates the events of M in an arbitrary order

Illustration

$AI =$ arbitrary other events $\to I\_running.A.B \to$
arbitrary other events $\to R\_commit.A.B \to$
arbitrary other events $\to AI$

Specification for a Correct Authentication of the Responder

$$AR_0 = \quad R\_running.A.B \to I\_commit.A.B \to AR_0$$

$$AR = \quad AR_0 \,|||\, RUN(\Sigma \setminus A_1)$$

where $A_1 = \{\!|\, R\_running.A.B, I\_commit.A.B \,|\!\}$
$\Sigma \mathrel{\widehat{=}}$ complete communication alphabet

and $RUN(M) \mathrel{\widehat{=}}$ infinite process that communicates the events of M in an arbitrary order

Illustration

$AR =$ arbitrary other events $\to R\_running.A.B \to$
arbitrary other events $\to I\_commit.A.B \to$
arbitrary other events $\to AI$

Proof of Correctness by Refinement

Tool Support

Automatic verification by the refinement checker FDR

Proof Obligations

$traces(SYSTEM) \subseteq traces(AR)$

damit gilt $AR \sqsubseteq_T SYSTEM$

$traces(SYSTEM) \not\subseteq traces(AI)$

damit gilt $AI \not\sqsubseteq_T SYSTEM$

Counterexample: Intruder Attack Scenario

Trace of the model checker

$\langle user.A.B,\ user.A.C,\ I\_running.A.C,$

$intercept.Msg_1.A.C.Encrypt_1.K_c.N_a.A,$ (1.1)

$R\_running.A.B,$

$fake.Msg_1.A.B.Encrypt_1.K_b.N_a.A,$ (2.1)

$intercept.Msg_2.B.A.Encrypt_2.K_a.N_a.N_b,$ (2.2)

$fake.Msg_2.C.A.Encrypt_2.K_a.N_a.N_b,$ (1.2)

$intercept.Msg_3.A.C.Encrypt_3.K_c.N_b,$ (1.3)

$fake.Msg_3.A.B.Encrypt_3.K_b.N_b,$ (2.3)

$R\_commit.A.B\rangle$

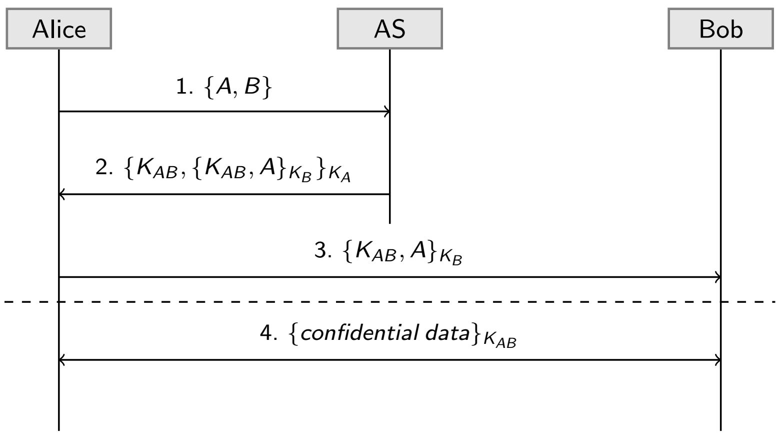
What is the cause of this counterexample?

$R\_commit.A.B$ occurs without a previous $I\_running.A.B$!

CSP/FDR Exercises

Task

➜ Model and verify the naive NSP (symmetric variant) using CSP/FDR!



How to use FDR?

1. If you plan to install FDR on your own computer, please follow the instructions at
   `http://www.cs.ox.ac.uk/projects/fdr/`
   Note the newest distribution ist FDR4
2. If you don't like to install FDR on your own machine, you can also use the installation on the computers of the pool room (use linux CentOS). Note that fdr3 is installed under the path
   /home/helke/tools/fdr/bin
   By invoking *fdr3* you start the GUI variant of the program
3. It is also possible to use FDR without GUI (only in batch-mode). To do this, you must call *refines*.

References

- Gavin Lowe: An Attack on the Needham-Schroeder Public-Key Authentication Protocol, Information Processing Letters, 1995.
- Gavin Lowe: Breaking and Fixing the Needham-Schroeder Public-Key Protocol using FDR,Tools and Algorithms for the Construction and Analysis of Systems, Springer Verlag, pages 147-166, 1996.
- C. A. R. Hoare: Communicating Sequential Processes. http://www.usingcsp.com/, Prentice Hall International Series in Computer Science, 1985.