Cryptographic Protocol Verification via Supercompilation (A Case Study)

Abdulbasit Ahmed
The University of Liverpool

Alexei P. Lisitsa
The University of Liverpool

Andrei P. Nemytykh
Program Systems Institute of Russian Academy of Sciences

VPT'13, 13 July 2013, Saint Petersburg, Russia

Outline

– Functional Modeling of Nondeterministic Computing Systems ↓

– Verification via Supercompilation ↓

– Prorgam Transformation for Program Verification (Related Works) ↓

– Our Contribution ↓

– The Needham-Schroeder Public Key Authentication Crypto-protocol ↓

– The Presentation Language ↓

– The Principles of Modeling ↓

– The Program Model of NSPK ↓

– The Intruder Behavior Logic ↓

– A Verification Attempt ↓

– G. Lowe's Corrected Version of NSPK ↓

– Discussion ↓

General Method for Functional Modeling of Non-deterministic Computing Systems (A.Lisitsa, A.Nemytykh, 2005)

- **Let S be a parameterized system (e.g., a protocol) and P be a safety property of S to verify;**

- **Write a program φ_S simulating execution of S for n steps, where n is an input parameter;**

- **If S is non-deterministic then let n be a string of characters labelling possible choices at the branching points;**

- **Given the value of n φ_s returns the state of the system S after execution n steps following the choices, provided by n.**

Outline

- Functional Modeling of Nondeterministic Computing Systems ↪
- **Verification via Supercompilation**
  - **Supercompilation**
  - **Parameterized Testing**
  - **Formal Models**
- Prorgam Transformation for Program Verification (Related Works) ↓
- Our Contribution ↓
- The Needham-Schroeder Public Key Authentication Crypto-protocol ↓
- The Presentation Language ↓
- The Principles of Modeling ↓
- The Program Model of NSPK ↓
- The Intruder Behavior Logic ↓
- A Verification Attempt ↓
- G. Lowe's Corrected Version of NSPK ↓
- Discussion ↓

Supercompilation

– *Super*vised *Compilation*.

– Semantic based program transformation technique (V.Turchin, 1960-70s).

– Can be used for optimization and specialization of (functional) programs.

– Much of the development has been done in the context of the Refal functional programming language.

– SCP4 is the most advanced implementation of supercompilation for Refal (A.Nemytykh, V.Turchin).

Supercompiler

– observes the behaviour of a functional program P running on partially defined input;

– unfolds a potentially infinite tree of all possible computations of P;

– reduces redundancy, e.g. by pruning *unreachable* fragments of code;

– folds the tree into a finite graph of parameterised configurations of P and transitions between them;

– basing on a graph of configurations constructs a new program, which is (almost) equivalent to the input program.

Resulting program defines a function which is an extention of the input function.

Verification via Supercompilation

- **V.Turchin (1986):**
  ". . . Proving the correctness of a program is theorem proving, so a supercompiler can be relevant. For example, if we want to check that the output of a function $F(x)$ ***always has the property*** $P(x)$***, we can try to transform the function*** $P(F(x))$ ***into an identical*** T ***. . ."***

- **The idea has not been tried for a long time for the problems interesting for verification community.**

Parameterised Testing

General technique for verification of parameterised systems (A.Nemytykh, A.Lisitsa, 2005)

- **Let S be a parameterized system (a protocol) and P be a safety property of S to verify.**

- **Write a program φ_S simulating execution of S for n steps, where n is an input parameter.**

- **If S is non-deterministic then let n be a string of characters labelling possible choices at the branching points.**

- **Given the value of n φ_s returns the state of the system S after execution n steps following the choices, provided by n.**

Parameterised Testing - II

- Let T_P be a testing program, which given a state s of S returns the result of testing the property P on s (*True* or *False*).

- Consider composition $T_P \circ \varphi_S$. It first simulates the execution of the system and then tests the property required.

"P holds in any possible state reachable by the execution of the system S from an initial state"

$\Leftrightarrow$

"the program $T_P(\varphi(n))$ never returns the value *False*, no matter what values are given to the input parameter".

Practical Implementation

- **Refal is used to implement $T_P(\varphi(n))$.**

- **The SCP4 supercompiler is used to transform $T_P(\varphi(n))$ to a form from which one can easily establish required property by the following syntactical check:**
  - **If resulting program does not contain the operator "return False;" ?**

This Approach Has Shown to Be Efficient

for the verification of various (classes of) parameterised and infinite-state protocols and systems:

- **Cache Coherence Snooping Protocols (Synopsis N+1, MSI, MESI, MOESI, Illinois, Berkley, Dragon, ...);**
- **Cache Coherence Directory Protocols (Steve German's server-client protocol);**
- **Java MetaLock Algorithm;**
- **Load Balancing Algorithm;**
- **Coverablity for Petri Nets;**
- **and others http://refal.botik.ru/protocols/**

Formal Models

- **A simplified theoretical model of the verification via supercompilation approach:**
  - **A.Nemytykh, A.Lisitsa, IJFCS Vol. 19, No. 04, 2008,**
    http://www.worldscientific.com/toc/ijfcs/19/04

- **The completeness of the method for the verification of coverability for Petri Nets:**
  - **A.Nemytykh, A.Lisitsa, RP'08, 2008,**
    http://www.csc.liv.ac.uk/~rp2008/programme.php
  - **A.Klimov, LNCS Vol. 7162, 2012,**
    http://rd.springer.com/book/10.1007/978-3-642-29709-0/page/1

Related Works

The history of verification of cryptographic protocols spans more than twenty years.

- **Dolev and Yao (1983) were the first to address the verification of cryptographic protocols by utilizing a formal model of the protocols and the environment.**

- **M. Leuschel et al. (1999,2000, ...) were the pioneers who suggested to apply a program specialization method for verification of various infinite state computing systems. The systems were modeled in terms of logic programs.**

Related Works-II

- A. Pettorossi, M. Proietti et al. (2001, ...) proposed to use constraint logic programs, which give more powerful means for dealing with infinite sets of states.

- A. Roychoudhury and C.R. Ramakrishnan in 2004 used fold/unfold transformations of logic programs for the verification of parameterized concurrent systems.

- G. W. Hamilton in 2007 described using of his distillation algorithm as a proof assistant for transformation of programs into a tail recursive form in which some properties of the programs can be easily verified by the application of inductive proof rules.

Related Works-III

– **A. Lisitsa and A. Nemytykh (2005, ...)** studied functional modeling and verification (by supercompilation) of global *safety* properties of nondeterministic parameterized cache coherence protocols.

– **A. Lisitsa and A. Nemytykh (2008) and A. Klimov (2012)** applied supercompilation to verification of Petri Nets models.

– **A. Ahmed (2008)** addressed the verification of the cryptographic protocols via supercompilation using the functional modeling of a variant of the Dolev-Yao model.

– **Antonina Nepeivoda (2013)** considered modeling and verifying of the ping-pong crypto-protocols. Verification of her program models uses generalization based on Turchin's relation (1988).

Our Contribution

We show how to extend this approach to the modeling and verification of cryptographic protocols.

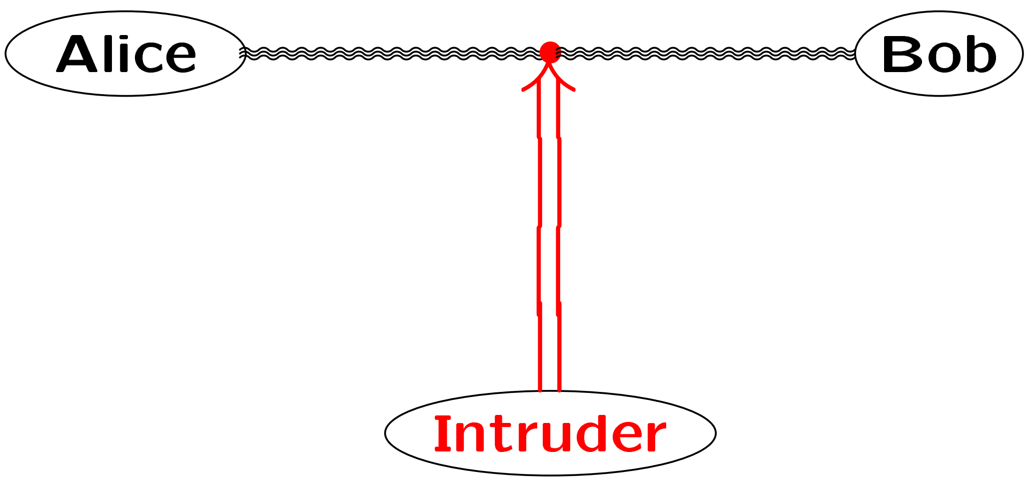
Our case study is
the Needham-Schroeder Public Key authentication Crypto-protocol:

- **the original version described by R. Needham and M. Schroeder in 1978;**

- **a modified (corrected) version given by G. Lowe in 1995.**

Our Contribution

R. Needham, M. Schroeder, 1978



The Main Novelty

– Complete parameterization of an intruder behavior.

Complete Parameterization of the Intruder Behavior vs. the Dolev Yao Model

– **This very conservative definition of tha attack does not lead to the generation of the spurious attacks on NSPK protocol. This surprising property of NSPK in the context of our modeling is interesting itself.**

– **The work of A. Ahmed addressed the verification of the cryptographic protocols via supercompilation using the functional modeling of a variant of the Dolev-Yao model.**

Outline

– Functional Modeling of Nondeterministic Computing Systems ↪

– Verification via Supercompilation ↪

– Prorgam Transformation for Program Verification (Related Works) ↪

– Our Contribution ↪

– **The Needham-Schroeder Public Key Authentication Crypto-protocol**

– The Presentation Language ↓

– The Principles of Modeling ↓

– The Program Model of NSPK ↓

– The Intruder Behavior Logic ↓

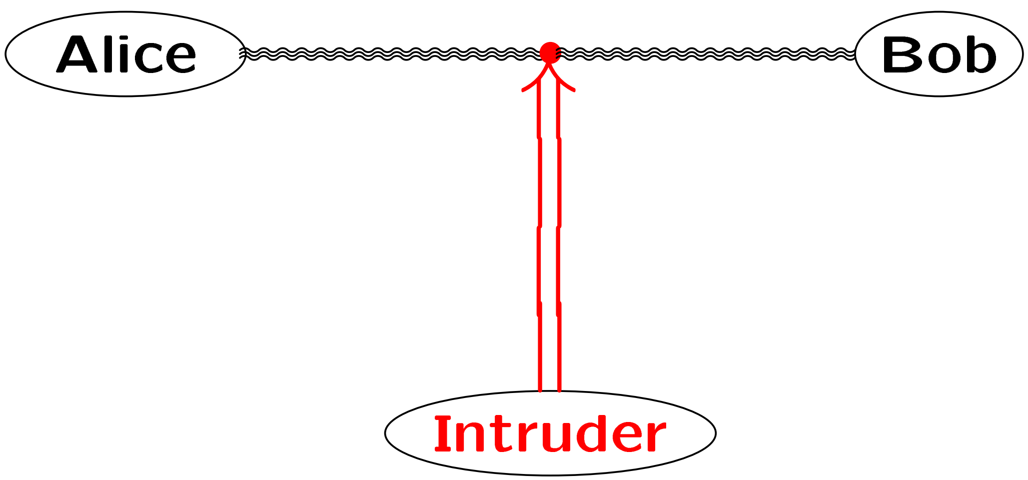
– A Verification Attempt ↓

– G. Lowe's Corrected Version of NSPK ↓

– Discussion ↓

The Needham-Schroeder Public Key Authentication Protocol (NSPK)

R. Needham, M. Schroeder, 1978



The Needham-Schroeder Public Key Authentication Protocol (NSPK)

R. Needham, M. Schroeder, 1978

– Two participants Alice (A) and Bob (B) who aim to authenticate each other by sending encrypted and signed messages via an open channel.

– The authentication is required even in the presence of an intruder (C). He is aware of the communication rules and tries to convince B that he (C) is A, observing the messages moving in the channel and sometimes replacing them.

NSPK-II

- **Every participant (A,B,C) has a pair of private and public keys assigned to him/her.**

- **Everyone can use a public key of anyone else (EA,EB,EC) to encrypt a message.**

- **Only the holder of the corresponding private key may decrypt such an encrypted message.**

- **The participants use random unique nonces rA, rB, rC. The nonces depend on the sessions. The probability of guessing the nonces is zero.**

Secure NSPK Evaluation

1. **A $\rightarrow$ B : EB(rA,A) - A initiates a communication session and sends a nonce rA signed with A and encrypted with the public key EB.**

2. **B $\rightarrow$ A : EA(rA,rB)**

3. **A $\rightarrow$ B : EB(rB)**

4. **After three messages exchange above participants B and A are assured that they communicate with each other.**

Any other messages received by the legal participants cause concern of an unauthorized access and interrupting the session.

Parameterization

- unknown of the messages of the legal participants;
- the messages generated by the intruder (C);
- an unknown number of the messages;
- the set $\mathcal{K}$ of all public keys.

We assume that A may initiate an exectution of the protocol by sending the first request to any participant whose public key belongs to $\mathcal{K}$.

The Verification Objective

is to prove that there exist no attacks on the protocol or otherwise to construct such an attack.

Definition 1 *An attack is a session of the protocol has successfully compeleted but at the same time an intruder took a part in the session.*

This very conservative definition of the attack does not lead to the generation of the spurious attacks on NSPK protocol. This surprising property of NSPK in the context of our modeling is interesting itself.

Outline

– Functional Modeling of Nondeterministic Computing Systems ↪

– Verification via Supercompilation ↪

– Prorgam Transformation for Program Verification (Related Works) ↪

– Our Contribution ↪

– The Needham-Schroeder Public Key Authentication Crypto-protocol ↪

– **The Presentation Language**

– The Principles of Modeling ↓

– The Program Model of NSPK ↓

– The Intruder Behavior Logic ↓

– A Verification Attempt ↓

– G. Lowe's Corrected Version of NSPK ↓

– Discussion ↓

The Presentation Language

Programs are strict term rewriting systems based on pattern matching.

The data set is a free monoid of concatenation with an additional unary constructor:

$d ::= [] \mid c \mid d_1 : d_2 \mid (d)$ — $c \in$ **identifiers**

The monoid of the terms:

$t ::= [] \mid c \mid v \mid f(\ args\ ) \mid t_1 : t_2 \mid (t)$ — $f \in$ **function names**

$args ::= t \mid t,\ args$ — $v \in$ **variables**

$s.var$ **ranges over identifiers,** $e.var$ **ranges over the whole data set,** $t.var$ **ranges over the data set excluding [].**

Outline

- Functional Modeling of Nondeterministic Computing Systems $\hookrightarrow$
- Verification via Supercompilation $\hookrightarrow$
- Prorgam Transformation for Program Verification (Related Works) $\hookrightarrow$
- Our Contribution $\hookrightarrow$
- The Needham-Schroeder Public Key Authentication Crypto-protocol $\hookrightarrow$
- The Presentation Language $\hookrightarrow$
- The Principles of Modeling $\hookrightarrow$
- **The Program Model of NSPK**
- The Intruder Behavior Logic $\downarrow$
- A Verification Attempt $\downarrow$
- G. Lowe's Corrected Version of NSPK $\downarrow$
- Discussion $\downarrow$

Modeling of the Dynamic of the Protocols

- **A non-deterministic protocol S is specified as a term rewriting system $\phi(i, \bar{x})$. i takes the initial state of S, $\bar{x}$ takes a finite sequence of actions of S. Given an action x_0 and a current state of S, ϕ computes the following state of S and goes on to the next action. ϕ returns a state of S after executing the $\bar{x}_0$.**

- **In the case of the NSPK protocol the state consists of the open channel's content and the protocol memory split into the memories of all participants of S. The open channel contains the only message.**

Privacy Policy for the Protocol Participants' Memory

- The protocol memory is a sequence p_1, p_2, q_3. p_i is the memory of the i-th legal protocol participant, q_3 is the intruder memory: Memory:t.A:t.B:e.memory. t.A ::= (A:e.A), t.B ::= (B:e.B)

- e.memory ::= (C:e.C) | []. C's memory may be omitted. That is a trick:

  - the term (C:e.C) informs C that the channel contains a message put by himself;

  - [] means the message was renewed by someone else.

Privacy Policy for the Protocol Participants' Memory - II

– **Memory:t.A:t.B:e.memory**. **t.A ::= (A:e.A)**, **t.B ::= (B:e.B)**

– The privacy policy for the memory is achieved by a programming discipline. Given a participant, (s)he performs his/her *given* step of the protocol with the only term rewriting step. The patterns of the sentences, which can realize this step, are not allowed to specify the part of the memory of the other participants.

– **Example:** The active participant is **B**.

- the memory pattern **Memory:(A:e.A):(B:B2):t.C** is allowed
- **Memory:(A:to:s.EB):(B:B2):t.C** is not

Privacy Policy for the Protocol Participants' Memory - III

- The intruder C is not able to spy (see above) in the memory of the legal participants. He cannot decrypt a message if he does not know the key decrypting the message. C never replaces his own message in the channel. C is able to synthesize messages from data taken from the pattern (through its variables' values), but he cannot split these values: if the pattern variables are e.xy, t.A, s.rB, then C may produce the term e.xy t.A e.xy (s.rB), but cannot specify (even partly) the values of these variables.

- The model may trace some information on the protocol evaluation. That may be useful for constructing an attack on the protocol.

The Program Model of NSPK

```
mainNSPK( Ms:t.1:t.2:e.cls, Key:s.EB ) =
     Test( (A:s.EB:(rA:A)): Loop( Ms:t.1:t.2:e.cls, Message:s.EB:(rA:A),
                                              Memory:(A:to:s.EB):(B:[])));
Loop( Ms, e.message, e.memory ) = refused;   /*1.*/
/*2a-b.*/
Loop(Ms:(B:[]):e.cls,Message:EB:(s.rA:A),Memory:(A:e.A):(B:[]):e.memory)
     = (B:EA:(s.rA:rB)) : Loop( Ms:e.cls, Message:EA:(s.rA:rB), Memory:(A:e.A):(B:B2) );
Loop( Ms:(B:[]):e.cls, t.message, t.memory ) = refused;
/*3a-b.*/
Loop( Ms:(A:e.A):e.cls, Message:EA:(rA:s.r), Memory:(A:to:s.EB):t.B:e.memory )
     = (A:s.EB:(s.r)) : Loop( Ms:e.cls, Message:s.EB:(s.r), Memory:(A:to:s.EB):t.B );
Loop( Ms:(A:e.A):e.cls, t.message, t.memory ) = refused;
/*4a-b.*/
Loop(Ms:(B:B2):e.cls,Message:EB:(rB),Memory:(A:e.A):(B:B2):e.memory)=(B:end):connection;
Loop( Ms:(B:B2):e.cls, t.message,  t.memory ) = refused;
Loop( Ms:(C:e.xy):e.cls, Message:EC:t.r, Memory:(A:e.A):(B:e.B) )     /*5.*/
  = (C:e.xy): Loop(Ms:e.cls, Message:e.xy,Memory:(A:e.A):(B:e.B):(C:C1));
Loop( Ms:(C:e.xy):e.cls, t.message, Memory:(A:e.A):(B:e.B):(C:C1) )   /*6.*/
     = Loop( Ms:e.cls, t.message, Memory:(A:e.A):(B:e.B):(C:C1) );

Test( connection ) = True;
Test( e.trace : refused ) = refused;
Test( (C:e.C):e.x:connection ) = (C:e.C):e.x:False;
Test( t.1:e.trace:connection ) = t.1 : Test( e.trace:connection );
```

The Program Model of NSPK: Input Point

```
mainNSPK( Ms:t.1:t.2:e.cls, Key:s.EB ) =
  Test((A:s.EB:(rA:A)): Loop( Ms:t.1:t.2:e.cls, Message:s.EB:(rA:A),
                                        Memory:(A:to:s.EB):(B:[])));
```

– The channel can contain the only message. Key:s.EB — the public key used by A to start.

– Information on the messages sent in the channel is stored in Memory — a sequence of the participants' storages. The information in a storage is available only to the participant corresponding to the storage. The storage for C may be empty.

– The initial encrypted message sent by A is s.EB:(rA:A). The message is encrypted with the s.EB of an addressee with whom A would like to initiate the authentication.

The Program Model of NSPK: Loop

```
/*1.*/
Loop( Ms, e.message, e.memory ) = refused;
/*2a.*/
Loop(Ms:(B:[]):e.cls,Message:EB:(s.rA:A),Memory:(A:e.A):(B:[]):e.memory)
     = (B:EA:(s.rA:rB)) : Loop( Ms:e.cls, Message:EA:(s.rA:rB), Memory:(A:e.A):(B:B2) );
/*2b.*/
Loop( Ms:(B:[]):e.cls, t.message, t.memory ) = refused;
/*3a.*/
Loop( Ms:(A:e.A):e.cls, Message:EA:(rA:s.r), Memory:(A:to:s.EB):t.B:e.memory )
     = (A:s.EB:(s.r)) : Loop( Ms:e.cls, Message:s.EB:(s.r), Memory:(A:to:s.EB):t.B );
/*3b.*/
Loop( Ms:(A:e.A):e.cls, t.message, t.memory ) = refused;
/*4a.*/
Loop(Ms:(B:B2):e.cls,Message:EB:(rB),Memory:(A:e.A):(B:B2):e.memory)=(B:end):connection;
/*4b.*/
Loop( Ms:(B:B2):e.cls, t.message,  t.memory ) = refused;
/*5.*/
Loop( Ms:(C:e.xy):e.cls, Message:EC:t.r, Memory:(A:e.A):(B:e.B) )
  = (C:e.xy): Loop(Ms:e.cls, Message:e.xy,Memory:(A:e.A):(B:e.B):(C:C1));
/*6.*/
Loop( Ms:(C:e.xy):e.cls, t.message, Memory:(A:e.A):(B:e.B):(C:C1) )
     = Loop( Ms:e.cls, t.message, Memory:(A:e.A):(B:e.B):(C:C1) );
```

Loop(Ms:e.cls, e.broadcast, Memory:e.memory) models the protocol dynamic.

- **The first argument ranges over the finite sequences of the messages abstracts. A message abstract includes its sender name and an abstract of the corresponding message (if needed).**

- **Loop sends a next message (e.broadcast) in the channel, modifies the memory and returns a trace of the messages passed over the channel followed by a flag.**

- **The trace is a sequence of (s.sender_name : e.ms_info).**

- **The flag is connection - authentication of the legal participants or refused - interruption of the current session.**

The Program Model of NSPK: Loop - (1)

Loop(Ms, e.message, e.memory) = refused;

– **The message sequence is empty. Mutual authentication is not established. The negotiation is interrupted.**

The Program Model of NSPK: Loop - (2a-b)

```
Loop(Ms:(B:[]):e.cls,Message:EB:(s.rA:A),Memory:(A:e.A):(B:[]):e.memory)
=(B:EA:(s.rA:rB)):Loop(Ms:e.cls,Message:EA:(s.rA:rB),Memory:(A:e.A):(B:B2));
Loop( Ms:(B:[]):e.cls, t.message, t.memory ) = refused;
```

The 1-st **B**'s message - a response to **A**'s request confirms the fact that **B** can decrypt the message received from **A**: **B**'s message contains the nonce **s.rA** sent by **A**. **B** traces his message as a part of the returning result. The memory is cleaned from possible intruder records.

- **(2a):** from **B**'s point of view, all the protocol rules are respected;

- **(2b): B** suspects an attack and interrupts the session.

The Program Model of NSPK: Loop - (3a-b)

```
Loop(Ms:(A:e.A):e.cls,Message:EA:(rA:s.r),Memory:(A:to:s.EB):t.B:e.memory)
= (A:s.EB:(s.r)):Loop(Ms:e.cls,Message:s.EB:(s.r),Memory:(A:to:s.EB):t.B);
Loop( Ms:(A:e.A):e.cls, t.message, t.memory ) = refused;
```

The 2-nd A's message. A checks the fact that his 1-st message was read. That is confirmed with the nonce rA. This A's response returns the nonce s.r back to the communication partner. The response is encrypted with the same public key used for the 1-st A's message: the key was early stored in the memory. The memory is cleaned from possible intruder records.

- **(3a): from A's point of view, all the protocol rules are respected;**

- **(3b): A suspects an attack and interrupts the session.**

The Program Model of NSPK: Loop - (4a-b)

```
Loop(Ms:(B:B2):e.cls,Message:EB:(rB),Memory:(A:e.A):(B:B2):e.memory)
                                              = (B:end):connection;
Loop( Ms:(B:B2):e.cls, t.message, t.memory ) = refused;
```

- **(4a):** The 2-nd B's message: this fact is confirmed with the memory. B checks the fact that his first message was read (by means of the nonce rB returned back to him). B decides that the person signed by A is the A. The authentication is established.

- **(4b):** B suspects an attack and interrupts the session.

The Program Model of NSPK: Loop - (5)

```
Loop( Ms:(C:e.xy):e.cls, Message:EC:t.r, Memory:(A:e.A):(B:e.B) )
  = (C:e.xy) : Loop( Ms:e.cls, Message:e.xy, Memory:(A:e.A):(B:e.B):(C:C1) );
```

- C checks in the memory that the last message sent in the channel was sent by someone else. That is the memory does not contain records written by C.

- C tries to impersonate a legal participant: by means of guessing a message e.xy. Here e.xy is *an arbitrary message*, therefore if C can decrypt the intercepted message t.r, then the message may include any information obtained from t.r.

- C records C1 in the memory, showing this current message is sent by himself. Otherwise the protocol runs in an infinite loop.

The Program Model of NSPK: Loop - (6)

```
Loop( Ms:(C:e.xy):e.cls, t.message, Memory:(A:e.A):(B:e.B):(C:C1) )
    = Loop( Ms:e.cls, t.message, Memory:(A:e.A):(B:e.B):(C:C1) );
```

- C checks in the memory that the last message sent in the channel was sent by himself. If the memory contains the record C1, then C does not replace the current message in the channel: otherwise the protocol evolution passes in an infinite loop.

- This case is the last in Loop. That means if the message was sent by someone else, then the program goes into deadlock (abnormal stop).

The Program Model of NSPK: Test

```
Test( connection ) = True;
Test( e.trace : refused ) = refused;
Test( (C:e.C):e.x:connection ) = (C:e.C):e.x:False;
Test( t.1:e.trace:connection ) = t.1 : Test( e.trace:connection );
```

This function checks correctness of a *fixed* NSPK evolution.

Outline

- Functional Modeling of Nondeterministic Computing Systems ↪
- Verification via Supercompilation ↪
- Prorgam Transformation for Program Verification (Related Works) ↪
- Our Contribution ↪
- The Needham-Schroeder Public Key Authentication Crypto-protocol ↪
- The Presentation Language ↪
- The Principles of Modeling ↪
- The Program Model of NSPK ↪
- **The Intruder Behavior Logic**
- A Verification Attempt ↓
- G. Lowe's Corrected Version of NSPK ↓
- Discussion ↓

The Intruder Behavior Logic

The Intruder Behavior Logic

```
Loop( Ms:(C:e.xy):e.cls, Message:EC:t.r, Memory:(A:e.A):(B:e.B) )
   = (C:e.xy) : Loop( Ms:e.cls, Message:e.xy, Memory:(A:e.A):(B:e.B):(C:C1) );
```

– **Guessing a message may include any subtle analyze of the messages our model completely parameterizes the intruder logic.**

– **Potential disadvantage: e.xy allows generating a message, which might be not relevant to the protocol. That may lead to generation of spurious attacks.**

– **A protocol is not secure if not only an attack was generated, but the attack is proven to be realized.**

The Intruder Behavior Logic - II

Such a spurious attack may be considered as a *"potential attack"*:

- **if the intruder might guess something, then he might break the negotiation confidentiality of the legal participants. Such an attack may also be very interesting for analyzing the protocols.**

Below we show that the NSPK logic model chosen above does not lead to the spurious attacks.

Outline

– Functional Modeling of Nondeterministic Computing Systems ↪

– Verification via Supercompilation ↪

– Prorgam Transformation for Program Verification (Related Works) ↪

– Our Contribution ↪

– The Needham-Schroeder Public Key Authentication Crypto-protocol ↪

– The Presentation Language ↪

– The Principles of Modeling ↪

– The Program Model of NSPK ↪

– The Intruder Behavior Logic ↪

– **A Verification Attempt**

– G. Lowe's Corrected Version of NSPK ↓

– Discussion ↓

A Verification Attempt

The NSPK program model P described above terminates on any input data,

– returning a result

– or falling in an abnormal state (pattern recognition impossible).

Claim: In the case the supercompiler SCP4 is able to transform P to a residual program P' such that P' has simple *syntactic* properties showing that P' is identically true on its domain, then the NSPK program model is secure.

Crucial Syntactic Property

E.g. such a syntactic property, in the presentation language terms, is lack of in P's right-side top-level passive subexpressions without the identifier False.

Otherwise the program model P' has to be additionally analyzed.

The Result of Specialization of the NSPK Program Model

The supercompiler generates a program P' containing two sentences with the identifier False.

One of them:

```
F122( (B:B2):e.140, e.142, EB:(rB) )
      = (A:EC:(rA:A)):(C:EB:(rA:A)):(B:EA:(rA:rB)):(A:EC:(rB)):
        e.142:(C:EB:(rB)):(B:end):False;
```

– the program model is not secure

– or the supercompiler is not able to solve the verification task.

Interactive Search for an Attack

We iterate supercompilation, modifying the input point of the program P.

mainNSPK(Ms:t.1:t.2:e.cls, Key:s.EB) = ... — original

mainNSPK(Ms:t.1:t.2:t.3, Key:s.EB) = ... — True
mainNSPK(Ms:t.1:t.2:t.3:t.4, Key:s.EB) = ... — True
mainNSPK(Ms:t.1:t.2:t.3:t.4:t.5, Key:s.EB) = ... — False

The residual program is a one-step program:

```
                    ...............
mainNSPK'( Ms:(C:EB:(rA:A)):(B:[]):(A:e.129):(C:EB:(rB)):(B:B2), Key:EC )
            = (A:EC:(rA:A)):(C:EB:(rA:A)):(B:EA:(rA:rB)):(A:EC:(rB))
              :(C:EB:(rB)):(B:end):False;
                    ...............
```

A Possible Attack

on NSPK is the following message sequence:

(A:EC:(rA:A)):(C:EB:(rA:A)):(B:EA:(rA:rB)):(A:EC:(rB)):(C:EB:(rB)):(B:end)

- **the considered sentence may be unreachable during interpretation of the residual program P';**

- **the constructed attack may be spurious.**


HACKER
DETECTED!!

The Possible Attack is the Classical G. Lowe's Attack (1995)

The intruder C using an authentication request from a participant A impersonates A in an authentication exchange with B.

1. **(A : EC:(rA:A))** - **A** initiates a session with **C**;

2. **(C : EB:(rA:A))** - **C** receives the 1-st message, decrypts it with his key, encrypts the result with **EB** and replaces the 1-st message with the changed one;

3. **(B : EA:(rA:rB))** - **B** taking into account that the 2-nd message signed by **A** sends the current message signed with the nonce **rB** and encoded with **EA**;

4. **(A : EC:(rB))** - **A** seeing that his previous message successfully decoded decides that **B'** key is really the key used for the 1-st message and sends confirmation of reading the 3-rd message, once again using the key **EC**;

5. **(C : EB:(rB)))** - **C** intercepts and decodes the 4-th message, he sends the (re)encoded message ensuring **B** that the last **B**'s message was read and he is a legal participant;

6. the technical **(B:end)** term just means that **B** receives the 5-th message and falsely decides that **C** is **A**.

Outline

- Functional Modeling of Nondeterministic Computing Systems ↪
- Verification via Supercompilation ↪
- Prorgam Transformation for Program Verification (Related Works) ↪
- Our Contribution ↪
- The Needham-Schroeder Public Key Authentication Crypto-protocol ↪
- The Presentation Language ↪
- The Principles of Modeling ↪
- The Program Model of NSPK ↪
- The Intruder Behavior Logic ↪
- A Verification Attempt ↪
- **G. Lowe's Corrected Version of NSPK**
- Discussion ↓

G. Lowe's Corrected Version of NSPK

The 2-nd step of NSPK described above can be specified more accurate: B → A : EA(rA,rB,EB) - B sends his public key EB to A. Now, at the 3-rd step, A may compare the received key with the key used for encoding his initial message.
The corresponding cases of the program must be corrected:

```
/*2a.*/
Loop( Ms:(B:[]):e.cls, Message:EB:(s.rA:A), Memory:(A:e.A):(B:[]):e.memory )
  = (B:EA:(s.rA:rB:EB)) :
    Loop( Ms:e.cls, Message:EA:(s.rA:rB:EB), Memory:(A:e.A):(B:B2) );
...
/*3a.*/
Loop(Ms:(A:e.A):e.cls,Message:EA:(rA:s.r:s.EB),Memory:(A:to:s.EB):t.B:e.memory)
  = (A:s.EB:(s.r)) : Loop(Ms:e.cls,Message:s.EB:(s.r),Memory:(A:to:s.EB):t.B);
```

The corrected version of NSPK has been successfully verified.

Outline

- Functional Modeling of Nondeterministic Computing Systems ↪
- Verification via Supercompilation ↪
- Prorgam Transformation for Program Verification (Related Works) ↪
- Our Contribution ↪
- The Needham-Schroeder Public Key Authentication Crypto-protocol ↪
- The Presentation Language ↪
- The Principles of Modeling ↪
- The Program Model of NSPK ↪
- The Intruder Behavior Logic ↪
- A Verification Attempt ↪
- G. Lowe's Corrected Version of NSPK ↪
- **Discussion**

Discussion

– **Verification via supercompilation is an empirically successful method for verification of parameterized systems.
A less parameterized approach has been applied to the verification crypto-protocols via supercompilation (A. Ahmed, 2008).**
http://www.csc.liv.ac.uk/~alexei/A.Ahmed.dissertation.pdf

– **The stricter rules of a protocol the simpler its program model. The weaker specifications imposed on a protocol the more complicated its open channel program model.**

– **The very conservative definition of tha attack does not lead to the generation of the spurious attacks on NSPK protocol. Describe a class of crypto-protocols having the similar property.**

Thank you!