ON INDUCTIVE APPROACH IN SECURITY PROTOCOL VERIFICATION

Ing. Pavel OČENÁŠEK, Doctoral Degree Programme (1)
Dept. of Information Systems, FIT, BUT
E-mail: ocenaspa@fit.vutbr.cz

Supervised by: Prof. Miroslav Švéda

ABSTRACT

This paper outlines some inductive methods that are used in verification of security protocols. The basic idea of inductive proof is outlined, followed by the description of the popular theorem prover Isabelle. The second part covers the description of spi-calculus which can be used to analyze security protocols as concurrent processes over communication channels.

1 INTRODUCTION

Security protocols are intended to let agents communicate securely over an insecure network. A crucial goal is to prevent a spy from reading the contents of messages intended for others (*secrecy*). Most security protocols also deal with *authenticity*. This means that if a message appears to be from agent A, then A sent precisely this message and it contains the indication of its freshness.

Recently, many formal methods [1] have been used for analyzing security protocols. We can identify two main approaches: state exploration and belief logics. In the first type of methods, the exhaustive search checks safety of all reachable states. In the second case we formalize what communicating agents can infer from the received messages. The inductive approach deals with both of them. From the model-based approach, we borrow a concrete notion of events, such as agent A is sending message X to the agent B. The second one allows us to use an idea of deriving guarantees from each message.

2 INDUCTIVE APPROACH

The inductive approach scales up to the analysis of real-world protocols thanks to its extensibility. Protocols are formalized as the set of all possible traces, which are lists of events such as “A sends message X to B”. Properties are proved by induction on traces, using the theorem prover (popular tool Isabelle).

The approach is oriented around proving guarantees, but their absence can indicate possible attacks.

2.1 OPERATORS PARTS, ANALZ AND SYNTH

Each of these three operators is defined inductively, as the least set closed under specified extensions. Each extends a set of messages *H* with other items derivable from *H*.

The set *parts H* is obtained from H by repeatedly adding the components of compound messages and the bodies of encrypted messages. It represents the set of all components of *H* that are potentially recoverable, perhaps using additional keys.

$$\text{Crypt K X} \in \text{parts H} \Rightarrow \text{X} \in \text{parts H}$$

$$\text{parts G} \cup \text{parts H} = \text{parts (G} \cup \text{H)} .$$

Another set *analz H* is obtained from *H* by repeatedly adding the components of compound messages and by decrypting messages whose keys are in *analz H*.

$$\text{Crypt K X} \in \text{analz H}, \text{K}^{-1} \in \text{analz H} \Rightarrow \text{X} \in \text{analz H}$$

$$\text{analz G} \cup \text{analz H} \subseteq \text{analz (G} \cup \text{H)}$$

$$\text{analz H} \subseteq \text{parts H} .$$

The last set *synt H* models the messages a spy could build up from elements of *H* repeatedly adding agent names, forming compound messages and encrypting with keys contained in *H*. Agent names are added because they are publicly known.

$$\text{X} \in \text{synth H}, \text{K} \in \text{H} \Rightarrow \text{Crypt K X} \in \text{synth H}$$

$$\text{K} \in \text{synth H} \Rightarrow \text{K} \in \text{H} .$$

2.2 MODELING A PROTOCOL

Paulson [2] [3] consider a security protocol and an unlimited population of agents. In particular, the agents include a spy who monitors the entire network and knows the long-term secrets of an unspecified set of compromised agents.

The free type key is introduced to represent cryptographic keys. In the symmetric-key setting, each agent is provided with a long-term key that is shared with the server

```
shrK : agent → key
```

whereas for asymmetric-key setting two functions are defined respectively for the private and the public keys of each agent.

A protocol description usually requires three additional rules. The first is the empty list – a trace denoted by []. Two other rules are needed to model fake messages and accidents.

If *evs* is a trace, $X \in synth(analz\ H)$ is a fraudulent message and $B \neq Spy$ then *evs* may be extended with the event

```
Says Spy B X
```

Here H contains all messages in the past trace. It includes the spy's initial state, which holds

the long-term keys of and arbitrary set of ‘bad’ agents.

If *evs* is a trace and *S* distributed in the session key *K* in a run involving the nonces *Na* and *Nb*, then *evs* may be extended with the event

```
Notes Spy {Na, Nb, K}
```

This rule models the loss of session keys.

2.3 INDUCTION

The principle of inductive proof is the following: Let N be the set of natural numbers and n ∈ N. To prove *P(n)* for each number n, prove *P(0)* and prove *P(x)* □ *P(Suc x)* for each n . For the set of traces, the induction principle says that *P(evs)* holds for each trace *evs* provided *P* is preserved under all the rules for creating traces.

We must prove *P[]* to cover the empty trace. For each of the other rules, we must prove an assertion of the form *P(evs)* □ *P(ev#evs)*, where event *ev* contains the new message.

2.4 ISABELLE

The approach has been automated using Isabelle/HOL [4] [5], an instantiation of the generic theorem prover Isabelle for higher-order logic. Isabelle is appropriate because it contains support for inductively defined sets and automatic tools related to them.

In Isabelle, there are various kinds of message items declared:

```
datatype agent = Server | Friend nat | Spy
datatype msg = Agent agent
      | Nonce     nat
      | Key key
      | MPair     msg msg
      | Crypt     key msg
```

To illustrate the syntax, we can define analz operator:

```
const analz :: msg set => msg set
inductive "analz H"
intrs
Inj  "X ∈ H ⇒ X ∈ analz H"
Fst  "{|X,Y|}∈analz H ⇒ X∈analz H"
Snd "{|X,Y|}∈analz H ⇒ Y∈analz H"
Decrypt "[ | Crypt K X∈analz H;  Key(invKey K)∈analz H | ] ⇒ X∈analz H"
```

In Isabelle, proofs are highly automated. One command can generate thousands of inferences. Small changes to protocols involve only small changes to proof scripts.

2.5 SPI CALCULUS

The spi-calculus [6] [7] is an extension of the pi-calculus with cryptographic primitives. It is designed for the description and analysis of security protocols. These protocols rely on cryptography and on communication channels with properties like authenticity and privacy. In

the spi-calculus a cryptographic protocol can be described as a concurrent process. Analysis of the traces generated by this process can be used to verify authentication and secrecy properties of the protocol.

The main drawback of trace analysis is that protocols usually can generate infinitely many traces. The reason is that the behavior of the environment is largely unpredictable. Rather than trying to describe this behavior as a specific process, it is sensible to simply assume that the communication network is totally under the control of the environment. The basic idea is to replace the infinitely many transitions arising from an input action by a single symbolic transition, and to represent the received message by a variable. In spi-calculus, the receiver of a message is written as *a(x).R*, where *a* is an arbitrary label, *x* is the input variable and *R* is the continuation.

For example, suppose that a process *P*, after receiving a message *x*, tries to decrypt *x* using key *k*, and, if this succeeds, calls *y* the result and proceed like *P'*.

This is written as $P =^{def} a(x).case\ x\ of\ \{y\}_k\ in\ P'$.

If we represent the state of the protocol as a pair $\langle \sigma, Q \rangle$, where σ is the trace of process' past actions and *Q* is a process term, the only two initial transitions of *P* will be:

$$\langle \varepsilon, P \rangle_s \rightarrow_s \langle a(x), case\ x\ of\ \{y\}_k\ in\ P' \rangle_s \rightarrow_s \langle a\langle \{y\}_k \rangle, P'[\{y\}_k / x] \rangle_s$$

where subscript *s* means "symbolic".

We assume an infinite set of names, to be used for communication channels, and an infinite set of variables. We let m, n, p, q and r range over names, and let x, y, and z range over variables. The set of terms is defined by the grammar:

L, M, N ::=	terms
n	name
(M, N)	pair
0	zero
suc(M)	successor
x	variable

The set of processes is defined by the grammar:

P, Q, R ::=	processes
M<N>.P	output
M(x).P	input
P | Q	composition
(νn)P	restriction
!P	replication
[M is N] P	match
0	nil
let (x, y) = M in P	pair splitting
case M of 0: P suc (x): Q	integer case

A composition P | Q denotes processes P and Q running in parallel. Each may interact with the other on channels known to both, or with the outside world, independently of the other.

A restriction *(νn)P* is a process that makes a new, private name n, which may occur in *P*, and then behaves as *P*.

If we have a communication protocol:

$A \rightarrow S$: c_{AB} on c_{AS}

$S \rightarrow B$: c_{AB}on c_{AB}

$A \rightarrow B$: M on c_{AB}

then the spi-calculus description is the following:

$A(M) \cong (\nu c_{AB}) c_{AS}\langle c_{AB}\rangle . c_{AB}\langle M\rangle$

$S \cong c_{AS}(x) . c_{SB}\langle x\rangle$

$B \cong c_{SB}(x) . x(y) . F(y)$

$Inst(M) \cong (\nu c_{AS})(\nu c_{SB})(A(M) \mid S \mid B)$

The method of using spi-calculus for verification yields decidability results for interesting classes of protocols and properties and is amenable to mechanization.

3 CONCLUSIONS

The inductive method is simple and general. Theorem proving seems to be, along with model checking, the most successful approach to the formal analysis of security protocols. It is intuitive and correctly deals with freshness and multiple executions.

The limitations of formal methods are obvious from e.g. attack to a recursive protocols. Here models idealize real world (assuming strong encryption). Recent research was provided in compositional methods. This approach seems to be necessary for different levels of abstraction - protocol messages, cryptographic algorithms should be verified separately. Our future work will focus on utilization of proposed approach for verification of wireless communication protocols based on the IEEE 802.11 standard.

REFERENCES

[1] Bella, Giampaolo: Inductive Verification of Cryptographic Protocols, University of Cambridge, 2000

[2] Paulson, L. C.: The inductive approach to verifying cryptographic protocols. Journal of Computer Security, Volume 6, Issue 1,2, pp.85-128, September 1998

[3] Paulson, L. C.: Proving Security Protocols Correct. In: Proc. of LICS'99, IEEE Computer Society Press, pp.370-383, 1999

[4] Bella, G., Paulson, L. C.: Using Isabelle to Prove Properties of the Kerberos Authentication System. University of Cambridge, 1997

[5] Paulson, L. C.: Isabelle: A Generic Theorem Prover. Springer Verlag 1994. LNCS 828

[6] Boreale, Michele: Symbolic analysis of cryptographic protocols in the spi-calculus, Universita di Firenze, 2000

[7] Abadi, M., Gordon A. D.: A calculus for cryptographic protocols: The spi calculus. Information and Computation, 1999