An Efficient Cryptographic Protocol Verifier Based on Prolog Rules

Bruno Blanchet, 2001

January 24, 2016

Mikael Elkiær Christensen
michri11@student.aau.dk

Department of Computer Science
Aalborg University
Denmark

The Problem

Mikael Elkiær Christensen

The Needham-Schroeder Public Key Protocol (1978):

1. $A \rightarrow S : A, B$
2. $S \rightarrow A : \{K_b, B\}_{K_s^{-1}}$
3. $A \rightarrow B : \{N_a, A\}_{K_b}$
4. $B \rightarrow S : B, A$
5. $S \rightarrow B : \{K_a, A\}_{K_s^{-1}}$
6. $B \rightarrow A : \{N_a, N_b, B\}_{K_a}$
7. $A \rightarrow B : \{N_b\}_{K_b}$

Man-in-the-middle attack presented by Gabin Lowe (1995).

Not The Problem

(CVE-2014-0160)

Overview

- Previously: Applied π Calculus, Multiset Rewriting, Model checking
  - Limiting runs, inefficient, non-automatic, restrictions
- Now: Prolog (First-order logic)
  - FOL: Generally, **sound**, but not **complete**
  - Uses custom resolution and unification
  - Makes approximations
  - Proves secrecy

Syntax

$M, N ::=$	terms
x	variable
$a[M_1, \ldots, M_n]$	name
$f(M_1, \ldots, M_n)$	function application
$F ::=$	fact
$p(M_1, \ldots, M_n)$	predicate application
$R ::=$	rule
$F_1 \wedge \cdots \wedge F_n \rightarrow F$	implication

$sk_A[], sk_B[]$
$k[x_1, \ldots, x_n]$

constructor	destructor
$pk_A = \mathbf{pk}(sk_A[])$	
$\mathbf{pencrypt}(m, pk(sk))$	$\mathbf{decrypt}(\mathbf{encrypt}(m, pk(sk)), sk) = m$
$\mathbf{sencrypt}(m, k)$	$\mathbf{sdecrypt}(\mathbf{sencrypt}(m, k), k) = m$
$\mathbf{sign}(m, sk)$	$\mathbf{getmess}(\mathbf{sign}(m, sk))$
$(\_, \ldots, \_)$	$\boldsymbol{i}\mathbf{th}((x_1, \ldots, x_n)) = x_i \mid i \in \{1, \ldots, n\}$
h	

Abilities of the attacker

Assumed the protocol is executed in the presence of an attacker that can:

- intercept all messages,
- compute new messages from the messages it has received, and
- send any message it can build.

Protocol

A protocol can be represented by three sets of rules:

1. Rules representing the computation abilities of the attacker

   $attacker(x_1) \wedge ... \wedge attacker(x_n) \rightarrow attacker(f(x_1, \ldots, x_n))$,

   $attacker(M_1) \wedge \cdots \wedge attacker(M_n) \rightarrow attacker(M)$

2. Facts corresponding to initial knowledge of the attacker

   $attacker(A[])$, $attacker(pk(sk_A[]))$

3. Rules representing the protocol itself

   $attacker(M_{j_1}) \wedge \cdots \wedge attacker(M_{j_n}) \rightarrow attacker(M_i)$.

Approximations

- New names are functions of messages previously received, unless altered.
- The same step of a protocol can be completed several times, yielding the same result, provided that the previous steps have been completed.

- Correctness still holds – intuitively, more attacker options and safe, still safe with less options.
- However, can lead to false attacks.

Multisets

- A hypotheses $F_1, \ldots, F_n$ of a rule are considered a multiset.
- A multiset of facts S is a function $S(F)$ yielding the number of repetitions of F in S.
- Giving a point-wise order on functions: $S \subseteq S' \Leftrightarrow \forall F, S(F) \leq S'(F)$.

Definition 1 (Rule Implication)

Mikael Elkiær Christensen

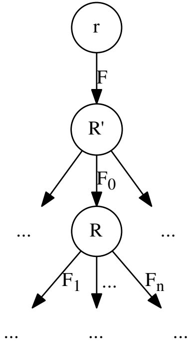
$$(H_1 \rightarrow C_1) \Rightarrow (H_2 \rightarrow C_2)$$
if and only if
$$\exists \sigma, \sigma C_1 = C_2, \sigma H_1 \subseteq H_2$$

Dept. of Computer Science
Aalborg University
Denmark

Definition 2 (Derivability)

Let F be a closed fact. Let B be a set of rules. F is derivable from B if and only if there exists a finite tree defined as follows:

1. Its nodes (except the root) are labelled by rules $R \in B$;
2. Its edges are labelled by closed facts;
3. If the tree contains a node labelled by R with one incoming edge labelled by F_0 and n outgoing edges labelled by $F_1, \ldots, F_n$, then $R \Rightarrow \{F_1, \ldots, F_n\} \rightarrow F_0$.
4. The root has one outgoing edge, labelled by F.



Prolog

$$\mathit{attacker}(\mathbf{pencrypt}(m, \mathbf{pk}(sk))) \wedge \mathit{attacker}(sk) \\ \rightarrow \mathit{attacker}(m)$$

First phase: completion of the rule base

1. For each $R \in B_0$, $B \leftarrow$ **(**add)(**elimdup**(R)B).
2. Let $R \in B$, $R = H \rightarrow C$ and $R' \in B$, $R' = H' \rightarrow C'$. Assume that there exists $F_0 \in H'$ such that:
   a) $R \circ_{F_0} R'$ is defined;
   b) $\forall F \in H, F \in_r S$;
   c) $F_0 \notin_r S$.

   In this case, we execute

   $$B \leftarrow \textbf{add}(\textbf{elimdup}(R \circ_{F_0} R'), B).$$

   This procedure is executed until a fixed point is reached.
3. Let $B' = \{(H \rightarrow C) \in B \forall F \in H, F \in_r S\}$.

Second phase: backward depth-first search

We define **derivablerec**(R, B'') by

1. **derivablerec**$(R, B'') = if \exists R' \in B'', R' \Rightarrow R$;
2. **derivablerec**$(\rightarrow C, B'') = \{C\}$ otherwise;
3. **derivablerec**$(R, B'') =$ $\cup\{$**derivablerec**(**elimdup**$(R' \circ_{F_0} R), \{R\} \cup B'')R' \in B', F_0$ such that $R' \circ_{F_0} R$ is defined $\}$ otherwise.

derivable$(F) =$ **derivablerec**$(\{F\} \rightarrow F,)$.

Experimental results

Protocol	Result	# Rules	Time (ms)
Needham-Schroeder public key	Attack	14	70
Needham-Schroeder public key corrected	Secure	14	60
Needham-Schroeder shared key	Attack	47	760
Needham-Schroeder shared key corrected	Secure	51	1190
...			

Time for questions...

AALBORG UNIVERSITY
DENMARK