A guide for Securify

Véronique Cortier

Abstract

We present a tool for automatically verifying secrecy properties on cryptographic protocols. It has been supported by the EVA project that aims at providing a toolbox for verifying cryptographic protocols.

1 Introduction

Securify is a tool for verifying secrecy properties on cryptographic protocols for an unbounded number of sessions, an unbounded size of messages, an unbounded number of participants and an unbounded number of nonces. It has been supported by the RNTL EVA project, which aims at providing a toolbox for verifying cryptographic protocols.

Securify computes sufficient conditions to ensure secrecy. Three tests are performed for every rule and every part of a sent message:

- either this part is a fresh generated nonce thus it cannot compromise any secret,
- or this part was already sent on the network, encrypted with at least the same set of keys,
- or this part is a secret but is encrypted with a protected key.

When none of these tests succeed, a backward search is done in order to obtain more information about the messages already sent on the network and the three tests are applied again.

The underlying algorithm has been described in [CMR01] but some hints are given here. Other tools for verifying cryptographic protocols can be found on the web page of the RNTL EVA project: `http://www-eva.imag.fr`

2 Interface

Two versions of Securify are available. The first one may be used through the web interface available at: `http://www-eva.imag.fr/outils1.html`. This version has some space and time limitations. The second and full version can be downloaded through the same web page. In this version, the user has no space or time limitations. In addition, he specifies the maximal number of backward search (explained later in this paper), while the number is always limited to 10 in the version available on the web.

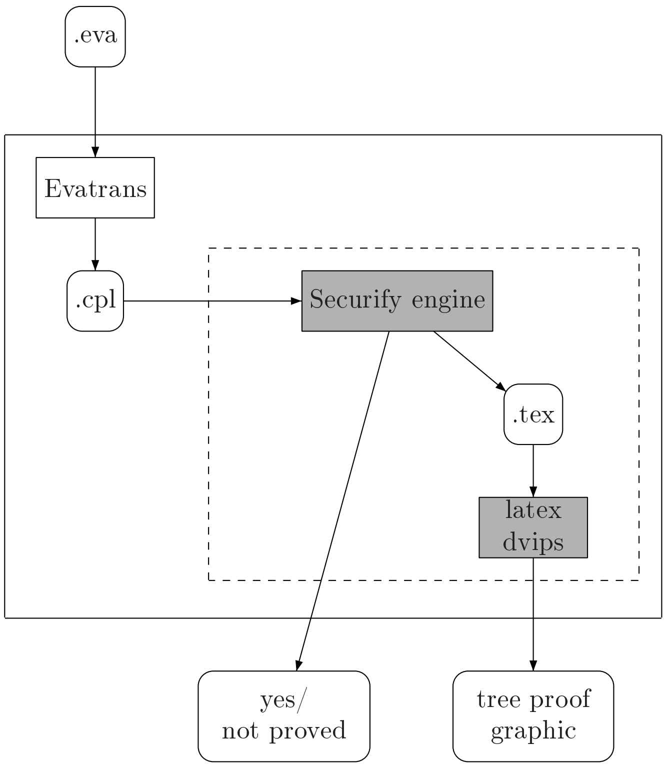
2.1 Input

The Securify input is a protocol specification written in the EVA language defined by D. Le Métayer and F.Jacquemard from Trusted Logic [JM01]. For example, the specification in the EVA language of the Needham-Schroeder protocol is given in Figure 1.

The protocols handled by Securify are typically those described in the Clark & Jacob survey [CJ97]. There are however some restrictions:

- Compound keys are not allowed. This implies that a message variable cannot be used in a key position;
- Long-term secret keys (like private keys or keys shared between an agent and a server) must not be used as plaintext;
- Long-term keys are either constant keys, or public or private agent's keys or else shared keys between an agent and a server.

2.2 Architecture



```
Needham_Schroeder_cles_publiques

  alg : asym_algo
  everybody knows alg

  A, B: principal

  basetype key
  Na, Nb : key

  keypair^alg PK, SK (principal)
  everybody knows PK

  A knows A, B, SK(A)
  B knows B, SK(B)

  {
    1. A -> B : {Na, A}_(PK(B))^alg
    2. B -> A : {Na, Nb}_(PK(A))^alg
    3. A -> B : {Nb}_(PK(B))^alg
  }

  s1. session *{A,B} A=A, B=B

  assume secret (SK(A)@s1.A), secret (SK(B)@s1.B),
         secret (SK(B@s1.A)), secret (SK(A@s1.B))

  claim *A*G secret (SK(A)@s1.A),
        *A*G secret (SK(B)@s1.B),
        *A*G secret (Na@s1.A),
        *A*G secret (Nb@s1.B)
```

Figure 1: Needham-Schroeder protocol described in the EVA language.

The eva file is first translated into an other model (EVA to CPL), more suitable for verification, by a front-end translator developped at the LSV/ENS Cachan by J. Goubault-Larrecq. Errors are reported in the `err` file.

Then the Securify engines tries to verify the specification and construct a tree proof or a tree proof attempt. This tree proof is described in a `tex` file which is compiled into a `ps` and a `pdf` file.

2.3 Output

On the standard output:

- Translation of the protocol in the MR model (explained below),
- Answer to the following question : does the protocol satisfy its requirements?
- Computation time.

In addition, three other files are produced (only the last one may be obtained through the web interface):

- The file.cpl contains the protocol specification in the CPL language;
- The file.err contains the error messages;
- The file.ps contains the proof attempt of the protocol cor- rectness.

3 Securify engine

3.1 MR model

Securify first translates the protocol described in `cpl` into the Millen & Rueß-model, whose description may be found in [MR00, CMR01]. This translation is written on the standard input. For example, the output corresponding to the `eva` file of figure 1 is described in figure 2.

It may help the user to verify that he obtained the wanted protocol.

The variables A_i stand for variables of sort agents, variables N_i for variables of sort nonces or keys, variables X_i stand for messages variables.

There are two kinds of events in the MR model: messages and spells. Spells are secrecy specification. For example, the spell `{Prv(A2),Prv(A1),N1}#{}` in the first rule of figure 2 means that if the values `Prv(A2),Prv(A1),N1` are initially unknown to the intruder then they have to remain unknown to the intruder.

3.2 Principle

Securify verifies that none of the protocol's rules compromises a secret. Three tests are performed for every rule and every part of a sent message:

```
            Formal protocol:
[
  [{Prv(A2),Prv(A1),N1}#{}] \\ If the values Prv(A2),Prv(A1) and
                               N1 are initially secret
  []                  \\ This field is used for fresh values
                         which are not wanted to be secret.
  [[[N1,A1]_Pub(A2)]] \\ Then the message [N1,A1]_Pub(A2)] is
                         added to the trace.
]
[
  [{Prv(A3),Prv(A4),N4}#{},[[N3,A3]_Pub(A4)]]
         \\ If the values Prv(A3),Prv(A4) and N4 are  initially
            secret and if the message [N3,A3]_Pub(A4)] has been sent
  []
  [[[N3,N4]_Pub(A3)]] \\ Then the message [N3,N4]_Pub(A3)] is
                         added to the trace
]
[
  [[[N1,N2]_Pub(A1)],{Prv(A2),Prv(A1),N1}#{},[[N1,A1]_Pub(A2)]]
  []
  [[[N2]_Pub(A2)]]
]
```

Figure 2: The Needham-Schroeder protocol expressed in the MR model.

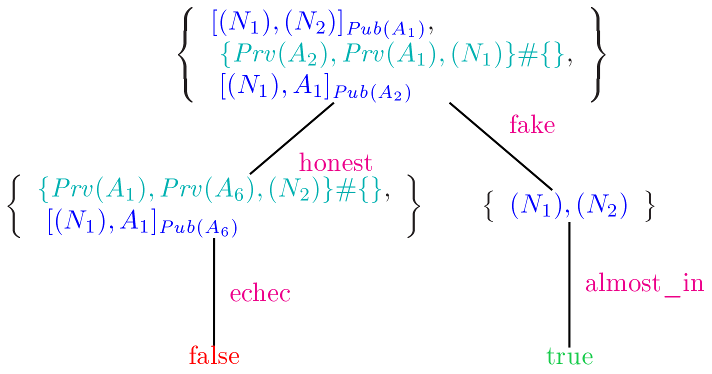


Figure 3: Tree proof attempt corresponding to the last rule of the protocol.

- either this part is a fresh generated nonce thus it cannot compromise any secret, this denoted by the label `new`;
- or this part was already sent on the network, encrypted with at least the same set of keys, this is denoted by the label `almost_in`;
- or this part is a secret but is encrypted with a protected key, this is denoted by the label `db`.

When none of these tests succeed, a backward search is done in order to obtain more information about the messages already sent on the network and the three tests are applied again. For example, let us consider what happens when we try to prove that revealing the nonce `N2` encrypted by the key `Pub(A2)` at the last step of the protocol does not compromised any secret (the tree proof attempt is displayed on figure 3). None of the three tests can be applied but we know that the message `[N1,N2]_Pub(A1)` must be in the trace. Thus, we distinguish two cases: either this message has been obtained be applying an honest rule (one of the rules of the protocol), either it has been built by the intruder. In the first case, looking at the rules, we deduce that the message `[N1,A1]_Pub(A6)` must also be in the trace and nothing else can be deduce: this induces a false node. In the second case, we deduce that the intruder must know the nonces `N1` and `N2`, thus we can apply our `almost_in` test: since no secret was compromised when the intruder knew `N2`, no secret is compromised by revealing `[N2]_Pub(A2)`.

A more complete information about the underlying theory may be found in [CMR01].

3.3 Tricks

- The tool usually works better when messages are typed;
- For some protocols, the number of backward search may be infinite;

- In the web page, the number of backward search is limited to 10. In the full version, the maximal number of backward search is set by the user.

3.4 Building an attack

When every leaf of the proof trees is labeled with `true`, Securify answers "The protocol satisfies the requirements", this means that the protocol satisfies the security specification written in the eva file. When one of the leaves is labeled with `false`, the proof failed but it does not mean that the protocol does not satisfies its requirements, it only mean that the algorithm was not able to conclude.

However, in many cases, the proof tree attempts enable the user to build an real attack. Consider again the tree proof attempt displayed on figure 3. The construction starts at the leaf labelled with `false`. Its parent node contains a spell and a message and corresponds exactly the left-hand-side of the second rule of the protocol. Now, we consider a (partial) run of the protocol where all the rules preceding this one in the protocol description are applied in the "natural" order. In this case, there is only one rule, the first one :

$$A_1 \to A_6 : \{N_1, A_1\}_{\mathsf{pub}(A_6)}$$

Next, we simulate an attack by following the branch from the `false` leaf up to its root. The parent node of the `false` leaf is directly connected with the root by an honest edge.

$$A_6 \to A_1 : \{N_1, N_2\}_{\mathsf{pub}(A_1)}$$

and N_2 should be secret between A_1 and A_6. Having reached the root of the tree, one applies the rule for which our algorithm fails.

$$A_1 \to A_2 : \{N_2\}_{\mathsf{pub}(A_2)}$$

A_1 sends the nonce N_2 to a wrong participant! Using this possibility, we can construct the following scenario, which corresponds to the man-in-the-middle attack:

$$\begin{tikzcd}
A_1 \arrow[r, "\{N_1, A_1\}_{\mathsf{pub}(A_2)}"] & A_2 \arrow[d, "\text{analyze synthesys}"] & \\
& A_2(A_1) \arrow[r, "\{N_1, A_1\}_{\mathsf{pub}(A_6)}"] & A_6 \\
A_1 & & A_6 \arrow[ll, "\{N_a, N_b\}_{\mathsf{pub}(A_1)}"'] \\
A_1 \arrow[r, "\{N_2\}_{\mathsf{pub}(A_2)}"] & A_2 \arrow[d, "\text{analyze synthesys}"] & \\
& A_2(A_1) \arrow[r, "\{N_2\}_{\mathsf{pub}(A_6)}"] & A_6
\end{tikzcd}$$

4 Results

The obtained results are summarized in the above tabular. The experimentations have been realized with a Pentium III, 933 Mhz, 256 Mo de RAM.

Protocols	Proof	maximal # of "back"	Time ms
Otway-Rees	Yes	0	0,35
Woo and Lam	Yes	0	0,11
Denning-Sacco	No	0	0,07
ISO Symmetric Key	Yes	0	0,15
Needham-Schroeder-Lowe	Yes	1	1,08
Needham-Schroeder	No	1	1,23
Kao-Chow	Yes	3	8,94

5 Limitations and futures of the tool

Suppose that N_1 is a nonce generated by an agent A, secret between A and B and that K_{ab} is a key generated by B, secret between A and B. Then the current algorithm is not able to verify the correctness of the protocol if the message $\{N_1\}_{K_{ab}}$ is revealed. However, a refinement of the algorithm, proved correct by Stéphanie Delaune, allow to deal with such protocols and will be implemented soon.

In addition, as stated in this paper, this tool can not handle compound keys but more sophisticated algorithms are currently studied to allow compound keys.

Moreover, we would like to handle some algebraic properties of cryptographic primitives like the exclusive or. The recent theoretical works on this subject could be incorporated in the algorithm.

References

[CJ97] J. Clark and J. Jacob. A survey of authentication protocol literature. `http://www.cs.york.ac.uk/˜jac/ papers/drareviewps.ps`, 1997.

[CMR01] V. Cortier, J. Millen, and H. Rueß. Proving secrecy is easy enough. In *Proceedings of the 14th Computer Security Foundations Workshop (CSFW'01)*, pages 97–108, Cape Breton, Nova Scotia, Canada, 2001. IEEE Computer Society Press.

[JM01] F. Jacquemard and D. Métayer. Langage de spécification de protocoles cryptographiques de eva : syntaxe concrète. Technical Report 1, Projet RNTL EVA, november 2001.

[MR00] J. Millen and H. Rueß. Protocol-independent secrecy. In *Proceedings of the 21st Symposium on Research in Security and Privacy (RSP'00)*. IEEE Computer Society Press, 2000.