AVISS — Automated Verification of Infinite State Systems

(IST-2000-26410)

Deliverable D3.2

# Preliminary definition, implementation and experimentation with the constraint logic theorem prover

# 1 Introduction

The objective of the third work-package WP3 of our project AVISS IST-2000-26410 is to experiment the chosen automated deduction techniques implemented in the prototype verification tool against the verification problems of the corpus [2] of security protocols.

The first step (*encoding*) is the definition of encodings that translates the protocol verification problems, obtained by applying the translator of WP2 to the corpus, into deduction problems falling into the scope of application of the chosen automated deduction techniques. (The architecture of the prototype is shown again in Figure 1.) These encodings, together with the translation mechanism set up in WP2, allows us to run the available automated deduction engines against the verification problems of the corpus (*experiments*). The experiments indicate ways to improve the encodings as well as the inference strategies implemented in the available automated deduction engines (*tuning*).

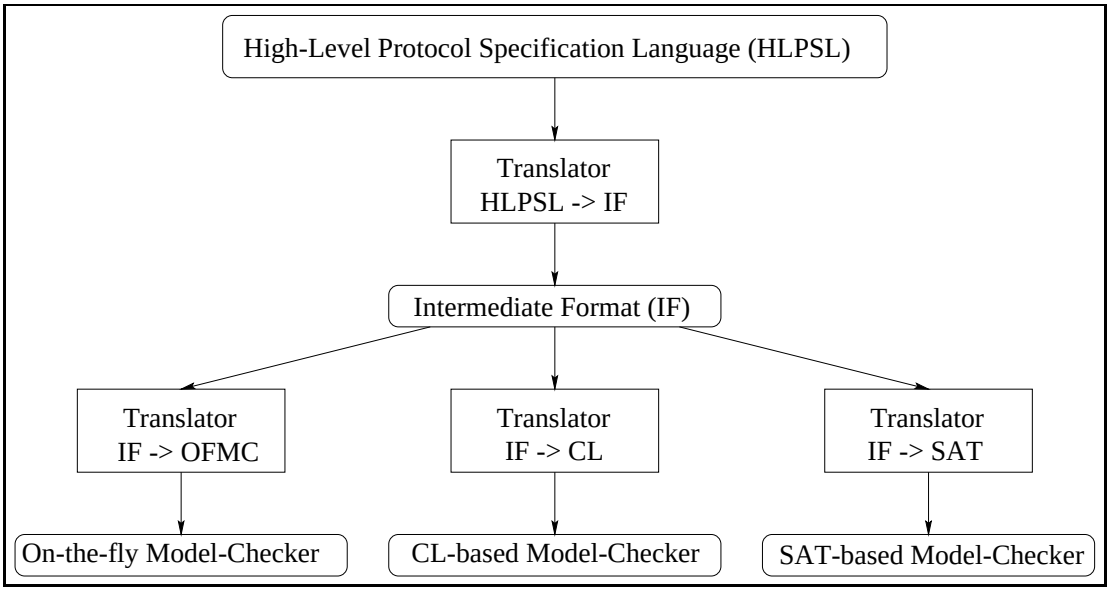


Figure 1: Architecture of the prototype verification tool

This deliverable D3.2 describes our implementation of task T3.2, i.e. define the encoding from the IF to the input format for the constraint-based theorem-prover, develop a prototype translator implementing the encoding, and experiment with problems from the corpus, tuning the encoding and/or the constraint-based theorem-prover.

# 2 Position in the project

The constraint logic theorem prover we employ is daTac.[1] Our translation task, from IF to a format suitable for daTac, is not difficult since daTac itself is a theorem prover using rewrite rules for deduction. After some experimentation, we have come to the conclusion that the rules for the intruder, as given by the HLPSL2IF compiler, were not suitable for using with our theorem prover since they generate more possibilities than we can handle, and they do not use unification, which is the strong point of daTac. Thus, we introduced protocol-independent intruder strategy, i.e. a set of rules that use unification. These rules will later be incorporated into the intermediate format, but we first have to discuss more with the other groups in order to find a description of this strategy compatible with the intermediate format (for a description of this strategy see [1]).

[1] `http://www.loria.fr/equipes/protheo/SOFTWARES/DATAC/`

The use of these rules leads to the current architecture of our part of the project, as given in Figure 2.

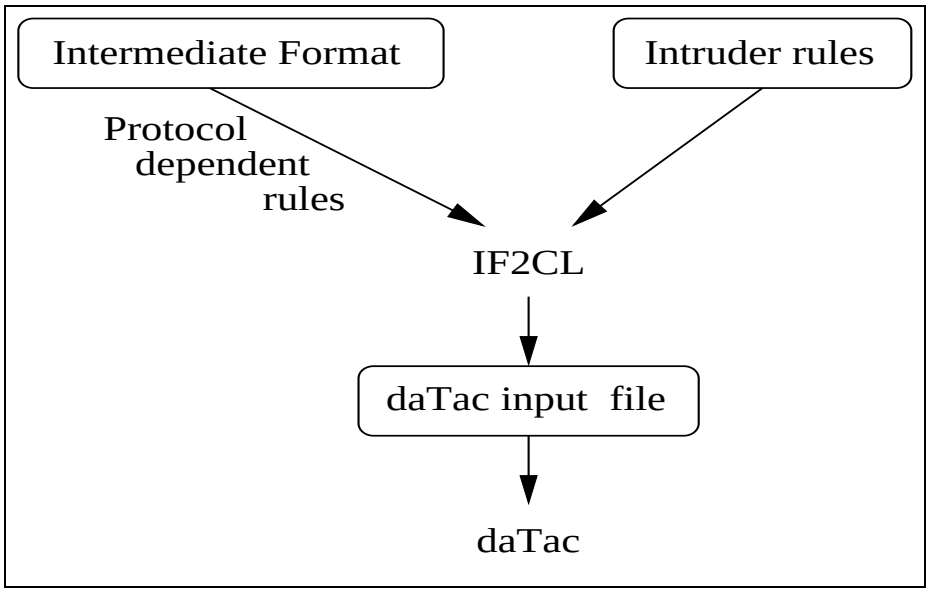


Figure 2: Architecture of the constraint logic part of the project

# 3 The IF2CL translator

The IF2CL] translator is a Perl [2] script that is used to:

1. parse the Intermediate Format of the description of the protocol, as given by HLPSL2IF;
2. include rules for the definition of the intruder;
3. add daTac specific instructions, such as properties of operators and method of deduction.

The translation from Intermediate Format rules to daTac rules is straightforward, since these two formats are in the same formalism. Assuming the reader is familiar with the Intermediate Format (as specified in deliverable D2.2), we can give the following example for the translation of a protocol rule:

```
# lb=Step_1, type=Protocol_Rules
h(s(xTime)).m(1,xr,xA,xB,crypt(xKb,c(xNa,xA)),xc2).w(1,xA,xB,...,...,xbool,xc)[3]
=>
h(xTime).m(2,xB,xB,xA,crypt(xKa,c(xNa,nonce(c(Nb,xTime)))),xc2).w(3,xA,xB,...,...,true,xc)
```

This rule corresponds to the receiving of the first message and sending of the second message in the simplified version of the Needham-Schroder public key protocol (see [2]):

$$A \rightarrow B : \{Na, A\}Kb$$
$$B \rightarrow A : \{Nb, Na\}Ka$$

[2] http://www.perl.com/

[3] We have replaced the knowledge of the principal with dots in order to simplify the presentation.

This rule is then translated, following our evaluation strategy, into:

```
# lb=Step_1, type=Protocol_Rules
State(s(xTime),
m(1,xr,xA,xB,crypt(xKb,c(xNa,xA)),xc2).ee,
w(1,xA,xB,...,...,...,xc).xw,
num(xnum).xi,
xp,
History(msg,xt)
) =>
State(xTime,
m(xStep,mr(I),xSender,xReceiver,xmesg,xsession).ee,
w(3,xA,xB,...,...,...,xc).xw,
diverted(c(xB,c(xA,crypt(xKa,c(xNa,nonce(c(Nb,xTime)))))))).xi.num(s(xnum)),
xp,
History(test,xt.mesg(u(xmesg).vtt.ii.seq,num(xnum).
diverted(c(xB,c(xA,crypt(xKa,c(xNa,nonce(c(Nb,xTime)))))))).xi))
)
```

daTac uses equational clauses, so the first thing we have to add to the rules, as they are given by HLPSL2IF, is a constructor for the state. We use the name **State** for this. In order to have simpler unification problems to solve, we also split the rule given by HLPSL2IF in several parts corresponding to the messages, the principals defined, the knowledge of the intruder, the history of this state. This last part is the only directly connected to the lazy intruder model.

The other changes to the rules given by HLPSL2IF] are connected with the lazy intruder strategy, and are:

1. a term `num(xnum)`, that keeps track of the number of already sent messages;
2. a field `History(...)` that keeps track of the formerly sent messages, as well as of the knowledge the intruder had at that time;
3. the `xp` field, used only when the `parallel` field of the High Level Protocol Specification Language is used;
4. one last transformation, which makes the intruder divert all messages sent by principals. The message term that was on the right hand side of the rule in the Intermediate Format is now replaced by a generic message term, and the knowledge the intruder could have yielded by diverting this message (that is, the sender of the message, the supposed receiver of the message and the message itself), is added into the diverted() term. This term is, during the verification, simplified and added to the knowledge of the intruder.

The `w(...)` term doesn't change. The results obtained using this strategy are given in the next section.

# 4 Verification using the daTac CL theorem prover

## 4.1 The daTac theorem prover

daTacis a theorem prover specialized for deduction in first-order logic with equality, and modulo theories composed with associative and commutative axioms. Its strategy is, in order to prove a property $P$ from a theory $T$, to derive consequences of $T \cup \{\neg P\}$ until it finds an empty clause (a contradiction). The deduction techniques used have been proved to be refutationally complete in first-order logic. The consequence is that, once given a theory $T$ and a property $P$ valid in this theory, daTac will *eventually* find a proof for $P$. The drawback is that it will not stop if $P$ is not valid.

## 4.2 Using daTac for cryptographic protocols verification

The architecture given in Figure 2 can easily be interpreted in terms of theorem proving. The inputs used for building the file for daTac are:

- a theory for the users of the protocol and an initial state, both expressed using the Intermediate Format;
- a theory for the intruder, expressed in the daTac format.

These two theories, once combined, give a theory $T$ for the execution of the protocol in an insecure environment. Our goal is to prove that given a specification for its environment (that is, the behavior of the intruder), the protocol definition is not robust enough to guarantee a property it claims to have. If this is the case, we say that this protocol has a flaw. A flaw can be expressed with a state $P$ that should not be reached. Currently, it can be either:

- a correspondence between principals flaw. In this case, one of the users believes he took part in a protocol session with another user, whereas the other user is not aware a protocol session took place;
- a secrecy flaw. Among the messages exchanged during the protocol, there is a part (a text, or a private key,...) that should have remained hidden, but that is known by the intruder.

It is now possible, using daTac, to see if the property $P$ can be proved using the theory $T$. Since daTac is refutationally complete, we can be sure that if there is a flaw, it will eventually be found.

## 4.3 Results obtained

We have already made some experimentation on several protocols from the corpus [2] and the results are encouraging, since we were able to find a lot of already known attacks. It is already possible to automatically retrieve the following flaws (the protocols and the attacks are illustrated in [2]):

| Protocol | Kind of Flaw | Time |
|---|---|---|
| Encrypted Key Exchange | Correspondence Between Principals | 130s |
| NSPK Exchange | Correspondence Between Principals | 13s |
| TMN | Correspondence Between Principals | 26s |
| Woo-Lam ($\pi$)-protocol | Correspondence Between Principals | 51s |
| Woo-Lam ($\pi - 3$)-protocol | Correspondence Between Principals | 8s |
| Woo-Lam Mutual Authentication | Correspondence Between Principals | 620s |
| SPLICE/AS | Correspondence Between Principals | 108s |
| Neumann-Stubblebine (Part 1) | Correspondence Between Principals | 6s |

Figure 3: Flaws that can be detected automatically

With small modifications to the daTac file, it is also possible to find flaws on the following protocols:

A compromised key attack cannot be expressed yet in the High Level Protocol Specification Language, and this explains why we had to modify the daTac input file. It happens to protocols whose aim is to exchange new keys. It is possible that such a protocol appears to be correct, but

| Protocol | Kind of Flaw | Time |
|---|---|---|
| Secure RPC | Compromised Key | 66s |
| RSA | Secrecy | 5s |
| Kao Chow | Compromised Key | 43s |
| Otway-Rees | Secrecy | 3s |

Figure 4: Flaws that can be detected after some small modifications to the daTac file

that if (by other means) the intruder gets to know a key once exchanged, he can force the users to re-use this same key. We capture this flaw in the following way:

1. specify a secrecy goal on that key in the High Level Protocol Specification Language;
2. (manual change) in the last step of the protocol, give the secret to the intruder. The rules given specify the changes to the state of the protocol. Therefore, we put the sub-term **secret(xsecret,f(xc))** in the left hand side, and replace it, in the right hand side, with the term **i(xsecret,f(xc))**.

The secrecy flaw of the RSA protocol is not automatically detected, because we do not assume in general special properties of RSA cryptographic algorithm, that is:

1. the fact that encryption and decryption use the same algorithm;
2. the fact that encryption is commutative.

Once these properties are added, it is possible to find the well-known flaw on the RSA protocol.

The changes required for the Otway-Rees protocol are more subtle. In the Intermediate Format, we assume the pairing constructor $c$ is left-associative. But in the Otway-Rees protocol, there is a flaw, depending on a type flaw, that occurs when this pairing operator is right-associative. We thus had to change the associativity of this operator, using the rule:

$$=> c(c(x1, x2), x3) = c(x1, c(x2, x3))$$

One shall note that type flaws often depend on such a property. For example, it is possible to find two different attacks on the corrected Needham-Schroeder public key protocol (short version) that use the same deduction rules, the only difference being that in one case, $c$ is right-associative, and in another case, it is left-associative.

# 5 Conclusion

There are still a lot of protocols that we cannot currently study automatically. Considering the changes made to the daTac input file, we however strongly believe it will possible to automate these changes, by adding information in the High Level Protocol Description Language. Among those changes, there is:

- the possibility to specify special properties of some operators (for example, the fact that encryption and decryption use the same algorithm in the RSA case);
- the possibility to include a notion of short term secret, that would be used to study the case where a session-dependent key can be used at a later time by the intruder;
- the extension of the High Level Protocol Specification Language to include timestamps, one-way authentication and other additional features.

# Bibliography

[1] Y. Chevalier and L. Vigneron. A Tool for Lazy Verification of Security Protocols. In *Proceedings of the Automated Software Engineering Conference (ASE'01)*, San Diego (USA), November 2001.

[2] J. Clark and J. Jacob. A survey of authentication protocol literature: Version, 1997.