# Combining two approaches for the verification of cryptographic protocols

Frédéric OEHL and David SINCLAIR

School of Computer Applications
Dublin City University
Dublin 9
Ireland
{foehl,dsinclai}@compapp.dcu.ie

4th October 2001

**Abstract**

This paper presents a new idea to verify the cryptographic protocols as easily as possible. The idea is to combine 2 existing methods, the "inductive approach" of Paulson and the "approximation" of Genet and Klay, to exploit the strengths the best of each method. We concentrate in particular on the transformation of the inductive specification to the input file used by the Timbuk library.

**Keywords**
*Cryptographic protocols - Inductive approach - Approximation technique - Combining*

# 1 Introduction

Today, lots of people have strong reservations about electronic shopping or other activities on the Internet where they need to give personal information. Why? Because they read or heard stories [Gui00, Sul00] about people who have had problems after they shopped on the Internet. They do not trust in the security of the network. They do not want to read their horoscope before each transaction to see if it is their lucky day!

One solution to this problem is to protect information on networks using cryptography. This means encoding a message with an encryption key before it is sent and decoding it after reception with the corresponding decryption key. But this is not sufficient, even with a very good cryptographic algorithm an intruder can intercept important information which may help him to decrypt messages. Why? Because information about which key was used to encode the message has to be sent to the receiver in order to enable him to use the right decryption key to recover the message. The communication protocols used to "securely" transfer this information can have flaws. So to secure network communications we also need secure and provably correct cryptographic protocols.

One way to design an efficient cryptographic protocol is to use formal methods for the specification and the verification of the protocol. Two essential properties that must be satisfied are the secrecy and the authentication properties. The secrecy property guarantees that an information is kept secret. The authentication property guarantees that a participant can identify the sender of a message. Several methods have been developed to do this job [BAN89, AG98, Pau98, Mon99, GK00, GL00].

In this paper, first we will present two of those methods, [Pau98] and [GK00]. Both approaches reason on the traces of protocols runs. Then we will explain the advantages of combining these two methods and show how they can be combined. Finally, we will introduce our future work.

# 2 Two formal verification techniques

## 2.1 The inductive approach

L. C. Paulson [Pau98] has developed a method based on the proof by induction. In his technique, the protocols are modelled by the set of all possible traces (a trace is the list of all the messages exchanged during a protocol run) and the assumption that there is an intruder or bad agent in the network. This intruder, based on the Dolev and Yao model [DY83], has access to all traces, can decrypt messages if he has captured the appropriate decryption keys and finally he can build and send fraudulent messages if he has an appropriate encryption keys. The properties that the protocol must satisfy are proved by induction on all possible traces that the protocol can generate. That means that after each protocol's step we verify if the properties are satisfied.

This method was implemented in Isabelle[1] theorem prover and used to verify the Needham-Schroeder [Pau98], Kerberos [BP97, BP98], TLS [Pau99] and some other protocols. The proofs of these protocols are available on the Isabelle's website[2].

To be more precise, for each protocol this method is used to verify:

- Possibility properties, which assure that message formats agree from one step to the next.
- Forwarding lemmas, which assure that the intruder will not learn anything new by seeing a message that an agent forwards and can not decrypt.
- Regularity lemmas, which assure that the intruder can not ever get hold of any agent's items, excluding the bad agents.
- Unicity theorems, which assure that session keys and timestamps uniquely identify their message.
- Secrecy theorems, which assure that if the intruder holds some session keys, he can not use them to reveal others.
- Authenticity theorems, which assure the authenticity of a session key.

## 2.2 The approximation technique

T. Genet and F. Klay [GK00] have developed a method for the analysis of cryptographic protocols based on tree automata, term rewriting systems (TRS) and an approximation technique. They use tree automata to model the networks and a term rewriting system to formalize the protocol's steps. Like in the inductive approach, there is an intruder in the network (this intruder has the same abilities as the intruder in the Paulson's method).

To verify the secrecy or authentication property, first they compute a superset of all the possible configurations of the network from the network's initial configuration (all the agents in the networks want to communicate with each other) with the term rewriting system and an extended version of the approximation technique introduced in [Gen98]. Then they use an automaton to model the negation of the property to be verified. Finally, they check the intersection of these automata. If the intersection is empty, they conclude that the protocol is safe otherwise they know that the protocol is insecure.

This method has been used to verify the Needham-Schroeder [GK00] and the Wide-Mouthed-Frog protocols (contact the authors for more information about this proof). A library (Timbuk[3]) for OCAML has been developed to support the tree automata and the approximation.

[1] http://isabelle.in.tum.de/index.html
[2] http://www.cl.cam.ac.uk/Research/HVG/Isabelle/library/HOL/Auth/
[3] http://www.irisa.fr/lande/genet/timbuk/index.html

## 3 Combining Isabelle and Timbuk

Now that we have provided a basic understanding of these two methods for verifying cryptographic protocols let us see why we would wish to combine these methods and how we can do that.

### 3.1 Why combine these two approaches?

By combining these two approaches, we hope to save time in the verification of protocols and simplify the proofs.

The inductive approach is a good method of verification, we can verify several properties. But in the inductive approach the secrecy and authenticity properties/theorems are very difficult to prove (to have an idea look the proof of the Needham-Schroeder protocol[4]). The proofs require an experienced user to inject the right lemma at the right time to make the proofs converge.

On the other hand, with the approximation techniques we can do a quick and semi-automatic (we only enter the TRS, the approximation function, the initial automaton) verification of those two properties. And we don't need to be an expert to write the TRS, the initial automaton and the approximation function. But with this method, if the properties are not satisfied, we must use another technique like the inductive approach to find the flaw.

### 3.2 How to combine?

Our idea is to combine the approximation and the induction approaches to exploit the advantages of each method. So:

1. We use the Timbuk library to verify the secrecy and authentication properties.
2. If the properties are satisfied, we used those results as axioms in the inductive method. If the properties are not satisfied, we use the inductive approach to find the flaw.
3. We use the inductive approach to prove the remainder properties (the proofs of those properties are highly automatic [Pau98]).

Because, the inductive approach is used as main method of verification, we'll use the specification file for Isabelle to generate the input file for Timbuk. In the rest of this paper we will see how an automatic program can be used to do this job (we keep the formalisms used in [Pau98] and [GK00]).

So first, let see what are the formats of the input files, then how we can build the Timbuk file from the Isabelle file using the Needham-Schroeder protocol as an example. So before our example, let us review the Needham-Schroeder protocol given in the Figure 1. In this protocol, two agents , Alice and Bob, want to establish a secure communication using a public keys infrastructure.

[4] http://www.cl.cam.ac.uk/Research/HVG/Isabelle/library/HOL/Auth/

| | |
|---|---|
| | Alice initiates a protocol run, sending a nonce Na and her name A to Bob. |
| Message 1: | A $\Longrightarrow$ B : $\{Na, A\}_{Kb}$ |
| | Bob responds to Alice's message with a further nonce Nb. |
| Message 2: | B $\Longrightarrow$ A : $\{Na, Nb, B\}_{Ka}$ |
| | Alice proves her existence by sending Nb back to Bob. |
| Message 3: | A $\Longrightarrow$ B : $\{Nb\}_{Kb}$ |

Figure 1: The Needham-Schroeder protocol

### 3.2.1 The formats

**3.2.1.1 Isabelle** In the Isabelle's file, we have the protocol's steps and three other operations:

- Nil, the initial trace of the protocol that is empty;
- Fake that model what the intruder can say;
- Oops that model the lost of information.

These three operations are not important to produce the Timbuk input file, so we will only look in more details the format of the protocol's steps.

For each step we find the preconditions to the execution of the operation and the step itself.

Consider the Isabelle file for the Needham-Schroeder protocol (Figure 2).

To have a better understanding of the Isabelle formalism, we give some explanations of the second step, *NS2*, of the protocol:

- $evs2$ is the trace of the protocol;
- *NB* is a nonce that has not been used before;
- "Says A' B {|Crypt (pubK B) {|Nonce NA, Agent A|}|}" is a message sent from A to B that is to be added to the trace;
- "Says B A {|Crypt (pubK A) {|Nonce NA, Nonce NB, Agent B|}|} # evs2" is the trace after the operation. The message "Says B A {|Crypt (pubK A) {|Nonce NA, Nonce NB, Agent B|}|}" is added to the trace $evs2$.

**3.2.1.2 Timbuk** In the Timbuk input file, we find a term rewriting system, a tree automaton and the approximation.

```
        (*Alice initiates a protocol run, sending a nonce to Bob*)
NS1:    "[ evs1 : ns_public; Nonce NA ~: used evs1 ] ==>
        Says A B {|Crypt (pubK B) {|Nonce NA, Agent A|}|} #
        evs1 : ns_public"

        (*Bob responds to Alice's message with a further nonce*)
NS2:    "[ evs2 : ns_public; Nonce NB ~: used evs2;
        Says A' B {|Crypt (pubK B) {|Nonce NA, Agent A|}|} : set evs2 ==>
        Says B A {|Crypt (pubK A) {|Nonce NA, Nonce NB, Agent B|}|} #
        evs2 : ns_public"

        (*Alice proves her existence by sending NB back to Bob.*)
NS3:    "[ evs3 : ns_public;
        Says A B {|Crypt (pubK B) {|Nonce NA, Agent A|}|} : set evs3;
        Says B' A {|Crypt (pubK A) {|Nonce NA, Nonce NB, Agent B|}|}:
        set evs3] ==> Says A B {|Crypt (pubK B) {|Nonce NB|}|} #
        evs3 : ns_public"
```

Figure 2: Inductive specification of the Needham-Schroeder protocol

*1. x initiates a communication with y:*
goal(x,y) -> U(LHS, mesg(x, y, encr(pubkey(y), x, cons(N(x, y), x))))

*2. u answers to NS1 by NS2:*
mesg(x, u, encr(pubkey(u), z, cons(v, x2))) -> U(LHS, mesg(u, x2, encr(pubkey(x2), u, cons(v, cons(N(u, x2), u)))))

*3. y answers to NS2 by NS3:*
mesg(x, y, encr(pubkey(y), z2, cons(N(y, z), cons(u, z)))) -> U(LHS, mesg(y, z, encr(pubkey(z), y, u)))

*4. after he received NS2, y believes that he has initiated a communication with z but he talks with z2:*
mesg(x, y, encr(pubkey(y), z2, cons(N(y, z), cons(u, z)))) -> U(LHS, c_init(y, z, z2))

*5. after he received NS3, y believes that he talks to z but he talks with z2:*
mesg(x, y, encr(pubkey(y), z2, N(y, z))) -> U(LHS, c_resp(y, z, z2))

Figure 3: Rules for the Needham-Schroeder protocol

**The term rewriting system (TRS)** is used to model the protocol's step (Figure 3) and all the operation which can be done by the intruder on the trace (Figure 4).

```
U(cons(x, y), z) -> U(LHS, add(x))

U(cons(x, y), z) -> U(LHS, add(y))

U(encr (pubkey(agt(o)), y, z), u) -> U(LHS, add(z))

U(encr (pubkey(agt(s(x))), y, z), u) -> U(LHS, add(z))

U(mesg(x, y, z), u) -> U(LHS, add(z))

add(x) -> x

U(x, U(y, z)) -> U(U(x, y), z)

U(U(x, y), z) -> U(x, U(y, z))

U(x, y) -> U(y, x)
```

Figure 4: Rules for the intruder's capacities

**The tree automaton** models the initial configuration of the network and all the agents who want to communicate with each other (c.f. *Part 1* Figure 5). The automaton models also the initial knowledge of the intruder and what he can do (c.f. *Part 2* Figure 5).

**The approximation** defines for each rule of the term rewriting system the approximation used to reduce the number of cases to deal with. In particular, there is no distinction between agents that are not explicitly specified. In the example all the messages sent and received between any agent, except Alice and Bob, are collapsed together. So we only have nine categories of messages for each protocol's step:

- from Alice to Bob;
- from Bob to Alice;
- from Alice to Alice;
- from Bob to Bob;
- from undefined agents to Alice;
- from undefined agents to Bob;
- from Alice to undefined agents;
- from Bob to undefined agents;
- all the remaining cases.

| | | |
|---|---|---|
| Automaton nspk | | |
| States | q0 q1 q2 q3 q4 q5 q6 q13 | |
| Final States | q13 | |
| Prior | | |
| | o -> q0 | s(q0) -> q0 |
| | A -> q1 | B -> q2 |
| | agt(q0) -> q3 | agt(q1) -> q4 |
| | agt(q2) -> q5 | |
| | | |
| Transitions | | |
| *Part 1* | U(q13, q13) -> q13 | |
| | goal(q4, q5) -> q13 | goal(q5, q4) -> q13 |
| | goal(q4, q4) -> q13 | goal(q5, q5) -> q13 |
| | goal(q3, q3) -> q13 | goal(q4, q3) -> q13 |
| | goal(q3, q4) -> q13 | goal(q5, q3) -> q13 |
| | goal(q3, q5) -> q13 | |
| | | |
| *Part 2* | agt(q0) -> q13 | agt(q1) -> q13 |
| | agt(q2) -> q13 | |
| | mesg(q13, q13, q13) -> q13 | |
| | cons(q13, q13) -> q13 | |
| | pubkey(q3) -> q13 | pubkey(q4) -> q13 |
| | pubkey(q5) -> q13 | |
| | encr(q13, q3, q13) -> q13 | |
| | N(q3, q3) -> q13 | N(q3, q4) -> q13 |
| | N(q3, q5) -> q13 | |

Figure 5: Initial automaton of the Needham-Schroeder protocol

To have an idea of the approximation function, let us see an example from the Needham-Schroeder. In the Figure 6 we have the approximation for the cases "Alice to Bob" and "remaining cases" of the first rule of the TRS. In both case we arrive to "mesg(x,y,q13)->q13" by introducing new states in the term "X" of "mesg(x,y,X)".

Now, we know the formats of the input files of each method (Isabelle and Timbuk). We can define our method for building the Timbuk input file from the Isabelle's file.

## 3.3 How to build

### 3.3.1 The TRS

To generate the TRS, we will use the Isabelle file. To be more precise we will use the file to find the protocol's steps rules.

The first thing to do is to cut off the message from the inductive specification. For example for the Needham-Schroeder, we should have Figure 7.

```
[U(LHS, mesg(q4,q5,encr(pubkey(q5),q4,cons(N(q4,q5),q4))))
-> q13] -> [LHS -> q13 N(q4,q5) -> q16 cons(q16,q4) -> q15
pubkey(q5) -> q14 encr(q14,q4,q15) -> q13
mesg(q4,q5,q13) -> q13]

[U(LHS, mesg(x,q3,encr(pubkey(q3),x,cons(N(x,q3),x))))
-> q13] -> [LHS -> q13 N(x,q3) -> q70 cons(q70,x) -> q69
pubkey(q3) -> q66 encr(q66,x,q69) -> q13
mesg(x,q3,q13) -> q13]
```

Figure 6: Examples of approximation

```
Says A B {|Crypt (pubK B) {|Nonce NA, Agent A|}|}

Says B A {|Crypt (pubK A) {|Nonce NA, Nonce NB, Agent B|}|}

Says A B {|Crypt (pubK B) {|Nonce NB|}|}
```

Figure 7: Information extracted from the Isabelle's file

In the approximation approach characters such as $x$, $u$, $z$, ... (c.f. Figure 3) are used as variables. As we want an automatic transformation of the Isabelle file to the input file for Timbuk, we will keep the characters $a$ and $b$ to refer to Alice (A) and Bob (B), and we'll use characters $a\_i$ with $i$ in the integers for the variables.

So from the information extracted, Figure 7, and with a set of rewriting rules (by example: Says X Y W $\Longrightarrow$ mesg(X,Y,W)), it's easy to generate the first terms (Figure 8)and the second terms (Figure 9) of the TRS for the Needham-Schroeder protocol. Then we put together those information to build the TRS for the protocol's steps (Figure 10).

```
Says A B {|Crypt (pubK B) {|Nonce NA, Agent A|}|}
                ==> mesg(a_5, b, encr(pubkey(b), a_4, cons(a_3, a)))

Says B A {|Crypt (pubK A) {|Nonce NA, Nonce NB, Agent B|}|}
                ==> mesg(a_8, a, encr(pubkey(a), a_7, cons(N(a,b), cons(a_6, b))))

Says A B {|Crypt (pubK B) {|Nonce NB|}|}
                ==> mesg(a_10, b, encr(pubkey(b), a_9, N(b,a)))
```

Figure 8: First terms of the rules

Now to complete our TRS, we add the intruder capacities (Figure 11) which model the Dolev-Yao intruder and are the same for all protocols.

```
Says A B {|Crypt (pubK B) {|Nonce NA, Agent A|}|}
              ==> mesg(a, b, encr(pubkey(b), a, cons(N(a,b), a)))

Says B A {|Crypt (pubK A) {|Nonce NA, Nonce NB, Agent B|}|}
              ==> mesg(b, a, encr(pubkey(a), b, cons(a_0, cons(N(b,a), b))))

Says A B {|Crypt (pubK B) {|Nonce NB|}|}
              ==> mesg(a, b, encr(pubkey(b), a, a_1))
```

Figure 9: Second terms of the rules

```
goal(a, b)
        ==> U(LHS, mesg(a, b, encr(pubkey(b), a, cons(N(a,b), a))))

mesg(a_5, b, encr(pubkey(b), a_4, cons(a_0, a)))
        ==> U(LHS, mesg(b, a, encr(pubkey(a), b, cons(a_0, cons(N(b,a), b)))))

mesg(a_8, a, encr(pubkey(a), a_7, cons(N(a,b), cons(a_6, b))))
        ==> U(LHS, mesg(a, b, encr(pubkey(b), a, a_6)))

mesg(a_8, a, encr(pubkey(a), a_7, cons(N(a,b), cons(a_6, b))))
        ==> U(LHS, c_init(a, b, a_7))

mesg(a_10, b, encr(pubkey(b), a_9, N(b,a)))
        ==> U(LHS, c_resp(b, a, a_9))
```

Figure 10: Rules for the protocol's steps

### 3.3.2 The tree automaton

The initial automaton is the same for all the asymmetric protocols without a sever. All the agents want to communicate with each other, so we just have Figure 5.

### 3.3.3 The approximation

We can split the approximation into:

- the protocol's steps;
- the invariant.

The invariant rules are easy to build because, like the rules for the intruder capacities, they are always the same in the TRS. The Figure 12 shows the invariant rules for the Needham-Schroeder protocol.

The approximation rules of the protocol's steps require a computation from the second terms of the TRS's rules (the terms $U(LHS, ...)$ in the Figure 3). For

U(cons(x, y) , z) $\Longrightarrow$ U(LHS, x)

U(cons(x, y) , z) $\Longrightarrow$ U(LHS, y)

U(encr(pubkey(agt(0)), y, z), u) $\Longrightarrow$ U(LHS, z)

U(encr(pubkey(agt(s(x))), y, z), u) $\Longrightarrow$ U(LHS, z)

U(mesg(x, y, z), u) $\Longrightarrow$ U(LHS, z)

U(x, U(y, z)) $\Longrightarrow$ U(U(x, y), z)

U(U(x, y), z)) $\Longrightarrow$ U(x, U(y, z))

U(x, y) $\Longrightarrow$ U(y, x)

Figure 11: Rules for the intruder capacities

[U(LHS,c_init(x,y,z)) -> q13] -> [LHS -> q13 c_init(x,y,z) -> q13]

[U(LHS,c_resp(x,y,z)) -> q13] -> [LHS -> q13 c_resp(x,y,z) -> q13]

[U(LHS, add(x)) -> q13] -> [LHS -> q13 add(x) -> q13]

[U(U(x,y),y) -> q13] -> [U(x,y) -> q13]

[U(x, U(y,z)) -> q13] -> [U(y,z) -> q13]

Figure 12: Invariant rules

each rule, we have 9 approximations to build by initializing the sender and the receiver to the states:

- q4 and q5;
- q5 and q4;
- q4 and q4;
- q5 and q5;
- q3 and q4;
- q3 and q5;
- q4 and q3;
- q5 and q3;

- _ and q3.

The first term of the approximation functions is obtained by this initialization. As it is shown on the Figure 13 for the first approximation of the first rule of the TRS.

```
U(LHS, mesg(a, b, encr(pubkey(b), a, cons(N(a,b), a)))) -> [U(LHS,
mesg(q4,q5,encr(pubkey(q5),q4,cons(N(q4,q5),q4)))) -> q13]
```

Figure 13: First term of the approximation

The second term of the approximation is obtained with the TRS rule and a set of rewriting rules (by example: mesg(x,y,q13) $\Longrightarrow$ mesg(x,y,q13)->q13 or mesg(x,y,z) $\Longrightarrow$ mesg(x,y,z)->q13 if z=q3 or z=q4 or z=q5). In the Figure 14 we have the example for the first approximation of the first rule.

```
U(LHS, mesg(a, b, encr(pubkey(b), a, cons(N(a,b), a)))) -> [N(q4,q5) -> q14
cons(q14,q4) -> q15 pubkey(q5) -> q15 encr(q15,q4,q14) -> q13 mesg(q4,q5,q13)
-> q13 LHS -> q13]
```

Figure 14: Second term of the approximation

Similar to the TRS, we put together the first terms and the second terms of the approximation to build the final approximation of the protocol's step. We can see the result of the first approximation of the step in the Figure 15.

```
[U(LHS, mesg(q4,q5,encr(pubkey(q5),q4,cons(N(q4,q5),q4)))) -> q13]
-> [N(q4,q5) -> q14 cons(q14,q4) -> q15 pubkey(q5) -> q15 encr(q15,q4,q14)
-> q13 mesg(q4,q5,q13) -> q13 LHS -> q13]
```

Figure 15: An example of the approximation of the first protocol step

Now that we have our Timbuk file, we can use the approximation technique to check the secrecy and the authentication properties. Afterwards we can go back to Isabelle to add the properties as axioms and do the proofs of the other properties. If the approximation technique cannot verify these properties we need to do the inductive proofs of the secrecy and the authentication properties.

# 4 Conclusion

In the literature you can find several examples of combination of model-checking and theorem proving [McM99, DF94, Gor99, Gor00, RSS95, AJS99, AJS98, MN, Ham99]. As far as we can ascertain this is the first developed for the verification of cryptographic protocols.

In this paper, we have defined the foundation of our approach; namely using the *approximation* to check if the secrecy and authentication properties are satisfied by protocols. If they are, we use those results as axioms in the *inductive approach*. If they're not, we use the *inductive approach* to find the flaw. Finally, we used the *inductive approach* to prove other properties.

Currently the implementation of an automatic tool to generate Timbuk input files from Isabelle specifications is limited to the asymmetric protocols without a trusted third party. The rules used to build the TRS's rules for the protocol's steps are not sufficient to verify all the possible protocols. To have a complete method we need to add the rules for supporting the symmetric protocols with or without a server and the asymmetric protocols with a server. The last remark is also true for the rules that model the intruder capacities.

Our method does not try to improve both approaches or tools, but to combine the strengths of both approaches to simplify the work of verification and to keep the verification as complete as possible.

The next step is to develop an implementation of this approach. Initially it will be tested with the basic protocols such as Otway-Rees [ORt87] and TLS [Gro96b] and if the results are conclusive we will then analyze industrial protocols such as SET [Gro96a].

## References


[AG98] M. Abadi and A. D. Gordon. A calculus for cryptographic protocols: The spi calculus. Technical report, DIGITAL, Systems Research Center, N 149, January 1998.

[AJS98] Mark D. Aagaard, Robert B. Jones, and Carl-Johan H. Seger. Combining theorem proving and trajectory evaluation in an industrial environment. In *Design Automation Conference (DAC)*, pages 538–541. ACM/IEEE, July 1998.

[AJS99] Mark D. Aagaard, Robert B. Jones, and Carl-Johan H. Seger. Lifted-FL: A Pragmatic Implementation of Combined Model Checking and Theorem Proving. In *Theorem Proving in Higher Order Logics (TPHOLs99)*, number 1690 in Lecture Notes in Computer Science, pages 323–340. Springer-Verlag, 1999.

[BAN89] M. Burrows, M. Abadi, and R. Needham. A logic of authentication. Technical report, DIGITAL, Systems Research Center, N 39, February 1989.

[BP97] G. Bella and L. Paulson. Using isabelle to prove properties of the kerberos authentication system, 1997.

[BP98] Giampaolo Bella and Lawrence C. Paulson. Kerberos version IV: Inductive analysis of the secrecy goals. In J.-J. Quisquater, editor, *Proceedings of the 5th European Symposium on Research in Computer Security*, pages 361–375, Louvain-la-Neuve, Belgium, 1998. Springer-Verlag LNCS 1485.

[DF94] J. Dingel and T. Filkorn. Model checking for infinite state systems using data abstraction, 1994.

[DY83] D. Dolev and A. Yao. On the security of public-key protocols. *IEEE Transactions on Information Theory, 2(29)*, 1983.

[Gen98] Thomas Genet. Decidable approximations of sets of descendants and sets of normal forms. In *RTA*, pages 151–165, 1998.

[GK00] Thomas Genet and Francis Klay. Rewriting for cryptographic protocol verification. In *CADE: International Conference on Automated Deduction*, 2000. http://citeseer.nj.nec.com/genet99rewriting.html.

[GL00] Jean Goubault-Larrecq. A method for automatic cryptographic protocol verification (extended abstract). In *Proc. Workshop on Formal Methods in Parallel Programming, Theory and Applications (FMPP-TA'2000)*, number 1800 in Lecture Notes in Computer Science, pages 977–984. Springer-Verlag, 2000.

[Gor99] Mike J. C. Gordon. Programming combinations of deduction and bdd-based symbolic calculation. Technical report, University of Cambridge Computer Laboratory, N 480, December 1999.

[Gor00] Mike J. C. Gordon. Reachability programming in HOL98 using BDDs. In *The 13th International Conference on Theorem Proving and Higher Order Logics*. Springer-Verlag, 2000.

[Gro96a] SET Working Group. Set$^{TM}$ specification, books 1,2 and 3. 1996. http://www.setco.org/set_specifications.html.

[Gro96b] TLS Working Group. The tls protocol version 1.0. 1996. http://www.ietf.org/html.charters/tls-charter.html.

[Gui00] C. Guillemin. 3.7 millions de numéros de cartes bancaires piraté sur egghead.com ? *ZDNet FRANCE*, 12 2000. http://www.zdnet.fr.

[Ham99] Tobias Hamberger. Integrating theorem proving and model checking in isabelle/ioa. Technical report, T.U. Munich, Agust 1999.

[McM99] K. L. McMillan. A methodology for hardware verification using compositional model checking. Technical report, Cadance Berkeley Labs, April 1999.

[MN] Olaf Müller and Tobias Nipkow. Combining model checking and deduction for I/O-automata. pages 1–12.

[Mon99] David Monniaux. Abstracting cryptographic protocols with tree automata. Lecture Notes in Computer Science, pages 149–163. Springer-Verlag, 1999.

[ORt87] D. Otway, O. Rees, and E. timely. mutual authentication, 1987.

[Pau98] L. C. Paulson. The inductive approach to verifying cryptographic protocols. *Journal of Computer Security*, 6, 1998. http://www.cl.cam.ac.uk/users/lcp/papers/protocols.html.

[Pau99] Lawrence C. Paulson. Inductive analysis of the Internet protocol TLS. *ACM Transactions on Information and System Security*, 2(3):332–351, 1999.

[RSS95] S. Rajan, N. Shankar, and M.K. Srivas. An integration of model-checking with automated proof checking. In Pierre Wolper, editor, *Computer-Aided Verification, CAV '95*, volume 939 of *Lecture Notes in Computer Science*, pages 84–97, Liege, Belgium, June 1995. Springer-Verlag.

[Sul00] B. Sullivan. Credit card hacker has struck before. *MSNBC*, 12 2000. http://www.msnbc.com.