Automatic Verification of Cryptographic Protocols in First-Order Logic

JIHONG HAN, ZHIYONG ZHOU AND YADI WANG
Zhengzhou Information Science and Technology Institute
No.12 Shangcheng East Road, Zhengzhou, Henan
CHINA

Abstract: - In this paper, a new first-order logical framework and method of formalizing and verifying cryptographic protocols is presented. From the point of view of an intruder, the protocol and abilities of the intruder are modeled in Horn clauses. Based on deductive reasoning method, secrecy of cryptographic protocols is verified automatically, and if the secrecy is violated, attack scenarios can be presented through back-tracing. The method has been implemented in an automatic verifier, many examples of protocols have been analyzed in less then 1s.

Key-Words: - Cryptographic protocol, First-order logic, Automatic verification, Secrecy, Attack scenarios, Deductive reasoning

1 Introduction

A cryptographic protocol is a precisely defined sequence of communication and computation steps using cryptographic mechanism, its aim is ensuring the security of the transaction and communication in network or distributed systems. The rapid extending of the internet causes a growing need for cryptographic protocols, but it is well known that the design of such protocols is difficult and error-prone. Therefore, it is necessary to study formal analysis methods and automatic verification tools for the cryptographic protocols. Researchers have adopted many theories and techniques to build automatic verification tools. The theories are mainly derived from logic[1,3,6], algebra[7,8,9], complexity theory[11,15] and automata theory[13], the popular techniques are model checking[14,16,17] and theorem proving[18,20, 3]. Verifiers based on model checking suffer from the problem of the state space explosion, while verifiers based on theorem proving usually need manual intervention.

In this paper, we present a new formal approach for automatic verification of cryptographic protocols. This approach is fully automatic and terminable. The main contributions of the paper are: a general framework of formalizing cryptographic protocol and abilities of the intruder, a practical solving algorithm based on automatic reasoning, and a simple method to find the attack scenarios.

2 Related Work

The logic-based approach has been proved to be particularly well-suited for automation. Its early application in analysis of cryptographic protocol is Millen's Interrogator[19]. Using Prolog, Interrogator searches for instantiations of a goal signifying the intruder's knowledge of specified data which would lead to an insecure state. Interrogator can find protocol flaws successfully, but the search time varies significantly depending on the precise format of the protocol specification and the amount of information about the insecure state, and the search heuristics may prevent some flaws from being noticed.

Combining the benefits of the finite state analysis and the inductive method, C.Weidenbach develops the automated theorem prover SPASS[3], in which the protocols are formalized in monadic first-order Horn logic. Based on sort resolution, he also proves that parts of the used first-order fragments can be decided. By taking Neumann-Stubblebine protocol as an executing instantiation, he shows that SPASS can automatically prove security properties of the protocol and detect potential errors of an implementation.

Blanchet's verifier[12] is another efficient automatic tool. It is based on logic programming and abstractions, the protocol and attacker's abilities are specified by means of Prolog rules, cryptographic primitives are represented by constructors and destructors, fresh values are modeled as functions of previously received messages of the principal. An abstraction is made by forgetting the number of times a message appears and remembering the fact that it has appeared. The state of principals is not explicitly maintained, principals' rules are not ordered into runs, the same rule can be applied more than once and in different order with respect to the original protocol. These approximations may lead to giving "false attacks" despite they are rather rare. By a two-phase algorithm based on resolution and depth-first backward search, the verifier can prove the secrecy of cryptographic protocols.

3 Modeling cryptographic protocols

Modeling the cryptographic protocol is the first step of protocol verification. We use first-order theory to formalize cryptographic protocols and intruders. The final form of our method is similar to Blanchet's, but the modeling process is more regular and impersonal.

We assume that messages transmitted by each principal can be received by the intruder, and messages received by each principal can be known by the intruder. We test the security of cryptographic protocols at the standpoint of the intruder. The intruder holds some initial knowledge, he can get information by observing the communications between principals, and gain knowledge by computing on the basis of known information. So, the protocol representation includes three parts: initial knowledge of the intruder, message exchange of the protocol itself, and computation abilities of the intruder.

3.1 Syntax of protocol representation

Our cryptographic protocol theory is expressed by terms, predicate and implication rules.

The terms represent messages that are exchanged between participants of the protocol. Constant symbols, variables, and function symbols are used to build terms. By convention, we use upper case to denote constants, lower case to denote variables, function and predicate begin with an upper letter.

In order to deduct efficiently, we introduce term types to specify the structure of messages and perform type check before computation. Term types have the following structure.

$\tau, \tau'::=msg$	message
$\|princ$	principal
$\|nonce$	nonce
$\|ident$	identifier
$\|key$	key
$\|compmsg$	compound message
$key::=shK$	symmetric session key
$\|pubK$	public key
$\|privK$	private key
$\|longtK$	long-term key
$compmsg::=H[\tau]$	hash value
$\|SC[\tau, \tau']$	symmetric cipher
$\|AC[\tau, \tau']$	asymmetric message cipher
$\|SN[\tau, \tau']$	signature
$\|T[\tau, \tau']$	tuple

A term has type τ can be represented by t: τ.

Constant symbol can be A, B, S, I, … which represent principals and have type *msg*, or terms which represent keys, nonces, identifiers etc. A variable can represent any term. The typical function symbols appear as below:

$(m_1,\ldots,m_n)$: tuple of messages, where m_i(i=1,…n) has type *msg*.

Host(x): identity of x: *princ.*

Pk(x): public key of x: *princ.*

Sk(x): private key of x: *princ.*

E(m,k): encrypt m: *msg* with k: *shK*

PE(m,pk): encrypt m: *msg* with pk: *pubK*

SG(m, sk): sign m: *msg* with sk: *privK*

H(m): hash of m: *msg*

Hk(m, k): keyed hash of m under key k

X(m, n): bitwise exclusive-or of m and n

Inc(m): addition of m by 1

The function value have certain types too.

The predicate has only one form of *Intr*(M) which means "intruder knows M". The implication rules are used to formalize the protocol steps and intruder's computation abilities.

3.2 Protocol specification

A protocol is composed of some communication steps executed by protocol principals. Every principal plays a different role in the protocol. Typically, the roles can be protocol initiator and responder, there often exits a trusted third party in the protocol too. We use axioms to depict the communicating actions of each role. The intruder can know all communications between the protocol roles. When a role receives a message, the intruder also know it, on the premise of receiving messages, the role would generate a new message and transmit it, the new message can be known by intruder too. The predicate corresponding to role's message receiving can be affiliated to the axiom by logical connective $\wedge$, and the predicate corresponding to role's message transmitting can be the conclusion of the implication relation. The universal quantification $\forall$ is used to eliminate the limitation for protocol runs, and existential quantification $\exists$ is used to denote generating of key or nonce, and through renaming and consistency check, ensure the refreshness and boundlessness of the new value. For example, for the Denning-Sacco key distribution protocol which can be expressed as follow:

1. A→S: *A, B*
2. S→A: *Pk*(*A*), *Pk*(*B*)
3. A→B: *Pk*(*A*), *Pk*(*B*), *PE*(*SG*(*k*, *Sk*(*A*)), *Pk*(*B*))

4. B→A: $E(s, k)$

where A and B are two principals whose goal is to establish a shared private key *k*. *Pk* (A) and *Pk* (B) are public keys of A and B respectively, they are contained in their digital certificates distributed by the trusted server S. Step 4 is not really part of the protocol, we include it for the purpose of secrecy verification.

The input of protocol specification in our first order logic can be shown as below:

Φ_i : $\forall$A, B $(Intr(Host(A), Host(B)) \wedge Intr(Pk(A), Pk(B)) \rightarrow \exists k\ (Intr(pk(A), pk(B), PE\ (SG(k, Sk(A)), Pk(B)) \wedge Intr(E(s,k))))$ (1)

Φ_r: $\forall$A, B $\forall k\ (Intr(Pk(A), Pk(B), PE\ (SG(k, Sk(A)), Pk(B))) \rightarrow Intr(E(s,k)))$ (2)

Φ_s : $\forall$A, B $(Intr(Host(A), Host(B)) \rightarrow Intr(Pk(A), Pk(B)))$ (3)

The axiom Φ_i corresponds to the initiator role, Φ_r corresponds to the responder role, and Φ_s corresponds to the trusted server.

For brevity, we transform the axioms into Horn clauses and remove the redundant facts, gain a set of clauses:

$\forall$A, B $(Intr(Host\ (A), Host\ (B)))$ (4)

$\forall$A, B $\exists k\ (Intr(Pk\ (A), Pk\ (B)) \rightarrow Intr(PE\ (SG(k, Sk\ (A)), Pk\ (B))))$ (5)

$\forall$A, B $\exists$ k $(Intr(Pk\ (A), Pk\ (B)) \rightarrow Intr(E(s,k)))$ (6)

$\forall$A, B $\forall$k $(Intr(Pk\ (A), Pk\ (B), PE\ (SG(k, Sk\ (A)), Pk\ (B))) \rightarrow Intr(E(s,k)))$ (7)

$\forall$A, B $(Intr(Host\ (A), Host\ (B)) \rightarrow Intr(Pk\ (A), Pk\ (B)))$ (8)

Then, we use a new name to replace the existential quantifier by skolemization, for instance, k[*x*] stands for generating a new name k depending on *x*.

The initiator roles can not restrict their sent messages to be accepted only by someone, and responder roles can not know where their received messages really come from. Assuming A and B are legal principals and they are willing to talk to any principals, we can transform (4)—(8) by removing the existential quantifiers and universal quantifiers, making resolution and eliminating the rules which are implicated by other rules[12], finally we gain formulae:

$Intr\ (Host\ (x))$ (9)

$Intr\ (Pk\ (x)) \rightarrow Intr\ (PE\ (SG(k[Pk\ (x)], Sk\ (A)), Pk\ (x)))$ (10)

$Intr\ (Pk\ (x)) \rightarrow Intr\ (E(s, k[Pk\ (x)]))$ (10)

$Intr\ (PE\ (SG(k, Sk\ (A)), Pk\ (B))) \rightarrow Intr\ (E(s,k))$ (12)

$Intr\ (Host\ (x)) \rightarrow Intr\ (Pk\ (x))$ (11)

In order to construct the attack trace later, each Horn clause above is associated with a message number according to its conclusion, and the message number can be passed to the new rule inferred by resolution. We can notice that the rule(10) and (12) are just the same rules representing the protocol in [12]. From the instantiation of (9) and (13), we can obtain the intruder's initial knowledge in [12].

3.3 The intruder abilities

Following Dolev-Yao Model, the protocol is executed in the presence of an intruder that can intercept all messages, generate new messages from the messages he has received, and send messages whenever he wants to do so. An intruder can gain information either by passive means, such as eavesdropping and taking advantage of public information, or by active means such as encrypting, decrypting, reconstructing and replaying messages, impersonating other principals. So the computation abilities of the intruder can be represented as below:

Compose: $Intr\ (n_1) \wedge Intr\ (n_2) \wedge \ldots \wedge Intr\ (n_k) \rightarrow Intr\ (n_1, n_2, \ldots n_k)$ for every $k \geq 0$ (14)

Decompose: $Intr\ (n_1, n_2, \ldots n_k) \rightarrow Intr\ (n_i)$ for every $k \geq i \geq 0$ (15)

Encrypt: $Intr\ (k) \wedge Intr\ (x) \rightarrow Intr\ (E(x, k))$, for k: *key*, x:*msg* (16)

Decrypt: $Intr\ (E(x, k)) \wedge Intr\ (k) \rightarrow Intr\ (x)$, for k: *key*, x:*msg* (17)

Public Encrypt: $Intr\ (x) \wedge Intr\ (Pk\ (y)) \rightarrow Intr\ (PE(x, Pk\ (y)))$, for x:*msg*, y:*princ* (18)

Public Decrypt: $Intr\ (PE(x, Pk(y))) \wedge Intr\ (Sk(y)) \rightarrow Intr\ (x)$, for x:*msg*, y:*princ* (19)

Sign: $Intr\ (x) \wedge Intr\ (Sk(y)) \rightarrow Intr\ (SG(x, Sk(y)))$, for x:*msg*, y:*princ* (20)

Check Sign: $Intr\ (SG(x, Sk(y))) \wedge Intr\ (Pk(y)) \rightarrow Intr\ (x)$, for x:*msg*, y:*princ* (21)

Hash: $Intr\ (x) \rightarrow Intr\ (H(x))$ (22)

Key Hash: $Intr\ (x) \wedge Intr\ (k) \rightarrow Intr\ (H(x, k))$ (23)

Generate key: $Intr\ (Sk\ (x)) \rightarrow Intr\ (Pk\ (x))$, for x, y:*princ* (24)

Exclusive-or: $Intr\ (x) \wedge Intr\ (y) \rightarrow Intr\ (X(x, y))$ (25)

The intruder's abilities also contain his initial knowledge, such as:

Secret Key: $Intr(Sk(I))$ (12)

Public Key: *Intr*(*Pk*(I)) (13)

Here, *Sk*(I) and *Pk*(I) are the secret key and public key of the intruder. (26) and (27) denote that the intruder knows his own secret key and public key. In addition, the intruder can gain some initial information about the protocol and principals, such as public keys of others.

4 Verifying the secrecy of cryptographic protocol

We adopt deductive reasoning method to verify the secrecy property of cryptographic protocols. If the intruder can not obtain any information about message M through interacting with the protocol, we say that the protocol keeps the secrecy of M.

4.1 Verifying algorithm

we consider the secrecy property as a goal, and check whether it can be inferred from the known rules. The known rules form a rule base B containing the Horn clauses of protocol description and abilities of the intruder. If the goal can be inferred from the base, the sequence of rules applied will lead to the description of an attack scenario.

Definition 1 (Rule activation) Let F be a fact, A rule is activated if its conclusion is unifiable with F.

Definition 2 (Provability) Let F be a closed fact, and B be a set of Horn clauses. F is provable from B, if and only if there exists a finite proof tree defined as follow:

1. Its root node is F.
2. Its parent nodes are all conclusions of activated Horn clause. A parent node and his son nodes denote a rule, parent node is the conclusion and son nodes are premises.
3. Its leaf nodes are all closed facts which are contained in B or unifiable with facts in B.

Theorem If Intr(M) is not provable from the initial facts and rules of our notation, then M is secret.

Prove The functions of cryptographic primitives in our notation correspond to the constructor in [12], and rules of intruder's abilities implicate the function of destructor. Rules representing the protocol are logical equivalence with the model of Selinger[10], we make a transform on the assumption that principal A is the legal initiator and B is the legal responder, this make A cannot play the role of B and vice-versa. This is one of the remarks of Branchet[12] too. By giving explicit names to subformulas, Selinger' model can be translated to a linear logic model[2], from a linear logic model which is simplified with respect to the model[2], Branchet's logic programming rules can be inferred, so our notation embodies Branchet's prolog rules. Thus we can use the idea of secrecy of Branchet's[12], if Intr(M) is not provable from the logic programming rules of our notation, it is not derivable from the Branchet's notation, according to theorem 4 in [12], M is secret.

We use the converse deductive reasoning method to prove the secrecy, the prove tree is established by using Generalized Modus Ponens rule backwards. The simplified deductive algorithm can be described as below:

```
Derive (B,F) returns a series of applied rules and a set of substitutions
    Trace_rule = {}
    Derive_list(B, [F],{})
End
    Derive_list(B, qlist, curr_subst, Trace_rule)
    If qlist is empty then return curr_subst, Trace_rule
    Sub={}
    Sort(qlist)
    Q←head(qlist)
    for each atom sentence $P_i$' in B such that $\sigma_i \leftarrow$ Unify($P_i$', Q) succeeds and $\sigma_i$ is in consistent with curr_subst do
        Trace_rule = Trace_rule □ $P_i$'
        Sub=sub□ append(curr_subst, $\sigma_i$)
        end
    for each noncycle rule $R_i = P_1 \wedge P_2 \wedge \ldots P_n \rightarrow Q_i$ in B such that $\sigma_i \leftarrow$ Unify($Q_i$, Q) succeeds and $\sigma_i$ is in consistent with curr_subst do
        Trace_rule = Trace_rule ∪ $R_i$'
            sub= sub ∪ Derive_list(B, ($\sigma_i P_1$,
            $\sigma_i P_2, \ldots \sigma_i P_n$), append(curr_subst, $\sigma_i$) ,
            Trace_rule)
            end
    return the union of Derive_list(B,
            tail(qlist), curr_subst, Trace_rule)
            for each substitution ∈ sub
    end
```

A cycle rule is the rule which is activated just after their reverse computing rule has been activated. The Compose and Decompose, Encrypt and Decrypt, Public Encrypt and Public Decrypt, Sign and Check Sign rules are all mutual reverse. If the two reverse rules are activated in succession, it will cause non-termination. So these cases should be avoided. Otherwise, the search space is finite.

For the Denning-Sacco key distribution protocol described above, we can find a proof tree shown in fig.1.

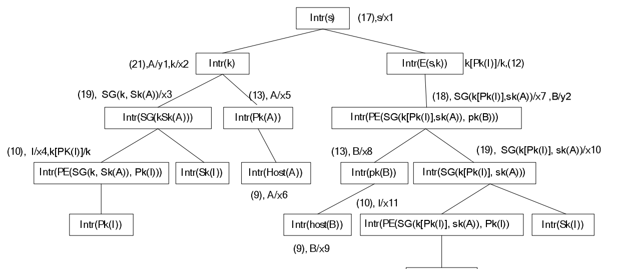

Fig. 1. A proof tree for secrecy of Denning-Sacco protocol

4.2 Finding attacks

From the proof tree for the secrecy of Denning-Sacco protocol, an intruder can know s which should be secret between principals A and B, if he knows the names of the principals and his own keys. We can find the attack trace by checking the recorded clause and substitution in reverse direction with that they are applied in the unification. Each clause associates a set of attack steps, the intruder obtains information by these steps, if he need some unknown information, a new session should be established. Each session has an identifier. The attack scenario is shown in table 1.

Table 1. A attack scenario of Denning-Sacco key protocol

Rule	Substitution	Attack scenario
Intr(*Pk* (x11)) → *Intr*(*PE* (SG(k[*Pk* (x11)], *Sk*(A)), *Pk* (x11)))	I / x11	1.A → I: A, I and I → S: A, I 2.S → I: *Pk* (A), *Pk* (I) and I → A: *Pk* (A), *Pk* (I) 3.A → I: *Pk* (A), *Pk* (I), *PE* (*SG*(k[*Pk* (I)], *Sk* (A)), *Pk* (I))
Intr(*PE*(x10, *Pk* (I))) ∧ *Intr*(*Sk*(I)) → *Intr*(x10)	*SG*(k[*Pk* (I)], *Sk* (A)) /x10	Decrypt *PE* (*SG*(k[*Pk*(I)], *Sk* (A)), PK), gain *SG*(k[*Pk*(I)] , *Sk* (A))
Intr(*Host*(x9))	B/x9	1'.A → I: A, B and I → S: A, B
Intr(*Pk* (B))	B/x8	2'.S → I: *Pk* (A), *Pk* (B) and I → A: *Pk* (A), *Pk* (B)
Intr(x7) ∧ *Intr*(*Pk*(y2)) → *Intr*(*PE*(x7, *Pk* (y2)))	*SG*(k[*Pk*(I)], *Sk* (A)) /x7 B/y2	Encrypt SG(k[*Pk*(I)] , *Sk* (A)) with *Pk* (B) Gain *PE* (*SG*(k[*Pk*(I)], *Sk* (A)), *Pk* (B))
Intr (*PE* (*SG*(k, sk(A)), *Pk* (B))) → *Intr*(*E*(s,k))	k=k[*Pk*(I)]	3'.A → I: *Pk* (A), *Pk* (B),*PE*(*SG*(k[*Pk* (B)], *Sk* (A)), *Pk* (B)) Replace *PE*(*SG*(k[*Pk* (B)], *Sk* (A)), *Pk* (B)) with *PE* (*SG*(k[*Pk*(I)], *Sk*(A)), *Pk* (B)) I → B: *Pk* (A), *Pk* (B), *PE* (*SG*(k[*Pk*(I)],*Sk*(A)), *Pk* (B)) B → I: *Intr*(E(s, k[*Pk*(I)])) I → A: *Intr*(E(s, k[*Pk*(I)]))
Intr(*SG*(x2, *Sk*(y1))) ∧ *Intr*(*Pk* (y1)) → *Intr*(x2)	A/y1, k= k[*Pk*(I)]/x2	Check sign of *SG*(*k*[*Pk*(I)] , *Sk*(A)), gain k[*Pk*(I)]
Intr(*E*(x1, k)) ∧*Intr*(k) → *Intr*(x1)	s/x1, k=k[*Pk*(I)]	Decrypt *E*(s, k[*Pk*(I)]), gain s

In our discussion, we have ignored the timestamp in the message *PE*(*SG*((k,TA), skA), pkB) of step 3, so this attacker only succeeds in the condition of timestamp is fresh.

5 Experimental results

Based on our first order logic theory for the cryptographic protocols, we implement a prototype of protocol verifier in Visual C++, and perform tests on a Pentium Ⅳ 1.86GHz, 512MB RAM, under Windows XP/2000/2003. By 22 protocols' verification, our method is proved to be efficient. These protocols are NSPK, NSSK, Otway-Rees, Wide Mouthed Frog, Yahalom, Denning-Sacco, needham, Andrew secure RPC, Carlsen's Secret Key Initiator and ISO four-Path Authentication protocol and their transmutations, The minimum time is 110ms(for Denning-Sacco protocol), maximum time is 1547ms(for NSSK protocol), most protocol can be verified in 1s. The number of rules approximates the Blanchet's. However, the visualizing of the first order proof search is the main superiority over other verifier.

6 Conclusion

Using first order logic to verify cryptographic protocols is an efficient and practical approach. We choose to study protocol representation and reasoning using first order logic because it is by far the most studied and best understood scheme in artificial intelligence. Based on Delov-Yao model, we have constructed a general framework for formalizing the protocol and abilities of the intruder in Horn clauses. Using deductive reasoning method, we have realized the secrecy verification of cryptographic protocols, and presented a method of constituting attack scenario. Our future work will focus on study of the optimization of the solving algorithm, the goal is to analyze complicated protocols and verify more security properties of cryptographic protocols in a uniform mechanism.

References:

[1] M. Burrows, M. Abadi, and R. Needham. A logic of authentication. *Proceedings of the Royal Society*, Series A, 1989, 426(1871),pp.233–271. Also appeared as SRC Research Report 39 and, in a shortened form, in ACM Transactions on Computer Systems 8, 1 (February 1990), pp.18-36.

[2] I. Cervesato, N. Durgin, P. Lincoln, J. Mitchell, and A. Scedrov. A meta-notation for protocol analysis. In P. Syverson, editor, *12-th IEEE Computer Security Foundations Workshop. IEEE Computer Society Press,* 1999.

[3] Weidenbach, C. 1999. Towards an automatic analysis of security protocols in first-order logic. *In 16th International Conference on Automated Deduction (CADE-16)*, H. Ganzinger, Ed. *Lecture Notes in Artificial Intelligence*, vol. 1632. Springer-Verlag, Berlin, Germany, pp.314-328.

[4] Durgin, N., Mitchell, J., and Pavlovic, D. 2001. A compositional logic for protocol correctness. *In*

14th IEEE Computer Security Foundations Workshop (CSFW-14). IEEE Computer Society, Los Alamitos, CA, pp.241–255.

[5] Debbabi, M., Mejri, M., Tawbi, N., and Yahmadi, I. 1997. A new algorithm for the automatic verification of authentication protocols: From specifications to flaws and attack scenarios. *In Proceedings of the DIMACS Workshop on Design and Formal Verification of Security Protocols.* Rutgers University, New Jersey.

[6] M.Bozzano, G.Delzanno. Automated Protocol Verification in Linear Logic. PPDP 2002.38-49.

[7] Steve Schneider. Security Properties and CSP. In *Proceedings of the 1996 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, May 1996, pp. 174-187.

[8] Abadi, M. and Fournet, C. 2001. Mobile values, new names, and secure communication. *In Proceedings of the 28th Annual ACM Symposium on Principles of Programming Languages (POPL'01).* ACM Press, New-York, NY, pp.104–115.

[9] Abadi, M. and Gordon, A. D. 1999. A calculus for cryptographic protocols: The spi calculus. *Information and Computation* 148, 1 (Jan.), 1–70. An extended version appeared as DigitalEquipment Corporation Systems Research Center report No. 149, January 1998.

[10] Selinger, P. 2001. Models for an adversary-centric protocol logic. *In Proceedings of the 1st Workshop on Logical Aspects of Cryptographic Protocol Verification (Paris, France)*, J. Goubault-Larrecq, Ed. Electronic Notes in Theoretical Computer Science, vol. 55(1). Elsevier, Amsterdam, The Netherlands, pp.73–88.

[11] M.Bellare and P.Rogaway. Provably secure session key distribution—the three party case. *In Proceedings of the 27th ACM Symposium on the Theory of computing,*1995.

[12] Blanchet, B. 2001. An efficient cryptographic protocol verifier based on Prolog rules. *In 14th IEEE Computer Security Foundations Workshop (CSFW-14).* IEEE Computer Society, Los Alamitos, CA, pp.82–96.

[13] D.Monniaux. Abstracting Cryptographic Protocols with Tree Automata. *In Static Analysis Symposium(SAS'99),*volume 1694 of Lecture Notes on Computer Science. Springer Verlag, Sept.1999,pp.149-163.

[14] Mitchell, J.C. Finite-state analysis of security protocols, *in A.J.Hu & M.Y.Vardi,eds,'Computer Aided Verification(CAV-98):10th International Conference',*Vol.1427 of LNCS, Springer,pp.71-76.

[15] Lincoln, P., Mitchell, J., Mitchell, M., and Scedrov, A. 1998. A probabilistic poly-time framework for protocol analysis. *In Proceedings of the Fifth ACM Conference on Computer and Communications Security.* ACM Press, New-York, NY, pp.112–121.

[16]W.Marrero,E.Clarke, and S.Jha. Model checking for security protocols.Technical *Report CMU-CS-97-139,School of Computer Science,* Carnegie Mellon University, May 1997.

[17]J.C.Mitchell, M.Mitchell, and U.Stern. Automated Analysis of Cryptographic Protocols Using Murφ. *In Proceedings of the 1997 IEEE Symposium on Security and Privacy*,1997,pp.141-151.

[18] Paulson, L. C. 1998. The inductive approach to verifying cryptographic protocols. *Journal of Computer Security* 6, 1–2, pp.85–128.

[19] J.K.Millen,S.C.Clark, and S.B.Freedman.The Interrogator: Protocol Security Analysis. *IEEE Transactions on Software Engineering*, SE-13(2), Feb.1987, pp.274-288.

[20] E.Cohen.TAPS: A First-Order Verifier for Cryptographic Protocols. CSFW 2000,pp.144-158.