Comparative Analysis of Formal Model Checking Tools for Security Protocol Verification

Reema Patel[1], Bhavesh Borisaniya[1], Avi Patel[1], Dhiren Patel[2], Muttukrishnan Rajarajan[3], and Andrea Zisman[3]

[1] Sardar Vallabhbhai National Institute of Technology Surat, India, {reema.mtech,borisaniyabhavesh,avi2687}@gmail.com
[2] Indian Institute of Technology Gandhinagar, India dhiren@iitgn.ac.in
[3] City University London, UK R.Muttukrishnan@city.ac.uk, a.zisman@soi.city.ac.uk

Abstract. With the proliferation of *universal clients* over Internet, use of security protocols is rapidly on rise to minimize associated risks. Security protocols are required to be verified thoroughly before being used to secure applications. There are several approaches and tools exist to verify security protocols. Out of these one of the more suitable is the Formal approach. In this paper, we give an overview of different formal methods and tools available for security protocol verification.

Keywords: Security Protocols, Formal Methods, State Exploration, Verification, Falsification.

1 Introduction

The usage of Internet has rapidly risen in our day-to-day life. Internet based applications such as online banking, online reservation, e-governance and many more use sensitive information and therefore need higher levels of security. To secure such applications, various security protocols have been developed. Security protocols ensure secure communication over a hostile network by performing security related functions using appropriate cryptographic primitives. Several protocols have been shown to have flaws in computer security literature. Badly designed security protocols make it easy for the intruder to observe, delay, redirect or replay the messages. In order to make these protocols secure they need to be verified using a systematic approach.As manual analysis of such security protocols is extremely difficult and time consuming, alternative methods and automatic tools are required for the protocol verification.

1.1 Two Approaches for Protocol Verification

Security protocols achieve secure communication using cryptographic primitives like *Key Distribution* or *Key Agreement*, *Authentication*, *Symmetric* or *Asymmetric Encryption*, *One Way Hash Function (OWHF)* or *Message Authentication Code* (MAC), etc. The main aim of the security protocol verification is to

N. Meghanathan et al. (Eds.): CNSA 2010, CCIS 89, pp. 152–163, 2010.

verify whether a security protocol satisfies the security requirements. Primarily, there are two approaches for security protocol verification [1]. They are computational approach and the formal method approach.

Computational Approach. Computational approach considers cryptographic primitives as a function on a string of bits. Security properties are defined in terms of probability of successful attacks.

The intruder is modeled as a *Turing machine* which has access to an oracle. However the oracle aims to find the decryption key. Protocol is considered good if oracle cannot find the decryption key or it will consume computational power to find the decryption key. Computational approach finds all probabilistic polynomial-time attacks but proofs are difficult to automate, which is quite hard to understand.

The Formal Method Approach. Contrast to a computational approach, Formal approach assumes perfect cryptography. It states that cipher-text can only be decrypted by applying the appropriate decryption key. Intruder is modeled as an agent that controls the network but cannot do a cryptanalysis [2].

There are a number of automatic security protocol verification tools based on formal approach. In formal approach, protocol description, security properties and intruder capabilities are defined formally. Formal method verification tool verifies the security protocol in the presence of the intruder, and if it is found to be faulty then it provides a counter example that represents how protocol violates the security requirement. Providing a counter example is an easy way to understand how attack can be performed on that protocol.

Rest of the paper is organized as follows: Section 2 discusses various approaches of formal verification of security protocols. Section 3 discusses different formal model checking tools. We give an example of protocol verification in section 4. In section 5 we give a comparative analysis of model checking tools. Finally section 6 gives the conclusion to this comparative study.

2 Formal Verification of Security Protocols

There are three approaches for formal verification of security protocols viz; *belief logic*, *theorem proving approach* and *state exploration* [2]. We hereby provide a brief overview of each approach.

2.1 Belief Logics

Belief logic was the first attempt to automate the verification of security protocols.

BAN Logic: BAN [3] logic, developed by Mike Burrows, Martin Abadi, and Roger Needham; is the most significant and prominent belief logic. It describes beliefs of communicating principles and evaluates those beliefs using inference

rules. Inference rules derive a final goal statement from all previously evaluated statements which represent either protocol is correct or not.

Its main characteristic is its simplicity but minor disadvantage is their assumptions. They include: strength of cryptography algorithm, principals are trustworthy, and principals can recognize and ignore their own messages. BAN Logic can only be used to verify authentication properties.

GNY: **G**ong, **N**eedham and **Y**ahalom extended the BAN logic [4]. BAN logic does not consider the release of message contents and the interaction of the protocol runs at different time. Thus a successful but complicated approach was proposed called GNY [4]. The GNY approach puts more considerations on the difference between content and meaning of messages. GNY logic has been difficult to apply, due to its many inference rules. GNY Logic also used to verify only authentication protocols.

BGNY: BGNY logic, introduced by **B**rackin [5] is an extended version of GNY. This belief logic is used by software that automatically proves the authentication properties of cryptographic protocols. Similarly to GNY logic, BGNY addresses only authentication. However, BGNY extends the GNY logic by including the ability to specify protocol properties at intermediate stages and being able to specify protocols that use multiple encryption and hash operations, message authentication codes, hash codes as keys, and key-exchange algorithms [6].

2.2 Theorem Proving Based Approaches

The Theorem proving approach considers a formal version of mathematical reasoning about the system. Tools which used this approach are based on either *Higher Order Logic* (HOL) or *First Order Logic (FOL).* Examples of theorem proving tools are HOL theorem prover, the ACL2 theorem prover, Isabelle theorem prover etc. Using a theorem prover, one formalizes the system (infinite number of principles running infinite number of sessions along with the intruder) as a set of possible communication traces. Theorem proving approach provides a formal proof [2]. This approach is not fully automatic and therefore time consuming approach. This approach provides poor support for flaw detection.

2.3 State Exploration

In this approach, a verification method searches all possible execution paths of the protocol and verifies each reachable state that satisfies some conditions. *Unbounded verification* considers unbounded number of principles and unbounded number of protocol sessions. So the state space generated from analyzing a protocol may be infinite. Here, intruder can also generate infinite number of messages. Such kind of a problem where the state space grows too large is known as state explosion. In *bounded verification*, principles and protocol sessions are fixed. Verification method finds out a particular state in which security property gets violated and a counter example is built by exploiting a trace from that incorrect

state to an initial state. State exploration methods are found to be powerful and automatic. Starting with an overview of Dolev-Yao method, followed by model checking tools which are used state exploration method, we discuss most important state exploration tools.

Dolev and Yaos Method Dolev-Yao method [7] mainly focuses on a formal model of an intruder. In this model, the network is assumed to be under the full control of an intruder who can read all message traffic, modify and destroy any messages, create any messages, and perform any operation, such as encryption or decryption that is available to legitimate users of the system. Most of the model checking tools use Dolev-Yao intruder model with minor changes.

3 Model Checking Tools

Majority of security protocol verification tools are based on model checking approach rather than on theorem proving approach. Some of the tools which use model checking approach based on state-exploration method are discussed below.

FDR/Casper: FDR (Failures-Divergence Refinement) is used to verify CSP (Communicating Sequential Processes) Programs. Gavin Lowe, found a flaw in NSPK (Needham-Schroeder Public-Key) protocol [8], and approached the verification of security protocols using Hoares calculus of CSP. FDR takes two CSP processes as input. One is the implementation of protocol, where all agents and intruder are modeled as CSP process. The second is specification of protocol which represents "correctly achieves authentication" or "secrecy" and models them as CSP process. It then verifies, whether implementation refines the specification or not. If FDR finds out that the specification is not met then it results the trace of system that does not satisfy specification. This trace consider as an attack on that protocol.

The process of converting a protocol description into CSP program is very difficult and time-consuming. For that Lowe developed a Casper compiler [9] that compiles abstract security protocol description into CSP program.

OFMC: Basin introduced On-the-Fly Model-Checker (OFMC) in 2003 [10]. OFMC is based on two lazy techniques: the first is lazy demand-driven search and second is the lazy intruder, which reduces the computational effort. Protocol description is specified in HLPSL (High-Level Protocol Specification Language) that allows user to write the protocol in *Alice-Bob* style notation. HLPSL2IF automatically translates it to IF (Intermediate Format) which OFMC takes as input.

Lazy demand-driven search uses lazy data types to model infinite state-space of protocol. Lazy data types model the protocol and attacker as infinite tree on the fly, in a demand driven way. The nodes of the tree are traces and children represent the next step of protocol or an action of an attacker. Properties of

nodes represent the security properties. Lazy intruder techniques model a lazy Dolev-Yao intruder whose actions are generated in a demand-driven way. Now OFMC is renamed as Open source Fixed-point Model-Checker.

CL-Atse: Chevalier et al. developed the Constraint-Logic based ATtack SEarcher [11]. It is OCaml-based (programming language) implementation of the deduction rules. These rules allow user to interpret and automatically execute the protocols in every possible way in the presence of Dolev-Yao intruder Capabilities.

Protocol description is specified as a set of rewriting rules in IF format. The main design goals of CL-Atse are modularity (easily extend the class of the protocols to be analysed) and performance (obtain the results using large number of protocol sessions). Any state-based properties (like secrecy, authentication etc) and algebraic properties of operators like XOR, exponentiation can be modelled and analysed.

SATMC: SAT-based Model Checker is an open and flexible platform for SAT-based bounded model checking [12]. Protocol descriptions are specified as rewrite formalism in IF format. SAT compiler generates the formula for each step of the protocol using encoding techniques. Each formula is then tested using SAT solver - whether formula is satisfiable or it leads to an attack. SATMC performs bounded analysis by considering finite sessions of the protocol with Dolev-Yao intruder capabilities.

TA4SP: Tree Automata based on Automatic Approximations for the Analysis of Security Protocols (TA4SP) was developed by Genet and Klay in 2000 [13], and is based on abstraction-based approximation method. Abstraction provides a way to prove correctness or security of a protocol by over-estimating the possibility of failure. This tool language represents an over-approximation or under-approximation of the intruder knowledge with an unbounded number of sessions.

The tool Is2TiF, composed in TA4SP automatically converts the input protocol specification file (HLPSL format) into Timbuk (a tree automata library). Tree automata library contains an approximation function, a term rewriting system representing the protocol, the intruder's abilities to analyze message and an initial configuration of the network. Timbuk taking the protocol specification file and then test, three kinds of conclusions can be deduced from the results: either a protocol is flawed (under-approximation), or a protocol is secured (over-approximation), or no conclusion can be raised.

AVISPA: AVISPA, a push-button tool for the Automated Validation of Internet Security-sensitive Protocols and Applications was developed by Basin *et al.*, in 2004 [14]. A number of protocol verification tools analyze small and medium-scale protocols such as Clark/Jacob library [15]. But AVISPA can analyze both small and medium-scale protocols as well as large scale Internet security protocols. Protocol description is specified in High Level Protocol Specification Language (HLPSL). Then HLPSL2IF converts it into rewrite IF format. Current

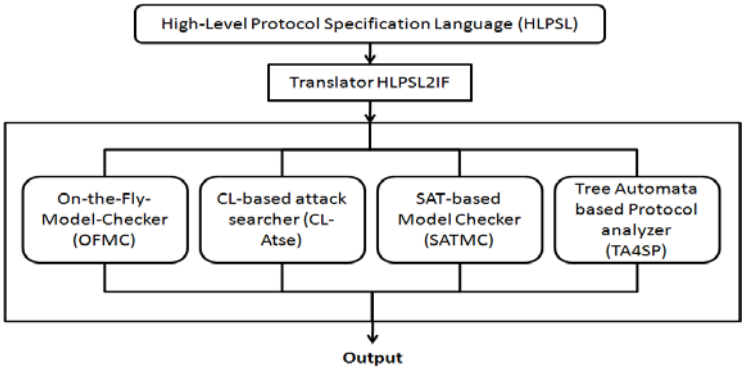

Fig. 1. The Architecture of the AVISPA tool

version of the AVISPA tool integrates four back-end tools which are: OFMC [10], CL-Atse [11], SATMC [12], and TA4Sp [13]. Now IF specifications are given as input to the all back-end tools. There is different output for each back-end tool. AVISPA tool provides a web-based graphical user interface that supports the editing of protocol specifications and allows the user to upload protocol specification. If an attack is found, the tool displays it using message-sequence chart which is very easy to understand. Figure 1 shows the architecture of AVISPA tool [14].

HERMES: Bozga *et al.* introduced HERMES in 2003 [16]. This tool is specialized to verify secrecy properties with unbounded number of sessions and principles. Hermes has no restriction on the size of messages.

Hermes provides a frond end (EVA). Hermes provides either an attack or proof tree for the protocol correction. Hermes uses three back-end tools to verify secrecy which are: Cryptographic Protocol Verification (CPV) [17], Hermes [16] and SECURIFY [18].

Interrogator: Interrogator, developed by Millen *et.al* in 1987 [19] is a Prolog program that searches for security vulnerabilities in network cryptographic key distribution protocols [20]. Here, a Protocol is modeled as communicating state machines in which intruder can destroy, intercept, and modify every message. Given a protocol description and a final state i.e. goal of the intruder in which message is supposed to be secret, the interrogator searches for every possible attack scenario with intruder abilities and verify the protocol against that final state.

NRL protocol analyzer: Catherine Meadows developed the NRL Protocol Analyzer [21] in US Naval Research Laboratory (NRL) in 1994. It is special-purpose tool that uses a theorem proving technique such as Inductive method for security protocol analysis. NRL uses the same model as interrogator. NRL

protocol analyzer allows an unbounded number of protocol sessions. It is not fully automatic tool because user has to provide a protocol specification and also insecure state that is, a state of the system in which security properties has been violated. Using backward search, the analyzer tries every single path to reach initial state from insecure state. If such path is found then reverse of this path will correspond to an attack on that protocol. But if attack is not found then backward search from insecure state to initial state goes into infinite loop and proves that "insecure state" is not reachable.

Brutus: Brutus [22] is a special-purpose tool for analyzing security protocols. The Protocol specification language used in Brutus is powerful enough to describe a variety of security properties. In this language user can specify what information participants (including intruder) must know and should not know. Brutus can only check a finite number of states and a full proof of protocol correctness cannot be obtained. Advantage of Brutus is that the intruder model is inbuilt into the tool so user has to not explicitly specify intruder abilities.

Murφ: The Murφ [23] verification system is a finite-state machine verification tool. Mur is also a description language. Murφ uses either breadth-first or depth-first search for the finite state space generation. The protocol specification is modeled in Murφ. Security properties can be specified by invariants, which are those Boolean conditions that are true in all reachable state. If a state is reached in which some invariant is violated, Murφ prints an error trace - a sequence of states from the start state to the state exhibiting the problem.

ProVerif: ProVerif, developed by Bruno Blanchet [24] represents protocol specification by Horn clauses. ProVerif accepts two kinds of input files: Horn clauses and a subset of the Pi-calculus. Protocol is analyzed with unbounded number of protocol sessions by using some approximations. Dolev-Yao intruder model is inbuilt into the tool so there is no need to explicitly model the attacker. Sometimes it can generate false attacks too [25].

Athena: Athena is an automatic model checking tool developed by Dawn Song and Perring in 2001 [26]. Thayer, Herog and Guttman proposed a Strand Space Model (SSM) for protocol representation [27]. Athena developed a new logic suitable for SSM for describing security protocols and their intended properties. Athena uses techniques from both model checking and theorem proving to extend SSM to represent protocol execution.

Athena allows arbitrary protocol configuration, e.g. unbounded number of principles and protocol sessions. To improve efficiency Athena uses pruning theorems for avoiding state-space explosion problem. One disadvantage of Athena is that the verification procedure is not guaranteed to terminate in unbounded verification. Termination can be forced by bounding the number of protocol sessions and the length of messages.

Scyther: Scyther and Scyther GUI were developed by Cas Cremers in 2007 [25]. Scyther is an automatic push-button tool for the verification and falsification of security protocols. Scyther provides a graphical user interface which incorporates the Scyther command-line tool and python scripting interfaces. Scyther tool takes protocol description and optional parameters as input, and outputs a summary report and display a graph for each attack. Figure 2 shows prototype of Scyther tool [25].

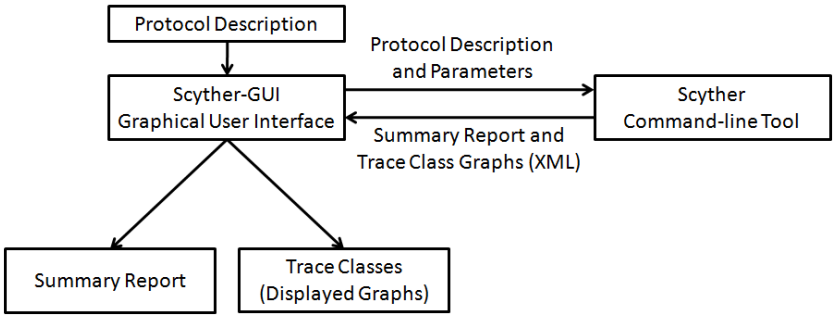

Fig. 2. The prototype of the Scyther tool

The description of a protocol is written in SPDL. Security properties are modeled as claim events. For the protocol verification, Scyther can be used in three ways. 1) Verification of claims: Scyther verifies or falsifies security properties. 2) Automatic claims: if user does not specify security properties as claim events then Scyther automatically generates claims and verifies them. 3) Characterization: Each protocol role can be "characterized". Scyther analyzes the protocol, and provides a finite representation of all traces that contain an execution of the protocol role.

Scyther generates attack graph for counter example. Scyther represents individual attack graph for each claim. It combines a number of novel features with the state-ofthe art performance viz; Scyther is guaranteed to terminate for an unbounded number of sessions, represents infinite sets of traces in terms of patterns, and facilitates multiprotocol analysis.

4 Protocol Verification Using Scyther

In this section we discuss how a protocol is verified using Scyther tool. Let us consider Wide-Mouthed-Frog (WMF) protocol. The WMF protocol is a server-based Symmetric key protocol which aims to provide shared key using trusted server and timestamp. The description of WMF protocol is as follows:

1. A → S : A, {Ta, B, Kab} Kas
2. S → B : {Ts, A, Kab} Kbs

In Scyther, a protocol is described in SPDL language in which each agent and the server is defined as a role. The WMF protocol specification in SPDL is given here.

```
#Protocol description in SPDL Language

usertype SessionKey,TimeStamp;
secret k: Function;
const Compromised: Function;
protocol wmf(A,B,S)
{
role A {
       const Kab: SessionKey;
       const Ta: TimeStamp;
       send_1(A,S, A, {Ta, B, Kab}k(A,S));
       claim_A1(A,Secret,Kab);
       }
role R {
       var Ts: TimeStamp;
       var Kab: SessionKey;
       read_2(S,B, {Ts, A, Kab}k(B,S));
       claim_B1(B,Secret,Kab);
       claim_B2(R,Nisynch);
       }
role S {
       var Kab: SessionKey;
       const Ts: TimeStamp;
       var Ta: TimeStamp;
       read_1(A,S, A,{Ta, B, Kab}k(A,S));
       send_2(S,B, {Ts, A, Kab}k(B,S));
       }
}
```

In first message, an agent A sends a message to S which contains his own identity i.e A, and Timestamp Ta, identity of B and session key Kab encrypted using symmetric key Kas. In SPDL, *send_1(A,S,A,{Ta,B,Kab}k(a,s))*; is the message .This same message is taken as a read event in server role. In second message, server replays to B and sends Timestamp Ts, As identity, and a session key Kab encrypted using symmetric key Kbs. This message is written as a send event in server role and as a read event in agent B role.

Security properties are defined as claim events. In this protocol session key Kab should be kept secret, in order to define *claim_A(A, secret, Kab)*. Scyther verifies WMF protocol and finds flaws in it and generates attack graph, which is shown in Figure 3.

Here Scyther finds out a replay attack which is already discovered by Lowe, as follows:

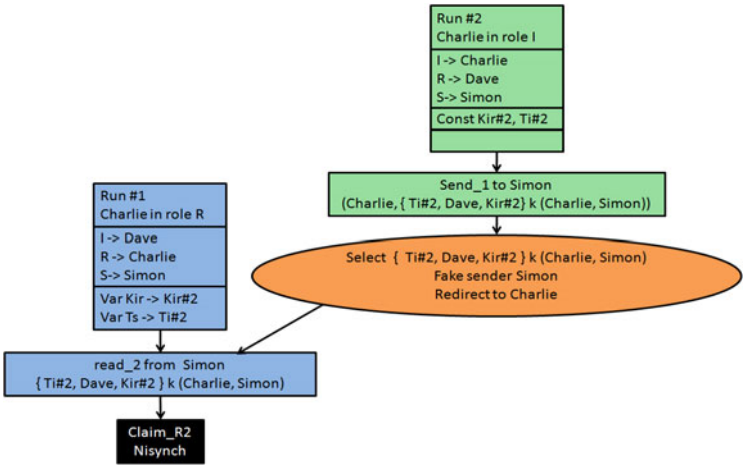

Fig. 3. Attack graph for WMF protocol

I.1 A → S: A, {Ta, B, Kab}Kas
I.2 S → B: {Ts, A, Kab} Kbs
II.2 I(S) → A: {Ta, B, Kab}Kas

In this attack, *A* thinks that *B* has established two sessions with him, when *B* thinks he has established only one session with *A*. *I.1* and *I.2* are actually correct messages of session *I*, however, there is also a possibility that intruder can reply the first message as if generated from the server.

5 Comparative Analysis of Formal Method Model Checking Tools

The comparison of formal model checking tools is given in Table 1 in relation to falsification, verification, and termination.

Tools perform falsification if it provides response when protocol is flawed. Verification of protocol can be taken as bounded if the tool verifies it with fixed number of principles and sessions or unbounded if number of principles and sessions are not fixed. While termination indicates that verification procedure will end successfully or not.

Athena is strong contender for protocol verification because it also provides bounded and unbounded verification but it's not publicly available.Publicly available tools such as FDR, OFMC, SATMC etc. also performs falsification,but their application is limted only to bounded verification. Hence, from comparison it's clear that AVISPA and Scyther are the most suitable tools for model checking.

Table 1. Comparison of formal model checking tools

Tool Name	Publicly Available	Falsification	Verification		Termination
			Bounded	Unbounded	
FDR/Casper	Yes	Yes	Yes	No	Yes
OFMC	Yes	Yes	Yes	No	Yes
CL-Atse	Yes	Yes	Yes	No	Yes
SATMC	Yes	Yes	Yes	No	Yes
TA4SP	Yes	Yes	No	Yes	Yes
Avispa	Yes	Yes	Yes	Yes	Yes
HERMES	Yes	Yes	No	Yes	Yes
Interrogator	No	Yes	Yes	No	Yes
NRL Protocol analyzer	No	No	No	Yes	Yes *
Brutus	No	Yes	Yes	No	Yes
Murφ	Yes	Yes	Yes	No	Yes
ProVerif	Yes	Yes	No	Yes	Yes
Athena(Bounded)	No	Yes	Yes	No	Yes
Athena(Unbounded)	No	Yes	No	Yes	No
Scyther	Yes	Yes	Yes	Yes	Yes

* It may not guarantee to terminate if the protocol is flawed.

6 Conclusion

Tools using formal approach are more appropriate for verifying security properties of a given protocol. Comparison shows that a fully automated Scyther and AVISPA formal verification tools are efficient to verify and falsify security protocols.

References

1. Abadi, M., Rogaway, P.: Reconciling two views of cryptography (the computational soundness of formal encryption). In: TCS 2000: Proceedings of the International Conference IFIP on Theoretical Computer Science, Exploring New Frontiers of Theoretical Informatics, pp. 3–22 (2000)
2. Juan Carlos, L.P., Monroy, R.: Formal support to security protocol development: A survey. Computacion y Sistemas 12(1), 89–108 (2008)
3. Burrows, M., Abadi, M., Needham, R.: A logic of authentication. ACM Trans. Comput. Syst. 8(1), 18–36 (1990)
4. Gong, L., Needham, R., Yahalom, R.: Reasoning about belief in cryptographic protocols. In: IEEE Symposium on Security and Privacy, p. 234 (1990)
5. Brackin, S.H.: A hol extension of gny for automatically analyzing cryptographic protocols. In: CSFW 1996: Proceedings of the 9th IEEE Workshop on Computer Security Foundations, p. 62 (1996)
6. Chen, Q., Zhang, C., Zhang, S.: Overview of security protocol analysis. In: Chen, Q., Zhang, C., Zhang, S. (eds.) Secure Transaction Protocol Analysis. LNCS, vol. 5111, pp. 17–72. Springer, Heidelberg (2008)

7. Dolev, D., Yao, A.C.: On the security of public key protocols. In: Annual IEEE Symposium on Foundations of Computer Science, pp. 350–357 (1981)
8. Lowe, G.: An attack on the needham-schroeder public-key authentication protocol. Inf. Process. Lett. 56(3), 131–133 (1995)
9. Lowe, G.: Casper: A compiler for the analysis of security protocols. Journal of Computer Security, 53–84 (1998)
10. Basin, D.A., Mödersheim, S., Viganò, L.: Ofmc: A symbolic model checker for security protocols. Int. J. Inf. Sec. 4(3), 181–208 (2005)
11. Basin, D.A.: Lazy infinite-state analysis of security protocols. In: Proceedings of the International Exhibition and Congress on Secure Networking - CQRE (Secure) 1999, pp. 30–42 (1999)
12. Armando, A., Compagna, L.: Satmc: A sat-based model checker for security protocols. In: Alferes, J.J., Leite, J. (eds.) JELIA 2004. LNCS (LNAI), vol. 3229, pp. 730–733. Springer, Heidelberg (2004)
13. Boichut, Y., Heam, P.C., Kouchnarenko, O., Oehl, F.: Improvements on the Genet and Klay Technique to Automatically Verify Security Protocols. In: Proc. Int. Workshop on Automated Verification of Infinite-State Systems (AVIS 2004), joint to ETAPS 2004, pp. 1–11 (2004)
14. Vigan, L.: Automated security protocol analysis with the avispa tool. Electronic Notes in Theoretical Computer Science 155, 61–86 (2006)
15. Clark, J.A., Jacob, J.L.: A survey of authentication protocol literature. Technical Report 1.0 (1997)
16. Bozga, L., Lakhnech, Y., Périn, M.: Hermes: An automatic tool for verification of secrecy in security protocols. In: Hunt Jr., W.A., Somenzi, F. (eds.) CAV 2003. LNCS, vol. 2725, pp. 219–222. Springer, Heidelberg (2003)
17. Goubault-Larrecq, J.: A method for automatic cryptographic protocol verification. In: IPDPS 2000: Proceedings of the 15 IPDPS 2000 Workshops on Parallel and Distributed Processing, pp. 977–984 (2000)
18. Cortier, V.: A guide for securify. Technical Report 13 (2003)
19. Millen, J.K., Clark, S.C., Freeman, S.B.: The interrogator: Protocol secuity analysis. IEEE Trans. Softw. Eng. 13(2), 274–288 (1987)
20. Tarigan, A., Rechnernetze, A., Systeme, V., Bielefeld, U.: Survey in formal analysis of security properties of cryptographic protocol (2002)
21. Meadows, C.: The nrl protocol analyzer: An overview. The Journal of Logic Programming 26(2), 113–131 (1996)
22. Clarke, E.M., Jha, S., Marrero, W.R.: Verifying security protocols with brutus. ACM Trans. Softw. Eng. Methodol. 9(4), 443–487 (2000)
23. Mitchell, J.C., Mitchell, M., Stern, U.: Automated analysis of cryptographic protocols using murφ. In: SP 1997: Proceedings of the 1997 IEEE Symposium on Security and Privacy, p. 141 (1997)
24. Blanchet, B.: An efficient cryptographic protocol verifier based on prolog rules. In: CSFW 2001: Proceedings of the 14th IEEE Workshop on Computer Security Foundations, p. 82 (2001)
25. Cremers, C.: Scyther - Semantics and Verification of Security Protocols. Ph.D. dissertation, Eindhoven University of Technology (2006)
26. Song, D., Berezin, S., Perrig, A.: Athena: a novel approach to efficient automatic security protocol analysis. Journal of Computer Security 9, 2001 (2001)
27. Thayer Fbrega, F.J., Herzog, J.C., Guttman, J.D.: Strand spaces: Why is a security protocol correct? In: Proceedings of the 1998 IEEE Symposium on Security and Privacy, pp. 160–171 (1998)