Implementing Protocol Verification for E-Commerce

B. Aziz, D. Gray, G. Hamilton, F. Oehl, J. Power and D. Sinclair

Abstract— **This paper presents a survey of the practical application of protocol verification techniques to applications in e-commerce. We concentrate in particular on logic-based approaches, and review the current state of the art as well as the prospects for realistic deployment of protocol verification techniques in the near future.**

Keywords— **keywords**

I. Introduction

A key component of e-commerce applications are the protocols that:
• identify the participants in a transaction;
• facilitate and enable the secure, reliable transfer of critical information; and
• provide for the non-repudiation of completed transactions

This paper reviews the current state of the art and presents the major techniques used todate to mathematically verify these protocols. In particular we focus on logic-based approaches, the tools used to support these techniques and highlight the protocols verified by these different techniques.

The formal verification of the protocols used to support e-commerce is not easy; so why should we formally specify and verify these protocols? The answer is to build trust and confidence in the users of the e-commerce systems. The users must believe that these protocols are secure and reliable. Trust and confidence are essential for the successful and widespread adoption of e-commerce. Todate the adoption of e-commerce has been greater in Business-to-Business (B2B) applications than Business-to-Consumer (B2C) applications. One reason for this is that many B2B applications are built on existing business relationships. In many B2C transactions the consumer is unknown to the business and both participants do not have a prior history on which this trust and confidence between the participants can be built. Formally verifying the behaviour of the protocols will help build this trust and confidence. The central rôle of trust and confidence in e-commerce has been recognised by the European Commission as typified by Commission proposal to fund research into building trust in confidence in the forthcoming 6th Framework Programme (FP6).

II. Theorem Proving

First some information shared by all the proof methods. In these methods, you use logical notations to model your protocol and the properties that it must satisfy. Then you use proof techniques (rewriting, simplification, induction, ...) to verify if the properties are satisfied by the protocol. With these methods, the verification is not limited by the number of actors and by the number of messages exchanged during protocol runs. But the verification with these methods can be long and can require an expert user (you need to inject the right lemma at the right moment in order to make the proof converge). It's very difficult to develop automatic tools to verify protocols by proof method.

We can split these methods into two categories:
• The inference construction methods which construct inferences using specialised logics based on a notion of knowledge and belief, that protocol participants can confidently reach desired conclusions;
• The proof construction methods which formally model the actual computations performed in protocols and prove theorems about these computations.

A. Inference construction methods

In this category you find all the approaches based on the modal logic used for the analysis of the evolution of knowledge and belief in distributed system.

The first logic developed for the verification of cryptographic protocols was "the logic of authentication"[1] (also called BAN logic). Michael Burrows, Martin Abadi and Roger Needham developed this logic in 1989. In this method, we reason about the beliefs of the agents in the network and the evolution of these beliefs after each protocol step. An example of this reasoning would be: "If Paul's received a message encrypted with the key K, and he believes that Alice and he share K, then he believes that Alice's sent the message".

So to verify a protocol with this method, we start by defining the initial beliefs of all the protocol's actors. Then after each protocol step you have access to new information. And with this new information and the logic inference rules, you find new beliefs by derivation. If the set of beliefs fits with the beliefs we want for our protocol, we assume the protocol has been proven correct. Otherwise, we may have discovered a security flaw in the protocol. The BAN logic has found flaws and redundancies in several protocols, including Needham-Schroeder public key (flaw) and Kerberos (redundancy).

Because we only reason about beliefs with this logic and don't attempt to model knowledge, we can only verify authentication property. That means we only verify that every actor can identify the sender of a message. This was one of the reasons of the development of other logics (another big reason is the lack of a complete semantics): GNY logic [2], the Abadi and Tuttle's logic [3], the Mao and Boyd's logic [4], AUTLOG logic [5] and SvO logic [6]. These logics are more complicate and more difficult to use than the BAN logic except for the Abadi and Tuttle, and SvO logics.

Other logics like Kailar's logic [7], and Kessler and Neumann's logic [8] have been used to verified e-commerce protocols. These logics are still based on the BAN logic, but they introduce the notion of accountability. Accountability is the property whereby the association of unique originator with an object or action can be proved to a third party (i.e.: a party who is different from the originator and the prover). The Kailar's logic [12] has been used to verify two versions of the Carnegie Mellon's Internet Billing Server protocol, the University of Southern California Information Science Institute's anonymous payment protocol and the SPX protocol. The SET [54](Secure Electronic Transaction) and the Payword protocols have been studied with the Kessler and Neumann's logic [8].

Now, you have an idea of the logics that are available to verify cryptographic protocols. But what tools could you use if you don't want to do the proofs by hand? The BAN logic was implemented in the theorem provers SETHEO [9] and EVES [10] (these two tools produce fully automatic proofs of protocols). Kindred [11] generated automatic checkers for both Kailar's logic and AUTLOG logic using Revere. Nevertheless, if you cannot find any tool developed for the logic you want to use, you still can implement it in a theorem prover like HOL [13] (Higher Order Logic) or SETHEO (Kessler and Neumann have started to implement there logic this tool, but it was a lot of work since SETHEO is actually not suited and the running time for most of their proofs was so bad that they decided to do them by hand) or any other theorem provers.

B. Proof construction methods

In this category, you find methods, which try to solve the problems of inference construction methods (lack of clear semantics) and model checking methods (state explosion).

In [14], D. Bolignano introduced a method based on the idea of trustable and un-trustable agents. So he has a set of trustable agents and one intruder. This intruder stores all the information exchange between the agents, can decrypt messages if he has the decryption key and he also can build and send fraudulent messages if he has the encryption key. Protocols are formalised as automata where each state is a n-uplet of agents' current state, and the transitions are the protocol steps. The protocol's properties are verified by induction the on automaton's states.

D. Bolignano used the theorem prover Coq [15] to implement this method and he presented an extension of his method for the e-commerce protocols [16].

L. C. Paulson [17] has developed a method based on the proof by induction. In this technique, the protocols are modelled by the set of all possible traces and you have the assumption, there is an intruder or bad agent in your network. This intruder has access to traces, can decrypt messages if he has caught the decryption key and finally he can build and send fraudulent messages if he has the encryption key. To verify protocols you verify that each protocol step preserves the desired properties.

This method was implemented in Isabelle theorem prover [18] and used to verify the Internet protocol TLS [19], the Kerberos protocol [20], [21], [22] and some other protocols. The proofs of these protocols are available on the Isabelle's website. At the moment, this technique is being used to verify the SET protocol in a project "Verifying E-Commerce Protocols"at University of Cambridge.

III. Model-checking

Model checking is an automated technique which, given a finite state model of a system and a property stated in some appropriate logical formalism (usually temporal logic), systematically checks the validity of the property. This involves traversing the state space of the system to check the property of interest. This is why this technique can be applied only to finite state systems. It also becomes intractable for large systems due to an explosion in the state space for such systems.

A number of different people have used this approach, either with general-purpose model checkers [29], [32], [33], [38], [25], [30], [40] or special-purpose model checkers [26], [36], [37], [35]. The first attempt to apply model checking to the verification of security protocols is by Dolev and Yao [26]. In this work, the protocol is modeled by defining a set of states and a set of transitions, and keeping track of an intruder, messages and the information known by each of the principals. This model is quite limited as it considers only secrecy, and models only encryption and decryption of messages, and adding or removing principals.

The Dolev-Yao model was extended by Meadows [36]. In this work, the protocol is modeled as a set of rules that describe how an intruder generates knowledge (either through applying encryption and decryption, or by receiving responses to messages it sends to the principals participating in the protocol). This model checker was still limited in that it did not allow the modeling of freshly generated nonces or session keys. These operations were added in the NRL Protocol Analyser [37]. This model checker also includes the states of the participants as well as the intruder. However, there is no systematic way of converting a protocol description into a set of transition rules. The model checker also relies heavily on the user during verification, a drawback more normally associated with theorem provers. The algorithms used in the NRL Analyzer are also not guaranteed to terminate, so a limit is placed on the number of recursive calls allowed for some of the model checking routines.

In [32], Lowe used the FDR model checker for CSP [28] to analyse the Needham-Schroeder public key authentication protocol [39] and succeeded in finding a previously unpublished error in the protocol. However, the model is parameterised by the nonces used by the participants, so it only models a single run of the protocol. In order to prove the general protocol correct, a theorem must be proven which states that the general protocol is insecure only if this restricted version is insecure.

Another scheme for the model checking of security protocols is proposed in [35]. Unlike the previous systems, where the protocol must be encoded in CSP or in term rewrite rules, protocol definitions are translated into a sequence of commands such as SEND, RECEIVE and NEWNONCE. All principals and an intruder are modeled as a sequence of such actions. These actions are interleaved to produce a system trace. These traces can be model checked over a finite number of runs of the system to see whether there is a reachable state which violates the security specification.

The work in [23] further expands the model which can be checked by the model checker to include a logic of belief. Principals are modeled as processes able to have beliefs. The specification of a principal therefore has two orthogonal aspects; a temporal aspect and a belief aspect. This framework is used to verify a property of the Andrew protocol.

In conclusion, model checking has a number of advantages and disadvantages when compared to theorem proving. One advantage is that little user interaction is required, with properties being verified automatically. Also, in the cases where a required property is broken, a model checker is able to indicate to the user the sequence of events which led to it being broken.

Disadvantages of the model checking approach are that it requires that systems have a finite number of states, and it also becomes intractable for large state systems. As a result, the examples of model checking security protocols given in this section have mostly only been applied to small examples such as the Needham-Schroeder public key authentication protocol [39]. One notable exception is the work of Leduc et al. using LOTOS [24] and the Eucalyptus tool-box [27] to analyze the Equicrypt protocol [31], which is a real system currently under design for use in controlling access to multimedia services in a public channel.

Another drawback of the model checking approach is that it is usually only applied to a model of a small system running the security protocol, together with a model of the most general intruder who can interact with the protocol, and then checking for attacks. However, if no attack is found, this only tells us that there is no attack on the small system being modeled; there still may be an attack on some larger system. One attempt to try and alleviate this problem can be found in [34].

IV. Static Analysis-Based Protocol Verification

Throughout the years, verification techniques and tools have attained a high level of maturity and enjoyed reasonable success, especially in areas related to computer hardware and telecommunication industries. Model checkers and theorem provers have become standard and attractive tools for hardware manufacturers to reveal subtle defects in systems and ensure the correctness and validation of their designs.

Such success has been helped by the cost-effective automated solutions such verification techniques bring with them. However there still remain a few obstacles that hinder a similar success in the software industry. Theorem provers require human intervention, where a level of expertise is needed to guide their operation. On the other hand, model checkers suffer from two major problems: the construction of finite state models and state explosion. The former is related to infinite state spaces, which are impossible to cover due to the way model checkers operate: by traversing all the possible states of the program. Therefore, a model is required with only a finite number of states. This approximation is always imprecise and could even lead to the wrong model. Additionally, if the number of states is too large (or will expand at a huge rate) beyond the space and time capabilities of the model checker, the latter problem arises.

Due to the fact that most model checkers accept inputs only in their own specialised languages, a semantic gap exists and has to be bridged between any non-finite-state software system and the language of a particular model checker. Most e-commerce protocols are specified with general-purpose languages, such as Java or COBOL. Such specifications have to be abstracted, using sophisticated program analyses and transformation techniques, into a finite mathematical model that is safe and correct. Such a process is usually characterised by errors and time-consumption resulting from the difficulty in encoding statements reflecting correctness requirements in the tool's language.

The state explosion problem in model checkers results from the large number of states to check, which is a prominent feature of the modern open software systems. Many state curbing techniques exist that add more abstraction to the model of the analysed system and therefore, introduce more difficulty and errors.

One major framework that has been initiated by the need to solve problems associated with model checkers is the Bandera project [42]. The project integrates the different static analysis and transformation techniques with model checkers while providing a pattern-based specification language [41] with which correctness requirements can be stated without ambiguity. Although the current implementation supports Java only (which is the *de facto* language of e-commerce anyway) and Web applications, the Bandera project is more of an open framework aimed at integrating programming languages to verification tools in the form of model checkers. This integration is made desirable by the automated model extraction in a safe and compact manner.

The idea to solve the state explosion problem is to do an abstraction of your system. That means only keep from the your model the information that are useful for the verification of your properties.

Here you have 2 concepts:

- Developing static analysis tools for the protocol verification;
- Developing tools that add static analysis in present model-checkers.

A. New tools

In this category, you find tools that use tree automata [52], Horn clauses [53], or else the Dominique Bolignanos model (in the project: Static Security Property Verifier) to model protocols.

With these methods protocols as Needham-Schroeder or even Skeme for the Horn clauses has been verified.

But the tools used are still on prototype stage.

B. Adding on present model-checker

The Bandera project provides a step towards the specification and automatic verification of security protocols written with Java-like languages. It is based on the component technology, which for the integration of new tools and hence, supports an open framework. However, in its current state, the Bandera project is described as a model extractor for Java, which builds a finite-state model that can be processed by model checkers. In doing so, it uses components that employ both syntactic and semantic static analysis techniques. Such techniques include mainly, program slicing [46], abstract interpretation [48], and partial evaluation [47].

Program slicing is a technique used for removing parts of a program that are not of interest to the model desired of that program. The technique works by identifying the interesting parts selected according to the model and then removing the rest. On the other, abstract interpretation [43], [44] is a mathematical framework for designing safe static approximations of program runtime semantics. The resulting semantics has as its semantic domain abstractions of the real concrete values. Main applications of the abstract interpretation lie in the design of compilers and analysers. Finally, partial evaluation [51] (also known as program specialisation) creates a specialised version of the inputted program by substituting values in that program that are already known at compile time. The resulting residual is much faster for execution. The Bandera project also supports components that are specialised in testing specific software units by using a testing environment called *verification harness*.

Other major projects related that have followed the same approach are JavaPathFinder [49] for translating Java source code into Promela, JCAT [45] for deadlock detection in Java program also by translating into Promela, and Feaver [50], which translates C programs into Promela as well.

V. Conclusion

In this paper we have have reviewed the major logic-based approaches used to verify security protocols used in e-commerce applications. In particular we have focussed on techniques that have tool support and have been actually used to verify some of the protocols used in e-commerce. These techniques and tools broadly fall into two categories; theorem provers and model checkers. The advantage of the model checkers is their ability to automatically prove a protocol without user direction, and in the case when the protocol fails, their ability to generate the sequence of events that invalidate the protocol. The main disadvantage of model checkers is that of state-space explosion. This has prevented model checkers from being used to verify large, or infinite state, systems. Theorem provers, on the other hand, do not suffer from this problem and they have been used to verify large protocols. The major disadvantage of theorem provers is that they require "expert" guidance. The work we are currently undertaking seeks to combine both techniques by embedding a model checker inside a theorem prover. Our goal is to develop a system that can verify large procotols but only requires minimal user direction.

As developers of e-commerce applications we must ensure that the protocols we use in our applications are secure and reliable. These protocols are complex entities and very stuble errors can be hidden deep in the protocol or introduced when the protocol is implemented. An important lesson on how easily this can happen occurred with the Needham-Schroeder protocol. When it was originally published it contained two nonces. Many commentators questioned the need for the second nonce and this nonce was "optimised" out of the protocol. This seemingly innocuous modification introduced a stuble flaw that allowed an attacker to access a secret shared between two others agents by intercepting one agent's messages while initiating a simultaneous with the other agent. This flaw was not discovered for 11 years. This salutary lesson needs to be kept in mind by designers and developers. SET [54] is arguably the "Rolls Royce" of payment protocols, but its widespread adoption has been prevented by its size and complexity. Just as in theNeedham-Schroeder protocol, simplifying or"optimising" SET could unknowingly introduce stuble flaws.

The formal verification of the protocols used in e-commerce is a vital component in the process of building trust and confidence in the users of e-commerce applications.

References

[1] M. Burrows and M. Abadi and R. Needham. *A Logic of Authentication.* DIGITAL, Systems Research Center, N. 39, February 1989, *http://www.research.digital.com/SRC/publications/.*

[2] L. Gong, R. Needham and R. Yahalom. Reasoning about Belief in Cryptographic Protocols. In *Proceeding of th 1990 IEEE Symposium on Security and Privacy*, 234–248, 1990, IEEE Compter Society Press.

[3] M. Abadi and M.R. Tuttle. A Semantics for a Logic of Authentication. In *Proceedings of the Tenth Annual ACM Symposium on Principles of Distributed Computing*, 201–216, 1991.

[4] W. Mao and C. Boyd. Towards Formal Analysis of Security Protocols. In *Proceedings of the 1993 IEEE Computer Security Foundations Workshop IV*, 147–158, 1993, IEEE Compter Society Press.

[5] V. Kessler and G.Wedel. AUTLOG-An advanced Logic of Authentication. In *Proceedings of the 1994 IEEE Computer Security Foundations Workshop VII*, 90–99, 1994, IEEE Compter Society Press.

[6] P. Syverson and P. C. van Oorschot. On unifying some cryptographic Protocol Logics. In *Proceedings of the 1994 IEEE Computer Security Foundations Workshop VII*, 14–29, 1994, IEEE Compter Society Press.

[7] R. Kailar. Reasoning about Accountability in Protocols for Electronic Commerce. In *Proceeding of th 1995 IEEE Symposium on Security and Privacy*, 236–250, 1995, IEEE Compter Society Press.

[8] V. Kessler and H. Neumann. A Sound Logic for Analysing Electronic Commerce Protocols. *ESORICS'98 Proceedings of the Fifth European Symposium on Research in Computer Security*, 345–360, 1998, Springer Verlag.

[9] J. Schumann. *Automatic Verification of Cryptographic Protocols Using SETHEO.* Technical Report AR-96-03, TU München, Institut für Informatik, 1996, *http://wwwjessen.informatik.tu-muenchen.de/ schumann/crypt.html.*

[10] D. Craigen and M. Saaltink. *Using EVES to Analyze Authentication Protocols.* Technical Report TR-96-5508-05, ORA Canada, March 1996, *http://www.ora.on.ca/eves/documentation.html.*

[11] D. Kindred. *Theory generation for Security Protocols.* Technical Report CMU-CS-99-130, Computer Science Department, Carnegie Mellon University, Pittsburg, PA, 1999, Ph.D. thesis.

[12] R. Kailar. Accountability in Electronic Commerce Protocols. *IEEE Transaction on Software Engineering*, 22(5):313–328, May 96.

[13] S. H. Brackin. A HOL Extension of GNY for Automatically Analysing Cryptographic Protocols. In *Proceedings of the 1996 IEEE Computer Security Foundations Workshop IX*, 62–76, 1996, IEEE Compter Society Press.

[14] D. Bolignano. An Approach to the Formal Verification of Cryptographic Protocols. *ACM Conference on Computer and Communications Security*, 106–118, 1996.

[15] B. Barras, S. Boutin, C. Cornes, J. Courant, J.-C. Filliatre, E. Gimenez, H. Herbelin, G. Huet, C. Munoz, C. Murthy, C. Parent, C. Paulin-Mohring, A. Saibi and B. Werner. *The Coq Proof Assistant Reference Manual : Version 6.1*, RT-0203, 1997.

[16] D. Bolignano. Towards the Formal Verification of Electronic Commerce Protocols. In *Proceeding of the 1997 IEEE Computer Security Foundations Workshop X*, 133–146, 1997, IEEE Compter Society Press.

[17] L. C. Paulson. The Inductive Approach to Verifying Cryptographic Protocols. *Journal of Computer Security*, V. 6, 85–128, 1998, *http://www.cl.cam.ac.uk/users/lcp/papers/protocols.html.*

[18] L. C. Paulson. *Isabelle: a generic theorem prover*, LNCS 828, 1994, Springer-Verlag Inc. .

[19] L. C. Paulson. Inductive analysis of the Internet protocol TLS. *ACM Transactions on Information and System Security*, V. 2, N. 3, 332–351, 1999.

[20] G. Bella and L. C. Paulson. Kerberos Version IV: Inductive Analysis of the Secrecy Goals In *Proceedings of the 5th European Symposium on Research in Computer Security*, Springer-Verlag LNCS 1485, Louvain-la-Neuve, Belgium, J.-J. Quisquater, 361–375, 1998.

[21] G. Bella and L. C. Paulson. Using Isabelle to prove properties of the Kerberos authentication system In *H. Orman and C. Meadows, editors, Workshop on Design and Formal Verification of Security Protocols.* DIMACS, 1997.

[22] G. Bella and L. C. Paulson. Mechanising BAN Kerberos by the Inductive Method. *Computer Aided Verification*, 416–427, 1998.

[23] M. Benerecetti and F. Giunchiglia. Model Checking Security Protocols Using a Logic of Belief. *TACAS 2000*, 519–534, 2000.

[24] T. bolognesi and E. Brinksma. Introduction to the ISO Specification Language LOTOS. *Computer Networks and ISDN Systems*, 14(1):25–59, 1987.

[25] Z. Dang and R. Kemmerer. Using the ASTRAL Model Checker for Cryptographic Protocol Analysis. In *Proceedings of the DIMACS Workshop on Design and Formal Verification of Security Protocols*, 1997.

[26] D. Dolev and A. Yao. On the Security of Public Key Protocols. *IEEE Transactions on Information Theory*, 29(2):198–208, 1989.

[27] H. Garavel. An Overview of the Eucalyptus Toolbox. In *COST247 Workshop*, 1996.

[28] A. Hoare. *Communicating Sequential Processes.* Prentice-Hall, 1985.

[29] R. Kemmerer, C. Meadows and J. Millen. Three systems for Cryptographic Protocol Analysis. *Journal of Cryptology*, 7(2), 1994.

[30] T. Kozlowski and S. Smolka. Digital Signatures With Encryption: Fact and Fiction (Extended Abstract). In *Conference Record of the 7th ACM Symposium on Principles of Programming Languages*, pages 81–94, 1990.

[31] S. Lacroix, J.-M. Boucqueau, J.-J. Quistater and B. Macq. Providing Equitable Conditional Access by Use of Trusted Third Parties. In *European Conference on Multimedia Applications, Services and Techniques*, pages 763–782, 1996.

[32] G. Lowe. Breaking and Fixing the Needham-Schroeder Public-Key Protocol Using FDR. In *Proceedings of TACAS*, LNCS 1055, pages 147–166, Springer-Verlag, 1996.

[33] G. Lowe and B. Roscoe. Using CSP to Detect Errors in the TMN Protocol. *IEEE Transactions on Software Engineering* 23(10): 659–669, 1997.

[34] G. Lowe. Towards a Completeness Result for Model Checking of Security Protocols. *Journal of Computer Security*, 7(1), 1999.

[35] W. Marrero, E. Clarke and S. Jha. A Model Checker for Authentication Protocols. In *Proceedings of the DIMACS Workshop on Design and Formal Verification of Security Protocols*, 1997.

[36] C. Meadows. Applying Formal Methods to the Analysis of a Key Management Protocol. *Journal of Computer Security*, 1:5–53, 1992.

[37] C. Meadows. The NRL Protocol Analyzer: An Overview. In *Proceedings of the Second International Conference on the Practical Applications of Prolog*, 1994.

[38] J. Mitchell, M. Mitchell and U. Stern. Automated analysis of Cryptographic Protocols Using Murϕ. In *IEEE Symposium on Security and Privacy*, 1997.

[39] R. Needham and M. Schroeder. Using Encryption for Authentication in Large Networks of Computers. *Communications of the ACM*, 21(12):993–999, 1978.

[40] W. Wu and M. Fumio. Model Checking Security Protocols. In *1999 Symposium on Cryptography and Information Security*, 1999.

[41] J. Corbett, M. Dwyer, J. Hatcliff, and Robby. A Language Framework For Expressing Checkable Properties of Dynamic Software. In *Proceedings of the SPIN Software Model Checking Workshop*, LNCS 1885, 2000.

[42] J. Corbett, M. Dwyer, J. Hatcliff, C. Pasareanu, S. Laubach, and H. Zheng. Bandera: Extracting Finite-state Models from Java Source Code. In *Proceedings of the 22nd International Conference on Software Engineering*, Limerick, Ireland, 2000.

[43] P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Conference Record of the 4th ACM Symposium on Principles of Programming Languages*, pages 238-252, Los Angeles, California, U.S.A., 1977.

[44] P. Cousot and R. Cousot. Abstract interpretation frameworks. *Journal of logic and computation*, 2(4):511-547, August 1992.

[45] C. Demartini, R. Iosif, and R. Sisto. A deadlock detection tool for concurrent Java programs. *Software - Practice and Experience*, 29(7), pages 577-603, July 1999.

[46] M. Dwyer, J. Corbett, J. Hatcliff, S. Sokolowski, and H. Zheng. Slicing Multi-threaded Java Programs: A Case Study. Kansas State University Computing and Information Sciences Tech Report 99-7, 1999.

[47] M. Dwyer, J. Hatcliff, and M. Nanda. Using Partial Evaluation to Enable Verification of Concurrent Software. Kansas State University Computing and Information Sciences Tech Report 97-15, 1997.

[48] J. Hatcliff, M. Dwyer, and S. Laubach. Staging Static Analyses Using Abstraction-based Program Specialization. Kansas State University Computing and Information Sciences Tech Report 98-5, 1998.

[49] K. Havelund and T. Pressburger. Model checking Java programs using Java PathFinder. *International Journal on Software Tools for Teaching Transfer*, 1999. to appear.

[50] G.J. Holzmann and M.H. Smith. Software model checking: Extracting verification models from source code. In *Proceedings of the FORTE/PSTV'99*, November 1999.

[51] M. Marinescu and B. Goldberg. Partial-evaluation techniques for concurrent programs. In *Proceedings of the ACM SIGPLAN Symposium on Partial Evaluation and Semantics-Based Program Manipulation (PEPM '97)*, pages 47-62. ACM, 1997.

[52] D. Monniaux. Abstracting Cryptographic Protocols with Tree Automata. In *In Proc. 6th Static Analysis Symposium*,149–163,1999.

[53] B. Blanchet. Abstracting Cryptographic Protocols by Prolog Rules (invited talk). In *8th International Static Analysis Symposium (SAS'01)*, July 2001.

[54] SET Working Group, *SET^{TM} Specification, books 1,2 and 3*, SETCO, http://www.setco.org/set_specifications.html

Mr. Benjamin Aziz holds a BSc in Electronics from the University of Garyounis, Libya, and an MSc in Computer Science from Trinity College, Dublin, Ireland. Currently, he is a PhD student in the Formal Methods group, School of Computer Applications, Dublin City University, Ireland. His PhD research is directed towards the area of static analysis of security properties for mobile and distributed systems using the abstract interpretation approach. Other main research interests include security and adaptability issues in integrating technologies, like CORBA. Mr Aziz is a student member of the IEEE.

Dr. David Gray holds B.Sc. and Ph.D. degrees from the Queens University of Belfast, N. Ireland. Currently he is a Senior Lecturer in Computer Applications at Dublin City University, Ireland. His main areas of research are security, formal methods and e-commerce. David is a member of the ACM.

Dr. Geoff Hamilton graduated from the University of Stirling, Scotland with a first class honours degree in 1989. He was awarded a Ph.D. from the University of Stirling in 1993. The area of his Ph.D. research was the optimisation of functional programs. Geoff joined the Computer Science Department in the University of Keele, England as a lecturer in 1993, and then moved to the School of Computer Applications, DCU in 1998, also as a lecturer. His current research activities are in the areas of formal specification and verification.

Mr. Frédéric Oehl holds a maîtrise and engineer title in Computer Science/Software Engineering from the University of Besançon, France, and a DEA in Computer Science, Process Control and Production Engineering from the University of Dijon, France. Currently he is a PhD student in the Formal Methods Group, School of Cumputer Application, Dublin City University, Ireland. The focus of his PhD research is the formal verification of cryptographic protocols. Other main research interests include the automata theory and the verification of infinite systems.

Dr. David Sinclair holds a BSc in Electronic Engineering from University College Dublin, Ireland and an MSc and PhD in Computing from Dublin City University, Ireland. He is a member of the Formal Methods Group in the School of Computer Applications, Dublin City University, and a founding member of the Secure and Distributed Systems Research Group. David's research interests in the design and verification of real-time, embedded and distributed systems follow from 8 years industrial experience in the design and development of embedded and distributed systems. The current focus of his research is the formal specification and verification of cryptographic systems and distributed systems. David is a member of the IEEE.