Selection of an Attack-Construction Engine to Enhance Security Protocol Analysis

Simon Lukell and Andrew Hutchison
Department of Computer Science
University of Cape Town
Rondebosch 7701
Ph: (021) 650 3127
Fax: (021) 689 9465
{slukell, hutch}@cs.uct.ac.za

Abstract— **In the context of providing an integrated environment for engineering of security protocols, the incorporation of an attack-construction analysis engine has been investigated. The purpose of such an engine is to search protocol specifications for possible replay attacks against it, returning a description of the attack if found. This kind of analysis complements the logic analysis tool already present in the environment, since it can find protocol vulnerabilities that the existing analysis is unable to detect. An investigation of six publicly available attack-construction tools was conducted, considering criteria such as capability, efficiency and usability. More project-specific factors, such as suitability for integration, also played an important part. The outcome of the investigation was that the constraint-based system by Corin & Etalle (based on an initial system by Millen & Shmatikov) was the most suitable.**

I. Introduction

Network security protocols make use of cryptographic techniques to achieve goals such as confidentiality, authentication, integrity and non-repudiation. However, the fact that strong cryptographic algorithms exist does not guarantee the security of a communications system [24]. In fact, it is recognised that the engineering of security protocols is a very challenging task, since protocols that appear secure can contain subtle flaws and vulnerabilities that attackers can exploit [1]. A number of techniques exist for the analysis of security protocol specifications. Each of the techniques currently available is not capable of detecting every possible flaw or attack against a protocol when used in isolation. However, when combined, they complement each other and allow a protocol engineer to obtain a more accurate overview of the security of a protocol that is being designed [14]. Previous projects of ours, in particular the Security Protocol Engineering and Analysis Resource (SPEAR) [5], and its successor, SPEAR II [23], introduced the concept of *multi-dimensional security protocol engineering*. Several aspects of cryptographic protocol engineering are collected into one application, which allows an engineer to rapidly construct, analyse and implement secure protocol designs.

The aspect of security protocol analysis in these projects was based on the *inference construction* techniques BAN [8] and GNY [13] modal logics respectively. As a step towards augmenting the analysis dimension of SPEAR II, it was decided that an *attack-construction* (AC) analysis component would be a valuable addition to the framework. Such a tool is capable of detecting protocol flaws that the current analysis is unable to find, and it gives the engineer a useful complementing view of a detected vulnerability. An AC tool searches a model of the protocol system, consisting of honest participants and a modelled malicious intruder, for possible replay attacks. If an attack is found, it returns a complete trace of the event that led to the attack, which is valuable when 'debugging' a specification. If no attack is found, the protocol is deemed secure against the analysed scenarios.

In this paper we present a survey of the attack-construction tools considered for integration with the SPEAR II framework, in which it could serve as one of several analysis engines. We identify functional attributes, which in combination give a relatively clear picture of the various capabilities of the tools. Additionally, a set of non-functional properties are considered, completing the overview. Following the tool survey, each attribute is given a relative degree of importance, forming the base on which a selection is made.

The main contribution of this paper is that we emphasise the application of the tools rather than their theoretical/ algorithmic constitution, which is the norm in other surveys such as in [20] and [26]. For a more technical description of the tools, these surveys and the original papers are more suitable. This overview is limited to publicly available AC analysis tools. We do not consider tools that do not return attack traces, and we omit analysers for which the software is not publicly available.

The remainder of the paper is organised as follows. In Section 2, a summary of the SPEAR II project is given followed by an overview of attack-construction analysis. Section 3 presents the set of features used in the evaluation, after which an overview of the considered tools is given in Section 4. The tool evaluation is described in Section 5, and the paper is concluded in Section 6.

II. Background

The rationale for the tool survey presented in this paper is the incorporation of an AC engine into the SPEAR II framework. In this section we describe this environment in more detail, followed by a brief introduction to the area of AC protocol analysis.

A. The SPEAR II Framework

A schematic overview of the SPEAR II framework is given in Figure 1. Completed modules within the framework are indicated by solid outlines, while possible future additions are denoted by grey outlines. This software engineering view of the tool is an intuitive representation that shows some

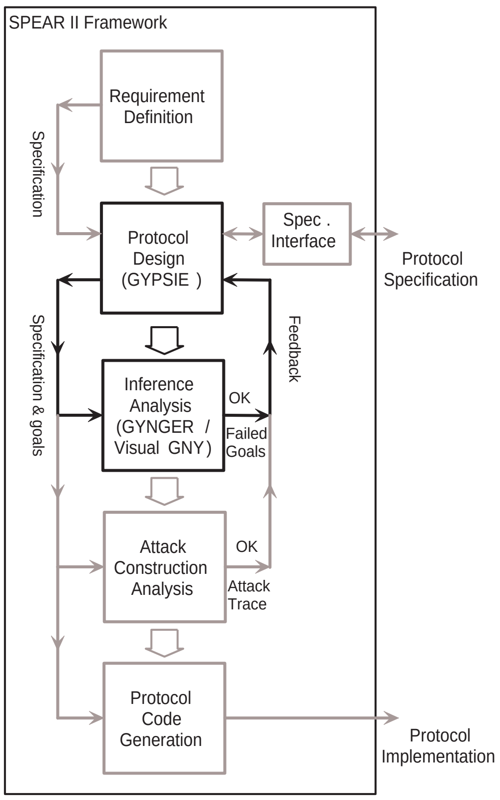

Fig. 1. The existing components of SPEAR II with proposed additions

relationships between its components, but it also indicates how the tool is used. In the figure, the large arrows between the modules indicate a natural work order when developing a security protocol, and the thinner arrows show what kind of information is conveyed between the modules. The figure suggests an iterative approach with the analysis modules feeding back results from analyses to the specification environment.

Currently SPEAR II consists of the following components integrated into one unified graphical interface: The **GYPSIE protocol specification environment** is designed for effective and accurate construction of cryptographic protocols and functions as the main interface of the SPEAR II application. **GYNGER** is a **Prolog-based analyser** that performs automatic analysis of protocols by using the GNY modal logic [13]. The **Visual GNY** environment was created to facilitate GNY-based protocol analysis and works in close conjunction with GYPSIE. In essence, Visual GNY functions as a user-friendly interface to the GYNGER analyser.

In order to increase the practical value of the tool, a number of additions can be made to the SPEAR II framework. From a software engineering perspective, a *protocol requirements tool* would assist the user to obtain an initial protocol specification from a set of requirements rather than having to define the specification. On the other end, an *implementation generation tool* would complete the protocol engineering process. We are busy developing such a tool as a specific related project [27]. One way of increasing the confidence in a protocol specification is to use external analysis tools. Security protocol specification languages, such as (CAPSL) [21] and HLPSL[16], support interfaces to several analysis tools. Therefore, a *protocol specification language interface* would also be a useful addition to the framework.

B. Attack-Construction Analysis

This method of analysis involves an explicit model of the protocol and a model of an intruder. There are many available formalisms that can be used to model the protocol, the participants, the intruder, their actions and the messages that they exchange [20]. However, all approaches use essentially the same basic assumptions about network communication and the capabilities of the attacker. These assumptions are based on the model introduced by Dolev and Yao [12], which gives the intruder the following capabilities: **read** any message and block further transmission; **decompose** a message into parts and remember them; **generate** fresh data as needed; **compose** a new message from known data and send it.

The intruder is only capable of obtaining encrypted information if he possesses the key to decrypt it with. This is referred to as the *perfect encryption assumption* [19], which is a means of isolating the protocol functionality from the cryptographic operations used in it. There are two different methods of using this model, namely searching the model forwards and searching it backwards. AC tools use a forward search strategy. They start in an initial state of a protocol environment and search the state space for insecure states exhaustively. The backward search, called *proof construction*, attempts to prove that a given insecure state of a protocol is unreachable.

The disadvantage with AC analysis lies in the big number of possible events that must be examined, also referred to as the *state space explosion problem*. Various optimisation techniques exist that limit the search space to a manageable size. The main advantage of this approach is that it is largely automatic, a property that agrees with the philosophy of the SPEAR II project. Proof construction attempts to avoid the exponential searches of AC, and to extend analyses to involve arbitrarily large numbers of participants and messages, with the disadvantage that it typically requires significant human insight and guidance.

III. Evaluation Framework

In this section, we identify various attributes that play a role in the evaluation, and ultimately the selection, of an AC tool. The first aspect considered, is the function of a tool, i.e. what it is capable (and not capable) of doing. Examples of such attributes are the cryptographic primitives the tool can model and which security properties it verifies. The other equally important aspect is the non-functional properties of a tool. Examples of non-functional attributes are performance, platform requirements and licensing.

A. Evaluated Properties

The most significant tool attributes in our study are of course the ones that form the boundary of the survey itself. Of the two limiting properties, one is functional, namely the capability of returning an attack trace to the user. The second limiting factor is non-functional, since it concerns the availability of the software.

1) Functional Properties: Functional properties of a system deal with what it is capable of doing. For our purposes they can be broken down to cryptographic primitives, bounded/ unbounded sessions, constructed keys, checked security properties, and termination.

Modelled *cryptographic primitives* are the basic building blocks in a protocol description. They include symmetric-key encryption/ decryption, public-key encryption/ signature, secure hash, and nonces. Some protocols use timestamps, which is a way of guaranteeing freshness without the need of two-way communication. Other protocols use arithmetic operations such as Diffie-Hellman exponentiation and XOR, which are usually not handled by analysis tools.

Some of the available tools can deal with an *unbounded* number of *sessions* of the protocol, with the benefit that no manual instantiation of the modelled honest principals is needed. In such a case, algebraic properties of the model are utilised to limit the search, which would otherwise continue indefinitely.

The notion of *constructed keys* is used in some protocols, for example in the case of the IKE protocol where each participant constructs only a part of the shared key that is negotiated. If the model does not support this, such protocols clearly cannot be verified properly.

A *type flaw* attack exploits a protocol's vulnerability to message component substitutions. A type flaw attack can be avoided in an implementation by typing the contents of the messages, but a secure specification is nevertheless preferable.

Two security properties can be verified in the model: *secrecy* and *authentication*. Secrecy is maintained if there is no state in which the modelled intruder gains knowledge of the secret message component. Authentication is verified through the use of a correspondence property of the protocol run. If a model contains a trace in which a participant has executed the protocol with a set of messages, and the intended counterpart's perception of the message contents does not match, then the authentication protocol has been attacked.

In the case of models with an unbounded number of parallel sessions of the protocol, the search space is clearly infinite. If the method of limiting the search of this model is incomplete, there is a risk that the search will not *terminate*. This is of course undesirable.

2) Non-functional Properties: These properties of a system describes not what the software will do, but how the software will do it. The properties for the evaluation are a chosen subset of the properties given in IEEE/ICS recommendations for requirements specifications [15]. The omitted non-functional system properties were deemed either not applicable, or insignificant for the purposes if this investigation.

Performance is the one non-functional property that has been given the most attention in literature, since it is used as a metric of the quality of the algorithm used in the tool. It is usually measured in the time taken for an attack to be discovered in a security protocol specification known to be flawed.

The *interface* property describes the language of the text input and output of the tool. This differs from the functional attribute, since languages with identical expressiveness can use different notations.

The *operational* property describes any aspects of the human operation of the system, i.e. the usability of the system, and the *resources* property describes what is required for the system to function. In our case, this is the external software that the tool depends on.

Portability is the property that describes on which platforms the tool can run on, and in some cases, which platforms it can be made to run on. Finally, the *legislative* property of software primarily deals with the what kind of licence applies to its use and distribution. Another significant property of freely distributable software is whether it is *open source software* (OSS).

	Casper	Constr	STA	SATMC	OFMC	Athena
SK Crypto	✓	✓	✓	✓	✓	✓
PK Encrypttion	✓	✓	✓	✓	✓	✓
PK Signature	✓	✓	✓	✓	✓	✓
Secure Hash	✓	✓	✓	✓	✓	✓
Timestamp	✓	×	×	×	×	×
Unb sessions	×	×	×	×	×	✓
Constr keys	×	✓	×	✓	✓	×
Type flaws	×	✓	×	✓	✓	×
Secrecy	✓	✓	✓	✓	✓	✓
Authentication	✓	✓	✓	✓	✓	✓
Termination	✓	✓	✓	✓	✓	×

TABLE I

COMPARISON OF TOOL FEATURES. ✓: AVAILABLE, ×: NOT AVAILABLE

	Casper	Constr	STA	SATMC	OFMC	Athena
Performance	-	0	–	+	+	+
Interface	+	+	–	+	+	–
Operational	+	0	–	+	+	–
Resources	FDR,H	Prolog	ML	SAT	none	∗
Portability	L	All	All	L	L/S	L
Open source	✓	✓	×	×	×	✓
License	?	free	u/d	GPL	u/d	free

∗ gcc, GNU make, Perl, CMU SMV, SML, Emacs

TABLE II

NON-FUNCTIONAL TOOL PROPERTIES. **–**: BELOW AVERAGE, **0**: AVERAGE, **+**: ABOVE AVERAGE, **H**: HASKELL, **L**: LINUX, **S**: SOLARIS, ✓: AVAILABLE, ×: NOT AVAILABLE, **?**: UNKNOWN, **U/D**: USE & DISTRIBUTE

IV. ATTACK-CONSTRUCTION ANALYSIS TOOLS

To our knowledge, there are six publicly available security protocol analysis tools that fall into the category of attack-construction. These are Casper/FDR, CASRUL/SATMC, CASRUL/OFMC, a constraints-based verifier (Constraints), STA and SyMP/Athena. The software together with the accompanying documentation and other required software were acquired via file transfer over the Internet. This section gives a brief summary of each tool, outlining the major features and properties of them. For a more technical account of the tools, the reader is advised to consult the original descriptions referred to in each case.

Table I gives an overview of the respective functional properties of the tools as described in the previous section, represented as a feature matrix. Table II presents the non-functional properties of the tools. In the table, properties such as performance and interface are given a + symbol if the property is 'above average', 0 for 'average', and - for 'below average'. These are relative estimates that are informally determined either by comparing the available tool descriptions, or by comparing practical experiences with the tools.

A. Casper/FDR

Casper [17] is a program that takes a description of a security protocol in a simple, abstract language, and produces a CSP description of the same protocol, which can be checked using the FDR general-purpose model checker [18]. FDR uses a finite state machine formalism. It establishes whether a property (in our case secrecy or authentication) holds by testing for the refinement of a transition system capturing the property by the machine. This tool is one of the most mature in the area of automatic security protocol analysis. It has found attacks on 20 protocols previously known to be insecure, as well as attacks on 10 other protocols originally reported as secure. Casper/FDR handles the conventional cryptographic primitives, and also has a formalism for timestamps. However, it does not model constructed keys and is unable of detecting type flaw attacks. Casper is open source software, requires a Haskell interpreter, and runs on Unix/Linux systems only. FDR is a commercial product provided by Formal Systems (Europe) Ltd. It runs under FreeBSD, Solaris and Linux.

B. Constraint-Based System

This tool is based on the approach of converting the reachability problem associated with finding an illegal state in the protocol model into a constraint-solving problem. It was first introduced in [22] and was later improved in [11]. The honest principals are modelled with the Strand Space formalism[], which is a model developed especially for the reasoning about security protocols, and the attacker is modelled with term set closure characterisation. The result is an intuitive tool that takes a simple input without the need of any pre-processing. It can handle the usual cryptographic primitives, and can model constructed keys. The tool is implemented in Prolog, so it can run on most platforms.

C. STA

STA (Symbolic Trace Analyzer) [7] is a symbolic model checker for security protocol specifications. Protocol properties are expressed in terms of traces generated by the protocol, e.g., ”every commit action of principal B happens only after the corresponding begin action of principal A”. Like most other analysis tools it has an in-built attacker, but there is no high-level specification interface, so the specification of protocols can be quite tedious. It requires a certain acquaintance with process algebras, although no deeper understanding of security is needed. STA is written in ML. Currently, shared-key, public-key cryptography and hashing are supported.

D. CASRUL/SATMC

Like Casper, CASRUL [9] is also a translator from a High-Level Protocol Specification Language (HLPSL), which produces a notation more suitable for analysis tools. It was developed as part of the European Project AVISS project [2], with the objective to build an industry-strength tool for efficient protocol analysis. CASRUL takes a protocol specification together with a definition of the actual system to be checked, including agents taking part in the system and the roles they play, the datatypes to be used, and the intruder’s abilities. The output, called Intermediate Format (IF), is a rewrite notation that can be handled with by automatic analysers. It is available for Linux, SunOS and Windows, but no source code is provided.

SATMC [3] is a SAT-based (boolean satisfiability) model checker for the analysis of security protocols. This approach uses the combination of a reduction of protocol insecurity problems to planning problems and well-known SAT-reduction techniques, providing an automatic model-checker for security protocols based on state-of-the-art SAT solvers. It takes either its own language SATE, or the output language of CASRUL (IF) as input, then performs the analysis with one of three SAT solvers. It is developed with SICStus Prolog, compiled into a stand-alone executable for Linux. It is published under the GNU licence, but no source code is available.

E. CASRUL/OFMC

The On-the-Fly Model Checker (OFMC) [4], was also developed as part of the AVISS project, hence it takes as input the Intermediate Format (IF) that is the output of the translator CASRUL. It combines the use of lazy data types and optimisations for modelling a lazy intruder, whose actions are generated in a demand-driven manner. This way, the search space is reduced without excluding any attacks, and the result is a very efficient analysis tool. It has constructed-key capability and it can detect type flaw attacks. The tool is free to use and distribute, and it is available for Linux and SunOS, with a Windows version being considered.

F. SyMP/Athena

The Athena tool [25] combines model-checking and interactive theorem-proving techniques with the strand-space model to reduce the search space and automatically prove the correctness of security protocols with arbitrary numbers of concurrent runs. Interactive theorem-proving in this setting allows one to limit the search space drastically by manually proving lemmas (e.g. ”the intruder cannot find out a certain private key, as it is never transmitted”). However, the amount of user interaction necessary to obtain such statements might be considerable. The unbounded number of runs comes at a price, since there is no guarantee that the the search will terminate. Moreover, like Casper/FDR2, Athena supports only atomic keys, and cannot detect type flaws. Athena forms part of the Symbolic Model Prover (SyMP) [6] tool, which is available for Linux, but should work on other Unix platforms too. It is distributed as SML source code. To compile it, gcc, GNU make, Perl, CMU SMV, and SML are required. Emacs is needed to run the interactive prover.

V. Analysis

We have established suitable attributes for the evaluation of the tools, and completed the tool overview. In this section, we select the tool most suitable for integration with SPEAR II. This process is performed in two stages. First, the relative importance of the attributes is established, after which the selection analysis is conducted.

A. Weighting of Evaluation Properties

We have identified a set of analysis tool properties, but they are not all equally important in the context integration with the SPEAR II environment. What remains is assigning each attribute with a ’degree of importance’, or a weighting, that will assist in the evaluation. A simple way of doing this is to establish three categories for each set of attributes.

1) Functional Properties: For functional properties, the following categories were used: ’essential’ (■), ’preferable’ (⊠) and ’optional’ (□). ’Essential’ means that the feature must be provided, ’preferred’, that it should be chosen if possible, and ’optional’ means that it is a feature that is not required. Table III gives an overview of the weighting of each category.

Functional Property	Weighting
Secret Key Encryption	■
Public Key Encryption	■
Public Key Signature	■
Secure Hash	■
Timestamp	☐
Unbounded sessions	☐
Constructed keys	⊠
Type Flaw	⊠
Secrecy check	■
Authentication check	■
Termination	■

TABLE III

TOOL FEATURE IMPORTANCE: ■: ESSENTIAL, ⊠: PREFERABLE, ☐: OPTIONAL

Non-functional property	Weighting
Performance	⊠
Interface	⊠
Operational	☐
Resource	⊠
Portability	■
Open source	■
Licence	■

TABLE IV

TOOL FEATURE IMPORTANCE: ■: VERY IMPORTANT, ⊠: CONSIDER, ☐: NOT IMPORTANT

All cryptographic primitives except timestamps are classified as essential. These constitute the core of the functionality of a tool. With fewer available primitives, a smaller set of protocols can be analysed. Timestamps are optional because of their limited use in protocols, but also for the fact that very few tools support them.

An unbounded number of sessions in the model is optional, since it has been shown that two instances achieve the same result as an infinite model [10]. Even though type flaw attacks are avoidable if the protocol implementation is typed, they should be avoided in a specification. Therefore, this attribute is categorised as preferable. Both secrecy and authentication checks are essential since they form the basis of the security protocol analysis. The analysis is required to result in either an attack trace, or a completed search, which requires that the engine terminates in all situations.

2) Non-Functional Properties: For These properties, the following categories were used: ’very important’ (■), ’consider’ (⊠) and ’not important’ (☐). ’Very important’ means that the property is a deciding factor, ’consider’, that it should be taken into account, and ’not important’ means that the property can be ignored. Table IV gives an overview of the weighting of each category.

Performance should be considered. If there is a choice between two tools, the tool that has a shorter execution time for the same task will be chosen. The interface property should be considered in a similar fashion. An output/input interface will have to be added to the SPEAR II tool, and a notation that is easier to translate to/from will facilitate this.

The usability of the tool is not important as such, since all interaction with the tool is intended to be automated via the SPEAR II user interface. The required resources in form of software is considered, based on the software engineering principle that it is desirable to keep the number of dependencies as low as possible.

The portability, or rather the ’installability’ is very important, since the current SPEAR II implementation runs only on Windows platforms. The legislative aspect of the tool is also very important, at least when considering the source code policy. Open source is desirable because of two reasons. Firstly, since the software is made for security analysis, it is essential that it can be verified to carry out what it claims to do. Secondly, the integration may be facilitated by modifications to the tool itself.

B. Selection

The tool overview, including the evaluation criteria together with their individual weighting, are used to select the most suitable tool. The method for the selection is intuitive: The tool attributes are considered in descending order of significance, excluding tools that do not fulfil the requirements. The remaining candidates are then compared on a feature-by-feature basis, after which the tool that performs the best in this comparison is selected.

1) Functional Properties: This part of the evaluation is trivial. All the tools share the ’essential’ features, with the exception of the Athena tool that does not guarantee termination of the search. The ’preferable’ attributes are shared by the Constraints, SATMC, and OFMC tools, with Casper and STA not offering these features.

2) Non-functional Properties: Of the ’very important’ properties, the platform requirement is the main factor in the whole selection process. This arises since the chosen system needs to co-exist with an existing software base, resulting in the exclusion of four out of the six candidates. The the only two remaining tools are the Constraints tool and STA. The open source property is satisfied by Casper, Constraints and Athena, and as far as licensing is concerned, the only tool that really stands out is Casper, not because of Casper itself, but for the commercial licence that applies to the FDR analyser.

Of the properties that fall into the ’consider’ category, the two AVISS tools (SATMC and OFMC) stand out in the performance section. This is not surprising considering that they are the most recently developed tools available. The two tools that lack in their interfaces, mainly because of their tedious input notations, are STA and Athena. Finally, the only tool that can operate without depending on other software is OFMC, although its use is significantly facilitated by incorporating the CASRUL pre-processor. The Constraints tool, STA and SATMC all depend on only one external program, of which Constraints have the advantage of SPEAR II already using an external Prolog engine for its logic analysis. Casper has two dependencies (Haskell and FDR), and Athena is dependent on even more external resources.

3) Results: The result from this evaluation is unambiguous. The most suitable tool is actually a sole survivor. The reason for the exclusion of the other candidates is not because they performed badly, but is mainly a result of the SPEAR II platform-dependence. The only exception here is the STA tool which was excluded mainly because of the open source issue. However, even in a comparison between Constraints and STA where the open source property is ignored, the result would be the same, because of the other non-functional attributes.

In fact, even in a scenario in which SPEAR II could run on any platform, the result would be the same. In this case the main deciding factor would be the open source property coupled with the licensing issue.

This is not to say that platform-versatility does not matter. An important realisation from this process was that many very useful tools for security protocol analysis are developed for the Linux/Unix platform.

VI. Conclusions

In this paper we have focussed our attention on tools developed for automatically checking secrecy or authentication in models of security protocols. The tools we were interested in were those that return either a confirmation that no flaws were found in the protocol, or, if a flaw is found, returns a trace of the attack. The survey was conducted with the view to select one of the tools for incorporation in a multi-dimensional security protocol engineering framework. A set of attributes were identified, which formed a basis for an evaluation of the tools. The attributes were assigned various degrees of significance to aid the analysis, after which the selection was made.

The contribution of this paper is not limited to the selection of a tool for a specific project, but rather can be generalised to an attempt at approaching the topic from an application point of view. Existing surveys provide good insights into the inner workings of the tools, (many of which are not even available for practical use), but little attention is given to any non-functional aspects other than their performance.

A weakness of this overview is the lack of practical comparison of the tools, regarding performance and their general impact on practical cases. Such a task would require a careful comparison of specifications, security properties, and of all optimisations used by the different tools. This task, however, goes beyond the scope of the present survey.

One useful insight gained from this survey is that a cross-platform implementation of the SPEAR II framework would further increase its value. A wider range of security protocol analysis tools could be interfaced with, resulting in a truly universal security protocol engineering resource.

References

[1] R. Anderson and R. Needham. Programming Satan's Computer. *Computer Science Today, Springer LNCS*, 1000:426–441, 1995.

[2] A. Armando, D. Basin, M. Bouallagui, Y. Chevalier, L. Compagna, S. Mdersheim, M. Rusinowitch, M. Turuani, L. Vigano, and L. Vigneron. The aviss security protocol analysis tool. In E. Brinksma and K. G. Larsen, editors, *Computer Aided Verification, 14th International Conference*, volume 2404 of *Lecture Notes in Computer Science*, pages 349–353, Copenhagen, Denmark, July 2002. Springer.

[3] A. Armando, L. Compagna, and P. Ganty. SAT-based Model-Checking of Security Protocols using Planning Graph Analysis. In K. Araki, S. Gnesi, and D. Mandrioli, editors, *Proceedings of the 12th International Symposium of Formal Methods Europe (FME)*, LNCS 2805, pages 875–893. Springer-Verlag, 2003.

[4] D. Basin, S. Mdersheim, and L. Vigan. An On-The-Fly Model-Checker for Security Protocol Analysis. In E. Snekkenes and D. Gollmann, editors, *Proceedings of ESORICS'03*, LNCS 2808, pages 253–270. Springer-Verlag, Heidelberg, 2003.

[5] J. Bekmann, P. D. Goede, and A. Hutchison. SPEAR: Security Protocol Engineering and Analysis Resources. In *DIMACS Workshop on Design and Formal Verification of Security Protocols*. Rutgers University, September 1997.

[6] S. Berezin. *Model Checking and Theorem Proving: a Unified Framework*. PhD thesis, Carnegie Mellon University, 2002.

[7] M. Boreale and M. Buscemi. A framework for the analysis of security protocols, 2001. An abstract appears in Proc. of WSDAAL 2001, Como, Italy.

[8] M. Burrows, M. Abadi, and R. Needham. A logic of authentication. *ACM Transactions on Computer Systems*, 8(1):18–36, 1990.

[9] Y. Chevalier and L. Vigneron. A tool for lazy verification of security protocols, 2001.

[10] H. Comon-Lundh and V. Cortier. Security properties: two agents are sufficient. In P. Degano, editor, *Programming Languages and Systems, 12th European Symposium on Programming, ESOP 2003, Proceedings*, volume 2618 of *Lecture Notes in Computer Science*, pages 99 – 113. Springer, April 2003.

[11] R. Corin and S. Etalle. An improved constraint-based system for the verification of security protocols. In M. Hermenegildo and G. Puebla, editors, *Int. Static Analysis Symp. (SAS), Madrid, Spain*. Springer-Verlag, Berlin., Sep 2002.

[12] D. Dolev and A. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, 29(2):198 – 208, 1983.

[13] L. Gong, R. Needham, and R. Yahalom. Reasoning about Belief in Cryptographic Protocols. In *Proceedings of the 1990 IEEE Symposium on Research in Security and Privacy*, pages 234 – 248, Oakland, California, 1990. IEEE Computer Society Press.

[14] N. J. Hopper, S. A. Seshia, and J. M. Wing. Combining theory generation and model checking for security protocol analysis. In *Post-CAV Workshop on Formal Methods in Computer Security*, July 2000.

[15] IEEE. Recommended practice for software requirements specifications, December 1993. IEEE Std 830- 1993.

[16] F. Jacquemard, M. Rusinowitch, and L. Vigneron. Compiling and verifying security protocols. In *Logic Programming and Automated Reasoning*, pages 131–160, 2000.

[17] G. Lowe. Casper: A compiler for the analysis of security protocols. In *PCSFW: Proceedings of The 10th Computer Security Foundations Workshop*. IEEE Computer Society Press, 1997.

[18] G. Lowe and A. W. Roscoe. Using CSP to detect errors in the TMN protocol. *Software Engineering*, 23(10):659–669, 1997.

[19] W. Marrero, E. Clarke, and S. Jha. A model checker types for authentication protocols. In *DIMACS Workshop on Design and Formal Verification of Security Protocols*. Rutgers University, September 1997.

[20] C. Meadows. Invariant generation techniques in cryptographic protocol analysis. In *PCSFW: Proceedings of The 13th Computer Security Foundations Workshop*. IEEE Computer Society Press, 2000.

[21] J. Millen. CAPSL: Common authentication protocol specification language. Technical Report MP 97B48, The MITRE Corporation, 1997.

[22] J. K. Millen and V. Shmatikov. Constraint solving for bounded-process cryptographic protocol analysis. In *ACM Conference on Computer and Communications Security*, pages 166–175, 2001.

[23] E. Saul. Facilitating the modelling and automated analysis of cryptographic protocols. Master's thesis, DNA Research Group, Computer Science Department, University of Cape Town, 2001.

[24] B. Schneier. Why cryptography is harder than it looks. *Information Security Bulletin*, 2(2):31 – 36, March 1997.

[25] D. X. Song, S. Berezin, and A. Perrig. Athena: A novel approach to efficient automatic security protocol analysis. *Journal of Computer Security*, 9(1/2):47–74, 2001.

[26] P. Syverson and I. Cervesato. The logic of authentication protocols. *Lecture Notes in Computer Science*, 2171:63 – 136, 2001.

[27] B. Tobler and A. C. M. Hutchison. Generation, analysis and verification of cryptographic protocol implementations. In L. L. Jan Eloff and H. Venter, editors, *3rd annual Information Security South Africa Conference*, pages 297–306, Pretoria, South Africa, July 2003. ISSA, ISSA.

Simon Lukell is currently pursuing an M. Sc. degree in the Department of Computer Science at University of Cape Town, where he also received his B. Sc. (HONS) degree in 2002. He is specialising on network security and more specifically on facilitating automatic analysis of security protocol specifications.

Andrew Hutchison received the Ph.D. degree in computer science from the University of Zurich, Switzerland, in 1996. He currently works for T-Systems, South Africa, and he is an Adjunct Professor in the Department of Computer Science, University of Cape Town. His research interests include security protocol analysis, intrusion detection, and performance modelling.