IJCAR 2004

Second International Joint Conference on Automated Reasoning

University College Cork, Cork, Ireland

Workshop Programme

Automated Reasoning for Security Protocol Analysis (ARSPA)

Alessandro Armando, Luca Viganò (Chairs)

WS 6 – July 4

Preface

This volume contains the proceedings of the First Workshop on Automated Reasoning for Security Protocol Analysis (ARSPA) held in Cork (Ireland) on July 4, 2004, in the context of the Second International Joint Conference on Automated Reasoning (IJCAR'04).

The workshop brought together researchers and practitioners from both the security and the automated reasoning communities, from academia and industry, who are working on developing and applying automated reasoning techniques and tools for the formal specification and analysis of security protocols.

There were 18 submissions of high quality. All the submissions were thoroughly evaluated, and an electronic program committee meeting was held through the Internet. The program committee selected 9 research contributions which are included in this volume. The workshop program was enriched by an invited talk "Believing the Integrity of a System" by Simon Foley, whose position paper is also included in this volume, and by two short presentations.

We would like to thank the people who contributed to the organization of the ARSPA Workshop. We are grateful to: the other members of the organizing and program committee (namely David Basin, Jorge Cuellar, and Michaël Rusinowitch) and the additional referees, who allowed us to review the papers in a very short time while maintaining a very high standard (namely Rafael Accorsi, Massimo Benerecetti, Maria Grazia Buscemi, Carlos Caleiro, Iliano Cervesato, Luca Compagna, Ricardo Corin, Giorgio Delzanno, Véronique Cortier, Paul Hankes Drielsma, Pierre Ganty, Steve Kremer, Heiko Mantel, Sebastian Mödersheim, Adriano Peron, and Graham Steel). We thank Achim Brucker for his technical support.

The Program Chairs of the ARSPA Workshop

Alessandro Armando
DIST – Università degli Studi di Genova

Luca Viganò
ETH Zurich

June 2004

This is a preliminary version. The final version will be published in Electronic Notes in Theoretical Computer Science
URL: www.elsevier.nl/locate/entcs

Contents

Part I

Full Presentations

Believing the Integrity of a System *(Invited Talk)*

Simon N. Foley [1,2]

Department of Computer Science
University College Cork
Ireland

Abstract

An integrity policy defines the situations when modification of information is authorised and is enforced by the protection mechanisms of a system. Traditional models of protection tend to define integrity in terms of ad-hoc authorisation techniques whose effectiveness are justified more on the basis of experience and "best practice" rather than on any theoretical foundation. In a complex application system it is possible that an integrity policy may have been incorrectly configured, or that the protection mechanisms are inadequate, resulting in an unexpected system compromise. This paper examines the meaning of integrity and and describes a simple belief logic approach for analysing the integrity of a system configuration.

Key words: Security, Integrity, Security Protocols, Belief logics, System Configuration.

1 Introduction

The 2001 Computer Crime and Security Survey [9] reported that 196 of the respondents to the survey could quantify their losses due to unauthorised use of computer systems at a total of US$378 million in the previous year. While access-control mechanisms, firewalls and so forth may help counter such losses, we can never be confident about security unless we are provided with some assurance of their effectiveness. Such assurance may be achieved, in part, by analysing whether a formal description of the system upholds certain security properties. These properties include confidentiality (no unauthorised release of information) and integrity (no unauthorised modification of information). The study of integrity as a formal security property has received little attention

[1] Supported by Enterprise Ireland Basic Research Grant Scheme SC/2003/7/ *FITNYSS Foundations for Integrity and Analysis.*

[2] Email: s.foley@cs.ucc.ie

This is a preliminary version. The final version will be published in Electronic Notes in Theoretical Computer Science
URL: www.elsevier.nl/locate/entcs

within the research community; confidentiality has been extensively studied and is the better understood of the two properties.

Early security research [3] characterised integrity in terms of read-write access controls between subjects and objects. This provides for a very coarse interpretation of integrity [30]; for example, once granted access to an account database, a bank clerk can make any change to the customer's account details. Access triples, well-formed transactions, and the principles of encapsulation [8,25], provide finer grained control by constraining the operations that a subject may carry out on an object: the bank clerk may execute only deposit or withdraw operations to access an account database.

Many integrity compromises are a result of 'insiders' executing fraudulent but authorised operations [9]. For example, the bank clerk executes an account deposit without lodging actual funds. Separation of duty [8,13,37,35] controls decrease the potential for fraud by involving at least two individuals at different points in a transaction: for example, by reconciling bank accounts and funds received each day, a supervisor detects and corrects the fraudulent deposit by the clerk. Role Based Access Control models [31,32] and authorisation models [2,21] provide integrity controls based on structures that organise related operations into roles and constrain the way that roles may be assigned and/or inherited by users; separation of duty is expressed within these models using role constraints.

These conventional security models describe *controls* for achieving integrity; they take an operational and/or implementation oriented approach by defining *how* to achieve integrity. No attempt is made to formalise a *property* that defines *what* is meant by integrity. For example, [8] recommends a combination of separation of duties, access-triples and auditing as a strategy for achieving integrity: it does it not attempt to address what is meant by integrity. Confidence is achieved to the extent that good design principles have been applied; there is no assurance that an integrity property is upheld. Thus, when we define a complex separation of duty policy we do not know, for certain, whether a dishonest user can bypass the intent of the separation via some unexpected circuitous, but authorised, route.

Jacob [20] formalises integrity as a functional property. This interpretation of integrity means that an integrity mechanism determines whether the current request for an operation is authorised based on the history of past authorisation requests that led to the current state.

In our research [14,15], we argue that integrity should be regarded as a non-functional property. Non-functionality means that in order to determine authorisation, it is necessary to examine *every possible* history of requests that could have reached the current state. Non-interference is an example of a non-functional property, and while non-interference and its derivatives have been extensively studied [12,11,17,28,29], designing and verifying security mechanisms that uphold non-functional properties is known to be difficult [24,34]. Thus, we argue that building mechanisms that uphold integrity (a

non-functional property) can be expected to be as difficult as building mechanisms that uphold non-interference.

This difficulty in properly analysing the integrity of a system is illustrated in [14,15], whereby, to provide integrity guarantees it is necessary to model the behaviour of both the system (with its protection mechanisms) and the *infrastructure* in which the system operates. Infrastructure is everything that serves the system requirements: software, hardware, users, and so forth. Even if a system is functionally correct, the infrastructure is likely to fail: software fails, users are dishonest, do not follow procedures, and so forth. The system and its security mechanisms must be designed to be resilient to these infrastructure failures. Only when a system is characterised in this way can it become possible to completely analyse whether a particular system configuration (including security policy) ensures integrity.

The non-functional approach to integrity proposed in [14,15] provides a formal algebraic semantics that requires detailed formal specifications to be provided for the system and its infrastructure. This requires considerable specification effort. The cost of such in-depth specification and subsequent analysis may be justified for small critical security mechanisms. However, we argue that such integrity analysis would not scale well to the configuration of a large and/or complex application system because it would be necessary to formally specify and reason about the potential behaviour of *every* infrastructure component, user and so forth.

2 Analysis of Security Configurations

We are interested in developing shallow and pragmatic security analysis methods for systems. This is achieved through the analysis of how a system is *configured*, rather than an analysis of its underlying mechanisms and protocols. Instead of concentrating on detailed semantics and complete formal verification of components, we are concerned more with with the ability to trace, at a practical level of abstraction, how component security requirements relate to each other and any overall security requirements. We believe that a complete security verification of a system is not achievable in practice; we seek some degree of useful feedback from an analysis that a particular system configuration is reasonable.

In [4] we describe a modelling approach that requires less semantic detail about the operation of the system and its infrastructure. Rather than attempting to model the complete behaviour of the system and infrastructure (as in [15]), only those components that are considered relevant to the security policy and configuration are modelled. This is done by modelling the system and infrastructure in terms of the *constraints* that they impose over security relevant components of the system. This results in a definition of integrity consistency that can be solved as a constraint satisfaction problem [23]. An advantage to expressing integrity analysis as a constraint satisfaction problem

is that there exists a wide body of existing research results on solving this problem for large systems of constraints in a fully mechanised manner. Constraints have been used in many practical analysis tools, such as Concurrent Engineering and Computer Aided Verification.

Another example of a 'shallow' configuration analysis is given in [33]. This approach searches for possible conflicts between separation of duty, user role assignment and role inheritance rules in RBAC models. [4] provides an (albeit abstract) semantics for integrity that may be used to determine whether the separation of duty controls actually achieve external consistency and whether the application uses integrity mechanisms correctly.

3 Towards a Logic of Integrity

We argue that the integrity-algebraic approach [14,15] does not scale well to large complex systems. In this paper we propose a logic-based approach: such an approach facilitates a high-level analysis of a system that is too complex to be amenable to algebraic analysis. The basis for this approach rests on our conjecture that belief logics [19] can be used to provide a complementary approach to integrity/external consistency analysis.

Studying the effects of normal versus abnormal infrastructure behaviour of systems has been successfully applied to cryptographic security protocols, for example [10,26,27]. Analysis is based on a generic behaviour (called "spy" [27] or "adversary" [10]) that characterises the untrusted network (abnormal infrastructure) over which a security protocol operates. Algebraic integrity analysis [15] generalises this by considering a complex adversary that characterises the combined threats from the infrastructure that a protection mechanism must withstand. This viewpoint—that many different adversaries with differing behaviours should be modelled in the environment—is also adopted in [1] in the analysis of security protocols.

Belief logics such as [6,18,36] have also been successfully used to analyse beliefs held between participants in cryptographic security protocols. In these logics the behaviour of the adversary is implicit in the deduction rules and in the stated beliefs of the principals. Therefore, integrity analysis based on belief-logic is possible if one reflects the 'infrastructure adversary' in terms of suitable logic deduction rules and in terms of the beliefs held by principals. We are developing a belief logic that can be used to analyse integrity. The following example sketches how it might be used.

Let P, A, B, C represent principals, ϕ represent a formulae of the logic and X represents a message (which can also be a formulae). The logic has the following formulae constructions.

- $P \mid\!\equiv X$: P *believes*, or would be entitled to believe X.
- $P \mid\!\sim X$: P at some time *said/sent* a message that indicated X.
- ΥX: X is *consistent* with some real world value. We use this to characterise

integrity in terms of *external consistency*, which is informally described in [8] as *"the correct correspondence between data objects and the real world"*, for example, a bank account balance corresponds to the customer's own belief about the account based on withdrawals and deposits.

We assume that formulae can be derived using the usual propositional logic operators $\vee$, $\wedge$ and $\rightarrow$, and the usual K-Axiom of modal logics:

$$\frac{P \models (\phi_1 \rightarrow \phi_2), P \models \phi_1}{P \models \phi_2}$$

Consider a customer C making a deposit (via an envelope) to an ATM machine A. The relationship between these principals are described in terms of beliefs with respect to the belief modality of the customer C.

The customer believes that the ATM A registers her deposit correctly:

$$C \models A \mathrel{|\!\sim} C \mathrel{|\!\sim} \Upsilon \mathsf{dep} \tag{1}$$

This reflects the belief of the customer that when the transaction is complete the ATM has said/registered that the customer made (said) a consistent deposit.

The customer believes that once the bank (for simplicity, A) is satisfied with (believes) the consistency of the deposit then it updates the customer account acct.

$$C \models ((A \models \Upsilon \mathsf{dep}) \rightarrow \Upsilon \mathsf{acct}) \tag{2}$$

These two logical formulae represent the operational beliefs held by C concerning a deposit transaction.

The customer believes that the ATM is honest: it says only what it believes [3]:

$$C \models ((A \mathrel{|\!\sim} \phi) \rightarrow (A \models \phi)) \tag{3}$$

Thus, formula 1 reduces to

$$C \models A \models C \mathrel{|\!\sim} \Upsilon \mathsf{dep} \tag{4}$$

The customer also believes that the bank believes she honest, that is, she will only say things that can be believed.

$$C \models A \models ((C \mathrel{|\!\sim} \phi) \rightarrow \phi)$$

If the customer believes that her deposit is correct ($C \models \Upsilon \mathsf{dep}$), then given the nature of the system and infrastructure as described above, the following can be deduced within the logic.

$$C \models \Upsilon \mathsf{acct}$$

[3] Note that, for the sake of simplicity, we do not consider how *recently* A said a message, or whether it is a replay of some earlier message.

That is, the customer believes that her account has integrity as a result of the transaction.

This deduction is possible because the ATM and customer are individually honest and competent. If the customer does not believe that the bank will honestly reconcile the amount in the envelope then it is not possible to deduce this account consistency. For example if the customer did not believe that A is honest (deletion of formula (3)) then the goal cannot be deduced. This result is as expected and it indicates that further operational checks and balances may be needed in the system, if account consistency is to be ensured in the presence of such infrastructure dishonesty.

For example, suppose that a bank clerk B validates the deposit in the ATM. After a transaction, the customer believes the clerk confirms the amount in the envelope

$$C \models B \mid\sim C \mid\sim \Upsilon \mathsf{dep} \tag{5}$$

However, suppose that only one of A and B can be considered honest. Formula (3) is replaced by

$$\begin{aligned} C \models (((A \mid\sim \phi) \rightarrow (A \models \phi)) \\ \vee ((B \mid\sim \phi) \rightarrow (B \models \phi))) \end{aligned} \tag{6}$$

Furthermore, suppose that the software that reconciles what A and B say about a deposit has a way to detect which is honest [4]. Thus, formula (2) is replaced by

$$C \models (((A \models \Upsilon \mathsf{dep}) \vee (B \models \Upsilon \mathsf{dep})) \rightarrow \Upsilon \mathsf{acct}) \tag{7}$$

These last two formulae correspond to a separation of duty-like rule on the transaction and it becomes possible to derive the original goal $C \models \Upsilon \mathsf{acct}$.

Note that a complete analysis should also consider the belief modality with respect to bank: can it be defrauded by dishonest customers?

Our example is grossly simplified. We are developing an expressive integrity logic, making it possible to properly express beliefs about operations and/or techniques that are used to guarantee integrity, such as cryptography, separation of duty, well-formed transactions, auditing and so forth.

4 Discussion and Conclusion

An abstract logic based approach facilitates high-level analysis of a system that is too complex to be amenable to algebraic analysis. The logic approach follows the strategy to make only the needed distinctions and no more and simplifies the specification and analysis of integrity.

A potential weakness of taking the logic approach is that, like existing authentication logics, it is possible that an integrity logic will miss classes

[4] For simplicity, we have abstracted away the other bank principals that determine this when they resolve conflicts that arise between the Clerk and ATM.

of attacks that could be identified by the algebraic approach in [15]. In the integrity logic we can model the infrastructure in terms of beliefs about the honesty and competency of the components. While limited in expression, this results in specifications that are relatively easy to develop and analyse. This is contrasted with the algebraic approach [15] which requires the behaviour of each principal/infrastructure to be fully specified, leading to unwieldy specifications that are difficult to develop and require a high degree of expertise to verify. The logic approach trades off completeness of analysis against ease of use and the ability to conduct fully automatic analysis of complex systems [5,22].

The logic of integrity outlined in this paper is an adaptation of the Simple Logic [7] which uses an expressive but very simple logical system. We are currently exploring automated analysis techniques; we have implemented an automatic verification tool for the Simple Logic using Theory Generation [22]. In [38] an automatic protocol generator is described that uses synthesis rules to compute protocols within the Simple Logic. Future research will explore how such techniques can be used to automatically synthesis acceptable security (integrity) policy configurations that uphold external consistency.

It is evident from the example above that if a configuration policy (of the system/infrastructure) is to be specified in terms of low-level beliefs then the exercise has the potential to be tedious and error prone (although, we believe less problematic than the algebraic approach). A topic of future research is to investigate the potential of using requirements elicitation techniques in the specification of security policy configurations. In particular, [16] takes a Safety Engineering approach in which Hazard Analysis is used for the elicitation of security protocol requirements.

Acknowledgements

Thanks to Stefano Bistarelli for many useful discussions on the nature of Integrity. Development and implementation of the Theory Generation encoding of the Simple Logic verifier was by Daithi O'Crualaoich and its adaptation to an automatic protocol generator by Hongbin Zhou.

References

[1] G. Bella and S. Bistarelli. Soft Constraints for Security Protocol Analysis: Confidentiality. In *Proc. of the 3rd International Symposium on Practical Aspects of Declarative Languages (PADL'01)*, LNCS 1990, pages 108–122. Springer-Verlag, 2001.

[2] E. Bertino et al. An authorization model and its formal semantics. In *Proceedings of the European Symposium on Research in Computer Security*, pages 127–142. Springer LNCS 1485, 1998.

[3] K.J. Biba. Integrity considerations for secure computer systems. Technical Report MTR-3153 Rev 1 (ESD-TR-76-372), MITRE Corp Bedford MA, 1976.

[4] S. Bistarelli and S.N. Foley. Analysis of integrity policies using soft constraints. In *Proceedings of IEEE Workshop Policies for Distributed Systems and Networks*, pages 77–80, June 2003.

[5] Stephen H. Brackin. A HOL extension of GNY for automatically analyzing cryptographic protocols. In *PCSFW: Proceedings of The 9th Computer Security Foundations Workshop*. IEEE Computer Society Press, 1996.

[6] M. Burrows, M. Abadi, and R. M. Needham. A logic of authentication. Technical Report Report number 39, Digital Systems Research Center, February 1989.

[7] L. Buttyán, S. Staamann, and U. Wilhelm. A simple logic for authentication protocol design. In *Proceedings of the 11th IEEE Computer Security Foundations Workshop (CSFW '98)*, pages 153–163, Washington - Brussels - Tokyo, June 1998. IEEE.

[8] D. D. Clark and D. R. Wilson. A comparison of commercial and military computer security models. In *Proceedings Symposium on Security and Privacy*, pages 184–194. IEEE Computer Society Press, April 1987.

[9] Computer Security Institute/US Federal Bureau of Investigation. *Computer Crime and Security Survey*, 2001.

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

[11] R. Focardi and R. Gorrieri. A classification of security properties for process algebras. *Journal of Computer Security*, 3(1):5–33, 1995.

[12] S.N. Foley. A universal theory of information flow. In *Proceedings 1987 IEEE Symposium on Security and Privacy*, pages 116–121. IEEE Computer Society Press, 1987.

[13] S.N. Foley. The specification and implementation of commercial security requirements including dynamic segregation of duties. In *ACM Conference on Computer and Communications Security*, pages 125–134, 1997.

[14] S.N. Foley. Evaluating system integrity. In *Proceedings of the ACM New Security Paradigms Workshop*, 1998.

[15] S.N. Foley. A non-functional approach to system integrity. *Journal on Selected Areas in Communications*, 21(1), Jan 2003.

[16] Nathalie Foster and Jeremy Jacob. Hazard analysis for security protocol requirements. In *Advances in Network and Distributed Systems Security, IFIP TC11 WG11.4 First Annual Working Conference on Network Security*, IFIP Conference Proceedings, 2001.

[17] J. A. Goguen and J. Meseguer. Unwinding and inference control. In *Proceedings 1984 IEEE Symposium on Security and Privacy*, pages 75–86. IEEE Computer Society, 1984.

[18] Li Gong, R. M. Needham, and R. Yahalom. Reasoning about belief in cryptographic protocols. In D. Cooper and T. Lunt, editors, *Proceedings 1990 IEEE Symposium on Security and Privacy*, pages 234–248, May 1990.

[19] G.E. Hughes and Cresswell M.J. *An Introduction to Modal Logic*. University Paperbacks, Methuen Press, 1968.

[20] J.L. Jacob. The basic integrity theorem. In *Proceedings of the Computer Security Foundations Workshop IV*, pages 89–97. IEEE Computer Society Press, June 1991.

[21] S. Jajodia et al. A logical language for expressing authorizations. In *Symposium on Security and Privacy*, pages 31–42. IEEE Press, 1997.

[22] D. Kindred and J.M. Wing. Theory generation for security protocols. *ACM TOPLAS*, July 1999.

[23] A.K. Mackworth. Constraint satisfaction. In S.C. Shapiro, editor, *Encyclopedia of AI (second edition)*, pages 285–293. John Wiley & Sons, 1992.

[24] J. McLean. A general theory of composition for trace sets closed under selective interleaving functions. In *Proc. IEEE Symposium on Research in Security and Privacy*, pages 79–93, 1994.

[25] R. Needham. Later developments at cambridge: Titan, cap and the cambridge ring. *Annals of the History of Computing*, 14(4):57, 1992.

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

[27] A.W. Roscoe. Using intensional specifications of security protocols. In *Proceedings of the Computer Security Foundations Workshop*, pages 28–38. IEEE Press, 1996.

[28] A.W. Roscoe, J.C.P. Woodcock, and L. Wulf. Non-interference through determinism. *Journal of Computer Security*, 4(1), 1995.

[29] P.Y.A. Ryan and S.A. Schneider. Process algebra and non-interference. In *IEEE Computer Security Foundations Workshop*, pages 214–227, 1999.

[30] R. Sandhu. A perspective on integrity mechanisms. In *Proceedings of the Computer Security Applications Conference*, page 279, 1989. Panel Discussion paper.

[31] R Sandhu et al. Role based access control models. *IEEE Computer*, 29(2):38–47, 1996.

[32] R. Sandhu, D. Ferraiolo, and R. Kuhn. The nist model for fole-based access-control. In *Proceedings of the ACM Workshop on Role-Based Access Control*, 2000.

[33] A. Schaad and D. Moffett. The incorportation of control principles into access control policies. In *Workshop on Policies for Distributed Systems and Networks*, Bristol, UK, 2001.

[34] F.B. Schneider. Enforcable security policies. *ACM Transactions on Information and Systems Security*, 3(1):30–50, February 2000.

[35] R. Simon and M. Zurko. Separation of duty in role based environments. In *Proceedings of the Computer Security Foundations Workshop*, pages 183–194. IEEE Press, 1997.

[36] P. Syverson and P.C. van Oorschot. On unifying some cryptographic protocol logics. In *Proceedings of IEEE Computer Security Foudations Workshop*, pages 14–28, 1994.

[37] U. S. Department of Defense. Integrity-oriented control objectives: Proposed revisions to the trusted computer system evaluation criteria (TCSEC). Technical Report DOD 5200.28-STD, 1991.

[38] H. Zhou and S.N. Foley. Fast automatic synthesis of security protocols using backward search. In *ACM Workshop on Formal Methods for Security Engineering*, Washington, DC, USA, 2003.

Extending Security Protocol Analysis: New Challenges [1]

Mike Bond [2] Jolyon Clulow [3]

Computer Laboratory
University of Cambridge
Cambridge, UK.

Abstract

We argue that formal analysis tools for security protocols are not achieving their full potential, and give only limited aid to designers of more complex modern protocols, protocols in constrained environments, and security APIs. We believe that typical assumptions such as perfect encryption can and must be relaxed, while other threats, including the partial leakage of information, must be considered if formal tools are to continue to be useful and gain widespread, real world utilisation. Using simple example protocols, we illustrate a number of attacks that are vital to avoid in security API design, but that have yet to be modelled using a formal analysis tool. We seek to extract the basic ideas behind these attacks and package them into a wish list of functionality for future research and tool development.

Key words: Security APIs, Formal Methods, Protocol Analysis, Perfect Encryption, Information Leakage

1 Introduction

Security protocols have been designed, studied and attacked for over thirty years. Today, formal analysis is becoming a popular tool for assisting in the design process. However, the assumptions that formal tools make and the restrictions they put on the description and analysis of behaviour conspire to limit their scope – preventing their application to harder protocol design problems of today. In particular, the design of security APIs as well as conventional protocol design in constrained environments (such as within embedded systems) cannot benefit fully from the existing tools because of these impractical

[1] The authors wish to acknowledge the generous funding of the CMI Institute and the Cecil Renaud Educational and Charitable Trust.

[2] Email: Mike.Bond@cl.cam.ac.uk

[3] Email: Jolyon.Clulow@cl.cam.ac.uk

This is a preliminary version. The final version will be published in Electronic Notes in Theoretical Computer Science
URL: www.elsevier.nl/locate/entcs

assumptions and restrictions. While designers can achieve security through systematic application of rules of thumb for fulfilling the assumptions, the results tend to be over-engineered and impractical to deploy. Instead, we propose that the time has come to extend these tools to relax the assumptions on the models they analyse.

We describe some well-known mistakes that need to be avoided in good protocol and API designs, yet which cannot be reasoned about in the abstract models used by formal tools. These mistakes are of particular significance as they are regularly discovered within security APIs, but are illustrated for simplicity with example security protocols.

We believe that formal reasoning about many of these lower-level attacks is possible, and present this as a challenge for the formal methods community to adapt and extend their tools, to assure their continued usage and eventual widespread acceptance into the design process.

2 Analysing Protocols with Formal Methods

Numerous tools and techniques for formal analysis of security protocols exist; they can be broadly split into model checkers, theorem provers and formal logics.

- *Model checkers* explore a state space, examining methodically whether certain requirements hold in each state of the model. Some can also reason about equivalence between state space models, or use mathematical techniques to reason about entire sets of states simultaneously. Theoretically, they can examine the entire state space and can give a similar assurance of correctness as that provided by a theorem prover (in practise however, the problems set by users are often too hard and defeat analysis, or are deliberately simplified to ensure solubility).
- *Theorem provers*, in contrast, search at a higher level of abstraction for chains of logic that constitute a compelling proof that a particular property always holds. Alternatively, they may find a counter-example in the process. Various proof search strategies are used, often based on the basic resolution strategy proposed by Robinson [19].
- Finally, *formal logics* provide the user with notation and precise definitions of properties, which help the user to perform intuitive reasoning more rigorously on paper.

The tools vary both in raw reasoning power and rigour. Some formal tools have been designed for *assurance* – prizing most of all that any answer that they produce is truly correct; that no special cases or peculiar conditions are missed by the tool (or any of its optimisations) that might affect the validity of the answer. This is often reflected in the tool's precise and pedantic specification language. On the other hand, some tools focus upon efficient searching methodologies and use powerful heuristics similar to AI strategies

to control their searches. Out of the dozens of existing tools, many have had notable success aiding human analysis, and some have directly or indirectly discovered new attacks.

In particular, Lowe had considerable success applying the FDR tool to model security protocols, finding a previously unknown attack on the Needham-Schroeder Public Key protocol [14]. The FDR tool reasons about refinement properties between models based upon communicating sequential processes; Lowe later built a high-level interface language, CASPER [15], for specifying security protocols and their properties.

Meadows describes several type-confusion attacks against the Group Domain of Interpretation Protocol [17] that were found with the NRL protocol analyser. One she describes as found *automatically*, whilst the other more feasible variant was identified by a human during the course of the analysis. Mitchell *et al.* used their own tool, Murϕ, to analyse successfully a number of classic protocols for known flaws. Following this, they analysed the secure sockets layer (SSL) protocol, identifying a previously unknown 'anomaly' in the version 3.0 specification [18].

Finally, there are reasoning logics such as the BAN logic [12]. Though instrumental in the understanding and development of formal analysis of security protocols, the BAN logic itself has not been widely used to identify flaws; its main role has been in retrospective analysis. Individual researchers regularly use custom extensions and improved logics [13,1,20] in the protocol design process.

It remains rare that vulnerabilities are found purely through automated analysis, as most analysts are highly-skilled, and spot the flaw during the process of formal specification. When the automated analysis phase does find a new attack outright, there is usually a special circumstance: for example, when the analysts are the tool creators, and their primary goal is to explore the powers of their own tools, perhaps by experimenting on protocols of their own design.

We explored the existing tools for security protocols analysis, applying them to security APIs, initially just modelling known attacks rather than searching for new ones. However, nearly every tool fell short in some category of functionality – either they had difficulty 'finding' the attack, or it simply could not be expressed. These attacks ranged in degrees of complexity, but some of the most trivial and obvious weaknesses from the perspective of a security API designer caused a remarkable amount of trouble.

3 From Protocols to Security APIs

Security API analysis closely resembles the analysis of protocols. Consider a cryptographic processor (imagine a PC in a safe) that is network-attached and is used as a service by one or more users, quite similar in concept to a trusted third-party in a security protocol. The device makes its functionality available

through an *Application Programming Interface (API)*. Users can thus offload their sensitive, and often computationally expensive security operations to a dedicated, optimised device. This device may have additional benefits and responsibilities not found in a normal PC, e.g. a cut-down operating system hardened against attack, and special storage for cryptographic keys, possibly in tamper protected memory. Indeed, international funds-transfer and credit card infrastructures are based around devices such as these.

When a user participates in a protocol run, the user may make a number of individual calls to the security device to process the received message and generate the response. We can consider these API interactions as runs of the special protocol between the user and the trusted third-party device. This underlying API will, in general, have finer granularity than the protocol it supports since interpreting a message and generating the response may involve many cryptographic operations, each potentially encapsulated in a single API call. Furthermore, the security device APIs are typically designed for flexibility (the manufacturer of the device has economic incentives to maximise market size and limit the development costs of associated with individual customisation) – so one can thus construct a great many protocols from a given set of API calls. A typical security API may have a few hundred calls, each parameterised, many of which could be expressed as a simple two-party protocol between user and device.

The daunting size complexity of security APIs, coupled with the obvious similarities with protocols, makes application of automated methods developed for protocols an attractive option. Indeed this is not a brand new idea: Longley and Rigby described early work in 1992 using an automated tool based on PROLOG to analyse security APIs of key management systems [23] (it is only since 2000 that security APIs have been subjected to widespread scrutiny). In fact, a surprising number of vulnerabilities have been discovered in security APIs over the past decade [2,3,5,6,7,10], many of which form part of mission critical, commercially important systems: attacks against ATM networks and banking infrastructures in these papers are good examples. With the increasing commoditisation of security and security products, the need for automated methods to verify the security of systems in the absence of true domain specialists is increased. Financial industries in particular would welcome improved assurance. We are also observing a trend in security APIs toward extensibility – users may develop their own calls which can be downloaded to the device and added to the API. Again there is a need to verify that security is not compromised by these extensions.

In this paper, we seek to extract the basic ideas behind these attacks and package them into a wish list of functionality for future tools – functionality which we believe is not being sufficiently addressed by the current avenues of development in protocols analysis tools.

4 Perfect Encryption

Is $\{X\}_K$ secure? The typical automated reasoning tool assumes so. There is the hidden, albeit obvious, assumption that the cipher used is secure and the key sufficiently long to prevent a brute force attack. However, perfect encryption is not necessarily a reasonable assumption on many platforms. DES lives on in many environments while AES is not yet as widely supported as one might have hoped, particularly in small and low cost devices. Indeed many embedded systems often still use even more lightweight ciphers for specific applications, for instance, wireless car key-fobs where every bit transmitted is expensive in power consumption. The problem of key length is not limited to low cost devices: the PKCS #11 standard was recently shown to not explicitly preclude the option of encrypting a secret key under a weaker algorithm or shorter key, which would result in degraded security [11]. Many legacy systems migrated from DES to 3DES without properly thinking through storage requirements for the longer keys. For example, the double length key $K = \langle K_L, K_R \rangle$ would be commonly encrypted as $\{K\}_{KM} = \langle \{K_L\}_{KM}, \{K_R\}_{KM} \rangle$. Since there is no cryptographic binding between the two halves, one can easily cut and paste them. For example, an attacker could create two keys $\langle \{K_L\}_{KM}, \{K_L\}_{KM} \rangle$ and $\langle \{K_R\}_{KM}, \{K_R\}_{KM} \rangle$. Each key is effectively a single length key, and a brute force attack is thus significantly easier. The ability to identify such issues through automated analysis of security APIs would be desirable. Extending existing tools to reason about 3DES as 3 individual crypto operations would appear to be easy. It seems that the challenge here is not to merely be able to reason but rather to reason efficiently, as well as encoding the problem concisely and being easy to use.

There are also attacks on these systems which can do better than a straight attack against the cipher algorithm. Some require a data set that exhibits or satisfies a given condition: for example, a set of keys with the same plaintext encrypted under each, to effect a *parallel key search*. The use of parallelism in an exhaustive search was described in [22]. Given many keys, each encrypting the same text, it is possible to search for all keys in parallel (or simultaneously) using an exhaustive key search machine. The key search machine iteratively generates keys and then performs a trial encryption of the common plaintext. It then compares the resulting ciphertext with the data set and searches for a match (decryption can of course be used in the same way, in place of encryption). In this process, it searches for many keys in parallel at the expense of only a single DES operation. If searching for 2^n keys, we can expect to encounter on average one in 2^{55-n} operations. This technique was used in [6,5] against several security APIs.

We have encountered many APIs in the wild that are vulnerable to parallel key searches through a number of different calls. For example, Figure 1 is a simple protocol that encrypts a user supplied value X using an offset i applied

to the key.

$$A \longrightarrow S: \quad X, \{K\}_{KM}, i$$
$$S \longrightarrow A: \quad \{X\}_{(K \oplus i)}$$

Fig. 1. The Encrypt using a Key Offset Protocol

It is trivial to generate the data set required to mount the attack. The protocol shown in Figure 2 facilitates a user transmitting an encrypted message to a target user with whom he does not share a key. This is achieved through the use of a trusted server that has keys established with all the users. The server accepts an encrypted message from the initiating user, decrypts the message and then re-encrypts it for specified target user using the keys it shares with both users. If there exist sufficiently many valid identities for the target j an attacker can again easily generate the required data set.

$$A \longrightarrow S: \quad \{X\}_{K_{AS}}, A, j$$
$$S \longrightarrow A: \quad \{X\}_{K_{jS}}$$

Fig. 2. The Translate Protocol

This simple weakness is not limited to only APIs but also some established protocols. The venerable Needham-Schroeder protocol shown in Figure 3 is similarly vulnerable. Eve iteratively impersonates the identities of all possible parties A_j. For each A_j Eve initiates a protocol run with the server S supplying the same value X in place of the random nonce R_A. The server responds with $\{X, \ldots\}_{K_{A_jS}}$. Eve can now perform the parallel search offline to recover the value of a K_{A_jS}.

$$A \longrightarrow S: \quad A, B, R_A$$
$$S \longrightarrow A: \quad \{R_A, B, K_{AB}, \{K_{AB}, A\}_{K_{BS}}\}_{K_{AS}}$$
$$A \longrightarrow B: \quad \{K_{AB}\}_{K_{BS}}$$
$$B \longrightarrow A: \quad \{R_B\}_{K_{AB}}$$
$$A \longrightarrow B: \quad \{R_B - 1\}_{K_{AB}}$$

Fig. 3. The Needham-Schroeder Protocol

$$E \longrightarrow S: \quad A_j, B, X$$
$$S \longrightarrow E: \quad \{X, B, K_{A_jB}, \{K_{A_jB}, A\}_{K_{BS}}\}_{K_{A_jS}}$$

Fig. 4. Using the Needham-Schroeder Protocol to effect a parallel key search

Formal tools capable of analysing protocols and APIs under which the necessary data could be obtained to perform such an attack would be use-

ful. Alternatively, one could calculate a numerical bound which limits the parameters of the system and thereby ensures security.

5 Information Leakage

Information leakage is usually thought of in the context of side-channel analysis of physical devices engaged in security protocols. However protocols themselves may leak a small amount of information, perhaps a few bits or a fraction of a bit, which if identified, can be recovered and accumulated through repeated protocol runs (or sequences of API calls), eventually revealing an entire secret, or bringing it within range of a brute-force search. There are strong similarities between these channels and conventional side channels exploited during power analysis, timing or emissions attacks. However, a crucial difference is that key material processed in cryptographic algorithms is often processed iteratively, so observation of a characteristic may permit the identification of an explicit bit of the secret. Repetition is used in the attack process to target other distinct bits of the key, or to reduce noise from the analysis. In the case of information leakage through protocols, the correspondence between the data leaked and the key bits of some secret may be much more complex. A non-trivial algorithm may be required to convert the information revealed about the secret into knowledge of the secret itself. A good example of this is the game MasterMind (TM). One player choses a pattern of four coloured pegs that he fixes. The second player tries in an iterative manner to guess the pattern. With each guess, the first player tells the second player the number of pegs of the correct colour in the correct place and the number of pegs of the correct colour in the incorrect place. In effect, the second player is given partial information about the secret. The player's objective is to develop an efficient strategy to turn this information into knowledge of the secret itself. Incidentally, the strategy used in the decimalisation table attack [10,8] to extract the digits of a customer PIN from a bank security device is remarkably similar to that of the MasterMind game.

In practice, common sources of leaked information are error conditions and codes. These can be explicitly revealed through a message or indirectly revealed through timing patterns indicating the premature halting or abnormal execution of an operation. Bleichenbacher [25] and Manger [24] have both proposed attacks on the various RSA padding schemes: in these protocols an error response, or the timing of an error response leaks information about the plaintext, allowing a chosen ciphertext attack on RSA. Often such a leak is not visible or not explicitly stated during design process but becomes visible during coding or development. The attacker is often faced with the challenge that sometimes it may be the case that we can readily determine that there exists some form of information leakage but not be able to clearly identify what information is being leaked nor how it can be exploited.

Another example lies in the ANSI X9.8 standard for encrypting PINs un-

der DES. The PIN is formatted into an 8 byte buffer with control information and padding. This buffer is then exclusive-ORed with the customer's personal account number (PAN) before being encrypted under the DES key. We represent this as $\{P \oplus A\}_{K_B}$ where K is the key, P the formatted PIN buffer and A the PAN. Whenever the encrypted PIN is verified or re-encrypted under a different zone key, the process is reversed. As an intermediate step the PIN is extracted and and integrity check is applied. Since each digit of the PIN is meant to be a decimal digit in the range 0 to 9, the check simply tests that each hexadecimal PIN digit extracted from the decrypted buffer is less than 10. Should this test fail, it means that either the PAN, key or encrypted PIN is incorrect or corrupted.

Essentially we can simplify this and represent it in a simple protocol that, given a user specified value for the PAN (which we denote X), tests whether the recovered PIN is decimal. For a single digit PIN, the protocol can be described as follows.

$$A \longrightarrow B: \quad X, \{P \oplus A\}_{K_B}$$
$$B \longrightarrow A: \quad (P \oplus A) \oplus X < 10$$

Fig. 5. The PIN integrity Check Protocol

It was not necessarily the implicit intention of the authors of the ANSI X9.8 standard to create this protocol, but it results as a consequence. At first glance, the integrity check seems innocent and benign enough, and indeed perhaps a useful addition that detects unintended errors. However, with a little thought it becomes obvious that repeated execution of this protocol with different values of X quickly leads to the identification of the set $\{P, P \oplus 1\}$. This can clearly be seen from Table 6. A given value of $P \oplus A$ results in a unique pattern of passes and fails.

Clearly a method that identifies potential leakages of information is required and some understanding of how this information might be used. Actually constructing an algorithm that assimilates the leaked information and reconstructs the underlying secret might be lofty goal since this could require considerable creativity. Perhaps a more realistic aim would be to identify the rate at which information is lost and establish a bound on the security of the protocol.

6 Protecting Low-Entropy Data

Low entropy-data (weak secrets and/or guessable data) is a common target of attack in fielded computer systems. Cryptographic protocols often have to work with this vulnerable data – be it in conventional security protocol interactions for authenticating principals with weak passwords, boot-strapping strong session key agreement, or in larger systems with security concerns, for instance interrogating an encrypted, anonymised database.

		$P \oplus A$				
		0,1	2, 3	4, 5	6, 7	8, 9
	0,1	Pass	Pass	Pass	Pass	Pass
	2,3	Pass	Pass	Pass	Pass	Fail
	4,5	Pass	Pass	Pass	Pass	Fail
X	6,7	Pass	Pass	Pass	Pass	Fail
	8,9	Pass	Fail	Fail	Fail	Pass
	A,B	Fail	Pass	Fail	Fail	Pass
	C,D	Fail	Fail	Pass	Fail	Pass
	E,F	Fail	Fail	Fail	Pass	Pass

Fig. 6. Table identifying the PIN using the PIN integrity Check Protocol

Formal automated reasoning about weak secrets is tentatively being integrated into our existing toolset. Lowe describes how FDR can reason effectively about "guessable secrets", and describes analysis of three security protocols which bootstrap from a weak secret [16]. Lowe considers the scenario where the weak secret is used as a key and the "guessing" occurs offline. We propose that the model can be extended to include:

- guessing attacks against weak secrets protected by strong keys (i.e., the weak secrets as data),
- guessing attacks that allow for false positives or are based on statistical information, and
- online guessing.

Furthermore, the applications of weak secrets analysed so far are restricted to authentication and key-establishment – we are finding that weak secrets manipulated by security APIs are much more vulnerable.

In [7,9], Bond describes several statistical attacks on customer PINs manipulated by Security APIs in bank computer systems. These four digit PINs are stored in encrypted 'PIN blocks', and are certainly guessable secrets. The attacks exploit known statistical characteristics of customer PINs introduced by a poorly-chosen PIN generation procedure becoming visible in the distribution of encrypted data blocks themselves. It is the range of manipulations that can happen to these weak secrets which presents the challenge for analysis. Not only can guesses against the weak secret be verified, but the secret can have mathematical operations performed on it (e.g. addition of constants, truncation), small quantities of known constants can be added to the set of weak secrets, and must not permit compromise of the others. This richness of manipulation is reminiscent of personal data stored in public databases.

Take for example, Anderson's summary of problems with the release of personal medical records to permit research: he describes attacks that can be performed on database systems to discover some weak secret (for example, is a person infected with HIV) [4]. It turns out that there are difficult trade-offs between allowing legitimate research requests and preventing abuse. There has also been substantial quantitative work done in the preservation of anonymity in census data [21] (which is more rigidly structured than medical data), which could be brought to bear.

A tough challenge will be to extend the work already done on guessable passwords to develop a more generic framework for reasoning about information flow through security protocols. New tools and techniques must be able to cope with leakage that may be both necessary and acceptable, but still provide assurance that the total rate of leakage cannot exceed some limits.

7 Conclusions

The formal study of security protocols has yielded many successes: from the accurate definition and reasoning about key concepts, to secure and efficient algorithms for authentication, key-establishment, and key-distribution . These categories of problem have largely been solved. Promising new work is extending formal analysis to deal with ever more complex and specialist protocols (e.g. for multi-party signature, or group key distribution), however there is another direction in which they can extend, that we believe yields much more practical value. The everyday problems involved in security API design, and in re-inventing existing protocols in new constrained mobile environments present exciting new challenges for tool makers and formal analysts. We believe that tools that can begin to address the three challenges we have presented would be invaluable to todays protocol and security API designers, as they face the new security challenges of tomorrow's mobile, ubiquitous and trusted computing.

References

[1] M. Abadi, M. Tuttle "A semantics for a logic of authentication", 10th ACM Symposium on Principles of Distributed Computing, p. 201–216, ACM Press, 1991

[2] R. Anderson "Why Cryptosystems Fail", Communications of the ACM vol 37 no 11 (November 1994), p. 32–40, 1994

[3] R. Anderson "Correctness of Crypto Transaction Sets", 8th International Workshop on Security Protocols, Cambridge, UK, April 2000

[4] R. Anderson "Security Engineering", Wiley, 2001

[5] M. Bond, R. Anderson, "API Level Attacks on Embedded Systems", IEEE Computer Magazine Oct 2001, p. 67–75

[6] M. Bond "Attacks on Cryptoprocessor Transaction Sets", CHES 2001, Springer LNCS 2162, p. 220–234

[7] M. Bond "Understanding Security APIs", Phd. Thesis, University of Cambridge http://www.cl.cam.ac.uk/~mkb23/research.html, 2004

[8] M. Bond, P. Zielinski "Decimalisation Table Attacks for PIN Cracking", University of Cambridge Computer Laboratory Technical Report TR-560, Jan 2003

[9] M. Bond, J. Clulow "Encrypted? Randomised? Compromised? (When Cryptographically Secured Data is not Secure)", Workshop on Cryptographic Algorithms and their Uses, 2004 (to appear)

[10] J. Clulow "The Design and Analysis of Cryptographic APIs", MSc. Dissertation, University of Natal, South Africa, http://www.cl.cam.ac.uk/~jc407/ , 2003

[11] J. Clulow, "On the Security of PKCS#11", CHES Workshop 2003, Cologne, Germany, LNCS 2779 pp. 411–425

[12] M. Burrows, M.Abadi, R. Needham, "A Logic of Authentication", ACM Transactions on Computer Systems, 1990, pp. 18–36

[13] L. Gong, R. Needham, R. Yahalom "Reasoning about belief in cryptographic protocols", IEEE Symposium on Security and Privacy, p. 234–248, IEEE Computer Society Press, 1990

[14] G. Lowe "Breaking and fixing the Needham-Schroeder public key protocol using FDR" Tools and Algorithms for the Construction and Analysis of Systems, p. 147–166, Springer-Verlag, 1996

[15] G. Lowe "Casper: A compiler for the analysis of security protocols", 10th IEEE Computer Security Foundations Workshop, p. 31–43, IEEE Computer Society Press, June 1997

[16] G. Lowe "Analysing Protocols Subject to Guessing Attacks", Workshop on Issues in the Theory of Security (WITS), 2002

[17] C.Meadows "A Procedure for Verifying Security Against Type Confusion Attacks." Proceedings of the 16th IEEE Computer Security Foundations Workshop, IEEE Computer Society Press, June 2003

[18] J.Mitchell, V.Shmatikov, U.Stern "Finite-state analysis of SSL 3.0", 7th USENIX Security Symposium, 1998

[19] J. Robinson, "A machine-oriented logic based on the resolution principle", Journal of the ACM, 12(1) p. 23–41, 1965

[20] P. Syverson, P. van Oorschot "On unifying some cryptographic protocol logics", IEE Symposium on Security and Privacy, p. 14–28, IEEE Computer Society Press, 1994

[21] American Statistical Association Privacy, Confidentiality and Data Security Website http://www.amstat.org/comm/cmtepc/index.cfm

[22] Electronic Frontier Foundation, "Cracking DES:Secrets of Encryption Research, Wiretap Politics & Chip Design", O' Reilly, 1998

[23] D. Longley, S. Rigby, "An Automatic Search for Security Flaws in Key Management Schemes", Computers and Security, March 1992, vol 11, pp 75–89

[24] J. Manger, "A Chosen Ciphertext Attack on RSA Optimal Asymmetric Encryption Padding (OAEP) as Standardized in PKCS #1 v2.0", Proceedings of the 21st Annual International Cryptology Conference on Advances in Cryptology, Springer-Verlag, pp 230–238, 2001

[25] D. Bleichenbacher, "Chosen Ciphertext Attacks Against Protocols Based on the RSA Encryption Standard PKCS #1", Proceedings of the 18th Annual International Cryptology Conference on Advances in Cryptology,Springer-Verlag, pp 1–12, 1998

Synthesising Efficient and Effective Security Protocols

Chen Hao [1] John A. Clark [2] Jeremy L. Jacob [3]

Department of Computer Science
University of York
YORK, YO10 5DD, United Kingdom

Abstract

The logical correctness of security protocols is important. So are efficiency and cost. This paper shows that meta-heuristic search techniques can be used to synthesise protocols that are both provably correct and satisfy various non-functional efficiency criteria. Our work uses a subset of the SVO logic, which we view as a specification language and proof system and also as a "protocol programming language". Our system starts from a set of initial security assumptions, carries out meta-heuristic search in the design space, and ends with a protocol (described at the logic level) that satisfies desired goals.

Key words: Security Protocols, Belief Logic, Automated Protocol Synthesis, Meta-heuristic Search, Non-functional Requirements.

1 Introduction

The correctness of a security protocol is a crucial design goal, but other, non-functional, criteria (most typically efficiency criteria) are also of considerable importance. The definition of "efficiency" will vary according to circumstance. Furthermore, designers often wish to find an efficient way of implementing a specification and they may also want to explore the consequences of making different initial assumptions. This paper demonstrates a framework for the automated synthesis of provably secure *and* efficient security protocols. Our framework views design synthesis as a numerical optimisation problem. We use the established global optimisation technique of simulated annealing [10] (see subsection 4.1 for a description) to perform the search. The framework extends previous work of the authors [5,6,7] which addressed only issues of

[1] E-mail: chenhao@cs.york.ac.uk
[2] E-mail: jac@cs.york.ac.uk
[3] E-mail: jeremy.jacob@cs.york.ac.uk

This is a preliminary version. The final version will be published in Electronic Notes in Theoretical Computer Science
URL: www.elsevier.nl/locate/entcs

logical correctness. Our previous work used BAN logic. In this paper, we use SVO logic [13], a more sophisticated logic than BAN logic, and thus giving greater confidence in the practical security of discovered protocols.

2 Protocols and Belief Logics

Since the late 1980's, formal methods have been used in the field of security protocols [4]. Recent approaches to the use of formal methods in the design of security protocols include finite-state model checking and belief logics [12]. In this paper, we will concentrate on protocol security belief logics, which formalise what a principal may infer from messages received.

Belief logics have played a crucial role in the development of protocol analysis as a research topic. The earliest, and best known, is the logic due to Burrows, Abadi and Needham [3], generally referred to as BAN logic. In BAN logic, the assumptions and goals of the protocols are specified as beliefs of principals. A message comprises a set of beliefs held by its sender. The logic provides various inference rules that define how a recipient's belief state may legitimately evolve upon receipt of a message. Informally, a message of a sender's beliefs can be received, but it may be encrypted. If a principal has the key, then the plaintext may be obtained. However, the principal has no assurance where the plaintext comes from, unless the message is signed. If a message is signed, then a principal may assume that the signing principal once said the plaintext. But he does not know when the message was said, unless it is fresh, in which case, the principal may assume the signer effectively, says the message at the same time it is received (that is, the message is timely, and thus not a replay). Still, the message may not come from a principal trusted to have the authority to make such statements. If it does, then whatever the message asserts is considered accurate.

BAN logic is simple and elegant and allows for very short abstract proofs. However, it does not allow analysis at the level of sophistication required by some analysts. Accordingly, various other, related, logics have been developed to fill some of the perceived gaps. This has generally been at the expense of increased complexity (for example, GNY logic [9] has more than 50 inference rules). Perhaps the most significant of subsequent logics has been SVO which aims to efficiently unify previous logics (BAN [3], GNY [9], AT [1] and VO [14]). SVO does not simply add on new notation and rules to capture the additional scope of these logics. Rather, its aim is to produce a model of computation and a logic that is sound with respect to that model, whilst still keeping the expressiveness of the various BAN extensions. We shall use a subset of SVO logic as the basis for our protocol synthesis framework. The parts of SVO we need for the examples in this paper are given in subsection 2.1.

2.1 SVO Notation

The language of SVO logic consists of the following expressions:

- P believes X: P may act as if X is true.
- P has X: X is a message that P can see. This includes messages initially available to P, received by P, freshly generated by P, and constructible by P from the above.
- P received X: P has received a message that contains X, and P can retrieve X from the message; this may require decryption.
- P said X: The principal P at some time sent a message including the statement X.
- P says X: X is a statement P said very recently. That is, P must have said X since the beginning of the current epoch.
- P controls X: P has jurisdiction over X. The principal P is an authority on X and should be trusted on this matter. An example of jurisdiction is that principals may believe that a key distribution server has jurisdiction over statements about the quality of keys.
- fresh(X): X has not been sent in a message at any time before the current run of the protocol. This is usually true for nonces.
- $P \xleftrightarrow{k} Q$: k will never be discovered by any principal but P, Q, or a principal which is trusted by P or Q.
- $\{X\}_k$: This is the notation for encryption of X by key k. It is assumed that encrypted messages are uniquely readable and verifiable by holders of the right keys. Similarly, encrypted messages can only be created by a principal with the appropriate keys.

2.2 SVO Axioms

When a principal receives a message, the logic provides twenty two axioms that indicate what new beliefs this principal may infer from the message contents. The axioms we we need for the examples in this paper are given below.

Belief Axiom

1. $(P \text{ believes } \varphi \wedge P \text{ believes } (\varphi \Rightarrow \psi)) \Rightarrow P \text{ believes } \psi$.

Source Association Axiom

2. $(P \xleftrightarrow{k} Q \wedge R \text{ received } \{X^Q\}_k) \Rightarrow (Q \text{ said } X \wedge Q \text{ has } k)$.

Keys are used to deduce the identity of the sender of a message. The superscript Q in the axiom indicates that the message is from Q (rather than P). The axiom applies when any principal R receives $\{X^Q\}_k$.

Receiving Axioms

3. P received $(X_1, \ldots, X_n) \Rightarrow P$ received X_i, for each $i \in \{1, \ldots, n\}$.

If a principal received a concatenation of messages, then the principal has received each component.

4. $(P$ received $\{X\}_k \wedge P$ has $k) \Rightarrow P$ received X.

If a principal received an encrypted message and has the key, then the principal also received the decrypted message.

Possession Axioms

5. P received $X \Rightarrow P$ has X

A principal possesses anything he received.

6. P has $(X_1, \ldots, X_n) \Rightarrow P$ has X_i, for each $i \in \{1, \ldots, n\}$.

If a principal possesses a concatenation of messages, then the principal possesses each component.

Saying Axioms

7. P said $(X_1, \ldots, X_n) \Rightarrow (P$ said $X_i \wedge P$ has $X_i)$, for each $i \in \{1, \ldots, n\}$.

8. P says $(X_1, \ldots, X_n) \Rightarrow (P$ said $(X_1, \ldots, X_n) \wedge P$ says $X_i)$, for each $i \in \{1, \ldots, n\}$.

Freshness Axiom

9. $\text{fresh}(X_i) \Rightarrow \text{fresh}(X_1, \ldots, X_n)$, for any $i \in \{1, \ldots, n\}$.

A concatenated message is fresh if one of its components is fresh.

Jurisdiction Axiom

10. $(P$ controls $\varphi \wedge P$ says $\varphi) \Rightarrow \varphi$.

Nonce-Verification Axiom

11. $(\text{fresh}(X) \wedge P$ said $X) \Rightarrow P$ says X.

Freshness promotes a message from having been said (sometime) to having been said during the current epoch.

Symmetric Goodness Axiom

12. $P \overset{k}{\longleftrightarrow} Q \equiv Q \overset{k}{\longleftrightarrow} P$.

2.3 GNY Recognisability Rule and Message Extension

SVO logic represents a protocol at the abstract level by asserting comprehension of received messages and interpretation of comprehended messages. These premises almost preclude automated reasoning in that a message of the same format may carry different, context dependant, meaning in different

protocols. We avoid this limit by using the GNY *Recognisability Rule* and *Message Extension* [9].

2.3.1 GNY Recognisability Rule

A principal P would recognise X if P has certain expectations about the contents of X before he actually receives X. P may recognise a particular value (for example, his own identifier or nonce), a particular structure (for example, the format of a timestamp), or a particular form of redundancy. The GNY recognisability rule specifies the formulæ that a principal can believe to be recognisable, when given his beliefs about the recognisability of other formulæ.

$$P \text{ believes } \phi(X_i) \Rightarrow P \text{ believes } \phi(X_1, \ldots, X_n), \text{ for any } i \in \{1, \ldots, n\}.$$

If a principal P believes that a formula X is recognisable, then he is entitled to believe that the whole message is recognisable. In this paper, we use this rule as a replacement of the SVO comprehension assertion.

2.3.2 GNY Message Extension

Typical protocol specifications often include verbal description to the effect that a principal should proceed only if certain conditions hold or only if he holds certain beliefs. This can be regarded as a precondition of a message. The precondition of a formula X, represented by statement C, is described as $X \hookrightarrow (C)$, where C is called a *message extension*. When a receiver receives a message that contains $X \hookrightarrow (C)$, he should interpret it as such. We use the notion of *message extension* to replace the SVO interpretation assertion.

2.4 Illustrative Example

Figure 1 gives a set of initial assumptions that are held by principal A and a key distribution server S, and a feasible protocol.

In this example, a principal A obtains a new secure session key k_{ab} to communicate with another principal B from a key distribution server S. We start from a set of initial assumptions and intend to achieve the above goals. Firstly, A possesses their identifiers (that is, A, B and S). He also possesses a particular random number N_a (also known as a *nonce*), and a long term key k_{as} for communicating with S. Secondly, S possesses principals identifiers (A, B and S) and the long term key k_{as} as well as A. As S plays the role of a key distribution server, he also possesses the new session key k_{ab}. Thirdly, A believes that k_{as} is secure and N_a is a well formed nonce which is both fresh and recognisable. A also trusts S to provide k_{ab}, that is to say A believes that S has jurisdiction over the new session key. And finally, S believes that k_{as} is secure for communicating with A and k_{ab} is a good key that can be used between A and B.

The goals of this protocol are for A to obtain and believe k_{ab} is secure for communicating with B, that is, A has k_{ab} and A believes $A \overset{k_{ab}}{\longleftrightarrow} B$. So the

Initial Assumptions

A has (A, B, S, N_a, k_{as}) $\quad$ S has $(A, B, S, k_{as}, k_{ab})$

A believes $A \overset{k_{as}}{\longleftrightarrow} S$ $\quad$ A believes fresh(N_a) $\quad$ A believes $\phi(N_a)$

S believes $A \overset{k_{as}}{\longleftrightarrow} S$ $\quad$ S believes $A \overset{k_{ab}}{\longleftrightarrow} B$

A believes (S controls $A \overset{k_{ab}}{\longleftrightarrow} B$)

Goals

A has k_{ab} $\quad$ A believes $A \overset{k_{ab}}{\longleftrightarrow} B$

A Feasible Protocol

1. $A \rightarrow S : A, B, N_a$
2. $S \rightarrow A : \{N_a, k_{ab} \hookrightarrow (A \overset{k_{ab}}{\longleftrightarrow} B)\}_{k_{as}}$

Fig. 1. Initial assumptions, goals and a feasible protocol

protocol is a fragment of some key distribution protocol.

A believes N_a is a well formed nonce and may include it in the first message together with identifiers A and B. When the server S receives this message, he can possess the nonce N_a later. However, S could not infer anything from this message since the message is in plaintext. Now, S may reply to A with the second message that contains two components: the newly received N_a and $k_{ab} \hookrightarrow (A \overset{k_{ab}}{\longleftrightarrow} B)$. S encrypts this message using key k_{as}. Once A receives this message, he may decrypt it to reveal its contents. Then, by the *GNY Recognisability Rule*, the whole message is recognisable by A because N_a is recognisable. That is, A believes that he received $(N_a, k_{ab} \hookrightarrow (A \overset{k_{ab}}{\longleftrightarrow} B))$. So, A has k_{ab} is achieved by applying the SVO possession axioms. Then, by *GNY Message Extension*, A believes that he received $(N_a, A \overset{k_{ab}}{\longleftrightarrow} B)$. Next, using the *Source Association Axiom*, A concludes that S said $(N_a, A \overset{k_{ab}}{\longleftrightarrow} B)$. This message contains an assertion involving N_a, a nonce A believes to be fresh, so A may conclude the whole message is a fresh one. Then A may deduce that S says the whole message using the *Nonce-Verification Axiom*. In detail, A believes S says $(N_a, A \overset{k_{ab}}{\longleftrightarrow} B)$. Since A believes that S has jurisdiction over the new session key, A may now believe $A \overset{k_{ab}}{\longleftrightarrow} B$ using the *Jurisdiction Axiom*. Note that this protocol is at an abstract logic level and it is secure with respect to SVO logic at that level.

A possible concrete version of this protocol is:

1. $A \rightarrow S : A, B, N_a$
2. $S \rightarrow A : \{N_a, B, k_{ab}\}_{k_{as}}$

3 Efficiency of Security Protocols

Current research in security protocols has largely focused on the *security* of protocols, and there is very little published discussion on the issue of protocol *efficiency* (notable exceptions are Boyd & Mathuria [2] and Gong [8]). The treatment of efficiency or performance is generally given a low priority and is often rather ad hoc. One possible reason is that security protocols normally involve only a few messages, thus optimisation is not seen as a very urgent requirement. However, as Gong [8] points out, it is natural and beneficial to investigate whether a protocol that achieves security requirements in a particular environment is also in some sense minimal or optimal. For example, reducing one message from a five-message protocol represents a 20% reduction in the number of messages and possibly a similar amount of reduction of the overall running time of the protocol.

Boyd and Mathuria [2] define two sorts of efficiency: computational efficiency and communications efficiency. Computational efficiency is concerned with the computations that the principals need to engage in to complete the protocol. This will largely depend on the algorithms used to provide the cryptographic services. Communications efficiency is concerned with the number and length of messages that need to be sent and received during the running of a protocol. In some sense, computational efficiency is an issue that can be discussed at the concrete level whilst communications efficiency can be expressed at the abstract level. In our system, we define a collection of fitness functions for expressing efficiency issues (number of messages, encryption method, server interaction etc.) and ways of determining how these concerns have been achieved. Details of the fitness function are given in subsection 4.4.

4 Search Strategy

Given a specification (assumptions as precondition, goals as postcondition) we wish to search the space of protocols for those satisfying that specification. Any series of honest exchanges between two or more principals defines a feasible (with respect to the logic) protocol. We consider this set of protocols as the design space. It is clear that this space grows exponentially as the number of messages or the number of principals rise. The choices of the content of messages introduces further combinatorial complexity. For a synthesis technique to be scalable, it cannot be based on simple enumeration. Our previous work has shown that the protocol synthesis problem can be couched as a numerical optimisation problem [5,6,7]. This work was concerned only with the synthesis of logically correct protocols. In this paper we show how the approach can be extended to encompass non-functional criteria too. In the experiments reported in this paper we have used the well-established technique of *simulated annealing* [10] (although our implementation allows rapid interchange of optimisation techniques). Below is a brief introduction to the

$S := S_0$
$T := T_0$
repeat until stopping criterion is met
    repeat *MIL* **times**
        Pick $Y \in N(S)$ with uniform probability
        Pick $U \in (0, 1)$ with uniform probability
        if $f(Y) > f(S) + T \ln U$ **then** $S := Y$
    $T := \alpha \times T$

Fig. 2. Basic Simulated Annealing for Maximisation Problems

simulated annealing algorithm.

4.1 Optimisation Technique

In 1983 Kirkpatrick et al. [10] proposed *simulated annealing*, a search technique inspired by the cooling processes of molten metals. It merges hill-climbing with the probabilistic acceptance of non-improving moves to find a good state $S \in State$. The basic algorithm is shown in Figure 2.

The search starts at some initial state $S_0 \in State$. There is a control parameter $T \in \mathbb{R}^+$ known as the *temperature*. This starts high at T_0 and is gradually lowered, typically by *geometric cooling* (that is, by multiplying by a *cooling factor*, $\alpha \in (0, 1)$ at each iteration).

At each temperature, a number *MIL* (Moves in Inner Loop) of moves to new states are attempted. A candidate state Y is randomly selected from the neighbourhood $N(S)$ of the current state. The new state Y is accepted if it is better or only slightly worse than S, as measured by a fitness function $f \in State \to \mathbb{R}$. By "slightly worse" is meant "no more than $|T \ln U|$ lower". Here $U \in (0, 1)$ is a random variable, and so $T \ln U \in (-\infty, 0)$; the smaller T is, the more likely that this term is close to 0 and eventually only improving moves are accepted (that is, the technique reduces to hill climbing).

The algorithm terminates when some stopping criterion is met. Common stopping criteria, and the ones used for the work in this paper, are to stop the search after a fixed number *MaxIL* of inner loops have been executed, or else when some maximum number *MUL* of consecutive unproductive inner loops have been executed (that is, without a single move having been accepted).

Generally the best state achieved so far is also recorded (since the search may actually move out of it and subsequently be unable to find a state of similar quality). Another common improvement to efficiency is to generate U only when $f(Y) < f(S)$.

In the experiments, we used a geometric cooling rate of 0.97; the number of attempted moves at each temperature was 400, with a maximum of 300 iterations (temperature reductions) and maximum number of 50 consecutive unproductive iterations (that is, with no move being accepted). Further details

of the annealing algorithm we used in our system can be found in the earlier papers [5,6,7].

4.2 Protocol Representation and a Move Function

In our protocol synthesis system, a protocol is represented as a sequence of M messages, each of which is represented by an integer sequence. N principals, indexed $0 \ldots N-1$, participate in the protocol. Associated with each of the principals are vectors of its current beliefs and formulæ. Each of the M messages is represented by $F+3$ integers, $s, r, k, f_1, \ldots, f_F$. These represent the sender, the receiver, the key that the sender used to encrypt this message, and a series of F indices that reference formulæ currently possessed by the sending principal. So, the sender is $s \bmod N$; the receiver is $r \bmod N$; the key is $k \bmod NK$ (where NK is the number of keys possessed by the sender); and the first component in the message is $f_1 \bmod T$ etc., where the sender possesses T formulæ, indexed $0 \ldots T-1$. Formula 0 is the null formula (which allows us to model messages with fewer than F "real" formulæ). Key 0 is the null key (message in plaintext is treated as "encrypted" by the null key). The vectors of the receiver's current beliefs and formulæ are updated after each message is sent (see below). In this way, an arbitrary sequence of integers can be interpreted as a feasible protocol (senders only ever send formulæ they actually hold).

The above strategy allows a very simple move function for local search — simply randomly replace some of the integers involved in some messages.[4] Although any integer sequence gives rise to a feasible protocol, the protocol may not satisfy our required goals. The fitness function below measures how close it comes to achieving the required goals and our search seeks to find a protocol that satisfies all these goals.

4.3 Executing a Protocol

Assume that a protocol consists of M messages, each of which consists of F formulæ, and we start from the very beginning of this protocol. Firstly, we initialise the belief and formula state of the relevant principals involved in this protocol. Then, for each message in this protocol, we follow the steps below.

(i) Determine the sender, receiver, and the key under which the current message is encrypted. If this key is an appropriate one for communication between the sender and the receiver, then proceed with the rest of the current message, else ignore this message and proceed to the next message. The method of decoding the sender, receiver, and key is given in subsection 4.2.

[4] Note: A perturbed sequence may actually give rise to the same protocol as the original sequence. For example, if an integer denoting a sender has its value changed by a multiple of N it will be interpreted as the same principal as before.

(ii) Decode each of the F formulæ corresponding to the current message. For instance, the first formula in the message is $f_1 \bmod T$, where the sender currently holds T sendable formulæ.

(iii) Update the receiver's beliefs vector by applying the SVO axioms. Here we demonstrate what a principal A will do after it receives a message $\{X, B, N_a\}_{k_{ab}}$ from another principal B (assume both A and B initially hold k_{ab}, and they both believe that k_{ab} is a good key for communication). Firstly, after decryption, A recognises that this message is comprehensible, thus B said X, B said B, B said N_a, B has X, B has B and B has N_a are added to A's belief vector (this represents A believes B said X, A believes B said B, A believes B said N_a, A believes B has X, A believes B has B and A believes B has N_a) whilst X, B and N_a are added to A's formula vector.[5] After this, A examines the set of received formulæ to see whether any of them are believed to be fresh. In this case, A retrieves the formula N_a, and as A believes the nonce N_a is fresh, then the whole message is regarded as fresh.[6] As the message is fresh, B says X, B says B and B says N_a are added to A's belief vector.[7] Similarly, other axioms now may be applied to deduce further beliefs until no further beliefs can be created.

(iv) Record the number of required goals achieved after this message has been analysed.

Once a protocol has been executed in the above way, the fitness of this protocol can be calculated as given in subsection 4.4.

4.4 The Fitness Function

The fitness function is used to guide the search for a 'good' solution, that is, the fitness function must tell us how good a candidate solution is. We use fitness functions, $f(P)$, which are sums of a *security fitness function*, s and an *efficiency fitness function*, e.

$$f(P) = s(P) + e(P)$$

The function $s(P)$ is defined as follows:

$$s(P) = \sum_{i=1}^{N} (\sigma + \delta(i)) \times G(P, i)$$

Here N is the maximum number of messages we allow in any protocol; $G(P, i)$ is the number of new required security goals that message i of P achieves; σ is, typically, a large constant that weights security much more heavily than efficiency and the $\delta(i)$ are weights among the individual messages. These weights,

[5] By applying the SVO *Receiving Axioms*, *Source Association Axiom* and *Possession Axiom*.
[6] By applying the SVO *Freshness Axiom*.
[7] By applying the SVO *Nonce-verification Axiom*.

Strategy	Weight						
	$\delta(1)$	$\delta(2)$	$\delta(3)$	$\delta(4)$	$\delta(5)$	$\delta(6)$	$\delta(7)$
EC	640	320	160	80	40	20	10
UC	160	160	160	160	160	160	160

Table 1
Weighting Strategies for $N = 7$

$\delta(i)$, were chosen to represent one of two strategies for finding protocols, *early credit (EC)* for achieving goals early in the protocol and *uniform credit (UC)* which does not favour one message over another; see Table 1. Further details of these strategies are given by Clark and Jacob [6,7].

The function $e(P)$ is a also a sum of fitness functions, one for each kind of fitness we consider.

$$e(P) = m(P) + c(P) + r(P)$$

The fitness function $m(P)$ punishes protocols with many messages.

$$m(P) = \mu \times M(P)$$

where $M(P)$ is the highest index of a message of P that contributes new goals and $\mu < 0$ is the weight we give to this.

The function $c(P)$ punishes protocols with more encryption.

$$c(P) = \kappa \times C(P)$$

where $C(P)$ is the number of encryptions in P and $\kappa < 0$ is the weight we give to this.

The function $r(P)$ punishes numbers of interactions with particular principals.

$$r(P) = \sum_{a \in A(P)} \rho(a) \times R(P, a)$$

where $R(P, a)$ is the number of rounds involving principal a in P, $A(P)$ is the set of principals in P and $\rho(a) < 0$ are weights which allow us to declare that interactions with some principals (for example, the server) are worse than others.

5 Experimental Method and Results

This section reports the results of applying the technique described above to the derivation of three-party symmetric key distribution protocols.

5.1 Assumptions

Three parties participate in this key distribution protocol: A, B and S. Their identifiers are held by each other initially.

A holds a long term secure key k_{as} that he believes to be secure for communicating with the key distribution server S. A maintains his own nonce N_a that he believes to be fresh. A is able to recognise his own identifier A and nonce N_a.

Similar to principal A, principal B holds a long term secure key k_{bs} that he believes to be secure for communicating with S. B maintains his own nonce N_b that he believes to be fresh. B is able to recognise his own identifier B and nonce N_b.

The server S holds k_{as}, k_{bs} and believes that they are secure for communicating with A and B respectively. S also holds k_{ab} and believes that it is a good key that can be used between A and B. The assumptions are:

A has (A, B, S)	A has k_{as}	A believes $A \overset{k_{as}}{\longleftrightarrow} S$
A has N_a	A believes $\text{fresh}(N_a)$	
A believes $\phi(A)$	A believes $\phi(N_a)$	A believes S controls $A \overset{k_{ab}}{\longleftrightarrow} B$
B has (A, B, S)	B has k_{bs}	B believes $B \overset{k_{bs}}{\longleftrightarrow} S$
B has N_b	B believes $\text{fresh}(N_b)$	
B believes $\phi(B)$	B believes $\phi(N_b)$	B believes S controls $A \overset{k_{ab}}{\longleftrightarrow} B$
S has (A, B, S)		
S has k_{as}	S believes $B \overset{k_{as}}{\longleftrightarrow} S$	
S has k_{bs}	S believes $B \overset{k_{bs}}{\longleftrightarrow} S$	
S has k_{ab}	S believes $B \overset{k_{ab}}{\longleftrightarrow} S$	

5.2 Goals

The first set of goals requires that at the end of the protocol run A and B must each believe that it possesses a good key k_{ab} for session communication, and that each believes the other possesses the same key.

A has k_{ab}	A believes $A \overset{k_{ab}}{\longleftrightarrow} B$
B has k_{ab}	B believes $A \overset{k_{ab}}{\longleftrightarrow} B$
A believes B has k_{ab}	B believes A has k_{ab}

5.3 Results

5.3.1 The first search

In our first search the annealing parameters given in subsection 4.1 were used. We took the maximum number of messages, $N = 7$. The weights we used for the fitness function are given in Table 2. We used the EC strategy in order to find a protocol that is able to satisfy security requirements as soon as possible.

σ	3000	Correctness is more important than efficiency
$\delta(i)$	EC	See Table 1 for the definition
μ	-200	
κ	-100	
$\rho(S)$	-100	S is the server
$\rho(A)$	-50	A is a client
$\rho(B)$	-50	B is a client

Table 2
Fitness function weightings for the first search

1. $A \rightarrow S : A, B, N_a$
2. $S \rightarrow A : \{N_a, k_{ab} \hookrightarrow (A \overset{k_{ab}}{\longleftrightarrow} B)\}_{k_{as}}$
3. $B \rightarrow S : B, A, N_b$
4. $S \rightarrow B : \{N_b, N_a, k_{ab} \hookrightarrow (A \overset{k_{ab}}{\longleftrightarrow} B)\}_{k_{bs}}$
5. $B \rightarrow A : \{B, N_b, N_a\}_{k_{ab}}$
6. $A \rightarrow B : \{N_b, A\}_{k_{ab}}$

Fig. 3. A symmetric key protocol found by our tool in the first search

Figure 3 shows one of the symmetric key protocols generated by the program. Only the core security relevant components of the protocol are presented. That is, we have removed components from the description that do not contribute to the goals. In addition, redundant components (where the same components are included twice or more in one message) have also been removed. (Currently, redundant components are removed by hand; automating their removal is under investigation.)

5.3.2 The second search

For our second search we asked our tool to try harder to minimise server interactions. To do this we doubled each of $\rho(S)$, $\rho(A)$ and $\rho(B)$ (that is, we let fitness function $r(P)$ carry more weight relative to the other properties) and used the UC strategy for the $\delta(i)$ (see Table 1). All other parameters are as in Table 2.

A new protocol (see Figure 4) with fewer server interactions than any protocol generated in the first search was found by our tool.

The first search might have produced the protocol in Figure 4, but without the demands given in the second search, it did not.

1. $A \rightarrow B : A, N_a$
2. $B \rightarrow S : A, N_a, B, N_b$
3. $S \rightarrow B : \{N_b, k_{ab} \hookrightarrow (A \overset{k_{ab}}{\longleftrightarrow} B)\}_{k_{bs}}$
4. $S \rightarrow A : \{N_a, N_b, k_{ab} \hookrightarrow (A \overset{k_{ab}}{\longleftrightarrow} B)\}_{k_{as}}$
5. $A \rightarrow B : \{A, N_a, N_b\}_{k_{ab}}$
6. $B \rightarrow A : \{B, N_a\}_{k_{ab}}$

Fig. 4. A protocol with fewer server interactions, found by our tool when asked to try harder to minimise server interactions

5.4 Comments

It is possible to ask our tool to find a protocol for an assumptions/goals pair that cannot be met, or is difficult to meet. In such cases, the search will terminate without producing a protocol or produce a protocol that only satisfies some of the goals.

One pleasing feature of the work is the speed at which protocols are generated. A typical run takes two minutes or less (on a Pentium Mobile 1.5G processor with 512MB memory, *Java*™ 2 SDK 1.4.2 and Borland JBuilder 9). This compares very favourably with other design synthesis approaches, for example model checking [11].

6 Conclusions and Further Work

The above work shows that meta-heuristic search approaches to secure protocol synthesis are potentially powerful and have the benefit of being rapid. Our tools generate candidate protocols rapidly and concrete refinements of them could be subjected to more detailed and sophisticated analysis (such as that provided by current model checking approaches). This would provide an interesting synthesis of current techniques. The protocols generated, although simple, are typical abstractions of protocols in the literature.

There is very little published work in the field of protocol efficiency. With our protocol synthesis approach, we have provided a framework for incorporating efficiency concerns into the synthesis approach (in terms of the number of messages, server interactions etc.).

However, we note that the efficiency of a protocol cannot be fully characterised independently of its implementation details. Thus, one topic of further work is to investigate more efficiency issues of security protocols and how to make tradeoffs among these concerns.

In this paper, we demonstrate that our system has the ability to synthesise three-party authentication and key transport protocols using symmetric key cryptography. Key agreement protocols have become much more popular than

key transport protocols in recent years. SVO logic has the ability to analysis key agreement protocols. We believe that our synthesis system can easily be extended to encompass public key scheme and synthesise key agreement protocols.

References

[1] Martin Abadi and Mark Tuttle. A semantics for a logic of authentication. In *Proceedings of the 10th ACM Symposium on Principals of Distributed Computing*, pages 201–216. ACM Press, 1991.

[2] Colin Boyd and Anish Mathuria. *Protocols for Authentication and Key Establishment*. Springer-Verlag, 2003.

[3] Michael Burrows, Martín Abadi, and Roger Needham. A logic of authentication. Technical Report 39, Digital Systems Research Center, 1989.

[4] Levente Buttyán. Formal methods in the design of cryptographic protocols (state of the art). Technical Report SSC/1999/038, EPFL SSC, 1999.

[5] Hao Chen, John A. Clark, and Jeremy L. Jacob. Automated design of security protocols. In *Proceedings of the 2003 Congress on Evolutionary Computation*, pages 2181–2188. IEEE Press, 2003.

[6] John A. Clark and Jeremy L. Jacob. Search for a solution: Engineering tradeoffs and the evolution of provably secure protocols. In *Proceedings of 2000 IEEE Symposium on Security and Privacy*, pages 82–95. IEEE Computer Society Press, 2000.

[7] John A. Clark and Jeremy L. Jacob. Protocols are programs too: The meta-heuristic search for security protocols. *Information and Software Technology*, 43(14):891–904, 2001.

[8] Li Gong. Efficient network authentication protocols: Lower bounds and optimal implementations. *Distributed Computing*, 9(3):131–145, 1995.

[9] Li Gong, Roger Needham, and Raphael Yahalom. Reasoning about belief in cryptographic protocols. In *Proceedings of 1990 IEEE Symposium on Security and Privacy*, pages 234–248. IEEE Computer Society Press, 1990.

[10] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.

[11] Adrian Perrig and Dawn Song. Looking for diamonds in the desert – extending automatic protocol generation to three-party authentication and key agreement protocols. In *Proceedings of the 13th IEEE Computer Security Foundations Workshop*, pages 64–76. IEEE Computer Society Press, 2000.

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

[13] Paul Syverson and Paul van Oorschot. A unified cryptographic protocol logic. Technical Report NRL Publication 5540–227, Naval Research Lab, 1996.

[14] Paul van Oorschot. Extending cryptographic logics of belief to key agreement protocols. In *Proceedings of the 1st ACM Conference on Computer and Communications Security*, pages 232–243. ACM Press, 1993.

On the Relative Soundness of the Free Algebra Model for Public Key Encryption

Christopher Lynch [1,2]

Department of Mathematics and Computer Science
Clarkson Universtity
Potsdam, NY 13699-5815

Catherine Meadows [3]

Naval Research Laboratory
Center for High Assurance Computer Systems
Code 5543
Washington, DC 20375

Abstract

Formal systems for cryptographic protocol analysis typically model cryptosystems in terms of free algebras. Modeling the behavior of a cryptosystem in terms of rewrite rules is more expressive, however, and there are some attacks that can only be discovered when rewrite rules are used. But free algebras are more efficient, and appear to be sound for "most" protocols. In [9] Millen formalizes this intuition for shared key cryptography and provides conditions under which it holds; that is, conditions under which security for a free algebra version of the protocol implies security of the version using rewrite rules. Moreover, these conditions fit well with accepted best practice for protocol design. However, he left public key cryptography as an open problem. In this paper, we show how Millen's approach can be extended to public key cryptography, giving conditions under which security for the free algebra model implies security for the rewrite rule model. As in the case for shared key cryptography, our conditions correspond to standard best practice for protocol design.

Key words: Cryptographic Protocol Verification.

[1] This work was done while the author was visiting the Naval Research Laboratory

[2] Email: clynch@clarkson.edu

[3] Email: meadows@itd.nrl.navy.mil

This is a preliminary version. The final version will be published in Electronic Notes in Theoretical Computer Science
URL: www.elsevier.nl/locate/entcs

1 Introduction

In most formal systems for cryptographic protocol analysis, it is typical to represent encryption operations in terms of a free algebra. Encryption, for example, may be represented as e(K,X), where K is the key and X is the plaintext. Decryption, however, is represented implicitly. For example, one may include a rule that says that if a principal knows the shared/symmetric key K and e(K,X), then it can also learn X.

Another approach is to represent both encryption and decryption explicitly. Encryption is represented, say, by e(K,X), while decryption is represented by d(K,X). The fact that decryption undoes encryption can be represented by a cancellation rule d(K,e(K,X)) = X. Since the decryption operator can also be used on unencrypted data, and the operations also cancel out when applied in reverse, we can also include the rule e(K,d(K,X)) = X.

This explicitness leads to a greater expressiveness, both in protocol representation and in constructing attacks. For example, in [5] Meadows analyzes a protocol that could not have been specified in the free algebra model, since it requires the application of the decryption operator to unencrypted data. Even when protocols do not require explicit representation of the decryption operator in this way, it is still possible that they may be subject to attacks that require its use. For example, in [9] Millen gives an example of a protocol that is secure in the free algebra model but insecure in the decrypt-extended model.

However, explicit representation of cancellation, although more expressive, can lead to a less efficient analysis. Moreover, many practical protocols appear to be immune to cancellation based attacks. Techniques such as recognizable formatting, probabilistic encryption, and so forth, appear to make unanticipated application of cancellation rules unlikely. Thus, most formal systems for cryptographic protocol analysis make use of the free algebra model, although they provide little formal justification for that approach.

In [9] Millen addresses this problem by giving necessary and sufficient conditions on protocols using symmetric encryption, under which if the secrecy property holds for a protocol in the free algebra model, then it also holds in the decrypt-extended model. The condition is specified on protocols written in a model based on parametric strand spaces and pattern matching. It has two subconditions. The first, called *purity* says that the decryption operator should not appear in the protocol specification. The second, called *EV-freeness*, says that there must be no application of an encryption operator to a variable in a specification. The EV-free property essentially says that a principal should not apply an encryption operator to a term unless it has been able to verify that that term has some kind of structure, so that it is not unknowingly applying the encryption operator to the result of applying a decryption operator. Since it is generally considered good practice to apply encryption only to data about which one knows something (e.g. it is data that satisfies certain formatting

conventions, or data that one has created oneself), it is reasonable to expect that most cryptographic protocols will satisfy EV-freeness.

Although Millen conjectured that similar results might hold for public key cryptography, he left it as an open question. As in shared key cryptography, it is possible to construct protocols vulnerable to attacks that rely on cancellation rules. Consider for example, a cryptosystem such as RSA in which signing and decryption are the same operation. In that case we have the identity $pke(privkey(U), pke(pubkey(U), X)) = X$, where pke is public encryption, $pubkey(U)$ is U's public encryption key, and $privkey(U)$ is both U's private signature key and its private decryption key. Consider now a simple notary public protocol in which a server is willing to sign any message it gets. Suppose that someone encrypts a message m with the server's public key and sends it. An intruder can send $pke(pubkey(s), m)$ to the server, which will output $pke(privkey(s), pke(pubkey(s), m)) = m$.

Such an attack relies on the cancellation properties of public key encryption, and can't be represented without it. But it is also the case that most practical protocols using public key cryptography are immune to this type of attack. Typically, public key pairs for signing are different from the pairs used for encryption, and digital signatures are applied to a one-way hash of a message, not the message itself. Finally, it is considered sound procedure never to sign a message that is entirely supplied by another party. Instead, a principal is usually required to add some material of its own before signing.

Such observations led us to believe that it should be possible to extend Millen's results to public key cryptography, basing our restrictions on the commonly accepted design principles described above. Our first, public key purity, which is analogous to Millen's purity, is that different key pairs should be used for encryption/decryption and signing/verification, even when the same algorithm is used for both signing and encryption, and that the private decryption key and the public signature verification key never appear in the protocol specification. The second of these, analogous to EV-freedom, called PEV-freedom, is that no occurrence of an encryption or signing operator contains a variable as an argument. We also introduce a third condition, called structure, which is sufficient without the other two, which says that no occurrence of an encryption or signing operator contains as an argument a variable or a term rooted with a public key operator. Note that the requirement that different key pairs be used for different types of operations assumes that principals know which key pairs are intended for what, and so rules out protocols that include the distribution of public keys. This is not the case when we require structure only, however.

The rest of the paper is organized as follows. In Section 2 we introduce the model. In Section 3 we prove the soundness results for public key pure, PEV-free protocols. In Section 4 we describe our plans for further work and make some conjectures. Due to page limits, we were unable to include much of the technical material for this paper. All of the proofs were left out, and all of the

material on *structured protocols* was left out. The full paper with all of that material can be found at `www.clarkson.edu/~clynch/papers/pubfree.ps/`

2 Protocol Model

2.1 Millen and Shmatikov's Constraint Model and its Application to the Free Algebra Model in Single Key Cryptography

Like Millen, we use a model based on parametric strands as used by Cervesato et al. in [1] and Song et al. in [12], and constraint solving as developed by Millen and Shmatikov in [10]. We refer the reader to [3] for a definition of strand spaces. In a parametric strand, message terms may contain variables. A variable will correspond to a subterm of a message term appearing in a negative node for which the receiver can verify no properties; thus any terms can be used to substitute for it. Variables can also appear in positive nodes of a strand, but only if they appeared earlier in a negative node.

Definition 2.1 A *constraint* is a pair $m : T$ where m is a term and T is a term set. A set of constraints $\mathcal{C}$ is *satisfied* by a substitution σ if, for all $m : T \in \mathcal{C}$, the intruder can derive $m\sigma$ from $T\sigma$.

A set of constraints and term sets can be constructed from a sequence of positive and negative nodes in a semibundle $\mathcal{B}$ that is compatible with the partial order imposed by the bundle ordering. Each positive node expands the last term set by the message it contains, while each negative node creates a constraint $m : T$ where m is the message in the node and T is the last term set, and the first term set is the set of terms known initially by the intruder. Such a constraint set is satisfied by a substitution σ, if and only if, for each initial sequence s of the constraint immediately preceding a negative node m, the intruder can generate $m\sigma$ from the positive nodes in $s\sigma$. In other words, the constraint set is satisfied by σ if and only if the intruder can generate a message expected by an honest principal from all the previous messages sent by honest principals and all the information known initially by the intruder. This, of course, is exactly how we use strand spaces to construct an attack on a protocol.

This makes it possible to define a bundle in a sense compatible with constraints.

Definition 2.2 A *semibundle* is a set of parametric strands. A total order $<$ on nodes in a semibundle is *compatible* with the semibundle if whenever N_1 precedes N_2 on a strand, then $N_1 < N_2$.

Definition 2.3 A *bundle* is a semibundle B such that there exists a total order $<$ on nodes compatible with B such that the set of constraints produced by the sequence of nodes induced by $<$ is satisfied. A semibundle is *solvable* if there is some bundle in which some ground instance of the semibundle is embedded.

In [9] Millen describes two constraint systems. One is based on a free algebra, with intruder operators consisting of concatenation, deconcatenation, encryption, and reversal of encryption. The other is based on an algebra (called the *decrypt-extended* algebra) that includes cancellation rules describing the effects of encryption and decryption on each other, and penetrator operators consisting of concatenation, deconcatenation, encryption, reversal of encryption, and the application of the decryption operator. The main theorem of that paper is that, given certain conditions, any bundle that is solvable in the second system is solvable in the first system. This boils down to showing under what conditions a constraint $m : T$ from the free algebra that is solvable in the cancellation algebra is also solvable in the free algebra. In [9] these conditions are that the protocol from which the constraints are derived should not contain any explicit use of the decryption operator (purity), and that it should contain no case of an encryption operator applied to a variable (EV-freeness). Note that solvability in the free algebra always gives us solvability in the decrypt-extended algebra, so that the result is an if-and-only-if. Moreover, the solution for the decrypt-extended algebra induced by the one for the free algebra introduces no strands belonging to honest principals. This has a result on the type of theorems we can prove, as we will see below.

Millen describes in [9] how this result can be used for proving that the free algebra is sound with respect to the decrypt-extended algebra for secrecy properties. One can create a special strand that describes the intruder learning a specified secret, such as a key. One can use his theorem to show that purity and EV-freeness implies that the intruder strand is solvable in the cancellation algebra if and only if it is solvable in the free algebra. Millen also mentions that the result could be used to prove soundness with respect to some authentication properties, although he does not go into detail about this.

We would like to characterize the types of attacks for which Millen's result and our results apply. To this end, we define a specification of an attack below.

Definition 2.4 We define an *attack specification* to be two sets of parametric strands, called the positive and the negative set. A *ground attack specification* is an attack specification in which all terms are ground.

Briefly, the positive set of an attack specification gives the strands that should be in the bundle, and the negative set gives the strands that should not be in the bundle. Thus, a specification of an attack on the secrecy of a key accepted by an initiator could be given in terms of a positive set containing two strands: an initiator strand in which the key is represented by the variable K, and a special intruder strand of the sort described by Millen in which the datum learned by the intruder is also represented by K. The negative set would be empty. However, a specification of an authentication protocol would include the authenticated strand in the positive set and the authenticating strand in the negative set.

The notion of attack specification is not explicitly set out in [10], but our definition captures the essence of the kinds of attacks Millen and Shmatikov handle in their constraint system. We note, however, that Millen and Shmatikov restrict themselves to ground attack specifications, or at least attack specifications in which all terms in the positive set are ground. We believe that it should be straightforward to extend their notion of an attack to non-ground attack specifications. This would allow us to characterize attacks in a somewhat more general way. However, when rewrite rules are used to describe the properties of a cryptosystem, the problem becomes a little more complex, since we need to keep track of which terms in an attack specification are assumed to be irreducible, and which terms may be allowed to be reducible. Though this would not necessarily be difficult to do (something like this is already done for the specification of insecure states in the NRL Protocol Analyzer) it could be tedious and somewhat beside the main goal of this paper. Thus we treat it as outside the scope of the paper and leave it for further work.

Millen's result, and our result as well, extends easily to any grounded attack specification with an empty negative set. It may not always extend to the case of a non-empty negative set, however, since the transformation that maps the cancellation algebra to the free algebra is not necessarily the identity, and it is possible that a strand that appears in an attack in the free algebra could be sent into the negative set by the transformation.

2.2 Free and Extended Algebras for Public Key Cryptography

As in Millen's case, our result is stated in terms of two term algebras. These are defined as follows:

Definition 2.5 The free algebra *FA* contains

(i) constants *pub*, *priv*, *enc* and *sig*. It also contains a set of names, a set of messages, and a set of nonces.

(ii) a pairing operator $[X, Y]$ and a public key encryption operator $pe(X, Y)$. The first argument to *pe* is the key, which we represent as $pk(N, P, S)$, where
- N is the name of the key (usually the principal who uses this key),
- P can be either *pub* or *priv*, indicating whether this is a public key or a private key, and
- S can be either *enc* or *sig*, indicating whether the key is used for encryption or for signing.

The second argument to *pe* is the message.

For example, the term $pe(pk(A, pub, enc), pe(pk(B, priv, sig), m))$ represents message m, signed with B's private key and then encrypted with A's public key. We use this notation to enforce the assumption that no key will be used for both signing and encryption in a protocol.

We only consider protocols where the second and third arguments of a *pk*

operator are not variables. This reflects the assumption that keys have a fixed purpose, it is not possible to change that purpose, and all principals will recognize what the purpose of a key is. This would hold, for example, in cases in which public and private keys are distributed beforehand by some trusted protocol. Since this is an assumption under which much cryptographic protocol analysis is done, we consider it reasonable, at least as a first approximation.

Definition 2.6 The extended algebra EFA, besides the above, includes the following set of equations E:

E1 $pe(pk(K, pub, enc), pe(pk(K, priv, enc), X)) = X$

E2 $pe(pk(K, priv, enc), pe(pk(K, pub, enc), X)) = X$

E3 $pe(pk(K, pub, sig), pe(pk(K, priv, sig), X)) = X$

E4 $pe(pk(K, priv, sig), pe(pk(K, pub, sig), X)) = X$

This set E models the properties of the encryption and signature operations. Let R be the rewrite system which results from orienting these equations from left to right as rewrite rules. This can be shown to be a confluent and terminating rewriting system representing E, using techniques similar to those in [5]. This means that two terms are equivalent modulo E if and only if they have the same unique normal form modulo R. We define $t \downarrow_R$ to be the normal form of t modulo R.

Let **D** be the following derivation rules:

D1 $(X, Y) \vdash X$

D2 $(X, Y) \vdash Y$

D3 $X, Y \vdash (X, Y)$

D4 $pe(pk(K, pub, enc), X), pk(K, priv, enc) \vdash X$

D5 $pe(pk(K, priv, sig), X), pk(K, pub, sig) \vdash X$

D6 $X, pk(K, pub, enc) \vdash pe(pk(K, pub, enc), X)$

D7 $X, pk(K, priv, sig) \vdash pe(pk(K, priv, sig), X)$

The list **D** expresses the ways in which the intruder can derive information in the free algebra FA.

We next construct a set of derivation rules **DE** which expresses what the intruder can learn in the equational extension EFA of the free algebra. We let **DE** be the set containing the derivation rules of **D**, not including rules **D4** and **D5**, plus the additional rules

DE1 $X, pk(K, priv, enc) \vdash pe(pk(K, priv, enc), X)$

DE2 $X, pk(K, pub, sig) \vdash pe(pk(K, pub, sig), X)$

If T is a set of terms, then we define $Deriv_{\mathbf{D}}(T)$ to be the set of terms that can be derived from T using the derivation rules of **D**. We define $Deriv_{\mathbf{DE}}(T)$ as the set of terms that can be derived from T using the derivation rules **DE**.

The derivations of $Deriv_{\mathbf{DE}}$ take place modulo the equational theory E.

In other words, given a derivation rule, $t_1 \cdots t_n \vdash t$ and a set of terms $s_1 \cdots s_n$, then we can derive a new term s if $s = t\sigma$ and $s_i =_E t_i\sigma$ for all i with $1 \leq i \leq n$. We assume that the substitution σ used in a derivation is irreducible by R. (If it were not, it could always be replaced by its reduced form.) After each derivation rule, we reduce the new term s by R.

Note that these derivation rules are analogous to the derivation rules given by Millen[9] for his result for secret keys. We only differ from them in that in **DE**, we do not explicitly add a rule to say that if a privately encrypted message is known, and if the corresponding key is known, then the message is known. We do not add this because it is a consequence of a derivation rule and a reduction rule. Similarly for messages publicly signed.

Any term that can be derived by **D** can also be derived by **DE**. The only thing that is not obvious is for rules 4 and 5 of **D** that do not exist in **DE**. But these rules are simulated by a derivation of **DE** plus a reduction with R.

2.3 Syntactic Properties of Protocol Specifications

We now define some basic syntactic properties of protocols which we will use to prove our results.

Definition 2.7 We call a term t *structured* if whenever the operator *pe* appears in t, the second argument of *pe* is not a variable or rooted with the operator *pe*.

An example of a structured term would be a tagged message such as in [4], in which every term is preceded by a tag describing its type. Another example would be a pair of terms.

Definition 2.8 We call a term t *public key pure* (or pk-pure) if it does not contain any term $pk(k, priv, enc)$ or $pk(k, pub, sig)$ where k may be any term. A set of terms is *pk-pure* if all its elements are pk-pure. A protocol is considered to be *pk-pure* if all terms appearing in the protocol are pk-pure.

Note that all pk-pure terms are irreducible by R.

Definition 2.9 A protocol is considered to be *PEV-free* if no occurrence of *pe* contains a variable as an argument.

3 pk-Purity and PEV-freeness Imply Soundness

We will only consider pk-pure protocols in this section, because we wish to model real protocols where decryptions are implicit. Similarly, the recognition of signatures is also implicit.

We will show that, if a protocol is pk-pure and PEV-free, and if T is pk-pure, then we will show that $Deriv_{\mathbf{DE}}(T) \subseteq Deriv_{\mathbf{D}}(T)$, which implies that $Deriv_{\mathbf{DE}}(T) = Deriv_{\mathbf{D}}(T)$. This will imply that the free algebra model finds all attacks that the extended model does.

First we will need a lemma that says that if a pk-pure irreducible term is PEV-free, then it remains irreducible after applying an irreducible substitution.

Lemma 3.1 *Let t be a pk-pure PEV-free term, and σ be a substitution such that σ is irreducible by R. Then $t\sigma$ is irreducible by R.*

We show that all derivations in **DE** preserve certain properties. To do that, we define a new rewrite system P which reduces all terms to a pk-pure term.

(i) $pe(pk(K, priv, enc), X) \rightarrow X$

(ii) $pe(pk(K, pub, sig), X) \rightarrow X$

We define $t \downarrow_P$ to be the normal form of t modulo P. Note that every pk-pure term is irreducible modulo P.

Lemma 3.2 *Let t be a pk-pure term, and σ be a substitution. If σ is irreducible by P, then $t\sigma$ is irreducible by P.*

This implies that $(t\sigma) \downarrow_P = t(\sigma \downarrow_P)$ if t is pk-pure.

Theorem 3.3 *Let $s_1, \cdots, s_n$ be terms, irreducible by R, such that s can be derived in one step from $s_1, \cdots, s_n$ in* **DE**. *Then either $s \downarrow_P$ can be derived in one step from $s_1 \downarrow_P, \cdots, s_n \downarrow_P$ in* **D** *and s is irreducible by R, or there exists an i such that $s \downarrow_P = s_i \downarrow_P$.*

Theorem 3.4 *Suppose that T is a set of terms irreducible by R. Let t be a term. If $t \in Deriv_{\mathbf{DE}}(T)$ then $t \downarrow_P \in Deriv_{\mathbf{D}}(T \downarrow_P)$. In addition, there exists an $s \in Deriv_{\mathbf{DE}}(T)$ such that s is irreducible by R and $t \downarrow_P = s \downarrow_P$.*

Corollary 3.5 *Suppose that T is a set of PEV-free pk-pure terms. Let t be a pk-pure PEV-free ground term. Let σ be a substitution irreducible by R. If $t\sigma \in Deriv_{\mathbf{DE}}(T\sigma)$ then $t(\sigma \downarrow_P) \in Deriv_{\mathbf{D}}(T(\sigma \downarrow_P))$.*

Corollary 3.6 *Let Pr be a pk-pure PEV-free protocol. Let B be a semibundle of Pr. If B is a bundle in EFA, then it is a bundle in FA.*

4 Conclusions

In this paper we extended Millen's result on the soundness of the free encryption model for shared key encryption to public key encryption as well. Although the proof differed in some important ways from Millen's we found that his general approach extended quite well to public key cryptography. Moreover, we found that our results, similarly to Millen's corresponded well to certain well-established best practices in cryptographic protocol design.

There are a number of ways in which these results could be extended, besides to attack specifications with nonempty negative sets as we noted in the previous section. The most obvious would be to include other best practices

(for example, the use of probabilistic encryption), or to extend our results to models in which both public and single key encryption are used, since most protocols in use today employ both. Moreover, there are a number of other special properties of cryptographic algorithms for which it would be useful to be able to abstract away from in a safe way. For example, the commutative property of Diffie-Hellman can add greatly to the expense of the analysis of a cryptographic protocol. In [7] a specification of Diffie-Hellman is used that abstracts away from the commutative property, but it has the disadvantage that the same key fragment computed by an initiator and a responder will have different representations. This should not be a problem if we can guarantee that a key fragment computed by an initiator will never be confused with one computed by a responder. We have some preliminary results in that direction, using techniques similar to those used in this paper. Other algebraic properties of interest that might be amenable to techniques such as the ones used in [9] and here include the homomorphic properties of RSA used to compute blind signatures, as well as the properties of Diffie-Hellman used in Group Diffie-Hellman. Indeed, an early paper by Even et al. [2] shows that it is safe to leave out the homomorphism property for RSA for a very restricted class of protocols known as ping-pong protocols, so we know that for this property the result is true at least for a limited case. It would also be useful to extend these results to systems that already use cancellation rules, such as the NRL Protocol Analyzer [6]. This would allow us to use cancellation rules when necessary, and the more efficient free algebras when not.

The approach taken in both Millen's and our paper, as well as the proposed work described above, is part of a more general plan that we have outlined in [8]. The general goal is to have a hierarchy of protocol models, each containing different amounts of detail. Given certain conditions on a protocol, one can prove that a less detailed model is sound with respect to the more detailed model, and perform one's analysis with the more efficient but less detailed model. If the conditions are not satisfied, one works with the more detailed model. This approach is not limited to more algebraic properties of cryptosystems; other possibilities, detailed in [8] include type flaw attacks (handled by Heather at al. in [4]), the possibility of key compromise, and cryptographic models. We consider the results in this paper to be a potential building-block in that edifice.

5 Acknowledgments

This work was supported by the Office of Naval Research.

References

[1] I. Cervesato, N. Durgin, P. Lincoln, J. Mitchell, and A. Scedrov. Relating strands and multiset rewriting for security protocol analysis. In *Proc. 13th IEEE*

Computer Security Foundations Workshop. IEEE Computer Society Press, 2000.

[2] S. Even, O. Goldreich, and A. Shamir. On the security of ping-pong protocols when implemented using the RSA. In *Advances in Cryptology: Proceedings of Crypto '85*. Springer-Verlag, 1985.

[3] F. Thayer Fábrega, J.. Herzog, and J. Guttman. Strand spaces: Why is a security protocol correct? In *Proc. 1998 IEEE Symposium on Security and Privacy*, pages 160–171. IEEE Computer Society Press, May 1998.

[4] J. Heather, S. Schneider, and G. Lowe. How to prevent type flaw attacks on security protocols. In *Proc. 13th IEEE Computer Security Foundations Workshop*. IEEE Computer Society Press, 2000.

[5] C. Meadows. Applying formal methods to the analysis of a key management protocol. *Journal of Computer Security*, 1(1), Jan. 1992.

[6] C. Meadows. The NRL Protocol Analyzer: an overview. *Journal of Logic Programming*, 26(2):113–131, 1995.

[7] C. Meadows. Analysis of the Internet Key Exchange Protocol using the NRL Protocol Analyzer. In *Proc. 1999 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, June 1999.

[8] C. Meadows. Towards a hierarchy of cryptographic protocol specifications. In *Proc. FMSE 2003: Formal Methods in Security Engineering*. ACM Press, 2003.

[9] J. Millen. On the freedom of decryption. *Information Processing Letters*, 86(6):329–333, June 2003.

[10] J. Millen and V. Shmatikov. Constraint solving for bounded process cryptographic protocol analysis. In *Proc. 8th ACM Conference on Computer and Communications Security (CCS '01)*, pages 166–175. ACM Press, 2001.

[11] L. Paulson. The inductive approach to verifying cryptographic protocols. *Journal of Computer Security*, 6(1):85–128, 1998.

[12] D. Song, S. Berizin, and A. Perrig. Athena: a novel approach to efficient automatic security analysis. *Journal of Computer Security*, 9(1):47–74, 2001.

Deciding the Security of Protocols with Commuting Public Key Encryption ⋆

Yannick Chevalier[a, 1] Ralf Küsters[b, 4] Michaël Rusinowitch[a, 2]
Mathieu Turuani[a, 3]

[a] *LORIA-INRIA, France*
54506 Vandoeuvre-les-Nancy cedex, France

[b] *Department of Computer Science, Stanford University,*
Stanford CA 94305, USA

Abstract

Many cryptographic protocols and attacks on these protocols make use of the fact that the order in which encryption is performed does not affect the result of the encryption, i.e., encryption is commutative. However, most models for the automatic analysis of cryptographic protocols can not handle such encryption functions since in these models the message space is considered a free term algebra. In this paper, we present an NP decision procedure for the insecurity of protocols that employ RSA encryption, which is one of the most important instances of commuting public key encryption.

Key words: Cryptographic Protocols, Extension of Dolev-Yao Model, Deduction modulo and Complexity.

1 Introduction

Most automatic analysis techniques for security protocols take as a simplifying hypothesis that the cryptographic algorithms are perfect: One needs the decryption key to extract the plaintext from the ciphertext, and also, a ciphertext can be generated only with the appropriate key and message (no collision). Under these assumptions and given a bound on the number of

⋆ This work was partially supported by PROCOPE, by the DFG, and by the FET Open Project IST-2001-39252 "AVISPA: Automated Validation of Internet Security Protocols and Applications", http://www.avispa-project.org/.

[1] email: chevalie@loria.fr
[2] email: rusi@loria.fr
[3] email: turuani@loria.fr
[4] email: kuesters@theory.stanford.edu

This is a preliminary version. The final version will be published in Electronic Notes in Theoretical Computer Science
URL: www.elsevier.nl/locate/entcs

protocol sessions, the insecurity problem is decidable (see e.g. [1,15,5,10]). However, it is an open question whether this result remains valid when the intruder model is extended to take into account even simple algebraic properties of low-level cryptographic primitives. This question is important since many security flaws are the consequence of these properties and many protocols are based on these operators (see, e.g., [16,14]).

Only recently the perfect encryption assumption for protocol analysis has been slightly relaxed. In [12], unification algorithms are designed for handling properties of Diffie-Hellman cryptographic systems. Although these results are useful, they do not solve the more general insecurity problem. In [7,8], decidability of security has been proved for protocols that employ exclusive or. In [6], we have extended this result to protocols that are based on Diffie-Hellman exponentiation. Diffie-Hellman exponentiation has also been studied in [13] and [3]. However, in the former work no decision procedure is provided and in the latter severe restrictions are imposed on the protocol and intruder model.

In this paper, we show that the insecurity problem for protocols that use commuting public-key encryption operators (such as RSA encryption with common modulus) admits an NP decision procedure for a finite number of sessions (see the main result in Section 4). In Section 2, we present a very simple protocol illustrating that protocols and attacks on these protocols may rely on the commutativity of encryption.

This problem can be related to the analysis of Diffie-Hellman protocols as studied in [6] since Diffie-Hellman exponentiation and commuting public-key encryption, which in case of RSA also involves exponentiation, share algebraic properties. However, there are significant differences.

First, the intruder capabilities differ. In case of public-key encryption the intruder is not able to compute the inverse of exponents, e.g., given a public key (n, e) and a cipher text $c = m^e \bmod n$, the intruder can not compute the private key d and then by computing $c^d \bmod n$ obtain m. Conversely, in the Diffie-Hellman setting, exponentiation is done modulo a publicly known prime, and thus, it is computationally feasible to compute the inverse of exponents, e.g., given $m = g^{a \cdot b}$ and b where g generates the multiplicative group induced by the prime p, the intruder can easily invert b modulo $p-1$ obtaining b^{-1} (in case an inverse exists) and by computing $m^{b^{-1}}$ obtain g^a.

Second, in [6] the intruder can not obtain inverses of messages, such as b^{-1} directly, but only use them in exponents. However, in the public-key setting we consider here, this would be unrealistic since inverses correspond to private keys, and of course, we need to allow the intruder to possess such keys (own private keys and private keys of dishonest principals).

As a result of these differences, the proofs differ as well. First, while roughly speaking in [6] we reduce the insecurity problem to solving linear equations in integers, we now obtain linear equations in non-negative integers. Also, we need to extend the intruder to allow private keys in her possession.

To minimize the changes necessary compared to the proof in [6], we consider private keys as atomic messages and extend the normalization function to make sure that in exponents public and private keys cancel each other out. This allows us to lift the proofs presented in [6] to the setting considered here.

2 Examples of Protocols Relying on Commutative Encryption

Let us illustrate by two simple examples given in [17] how commutative properties of public key encryption schemes are employed in cryptographic protocols.

The first protocol is due to Shamir. The aim of this protocol is to permit secure communication between two agents who neither share a symmetric key nor know the public key of the other agent. The protocol uses the commutativity property of the RSA encryption scheme:

$$\begin{aligned}
&1.\ A \to B:\ \{\mathsf{secret}\}^p_{K_A}\\
&2.\ B \to A:\ \{\{\mathsf{secret}\}^p_{K_A}\}^p_{K_B}\\
&3.\ A \to B:\ \{\mathsf{secret}\}^p_{K_B}
\end{aligned}$$

In this protocol, a common RSA modulus n is assumed. The public key of A is (n, K_A) and the one for B is (n, K_B). The message secret is some non-negative integer $< n$. The term $\{\mathsf{secret}\}^p_{K_A}$ stands for $\mathsf{secret}^{K_A} \bmod n$. By the algebraic properties of exponentiation, we have that $\{\{\mathsf{secret}\}^p_{K_A}\}^p_{K_B} = \{\{\mathsf{secret}\}^p_{K_B}\}^p_{K_A}$. In step 3 of the protocol, A computes $\{\{\{\mathsf{secret}\}^p_{K_A}\}^p_{K_B}\}^p_{K'_A} = \{\mathsf{secret}\}^p_{K_B}$ where K'_A is A's private key. Thus, the protocol itself uses the commutativity of encryption. Since B is not authenticated in this protocol, it is obvious that the intruder I can impersonate B, by simply playing B's role while using her own public key K_I.

A commutative public key encryption scheme or signature scheme may also be relevant in the case of group protocols. Inspired by the protocol given in [17], Chapter 23, consider a group of l agents. A trusted server generates two large prime numbers p and q, computes $n = p \cdot q$, and $l + 1$ numbers $k_0, \ldots, k_l$ such that:

$$k_0 \cdots k_l \equiv 1 \bmod (p-1) \cdot (q-1)$$

Each agent A_i, $1 \leq i \leq l$, receives the public keys $K_j = k_0 \cdot \ldots \cdot k_l \cdot {k_j}^{-1}$ for every j and the private key k_i. Note that

$$\{M\}^p_{k_0 \cdots k_l} = M,$$

and in particular,

$$\{\{M\}^p_{k_i}\}^p_{K_i} = M.$$

Once the key distribution is completed, a message can be signed by a subset $\{A_i\}_{i \in I, I \subseteq [1,..,l]}$ of the members of the group. For example, suppose $l = 4$ and

A_1 wants to sign a contract, say the message M, with A_2 and A_4. A possible message sequence is:

$$\begin{aligned}
&1.\ A_1 \to A_2 :\ \{M\}^p_{k_1}\\
&2.\ A_2 \to A_4 :\ \{\{M\}^p_{k_1}\}^p_{k_2}\\
&3.\ A_4 \to A_1 :\ \{\{\{M\}^p_{k_1}\}^p_{k_2}\}^p_{k_4}
\end{aligned}$$

On receiving the second message, A_4 can verify the signatures and identity the agents that have signed M by testing whether

$$\{\{\{\{M\}^p_{k_1}\}^p_{k_2}\}^p_{K_1}\}^p_{K_2} = \{\{\{\{M\}^p_{k_1}\}^p_{K_1}\}^p_{k_2}\}^p_{K_2} = M.$$

Agent A_4 can then also sign the contract using her private key k_4. The point here is that due to the commutativity property, A_4 does not need to know in what order the agents signed the message. Certainly, this protocol, when for instance used as a contract signing protocol, has many problems, which, however, we do not want to discuss here.

3 The Protocol and Intruder Model

The protocol and intruder model we describe here extend standard models for automatic analysis of security protocols in two respects. First, messages can be built using the operator $\{\_\}^p_\_$, which stands for encryption by a multiset of public/private keys described as a product of public/private keys. For instance, we have that $\{\{m\}^p_{K_A}\}^p_{K_B} = \{m\}^p_{K_A \cdot K_B} = \{m\}^p_{K_B \cdot K_A} = \{\{m\}^p_{K_B}\}^p_{K_A}$. In particular, we can model the commutativity of public key encryption. Second, in addition to the standard Dolev-Yao intruder capabilities, the intruder is equipped with the ability to perform this generalized encryption with any set of public or private keys she knows (we even allow arbitrary messages). For instance, if she happens to know A's private key K'_A and the message $c = \{m\}^p_{K_A}$, then she can compute $\{c\}^p_{K'_A} = \{m\}^p_{K_A \cdot K'_A} = \{m\}^p_1 = m$. In what follows, we provide a formal definition of our model by defining terms, messages, protocols, the intruder, and attacks.

3.1 Terms and Messages

The set of terms *term* is defined as the union of *roots* (also called *standard terms*) and *products* (also called *non-standard terms*) in the following grammar:

$$\begin{aligned}
\mathit{root} &::= \mathcal{A} \mid \mathcal{V} \mid \langle \mathit{root}, \mathit{root} \rangle \mid \{\mathit{root}\}^s_{\mathit{root}} \mid \{\mathit{root}\}^p_{\mathit{product}}\\
\mathit{product} &::= \mathit{root}^{\mathbb{N}} \mid \mathit{root}^{\mathbb{N}} \cdot \mathit{product}
\end{aligned}$$

where $\mathcal{A}$ is a finite set of constants (*atomic messages*), containing principal names, nonces, keys, and the constants 1 and secret; $\mathcal{K}$ is a subset of $\mathcal{A}$ denoting the set of public and private keys; $\mathcal{V}$ is a finite set of variables; and

$\mathbb{N}$ is the set of non-negative integers. We assume that there is a bijection $\cdot'$ on $\mathcal{K}$ which maps every public (private) key k to its corresponding private (public) key k'. The binary symbol $\langle \cdot, \cdot \rangle$ is called *pairing*, the binary symbol $\{\cdot\}^s_\cdot$ is called *symmetric encryption*, the binary symbol $\{\cdot\}^p_\cdot$ is *public key encryption*. Note that a symmetric key can be any standard term and that for public key encryption the key can be any non-standard term (product). The non-negative integers occurring in products are called *product exponents*.

Envision a term t as a tree structure where each internal node is labelled by a constructor and each leaf is either a variable or a constant. A term u is a *subterm* if the tree representing u is a subtree of the tree representing t and if u is a root term. We note $\mathrm{Sub}(t)$ the set of subterms of a term t. By extension if E is a set of terms we note $\mathrm{Sub}(E)$ the union for $t \in E$ of the sets $\mathrm{Sub}(t)$.

The *size* of a term t is denoted $|t|$ and is the size of the representation of t by a labelled DAG (*Directed Acyclic Graph*). It is linear with respect to the cardinal of $\mathrm{Sub}(t)$ plus the space needed to represent the coefficients in binary.

A *ground term* (also called *message*) is a term without variables. Like a term, it can be *standard* or *non-standard*. A *(ground) substitution* is a mapping from $\mathcal{V}$ into the set of *standard* (ground) terms. The application of a substitution σ to a term t (a set of terms E) is written $t\sigma$ ($E\sigma$), and is defined as expected.

We now formulate the algebraic properties of terms. Besides commutativity and associativity of the product operator we consider the following properties where t is a standard term, M_1, M_2 are products, $k \in \mathcal{K}$, k' is the corresponding inverse key to k, and z, z' are non-negative integers:

$$t^1 = t \qquad t \cdot 1 = t \qquad \{t\}^p_1 = t$$
$$t^0 = 1 \qquad t^z \cdot t^{z'} = t^{z+z'} \qquad \{\{t\}^p_{M_1}\}^p_{M_2} = \{t\}^p_{M_1 \cdot M_2}$$
$$1^z = 1 \qquad k \cdot k' = 1$$

A *normal form* $\ulcorner t \urcorner$ of a term t is obtained by exhaustively applying these identities from left to right. Note that $\ulcorner t \urcorner$ is uniquely determined up to commutativity and associativity of the product operator. Two terms t and t' are *equivalent* if $\ulcorner t \urcorner = \ulcorner t' \urcorner$. The notion of normal form extends in the obvious way to sets of terms and substitutions. We illustrate the notion of a normal form by some examples: If $a, b, c, d \in \mathcal{K}$, then i) $\ulcorner (a^2 \cdot b^1) \cdot b'^2 \urcorner = a^2 \cdot b'$, ii) $\ulcorner \{\{a\}^p_{b^1 \cdot c^1}\}^p_{c' \cdot d'^2} \urcorner = \{a\}^p_{b \cdot d'^2}$, and iii) $\ulcorner \{\{a\}^p_{b^3 \cdot c'^6 \cdot b'^3}\}^p_{c^6} \urcorner = a$. Recall that, for instance, b' denotes the decryption key corresponding to b.

One easily shows:

3.2 Protocols

Protocols are defined as follows.

Definition 3.1 A *protocol rule* is of the form $R \Rightarrow S$ where R and S are standard terms.

A *protocol* P is a tuple $(\{R_i \Rightarrow S_i \mid i \in \mathcal{I}\}, <_{\mathcal{I}}, E)$ where E is a finite normalized set of standard messages with $1 \in E$, the *initial intruder knowledge*, $\mathcal{I}$ is a finite (index) set, $<_{\mathcal{I}}$ is a partial ordering on $\mathcal{I}$, and $R_i \Rightarrow S_i$, for every $i \in \mathcal{I}$, is a protocol rule such that

(i) the (standard) terms R_i and S_i are normalized,

(ii) for all $x \in \mathcal{V}(S_i)$, there exists $j \leq_{\mathcal{I}} i$ such that $x \in \mathcal{V}(R_j)$, and

(iii) for every subterm $\{t_1\}^p_{t_2^{z_2} \cdots t_n^{z_n}}$ of R_i, there exists $r \in \{1, \ldots, n\}$ such that $\mathcal{V}(t_l) \subseteq \cup_{j <_{\mathcal{I}} i} \mathcal{V}(R_j)$ for every $l \in \{1, \ldots, n\} \setminus \{r\}$.

Condition 1. in the above definition is not an actual restriction. One obtains an equivalent protocol (in the sense that the same attacks are possible) by normalizing the R_i and S_i in the protocol rules in case they are not normalized already. Roughly speaking, Condition 2. guarantees that a principal can only output messages she has learned before. Finally, Condition 3. ensures that every single protocol rule can be applied deterministically to an input message. These conditions do not seem to exclude realistic protocols. See [6] for more detailed remarks on the above conditions. Note that in our model, a *protocol* corresponds to a specification in the Alice&Bob notation *together* with an instantiation. As a result, several sessions are modelled as only one protocol.

In our protocol model, the RSA protocol (Section 2) can formally be stated as follows where we assume that A runs one instance of the protocol as initiator and B runs one instance as responder. The protocol consists of three protocol rules denoted $(A, 1)$, $(A, 2)$, and $(B, 1)$ with

$$(A,1)\colon\ 1 \Rightarrow \{\mathsf{secret}\}^p_{K_A},$$
$$(A,2)\colon\ x \Rightarrow \{x\}^p_{K'_A}, \quad \text{and}$$
$$(B,1)\colon\ y \Rightarrow \{y\}^p_{K_B}$$

where $(A, 1)$ and $(A, 2)$ denote the first and second protocol step performed by A, respectively, and $(B, 1)$ denotes B's protocol step. The partial ordering is $<= \{((A, 1), (A, 2))\}$, i.e., it only satisfies $(A, 1) < (A, 2)$, and thus, makes sure that $(A, 1)$ must be performed before $(A, 2)$. The initial intruder knowledge is $\{1, K_I, K'_I\}$, i.e., besides the constant 1, the intruder knows her public and private key.

3.3 The Intruder Model and Attacks

Given a finite normalized set E of messages, the (infinite) set $\mathit{forge}(E)$ of messages the intruder can derive from E is the smallest set satisfying the following conditions:

(i) $E \subseteq \mathit{forge}(E)$.

(ii) If $\langle m, n\rangle \in \mathit{forge}(E)$, then $m \in \mathit{forge}(E)$ and $n \in \mathit{forge}(E)$. Conversely, if $m, n \in \mathit{forge}(E)$, then $\langle m, n\rangle \in \mathit{forge}(E)$.

(iii) If $\{m\}^s_n \in \mathit{forge}(E)$ and $n \in \mathit{forge}(E)$, then $m \in \mathit{forge}(E)$. Conversely, if $m, n \in \mathit{forge}(E)$, then $\{m\}^s_n \in \mathit{forge}(E)$.

(iv) If $m, m_1, \ldots, m_n \in \mathit{forge}(E)$, and $z_1, \ldots, z_n \in I\!N$ then $\ulcorner\{m\}^p_{m_1^{z_1}\cdots,m_n^{z_n}}\urcorner \in \mathit{forge}(E)$.

While the first three conditions are standard for Dolev-Yao intruders, the last condition, which gives the intruder the ability to perform commuting public key encryption, is new. We call this extended intruder the *RSA intruder*. Note that by performing pubic key encryption, the intruder can both encrypt and decrypt messages. Also note that if E is a set of normalized messages, then so is $\mathit{forge}(E)$.

We now define attacks. In an attack on a protocol P, the intruder (nondeterministically) chooses some execution order for P, i.e., a linearization of the protocol rules which is compatible with the partial ordering, and then tries to produce input messages for the protocol rules. These input messages are derived from the intruder's initial knowledge and the output messages produced by executing the protocol rules. The aim of the intruder is to derive the message secret.

Definition 3.2 Let $P = (\{R_j \Rightarrow S_j \mid j \in \mathcal{I}\}, <_{\mathcal{I}}, S_0)$ be a protocol. Then an *attack* on P consists of a linearization $\pi : R_1 \Rightarrow S_1, \ldots, R_k \Rightarrow S_k$ of the protocol rules in P assuming $k = \mathsf{Card}(I)$ which is compatible with $<_{\mathcal{I}}$ and a normalized ground substitution σ of the variables occurring in P such that

(i) $\ulcorner R_i\sigma\urcorner \in \mathit{forge}(\ulcorner S_0, S_1\sigma, ..., S_{i-1}\sigma\urcorner)$ for every $i \in \{1, \ldots, k\}$ and

(ii) $\mathsf{secret} \in \mathit{forge}(\ulcorner S_0, S_1\sigma, ..., S_k\sigma\urcorner)$.

In Definition 3.2 we restrict ourselves to $k = \mathsf{Card}(I)$ for simplifying notations. Considering $k < \mathsf{Card}(I)$ would amount to detect attack even with partial (i.e. unfinished) sessions. This kind of attacks can be captured too by analyzing protocols with some final steps removed.

The decision problem we are interested in is the following set of protocols:

$$\textsc{Insecure} := \{P \mid \text{ there exists an attack on } P\}.$$

It can easily be checked that the RSA-protocol as formally described in Section 3.2 is insecure according to our definition.

4 Main Theorem

The main result of this paper is the following:

Theorem 4.1 *For the RSA intruder, the problem* Insecure *is NP-complete.*

As mentioned in the introduction, the proof follows the same lines as the

one for Diffie-Hellman exponentiation presented in [6]. Here we only provide a very brief proof sketch.

NP-hardness can easily be established (see for instance [1]). The decision procedure first guesses an execution order of the protocol rules and a ground substitution σ of size polynomially bounded in the size of the protocol, and then checks whether condition 1. and 2. of Definition 3.2 are met. This check can be done in polynomial time.

The completeness and the complexity of this procedure can be proved in two steps. First, we prove that it is possible to check in polynomial time w.r.t. the sizes of $\mathcal{P}$ and σ whether a couple (π, σ) is an attack. Then we prove it is sufficient to consider substitutions σ of polynomial size w.r.t. the size of the protocol in order to prove whether there exists attacks on a given protocol $\mathcal{P}$.

4.1 Deciding whether (π, σ) is an attack

We want to have a procedure with a time complexity polynomial w.r.t. the sizes of $\mathcal{P}$ and σ. Since in Definition 3.2 we have $k \leq |\mathcal{P}|$, it is sufficient to prove that:

(i) for all terms t and all substitutions σ, we have:

$$|\ulcorner t\sigma \urcorner| \leq |t| + |\sigma|$$

(ii) for every set of terms E and for all term m, it can be decided in polynomial time w.r.t. $|E, t|$ if m is in $\mathit{forge}(E)$.

The first point is a consequence of the definition of the normalisation function. We give now a sketch of the proof of the second point.

The deduction power of the intruder can be modelled by an infinite set of ground deduction rules $l \rightarrow r$ where l is a set of terms and r is a term. A set L of such rules defines a transition relation between sets of terms as follows. Let E and E' be two sets of terms, then we have $E \rightarrow_L E'$ iff there exists $l \rightarrow r \in L$ such that $l \subseteq E$ and $E' = E, r$. The set $\mathit{forge}(E)$ can then equivalently be defined as the closure of E by the rules of L.

Let $\mathcal{F}$ be the set of constructors $\{\langle \_, \_ \rangle, \{\_\}^s_\_, \{\_\}^p_\_\}$. The path followed in [4] is then to associate to each constructor f in $\mathcal{F}$ a system of rewrite rules L_f. The first result is that for each constructor $f \in \mathcal{F}$, the transition relation between two sets E and F for the set of rewrite rules L_f can be decided in polynomial time w.r.t. $|E, E'|$. We just give here the proof for the commutative encryption operator $\{\_\}^p_\_$.

Proposition 4.2 *Let E and E' be two sets of terms and $f = \{\_\}^p_\_$. Then $E \rightarrow_{L_f} E'$ can be decided in polynomial time w.r.t. $|E, E'|$.*

Proof. First, we check that $E \subseteq E'$ and $E' \setminus E = \{t\}$. Then let r be the root of t, and E_r be the subset of E of terms of root r. For each $u \in E_r$, compute p_u, the product of t divided by the product of u. If each root term v in p_u is

in E, answer YES.

If no check succeeds, answer NO. □

Example.

Let $E = a, \{a\}^p_{k_a \cdot k'_c}, k_a, k'_a, k_b$ and $F = E \cup \{\{a\}^p_{k_b \cdot k'_c}\}$. We want to decide whether there exists a one-step transition from E to F. We have $E \subseteq F$ and $F \setminus E = \{\{a\}^p_{k_b}\} = \{t\}$. The root of t is the constant a. The subset of E of terms of root a is $E_a = a, \{a\}^p_{k_a \cdot k'_c}$.

- For a, we take the product equal to 1. Thus, $p_a = k_b \cdot k'_c$ and since $k'_c \notin E$, the check fails in this case;
- For $\{a\}^p_{k_a \cdot k'_c}$, we have:

$$p_{\{a\}^p_{k_a \cdot k'_c}} = (k_b \cdot k'_c)/(k_a \cdot k'_c) = k_b \cdot k'_c \cdot k'_a \cdot k_c = k_b \cdot k'_a$$

  In this case, $k_b \in E$ and $k'_a \in E$, and thus the check succeeds. The procedure returns with the answer YES.

There is a transition from E to E' iff there exists $f \in \mathcal{F}$ such that $E \rightarrow_{L_f} E'$. Thus, the one-step transition problem can be decided in polynomial time for the set of rewrite rules $\mathrm{L} = \cup_{f \in \mathcal{F}} L_f$. We now consider the derivation problem:

$$\mathrm{DERIVE} := \{(E, m) \mid m \in \mathit{forge}(E)\}$$

where E is a finite set of normalized standard messages and m is a normalized standard message (both given as DAG). It is equivalent to decide DERIVE and to decide whether there exists n sets of terms $E_1, \ldots, E_n$ such that $E_1 = E$ and $m \in E_n$ and for all $i \in \{1, \ldots, n-1\}$ we have $E_i \rightarrow_{\mathrm{L}} E_{i+1}$. A fundamental result is that if there exists such a sequence of transitions, then there exists one such that $E_n \subseteq \mathrm{Sub}(E, m)$. The proof consists in considering a sequence of transitions of minimal length $E_1 \rightarrow_{\mathrm{L}} \ldots \rightarrow_{\mathrm{L}} E_n$ such that $m \in E_n$ and to prove that each transition adds a subterm of E or of m. Such a transition sequence is called a *normal proof* in [8] or a *well-formed derivation* in [7]. Combining this result with Proposition 4.2, one obtains the following proposition:

Proposition 4.3 *For the RSA intruder,* DERIVE *can be decided in deterministic polynomial time.*

Proof. (sketch) The procedure consists in computing $F = \mathit{forge}(E) \cap \mathrm{Sub}(E, m)$. We start with $F_0 = E$, and F_{i+1} is computed from F_i by adding all the terms t in $\mathrm{Sub}(E, m)$ such that there exists a transition between F_i and F_i, t. The computation stops as soon as $F_{i+1} = F_i$. The existence of well-formed derivation then implies $F_i = \mathit{forge}(E) \cap \mathrm{Sub}(E, m)$. We have to decide a polynomial number of times a one-step transition problem. Thus, the total running time is polynomial w.r.t. $|E, m|$. □

4.2 Bounds on $|\sigma|$

The involved part of the proof of Theorem 4.1 is to show that when there is an attack on a protocol P, then there exists an attack of size polynomially bounded in the size of the protocol. This proof is done in two steps. First, it is shown that the number of subterms occurring in σ can polynomially be bounded in the size of P. Note that this does not bound the size of product exponents in σ. Therefore, in a second step, it is shown that the size of product exponents can polynomially be bounded in the size of P. This is done as follows: Given an attack with substitution σ, the product exponents in σ are replaced by variables (taking non-negative integers) yielding a symbolic substitution σ^Z. Now, we associate a linear Diophantine equation system in non-negative integers (of polynomial size in P) with the attack which constraints the variables in σ^Z (and auxiliary variables) such that when instantiating σ^Z by a non-negative solution of the equation system this also gives an attack on P. By [2], the size of the solutions can polynomially be bounded in the size of the equation system, and thus, P. This shows that the size of product exponents can polynomially be bounded in the size of P.

5 Conclusion

We have shown that the security problem for a class of protocols with commuting public key encryptions is in NP. This result was obtained by a reduction to the satisfiability of linear diophantine equations on $\mathbb{N}$. The result generalizes easily to the more common case where the set of keys yielding commuting encryption is a subset of the set of all keys. It would be interesting to characterize a class of algebraic properties that can be captured by our approach.

References

[1] R. Amadio, D. Lugiez, and V. Vanackere. On the symbolic reduction of processes with cryptographic functions. *Theoretical Computer Science*, 290(1):695–740, 2002.

[2] I. Borosh and L.B. Treybig. Bounds on positive integral solutions of linear Diophantine equations. In Proc. Amer. Math. Soc. 55, 299-304 (A6), 1976.

[3] M. Boreale and M.G. Buscemi. On the symbolic analysis of low-level cryptographic primitives: Modular exponentiation and the Diffie-Hellman protocol. In *Proc. of FCS 2003*.

[4] Y. Chevalier. Résolution de problèmes d'accessibilité pour la compilation et la validation de protocoles cryptographiques Mémoire de thèse, Université Henri Poincaré Nancy 1, 2003.

[5] Y. Chevalier and L. Vigneron. Towards Efficient Automated Verification of Security Protocols. In Proceedings of the Verification Workshop (VERIFY'01)

(in connection with IJCAR'01), Università degli studi di Siena, TR DII 08/01, pages 19–33, 2001.

[6] Y. Chevalier, R. Küsters, M. Rusinowitch, and M. Turuani. Deciding the Security of Protocols with Diffie-Hellman Exponentiation and Products in Exponents. In *Proc. of FSTTCS 2003*.

[7] Y. Chevalier, R. Küsters, M. Rusinowitch, and M. Turuani. An NP Decision Procedure for Protocol Insecurity with XOR. In *Proceedings of the Logic In Computer Science Conference LICS'03*, June 2003. Long version available as Technical Report RR-4697, INRIA, France.

[8] H. Comon-Lundh and V. Shmatikov. Intruder Deductions, Constraint Solving and Insecurity Decision in Presence of Exclusive or. In *Proceedings of the Logic In Computer Science Conference, LICS'03*, pages 271–280, 2003.

[9] R. Corin and S. Etalle. An Improved Constraint-based system for the verification of security protocols. In *Proceedings of the 9th Int. Static Analysis Symp. (SAS)*, LNCS 2477, pp. 326-341, Springer-Verlag, 2002.

[10] R. Küsters. On the Decidability of Cryptographic Protocols with Open-ended Data Structures. In *Proc. of CONCUR 2002*.

[11] R. Küsters and T. Wilke. Automata-based Analysis of Recursive Cryptographic Protocols. In *Proc. 21st Symposium on Theoretical Aspects of Computer Science (STACS 2004)*, LNCS 2996, pages 382–393, Springer-Verlag, 2004.

[12] C. Meadows and P. Narendran. A unification algorithm for the group Diffie-Hellman protocol. In *Proc. of WITS 2002*.

[13] J. Millen and V. Shmatikov. Symbolic protocol analysis with products and Diffie-Hellman exponentiation. In *Proc. of CSFW 16*, 2003.

[14] O. Pereira and J.-J. Quisquater. A Security Analysis of the Cliques Protocols Suites. In *Proc. of CSFW-14*, pages 73–81, 2001.

[15] M. Rusinowitch and M. Turuani. Protocol Insecurity with Finite Number of Sessions is NP-complete. In *Proc. of CSFW-14*, pages 174–190, 2001.

[16] P. Ryan and S. Schneider, An attack on a recursive authentication protocol: A cautionary tale. Information Processing Letters 65 (1998), 7-10.

[17] B. Schneier. *Applied Cryptography*. John Wiley & sons, New York, 1996.

Metareasoning about Security Protocols using Distributed Temporal Logic

Carlos Caleiro [1]

CLC, Department of Mathematics, IST, Lisbon, Portugal

Luca Viganò [2] David Basin [3]

Department of Computer Science, ETH Zurich, Switzerland

Abstract

We introduce a version of distributed temporal logic for rigorously formalizing and proving metalevel properties of different protocol models, and establishing relationships between models. The resulting logic is quite expressive and provides a natural, intuitive language for formalizing both local (agent specific) and global properties of distributed communicating processes. Through a sequence of examples, we show how this logic may be applied to formalize and establish the correctness of different modeling and simplification techniques, which play a role in building effective protocol tools.

Key words: Security protocols, protocol models, intruder models, distributed temporal logic, secrecy, authentication, metareasoning.

1 Introduction

Many security protocols have been proposed to help build secure distributed systems. Given how difficult it is for humans to predict all the possible ways for distributed computation to proceed, it is not so surprising that attacks have been found on many protocols that were originally believed to be secure. Due to the subtlety of the problem, the use of formal methods for analyzing security protocols has been gaining popularity, e.g. [1,2,5,10,13,14,15,16]. In this paper,

[1] Email: ccal@math.ist.utl.pt
[2] Email: vigano@inf.ethz.ch
[3] Email: basin@inf.ethz.ch

This work was partially supported by FCT and EU FEDER via the Project FibLog POCTI/MAT/37239/2001 of CLC, and by the FET Open Project IST-2001-39252 and BBW Project 02.0431, "AVISPA: Automated Validation of Internet Security Protocols and Applications".

This is a preliminary version. The final version will be published in Electronic Notes in Theoretical Computer Science
URL: www.elsevier.nl/locate/entcs

we report on how a suitable version of temporal logic for communicating agents can be used as a *metalevel tool* for analyzing security protocol models and properties.

Our starting point is the distributed temporal logic DTL of [9], which focuses on the expressibility of properties from the local point of view of each agent, and which we extend in order to also express global properties. Aside from its clean interpretation structures, which provide a simple, intuitive model of distributed systems, our reasons for using this logic are primarily threefold. First, it is well-suited for specifying and reasoning about communicating agents in distributed systems. Second, its temporal dimension can be effectively used to formalize and reason about interleaved protocol executions. Finally, its distributed dimension, with explicit agent identifiers, supports an elegant formalization of the different security goals that protocols are supposed to achieve, such as different forms of authentication and secrecy.

The logic we introduce here provides an *object level tool* where we can specify and reason about specific protocols and the properties that the protocols are supposed to establish. In particular, as we describe in [3,4], using the logic it is possible to specify a protocol-independent distributed communication model, on top of which protocols can be formally defined and analyzed. The principal aim of our work, however, is not the mere ad hoc analysis of specific protocols. Rather, our long-term objective is to use our logic as a *metalevel tool* for the comparative analysis of security protocol models and properties. Our logic provides a basis to rigorously investigate general metalevel properties of different protocol models by establishing modeling and analysis simplification techniques that contribute to the sound design of effective protocol validation tools. In this regard, we believe that our logic can contribute to clarifying the concepts involved by providing a basis for naturally representing and reasoning about the underlying computational models.

We anticipate several applications. The most direct consists of a rigorous account of different widely used simplification techniques, as we discuss in this paper. We prove here a general lemma about *secret data* that is similar to the secrecy theorems of [7,12]. We also obtain soundness and completeness results, with respect to typical security goals, for two model-simplification techniques: *one intruder is enough*, along the lines of [6], and the *predatory intruder*, a restriction on the behavior of the intruder, variants of which underly the trace models used in practice, e.g. [14]. While these results, mutatis mutandis, have already been shown for other particular formalisms, our logic provides a means for proving them in a general and uniform way within the same formalism, which paves the way for further general investigations. Our formalization has also allowed us to clarify aspects of these simplification properties that are often neglected or cannot be specified in the first place (e.g. concerning principals' identities and the way security properties are established).

We have also begun applying our logic to other metatheoretical investigations, such as developing appropriate partial-order techniques that may reduce

the (potentially infinite) state-space exploration involved in model-checking protocol properties (cf. [2]). This is work in progress and the first results are promising.

We proceed as follows. In §2 we introduce our distributed temporal logic. Using the logic, in §3, we define a protocol-independent distributed communication model, on top of which protocols and security goals can be formalized and analyzed, as shown in §4. In §5 we present metalevel results, and conclude in §6 with a discussion of related and future work.

2 Distributed temporal logic

DTL [9] is a logic for reasoning about temporal properties of distributed systems from the local point of view of the system's agents, which are assumed to execute sequentially and to interact by means of synchronous event sharing. Distribution is implicit, making it easier to state the properties of an entire system through the local properties of its agents and their interaction. Herein, we introduce a minor extension of DTL tailored also to support the smooth formalization and proof of global properties.

The logic is defined over a *distributed signature*

$$\Sigma = \langle Id, \{Act_i\}_{i\in Id}, \{Prop_i\}_{i\in Id}\rangle$$

of a system, where Id is a finite set of *agent identifiers* and, for each $i \in Id$, Act_i is a set of *local action symbols* and $Prop_i$ is a set of *local state propositions*. The *global language* $\mathcal{L}$ is defined by the grammar

$$\mathcal{L} ::= @_i[\mathcal{L}_i] \mid \bot \mid \mathcal{L} \Rightarrow \mathcal{L}\,,$$

for $i \in Id$, where the *local languages* $\mathcal{L}_i$ are defined by

$$\mathcal{L}_i ::= Act_i \mid Prop_i \mid \bot \mid \mathcal{L}_i \Rightarrow \mathcal{L}_i \mid \mathcal{L}_i \,\mathsf{U}\, \mathcal{L}_i \mid \mathcal{L}_i \,\mathsf{S}\, \mathcal{L}_i \mid @_j[\mathcal{L}_j]\,,$$

with $j \in Id$. Locally for an agent, U and S are respectively the *until* and *since* temporal operators. Actions correspond to true statements about an agent when they have just occurred, whereas state propositions characterize the current local states of the agents. Note that $@_j[\varphi]$ means different things depending on the context. If it is a global formula, it means that φ holds at the current local state of agent j. If it is a local formula appearing inside an $@_i$-formula then it is called a *communication formula* and it means that agent i has just communicated with agent j for whom φ held.

The interpretation structures of $\mathcal{L}$ are suitably labeled distributed life-cycles, built upon a simplified form of Winskel's *event structures* [19]. For brevity, we just give an outline of their definition here and refer to [3] for details. A *local life-cycle* of an agent $i \in Id$ is a pair $\lambda_i = \langle Ev_i, \rightarrow_i\rangle$, where Ev_i is the set of *local events* and $\rightarrow_i \subseteq Ev_i \times Ev_i$ is the *local successor relation*,

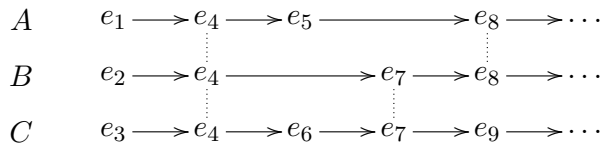

Fig. 1. A distributed life-cycle for agents A, B and C.

$$\pi_A(\emptyset) \xrightarrow{\alpha_A(e_1)} \pi_A(\{e_1\}) \xrightarrow{\alpha_A(e_4)} \pi_A(\{e_1, e_4\}) \xrightarrow{\alpha_A(e_5)} \pi_A(\{e_1, e_4, e_5\}) \xrightarrow{\alpha_A(e_8)} \cdots$$

Fig. 2. The progress of agent A.

such that the transitive closure $\rightarrow_i^*$ defines a well-founded total order of *local causality* on Ev_i. A *distributed life-cycle* is a family $\lambda = \{\lambda_i\}_{i \in Id}$ of local life-cycles such that the transitive closure $\rightarrow^*$ of $\rightarrow = \bigcup_{i \in Id} \rightarrow_i$ defines a partial order of *global causality* on the set $Ev = \bigcup_{i \in Id} Ev_i$ of all events. This last condition is essential since events can be shared by several agents at communication points.

We can check the progress of an agent by collecting all the local events that have occurred up to a certain point. This yields the notion of the *local configuration* of an agent i: a finite set $\xi_i \subseteq Ev_i$ closed under local causality, i.e. if $e \rightarrow_i^* e'$ and $e' \in \xi_i$ then also $e \in \xi_i$. The set Ξ_i of all local configurations of an agent i is clearly totally ordered by inclusion and has $\emptyset$ as the minimal element. In general, each non-empty local configuration ξ_i is reached, by the occurrence of an event that we call $last(\xi_i)$, from the local configuration $\xi_i \setminus \{last(\xi_i)\}$. We can also define the notion of a *global configuration*: a finite set $\xi \subseteq Ev$ closed for global causality, i.e. if $e \rightarrow^* e'$ and $e' \in \xi$ then also $e \in \xi$. The set Ξ of all global configurations constitutes a lattice, under inclusion, and has $\emptyset$ as the minimal element. Clearly, every global configuration ξ includes the local configuration $\xi|_i = \xi \cap Ev_i$ of each agent i. Given $e \in Ev$, note that $e{\downarrow} = \{e' \in Ev \mid e' \rightarrow^* e\}$ is always a global configuration.

An *interpretation structure* $\mu = \langle \lambda, \alpha, \pi \rangle$ consists of a distributed life-cycle λ plus families $\alpha = \{\alpha_i\}_{i \in Id}$ and $\pi = \{\pi_i\}_{i \in Id}$ of local labeling functions. For each $i \in Id$, $\alpha_i : Ev_i \rightarrow Act_i$ associates a local action to each local event, and $\pi_i : \Xi_i \rightarrow \wp(Prop_i)$ associates a set of local state propositions to each local configuration. We denote the tuple $\langle \lambda_i, \alpha_i, \pi_i \rangle$ also by μ_i.

Fig. 1 illustrates the notion of a distributed life-cycle, where each row comprises the local life-cycle of one agent. In particular, $Ev_A = \{e_1, e_4, e_5, e_8, \dots\}$ and $\rightarrow_A$ corresponds to the arrows in A's row. We can think of the occurrence of the event e_1 as leading agent A from its initial configuration $\emptyset$ to the configuration $\{e_1\}$, and then of the occurrence of the event e_4 as leading to configuration $\{e_1, e_4\}$, and so on; the state-transition sequence of agent A is displayed in Fig. 2. Shared events at communication points are highlighted by the dotted vertical lines. Note that the numbers annotating the events are there only for convenience since no global total order on events is in general imposed. Fig. 3 shows the corresponding lattice of global configurations.

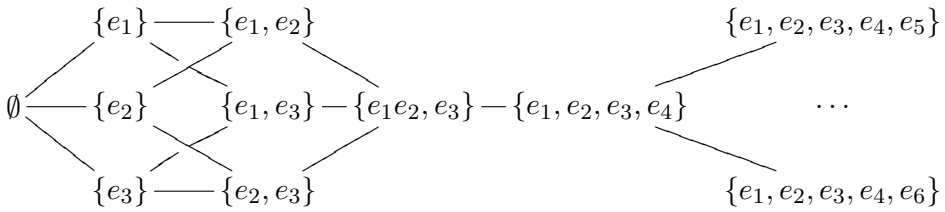

Fig. 3. The lattice of global configurations.

We can then define the *global satisfaction relation* at a global configuration ξ of μ as

- $\mu, \xi \Vdash @_i(\varphi)$ if $\mu, \xi|_i \Vdash_i \varphi$;
- $\mu, \xi \nVdash \bot$;
- $\mu, \xi \Vdash \gamma \Rightarrow \delta$ if $\mu, \xi \nVdash \gamma$ or $\mu, \xi \Vdash \delta$,

where the *local satisfaction relations* at local configurations are defined by

- $\mu, \xi_i \Vdash_i act$ if $\xi_i \neq \emptyset$ and $\alpha_i(last(\xi_i)) = act$;
- $\mu, \xi_i \Vdash_i p$ if $p \in \pi_i(\xi_i)$;
- $\mu, \xi_i \nVdash_i \bot$;
- $\mu, \xi_i \Vdash_i \varphi \Rightarrow \psi$ if $\mu, \xi_i \nVdash_i \varphi$ or $\mu, \xi_i \Vdash_i \psi$;
- $\mu, \xi_i \Vdash_i \varphi \,\mathsf{U}\, \psi$ if there exists $\xi_i'' \in \Xi_i$ with $\xi_i \subsetneq \xi_i''$ such that $\mu, \xi_i'' \Vdash_i \psi$, and $\mu, \xi_i' \Vdash_i \varphi$ for every $\xi_i' \in \Xi_i$ with $\xi_i \subsetneq \xi_i' \subsetneq \xi_i''$;
- $\mu, \xi_i \Vdash_i \varphi \,\mathsf{S}\, \psi$ if there exists $\xi_i'' \in \Xi_i$ with $\xi_i'' \subsetneq \xi_i$ such that $\mu, \xi_i'' \Vdash_i \psi$, and $\mu, \xi_i' \Vdash_i \varphi$ for every $\xi_i' \in \Xi_i$ with $\xi_i'' \subsetneq \xi_i' \subsetneq \xi_i$;
- $\mu, \xi_i \Vdash_i @_j[\varphi]$ if $\xi_i \neq \emptyset$, $last(\xi_i) \in Ev_j$ and $\mu, (last(\xi_i)\downarrow)|_j \Vdash_j \varphi$.

We say that μ is a model of $\Gamma \subseteq \mathcal{L}$ if $\mu, \xi \Vdash \gamma$ for every global configuration ξ of μ and every $\gamma \in \Gamma$. Other standard operators are defined as abbreviations, e.g. $\neg$, $\top$, $\vee$, $\wedge$, and

$\mathsf{X}\varphi \equiv \bot \,\mathsf{U}\, \varphi$	next	$\dagger \equiv \neg \mathsf{X} \top$	in the end
$\mathsf{Y}\varphi \equiv \bot \,\mathsf{S}\, \varphi$	previous	$* \equiv \neg \mathsf{Y} \top$	in the beginning
$\mathsf{F}\varphi \equiv \top \,\mathsf{U}\, \varphi$	sometime in the future	$\mathsf{F}_\circ \varphi \equiv \varphi \vee \mathsf{F}\varphi$	now or sometime in the future
$\mathsf{P}\varphi \equiv \top \,\mathsf{S}\, \varphi$	sometime in the past	$\mathsf{P}_\circ \varphi \equiv \varphi \vee \mathsf{P}\varphi$	now or sometime in the past
$\mathsf{G}\varphi \equiv \neg \mathsf{F} \neg \varphi$	always in the future	$\mathsf{G}_\circ \varphi \equiv \varphi \wedge \mathsf{G}\varphi$	now and always in the future
$\mathsf{H}\varphi \equiv \neg \mathsf{P} \neg \varphi$	always in the past	$\mathsf{H}_\circ \varphi \equiv \varphi \wedge \mathsf{H}\varphi$	now and always in the past

Fig. 4 illustrates the satisfaction relation with respect to communication formulas of our running example. Clearly $\mu, \emptyset \Vdash @_B[\psi \mathsf{U} @_A[\varphi]]$, because $\mu, \xi' \Vdash @_B[@_A[\varphi]]$. Note however that $\mu, \xi \nVdash @_B[@_A[\varphi]]$, although $\mu, \xi \Vdash @_A[\varphi]$.

Rules for proving invariants by induction can be established in our logic in the standard way (see [3,4]).

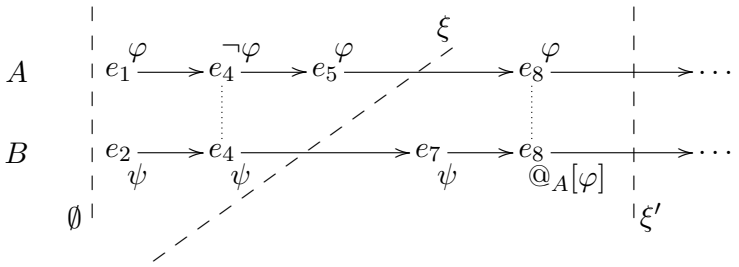

Fig. 4. Satisfaction of formulas.

3 The network model

We provide a specification of a generic open network where agents interact by exchanging messages through an insecure public channel. A *network signature* is a pair $\langle Princ, Name\rangle$, where $Princ$ is a finite set of principal identifiers $A, B, C, \ldots$, and $Name$ is a family $\{Name_A\}_{A\in Princ}$ of pairwise disjoint finite sets of *names*, corresponding to the possible aliases used by each principal (the importance of aliases will become clearer below, e.g. in §5.2). We use primed notation to denote names, e.g. writing A' to denote a name used by principal A. By abuse of notation, we also use $Name = \bigcup_{A\in Princ} Name_A$. Furthermore, we assume fixed two sets $Nonce$ and Key of "numbers" that can be used as *nonces* and *keys*, respectively, and whose members we denote by N and K, possibly with annotations. In general, we assume that several kinds of keys can coexist and that each key K has its own inverse key K^{-1}. *Messages*, which we denote by M, possibly with annotations, are built inductively from *atomic messages* (names and "numbers"), by concatenation $(\_;\_)$, which we assume to be associative, and encryption under a key K ($\{\_\}_K$). The set Msg of messages is thus defined by

$$Msg ::= Name \mid Nonce \mid Key \mid Msg; Msg \mid \{Msg\}_{Key}\,.$$

Note that we consider an equational signature with four sorts, namely the sort of messages and its subsorts names, nonces and keys, where we follow the usual *free-algebra assumption* so that syntactically different terms denote different messages.

Given a network signature $\langle Princ, Name\rangle$, we obtain a distributed signature by taking $Id = Princ \uplus \{Ch\}$, where Ch is the communication channel (used to model asynchronous communication), and defining the local alphabet of each agent (the principals and the channel) as follows. The signature of a principal A requires actions Act_A and state propositions $Prop_A$, where Act_A includes

- $send(M, B')$ — sending of the message M to B';
- $rec(M)$ — reception of the message M;
- $spy(M)$ — eavesdropping of the message M; and
- $nonce(N)$ — generation of the fresh nonce N,

and $Prop_A$ includes

- $knows(M)$ — knowledge of the message M.

Note that we do not explore the epistemic dimension of this knowledge.

For the channel Ch we do not require any state propositions, i.e. $Prop_{Ch} = \emptyset$, whereas the actions Act_{Ch} include

- $in(M, A')$ — arrival at the channel of the message M addressed to A';
- $out(M, A')$ — delivery of the message M from the channel to principal A; and
- $leak$ — leaking of messages.

The model could, of course, be extended in many ways. For example, we could include other kinds of message constructors (e.g. for hashing and exponentiation), or further actions and state propositions. We will consider such extensions in future work, where we will also include servers and further channels with distinct accessibility and reliability properties. For now, however, the above is enough to abstractly formalize and reason about the properties of communication between principals executing security protocols.

In the network model that we define, principals can send and receive messages, at will, always through the channel. If the principal A sends a message to B', then the message synchronously arrives at the channel, where it is stored for future delivery to B. If delivery ever happens, it must be synchronized with the corresponding receive action of B. However, the principal A can only send M to B' if A knows both the name B' and how to produce the message M. As usual, the knowledge of principals is not static. In addition to their initial knowledge, principals gain knowledge from the messages they receive and the fresh nonces they generate. Principals may also spy on messages being leaked by the channel and learn their content. We do not allow principals to explicitly divert messages, but we also do not guarantee that messages delivered to the channel are ever received.

To ensure that principals cannot learn messages in an ad hoc fashion, we specify that the *knows* propositions only hold where strictly necessary. To this end, we follow the idea underlying Paulson's inductive model [14], in accordance with the usual assumption of *perfect cryptography* (that the only way to decrypt an encrypted message is to have the appropriate key). We restrict attention to those interpretation structures μ such that, for every principal A, the following condition holds for all messages M and global configurations $\xi \in \Xi$ such that $\xi|_A \neq \emptyset$:

(K) $\mu, \xi \Vdash_A knows(M)$ iff $M \in synth(analz(\{M' \mid \mu, \xi \Vdash_A (\mathsf{Y}\, knows(M')) \vee rec(M') \vee spy(M') \vee nonce(M')\}))$,

where *analz* and *synth* are the functions representing how principals analyze or synthesize messages from a given set of messages (see, e.g., [14]). Note that **(K)** implies that, in every model $\mu = \langle \lambda, \alpha, \pi \rangle$ of the specification, π is completely determined by λ and α, given $\pi_A(\emptyset)$ for each $A \in Princ$. This is

equivalent to saying that the knowledge of each principal only depends on its initial knowledge and on the actions that have occurred. A number of other useful properties follow from **(K)**, e.g. for each principal $A \in \mathit{Princ}$:

(K1) $@_A[\mathit{knows}(M_1; M_2) \Leftrightarrow (\mathit{knows}(M_1) \wedge \mathit{knows}(M_2))]$;

(K2) $@_A[(\mathit{knows}(M) \wedge \mathit{knows}(K)) \Rightarrow \mathit{knows}(\{M\}_K)]$;

(K3) $@_A[(\mathit{knows}(\{M\}_K) \wedge \mathit{knows}(K^{-1})) \Rightarrow \mathit{knows}(M)]$;

(K4) $@_A[\mathit{knows}(M) \Rightarrow \mathsf{G}_\circ\, \mathit{knows}(M)]$;

(K5) $@_A[\mathit{rec}(M) \Rightarrow \mathit{knows}(M)]$;

(K6) $@_A[\mathit{spy}(M) \Rightarrow \mathit{knows}(M)]$; and

(K7) $@_A[\mathit{nonce}(N) \Rightarrow \mathit{knows}(N)]$.

To guarantee the freshness and uniqueness of the nonces generated by each principal, we further require the axioms

(N1) $@_A[\mathit{nonce}(N) \Rightarrow \mathsf{Y} \neg\, \mathit{knows}(M_N)]$,

(N2) $@_A[\mathit{nonce}(N)] \Rightarrow \bigwedge_{B \in \mathit{Princ} \setminus \{A\}} @_B[\neg\, \mathit{knows}(M_N)]$,

where M_N ranges over all the messages containing the nonce N. Together with (K7), **(N1)** and **(N2)** guarantee that every nonce is generated at most once, if at all, in each model, and always freshly (also taking into account the initial knowledge of all agents). The specification of the network model also comprises a number of axioms that characterize the behavior of the channel and of each principal $A \in \mathit{Princ}$:

(C1) $@_{\mathit{Ch}}[\mathit{in}(M, A') \Rightarrow \bigvee_{B \in \mathit{Princ}} @_B[\mathit{send}(M, A')]]$;

(C2) $@_{\mathit{Ch}}[\mathit{out}(M, A') \Rightarrow \mathsf{P}\ \mathit{in}(M, A')]]$; and

(C3) $@_{\mathit{Ch}}[\mathit{out}(M, A') \Rightarrow @_A[\mathit{rec}(M)]]$,

(P1) $@_A[\mathit{send}(M, B') \Rightarrow \mathsf{Y}(\mathit{knows}(M) \wedge \mathit{knows}(B'))]$;

(P2) $@_A[\mathit{send}(M, B') \Rightarrow @_{\mathit{Ch}}[\mathit{in}(M, B')]]$;

(P3) $@_A[\mathit{rec}(M) \Rightarrow @_{\mathit{Ch}}[\bigvee_{A' \in \mathit{Name}_A} \mathit{out}(M, A')]]$;

(P4) $@_A[\mathit{spy}(M) \Rightarrow @_{\mathit{Ch}}[\mathit{leak} \wedge \mathsf{P} \bigvee_{B' \in \mathit{Name}} \mathit{in}(M, B')]]$;

(P5) $@_A[\bigwedge_{B \in \mathit{Princ} \setminus \{A\}} \neg @_B[\top]]$; and

(P6) $@_A[\mathit{nonce}(N) \Rightarrow \neg\, @_{\mathit{Ch}}[\top]]$.

The channel axioms **(C1–C3)** are straightforward. They state that a message addressed to A' only arrives at the channel if it is sent to A' by some principal B; that the channel only delivers a message to A' if such a message for A' has previously arrived; and that if the channel delivers a message to A' then A receives it. The principal axioms are also simple. **(P1)** is a precondition for sending a message, stating that the sender must know both the message and the recipient's name beforehand. The next three formulas are interaction axioms. **(P2)** and **(P3)** state that the sending and receiving of messages, respectively, must be shared with the corresponding arrival and delivery actions

of the channel. **(P4)** guarantees that a spied message must have arrived at the channel, addressed to some recipient. The two final axioms limit the possible interactions: **(P5)** guarantees that principals never communicate directly (only through the channel), and **(P6)** states that nonce generating actions are not communication actions.

4 Modeling security protocols

Protocols are usually informally described by short sequences of messages that are exchanged by principals in order to achieve particular security goals in open, hostile environments. We model protocols on top of our network.

We illustrate protocol modeling by using a standard example: the (flawed) simplified Needham-Schroeder Public Key Protocol NSPK [10], which we present as the following sequence of message exchange steps.

$$\begin{aligned}
&(\text{step}_1)\ a \to b : (n_1).\ \{n_1; a\}_{K_b}\\
&(\text{step}_2)\ b \to a : (n_2).\ \{n_1; n_2\}_{K_a}\\
&(\text{step}_3)\ a \to b :\qquad\ \{n_2\}_{K_b}
\end{aligned}$$

In this notation, a and b are variables of sort name that denote the roles played in one execution of the protocol, and n_1 and n_2 are variables of sort nonce. The arrows represent communication from the sender to the receiver. The parenthesized nonces prefixing the first and second messages signify that these nonces must be freshly generated before the message is sent. Moreover, it is assumed that *public* and *private* keys have been generated and appropriately distributed: K_a represents the public key of a, whose inverse key should be private, i.e. known by no one but the principal using that name. Although other possibilities, such as shared keys, could be easily added to the model, we refrain from doing so here, for simplicity, and assume that these are the only existing keys.

Formalizing a protocol like the above involves defining the sequences of actions (*send*, *rec*, and *nonce*) taken by honest agents executing the protocol. Namely, for each role, we formalize the actions taken and the order in which they must be taken. In the case of NSPK, there are two roles: an initiator role *Init*, represented by a, and a responder role *Resp*, represented by b. Given distinct names A' and B', of principals A and B respectively, and nonces N_1 and N_2, the role instantiations should correspond to the execution, by principal A, of the sequence of actions $\text{run}_A^{Init}(A', B', N_1, N_2)$:

$$\langle \mathit{nonce}(N_1).\mathit{send}(\{N_1; A'\}_{K_{B'}}, B').\mathit{rec}(\{N_1; N_2\}_{K_{A'}}).\mathit{send}(\{N_2\}_{K_{B'}}, B')\rangle\,,$$

and to the execution, by principal B, of the sequence $\text{run}_B^{Resp}(A', B', N_1, N_2)$:

$$\langle \mathit{rec}(\{N_1; A'\}_{K_{B'}}).\mathit{nonce}(N_2).\mathit{send}(\{N_1; N_2\}_{K_{A'}}, A').\mathit{rec}(\{N_2\}_{K_{B'}})\rangle\,.$$

In the remainder of the paper, we use $w = \langle w_1.w_2.w_3 \dots \rangle$ to denote a (possibly infinite) sequence composed of the elements $w_1, w_2, w_3, \dots$, and we use $|w|$ to denote its length. Of course, $\langle\rangle$ denotes the empty sequence and $|\langle\rangle| = 0$. We assume that $|w| = \infty$ if w is infinite. We write $w \centerdot w'$ to denote sequence concatenation, provided that the first sequence is finite.

In general, a protocol description like the one above may involve j name variables $a_1, \dots, a_j$, corresponding to j distinct roles, and k nonce variables $n_1, \dots, n_k$, and consist of a sequence $\langle \text{step}_1 \dots \text{step}_m \rangle$ of message exchange steps, each of the form

$$(\text{step}_q) \quad a_s \to a_r \ : \ (n_{q_1}, \dots, n_{q_t}).\ M\,,$$

where M can include any of the name and nonce variables. A *protocol instantiation* is a variable substitution σ such that each $\sigma(a_i) \in \mathit{Name}$, each $\sigma(n_i) \in \mathit{Nonce}$, and σ is injective on name variables, i.e. if $i_1 \neq i_2$ then $\sigma(a_{i_1}) \neq \sigma(a_{i_2})$. We extend σ to messages, actions, sequences, and formulas in the natural way. Each instantiation prescribes a concrete sequence of actions to be executed by each of the participants in a run of the protocol: for each role i, if $\sigma(a_i) \in \mathit{Name}_A$ then we have the corresponding sequence $\text{run}^i_A(\sigma) = \sigma(\text{step}^i_1) \centerdot \dots \centerdot \sigma(\text{step}^i_m)$ where

$$\text{step}^i_q = \begin{cases} \langle \mathit{nonce}(n_{q_1}) \dots \mathit{nonce}(n_{q_t}).\mathit{send}(M, a_r) \rangle & \text{if } i = s, \\ \langle \mathit{rec}(M) \rangle & \text{if } i = r, \\ \langle\rangle & \text{if } i \neq s \text{ and } i \neq r. \end{cases}$$

We can easily formalize in the logic the complete execution by principal A of the run corresponding to role i of the protocol. If $\text{run}^i_A(\sigma) = \langle \mathit{act}_1 \dots \mathit{act}_n \rangle$ then we can consider the local formula $\text{role}^i_A(\sigma)$:

$$\mathit{act}_n \wedge \mathsf{P}(\mathit{act}_{n-1} \wedge \mathsf{P}(\dots \wedge \mathsf{P}\, \mathit{act}_1) \dots)\,.$$

In general, if we denote the set of all protocol instantiations by Inst, we can define the set Runs^i_A of all possible concrete runs of principal A in role i, and the set Runs_A of all of A's possible concrete runs in any of the j roles:

$$\mathit{Runs}^i_A = \bigcup_{\sigma \in \mathit{Inst}} \{\text{run}^i_A(\sigma) \mid \sigma(a_i) \in \mathit{Name}_A\} \qquad \text{and} \qquad \mathit{Runs}_A = \bigcup_{i=1}^{j} \mathit{Runs}^i_A\,.$$

It should be clear that $\mu, \xi \Vdash @_A[\text{role}^i_A(\sigma)]$ if and only if A has just completed the required sequence of actions $\text{run}^i_A(\sigma)$ at ξ. Often, in examples, we will use $\bar{a} = \langle a_1 \dots a_j \rangle$ and $\bar{n} = \langle n_1 \dots n_k \rangle$, and write $\text{run}^i_A(\sigma(\bar{a}), \sigma(\bar{n}))$ instead of $\text{run}^i_A(\sigma)$, and $\text{role}^i_A(\sigma(\bar{a}), \sigma(\bar{n}))$ instead of $\text{role}^i_A(\sigma)$.

4.1 Honesty

We take an external view of the system, and consider a *protocol signature* to be a triple $\langle Hon, Intr, Name\rangle$ where Hon and $Intr$ are disjoint sets of *honest* and *intruder* principals, and $\langle Hon \cup Intr, Name\rangle$ is a network signature such that every honest principal has exactly one name. Note that this implies that no honest agent will ever play two different roles in the same run of a protocol. Without loss of generality, we assume that $Name_A = \{A\}$ for every $A \in Hon$. This implies that if we know that a principal A is honest then we always write A instead of A'. We assume also that the private key of each honest principal is initially only known by that principal. This can be achieved by the axioms **(Key1)** and **(Key2)** below, where $A \in Hon$:

(Key1) $@_A[* \Rightarrow knows(K_A^{-1})]$; and

(Key2) $@_B[* \Rightarrow \neg\, knows(M)]$, for every $B \in Princ \setminus \{A\}$ and every M containing K_A^{-1}.

Models of a protocol are those network models where, furthermore, all honest principals strictly follow the rules of the protocol. That is, for every $A \in Hon$, if the local life-cycle of A is $e_1 \rightarrow_A e_2 \rightarrow_A e_3 \rightarrow_A \ldots$, then the corresponding (possibly infinite) sequence of actions

$$w(A) = \langle \alpha_A(e_1).\alpha_A(e_2).\alpha_A(e_3)\ldots\rangle$$

must be an interleaving of prefixes of sequences in $Runs_A$, but using distinct fresh nonces in each of them. Formally, we say that two sequences of actions w and w' are *independent* provided that if $w_i = nonce(N)$, for some $i \leq |w|$ and $N \in Nonce$, then $w'_j \neq nonce(N)$ for every $j \leq |w'|$. The requirement on protocol models can now be rigorously defined. For each $A \in Hon$, there must exist a set $W \subseteq Runs_A$ of pairwise independent sequences such that for every $i \leq |w(A)|$ it is possible to choose $w \in W$, $j \leq |w|$ and $i_1 < \cdots < i_j = i$ satisfying $w(A)_{i_k} = w_k$ for all $k \leq j$.

Note that this is essentially equivalent to approaches such as [14], where the behavior of an honest agent A is defined inductively in such a way that the jth action of a sequence $w \in Runs_A$ can be executed only if the previous $j-1$ actions have already been executed, or to strand spaces [17,18] where essentially the same sequences of $Runs_A$ are used to model honest agents. In all cases, the intruders (*attackers* or *penetrators*) can act freely, according to the standard Dolev-Yao capabilities.

In the case of the NSPK protocol, this means that the life-cycle of each honest agent must be built by interleaving prefixes of sequences of the form $\mathrm{run}_A^{Init}(A, B', N_1, N_2)$ or $\mathrm{run}_A^{Resp}(B', A, N_1, N_2)$, where no two such initiator runs can have the same N_1, no two responder runs can have the same N_2, and the N_1 of an initiator run must be different from the N_2 of any responder run.

4.2 Security goals

The aim of protocol analysis is to prove (or disprove) the correctness of a protocol with respect to the security goals that the protocol should achieve. For instance, the *secrecy* of the critical data exchanged during an execution of the protocol among its participants is one such goal. In addition, an honest principal running the protocol should be able to *authenticate* the identities of its protocol partners by examining the messages he receives. There are many approaches to specifying secrecy and authentication in the literature, depending in part on the underlying model used. However, the various approaches mostly agree on the general picture. Below, we show how to formulate the required secrecy and authentication goals of protocols in the general case, illustrating them by means of the NSPK protocol.

As usual, given a security goal, we call an *attack* on a protocol any protocol model μ and configuration ξ for which the formula expressing the goal does not hold. Let us start with secrecy.

Secrecy

We can formalize that the messages in a finite set S will remain a shared secret between the participants $A_1, \ldots, A_j$ after the complete execution of a protocol instantiation σ, with each $\sigma(a_i) \in \mathit{Name}_{A_i}$, by the formula $\mathrm{secr}_S(\sigma)$:

$$\bigwedge_{i=1}^{j} @_{A_i}[\mathsf{P}_\circ\ \mathrm{role}^i_{A_i}(\sigma)] \Rightarrow \bigwedge_{B \in \mathit{Princ} \setminus \{A_1,\ldots,A_j\}} \bigwedge_{M \in S} @_B[\neg\, \mathit{knows}(M)].$$

Of course, this property can only be expected to hold in particular situations. Assume that all the participants are honest, i.e. each $A_i \in \mathit{Hon}$ and so $\mathit{Name}_{A_i} = \{A_i\}$. One might then expect that the "critical" nonces generated during that run will remain a secret shared only by the participating principals. Indeed, being honest, they will not reuse those nonces in further protocol runs. Using the logic, we can check the property $\mathrm{secr}_{\sigma(F)}(\sigma)$ for the relevant set of fresh nonce variables $F \subseteq \{n_1, \ldots, n_k\}$. As before, we sometimes write $\mathrm{secr}_{\sigma(F)}(\sigma(\bar{a}), \sigma(\bar{n}))$ instead of $\mathrm{secr}_{\sigma(F)}(\sigma)$.

In the case of the NSPK protocol, this would amount to requiring that $\mathrm{secr}_{\{N_1,N_2\}}(A, B, N_1, N_2)$ holds, with A and B both honest.

Authentication

There are many possible notions of authentication (see, e.g., [11]). However, most authors agree that authentication expresses some kind of correspondence property between the messages an agent receives in a protocol run and the messages that other participants of the same run are supposed to send. The typical authentication goal states that if an honest principal A completes his part of a run of a protocol in role i, with certain partners and data, then

it must be the case that these partners have also been actively involved by sending to A the messages that he received.

Given a protocol instantiation σ such that $\sigma(a_i) = A \in \mathit{Hon}$ and $\sigma(a_j) \in \mathit{Name}_B$, the property that A authenticates B in role j at step q of the protocol can be defined in our logic by the formula $\mathrm{auth}^{i,j,q}_{A,B}(\sigma)$, which is

$$@_A[\mathrm{role}^i_A(\sigma)] \Rightarrow @_B[\mathsf{P}_\circ\ \mathit{send}(\sigma(M), A)], \text{ if } B \text{ is honest, and}$$

$$@_A[\mathrm{role}^i_A(\sigma)] \Rightarrow \bigvee_{C \in \mathit{Intr}} @_C[\mathsf{P}_\circ\ \mathit{send}(\sigma(M), A)], \text{ if } B \text{ is dishonest,}$$

assuming that the protocol step_q requires that a_j sends the message M to a_i. Note that if we consider only one dishonest principal, as is usual, this distinction vanishes, but our formalization is more general and we will make use of this generality later (see Proposition 5.3 below). We should therefore require $\mathrm{auth}^{i,j,q}_{A,B}(\sigma)$ to hold whenever step q is considered essential for authentication. As before, we sometimes write $\mathrm{auth}^{i,j,q}_{A,B}(\sigma(\bar{a}), \sigma(\bar{n}))$ instead of $\mathrm{auth}^{i,j,q}_{A,B}(\sigma)$.

In the case of the NSPK protocol, assuming for the moment that only one dishonest principal exists, we could specify for honest A acting as initiator, the authentication of the responder at step 2 using $\mathrm{auth}^{\mathit{Init},\mathit{Resp},2}_{A,B}(A, B', N_1, N_2)$:

$$@_A[\mathrm{role}^{\mathit{Init}}_A(A, B', N_1, N_2)] \Rightarrow @_B[\mathsf{P}_\circ\, \mathit{send}(\{N_1; N_2\}_{K_A}, A)]\,,$$

and for honest B acting as responder, the authentication of the initiator at step 3 using $\mathrm{auth}^{\mathit{Resp},\mathit{Init},3}_{B,A}(A', B, N_1, N_2)$:

$$@_B[\mathrm{role}^{\mathit{Resp}}_B(A', B, N_1, N_2)] \Rightarrow @_A[\mathsf{P}_\circ\, \mathit{send}(\{N_2\}_{K_B}, B)]\,.$$

This last property fails in the man-in-the-middle attack on NSPK [10], as we show in [3,4].

5 Metalevel analysis of the model

Our protocol analysis framework is based on a logic that is not specifically tailored to security protocols, and we are thus not bound to any assumptions about the underlying protocol model. Rather, we can use our logic to specify and reason about different assumptions, proving different metalevel properties of security protocol models, as well as the correctness of different model simplification techniques, within one and the same formalism in a uniform way. We develop our proofs in the context of the general network model we have defined above, with explicit asynchronous communication through the channel, and where intruders are modeled as agents within the system.

In this section, we give three substantial examples of formally reasoning about simplification techniques for protocol models. These examples are interesting in their own right. Moreover, they also illustrate how our approach

can help to clarify a number of underlying concepts that are often left implicit, or neglected, when considering such simplifications within other approaches.

5.1 Secret data

The following lemma is an example of the kind of metalevel property that any suitable network model should enjoy. Let $S \subseteq Msg$ be a set of *secret* atomic messages (names, nonces, and keys), and denote by Msg_S the set of *S-secure messages*, i.e. all messages where items from S can only appear if under the scope of an encryption with a key whose inverse is also in S. It should be clear that Msg_S contains precisely the messages that can be securely circulated in the network without danger of compromising any of the secrets in S. Indeed, $synth(analz(Msg_S)) = Msg_S$ and $Msg_S \cap S = \emptyset$.

More specifically, the following lemma states that under the assumption that no principal in G will ever send an S-insecure message and that all the nonces in S are freshly generated among the principals in G, if at some point the S-insecure data is unknown outside of G, then it will forever remain so.

Lemma 5.1 (Secret Data) *Assume that $G \subseteq Princ$ is a group of principals, μ is a network model such that $\mu \Vdash \bigwedge_{A\in G} @_A[\neg\, send(M, C')]$ for every $M \notin Msg_S$ and every name C', and $\mu \Vdash \bigvee_{A\in G} @_A[* \Rightarrow \mathsf{F}\, nonce(N)]$ for every nonce $N \in S$. If*

$$\mu, \xi \Vdash \bigwedge_{B\in Princ\setminus G} @_B[\neg\, knows(M)] \text{ for every } M \notin Msg_S,$$

then also

$$\mu, \xi \Vdash \bigwedge_{B\in Princ\setminus G} @_B[\mathsf{G}_\circ\, \neg\, knows(M)] \text{ for every } M \notin Msg_S.$$

Proof. By induction on configurations $\xi' \supseteq \xi$. Assuming the base case, as given, it suffices to prove that, given $\xi' \supseteq \xi$, if $\mu, \xi' \Vdash @_B[\neg\, knows(M)]$ for every $M \notin Msg_S$ and every principal $B \notin G$, and $\xi' \cup \{e\} \in \Xi$, then also $\mu, \xi' \cup \{e\} \Vdash @_B[\neg\, knows(M)]$ for every $M \notin Msg_S$ and $B \notin G$.

Suppose, by absurdity, that $\mu, \xi' \cup \{e\} \Vdash @_B[knows(M)]$ for some $M \notin \mathcal{M}_\mathcal{S}$ and $B \notin G$. Then it must be the case that $e \in Ev_B$ and so the local configuration of all other principals does not change. Moreover, $\alpha_B(e)$ cannot be a sending action since this would not change the local state of B. If $\alpha_B(e)$ is either $rec(M'')$ or $spy(M'')$ then it must be the case that $M'' \notin Msg_S$, but, since it had to have been previously sent to the channel, this is impossible. Indeed, by assumption, principals in G never send such messages, and no other principal could have sent it before. Hence, it must be a $nonce(N)$ action for some $N \in S$. But this contradicts the fresh nonce axioms because, by assumption, N is generated by some principal in G. Thus, $\mu, \xi' \Vdash @_B[\neg\, knows(M)]$ for every $M \notin \mathcal{M}_\mathcal{S}$, $B \notin G$ and $\xi' \supseteq \xi$, and the result follows. □

Note that the set $Msg \setminus Msg_S$ of S-insecure messages forms precisely what has been called an *ideal* in the context of strand spaces [17], whereas the set Msg_S of S-secure messages is the corresponding *coideal*, in the terminology

of [7,12]. In fact, Lemma 5.1 is a general result about the flow of data in the network, which is independent of protocols. The result can of course be used to reason about secrecy properties in protocol models, providing a result that is very similar to those found in [7,12]. Indeed, under reasonable conditions, the secrecy of generated nonces can be easily seen to hold.

Proposition 5.2 (Secrecy) *A given protocol guarantees* $secr_{\sigma(F)}(\sigma)$ *for an instantiation* σ *with only honest participants* $\sigma(a_1) = A_1$, ..., $\sigma(a_j) = A_j$, *provided that all the messages ever sent by* $A_1, \ldots, A_j$ *in any protocol run are* $(\{K_{A_1}^{-1}, \ldots, K_{A_j}^{-1}\} \cup \sigma(F))$*-secure.*

Proof. The result follows by an application of the Secret Data Lemma 5.1, using $G = \{A_1, \ldots, A_j\}$ and $S = \{K_{A_1}^{-1}, \ldots, K_{A_j}^{-1}\} \cup \sigma(F)$. Let μ be a protocol model, ξ a global configuration, and assume that $\mu, \xi \Vdash \bigwedge_{i=1}^{j} @_{A_i}[\mathsf{P}_\circ\ \mathrm{role}_{A_i}^{i}(\sigma)]$. The assumption that $A_1, \ldots, A_j$ will only send S-secure messages is the first precondition for the application of the lemma. The second precondition of the lemma follows immediately from the fact that all the corresponding roles of the protocol have been completed and therefore all the nonces in S are generated in μ among the principals in G.

Take the initial configuration $\emptyset$. Clearly, no principal outside G initially knows S-insecure messages. For the nonces it is trivial as they are generated in the model; for the keys it follows from the axioms **(Key1)** and **(Key2)**. By the lemma, we conclude that $\mu, \emptyset \Vdash @_B[\mathsf{G}_\circ \neg\, knows(M)]$ for every $B \in Princ \setminus G$ and $M \notin Msg_S$. In particular, we have that $\mu, \xi \Vdash \bigwedge_{B \in Princ \setminus \{A_1, \ldots, A_j\}} \bigwedge_{N \in \sigma(F)} @_B[\neg\, knows(N)]$. □

Note that our assumption here that all the messages ever sent by $A_1, \ldots, A_j$ in any protocol run are $(\{K_{A_1}^{-1}, \ldots, K_{A_j}^{-1}\} \cup \sigma(F))$-secure is essentially equivalent to *discreetness* in the terminology of [7,12].

5.2 One intruder is enough

In the following, we distinguish between *one-intruder* and *many-intruder* protocol signatures and models, depending on whether *Intr* is a singleton or not. Indeed, most approaches to protocol analysis only consider one-intruder models. Below, we show that this simplification is adequate. We achieve this by postulating a unique intruder Z who controls the activity of all dishonest principals, by making Z inherit the initial knowledge of all of them and perform, in some compatible sequential order, all the actions each of them performed. Of course, this transformation should be transparent to honest agents.

Proposition 5.3 (One intruder is enough) *The restricted class of one-intruder models is fully representative in the following sense: any attack on a protocol in a many-intruder model can be mapped to a corresponding attack in a one-intruder model.*

Proof. Let $cp\Sigma = \langle Hon, Intr, Name\rangle$ be a many-intruder signature and assume that an attack on the security goal γ happens at configuration ξ of $\mu = \langle \lambda, \alpha, \pi\rangle$. Consider the one-intruder signature $cp\Sigma' = \langle Hon, \{Z\}, Name'\rangle$, with $Name'_Z = \bigcup_{A\in Intr} Name_A$, and build $\mu' = \langle \lambda', \alpha', \pi'\rangle$ as follows: $\mu'_A = \mu_A$ for every $A \in Hon$; $\mu'_{Ch} = \mu_{Ch}$; and $\lambda'_Z = \langle Ev_Z, \rightarrow_Z\rangle$ where $Ev_Z = \bigcup_{A\in Intr} Ev_A$ and $\rightarrow_Z$ is the successor relation associated to some discrete linearization $\langle Ev_Z, \rightarrow^*_Z\rangle$ of $\langle Ev_Z, \rightarrow^*\rangle$ that has $\bigcup_{A\in Intr} \xi|_A$ as a local configuration, $\alpha'_Z(e) = \alpha_A(e)$, where $A \in Intr$ is the unique principal such that $e \in Ev_A$, and $\pi'_Z(\emptyset) = \bigcup_{A\in Intr} \pi_A(\emptyset)$. It is straightforward to check that μ' is a one-intruder model of the protocol and ξ is still a configuration. We now show that:

(i) $\mu, \xi \Vdash @_A[\varphi]$ iff $\mu', \xi \Vdash @_A[\varphi]$, for every $A \in Hon$ and every $\varphi \in \mathcal{L}_A$ that does not include communication subformulas; and
(ii) $\mu, \xi \Vdash \bigvee_{A\in Intr} @_A[\mathsf{P}_\circ\ act]$ iff $\mu', \xi \Vdash @_Z[\mathsf{P}_\circ\ act]$, for every action act.

Property (i) is an immediate consequence of the fact that $\mu'_A = \mu_A$ for every $A \in Hon$, if we note that, by definition, the satisfaction of a local formula without communication subformulas only depends on the local life-cycle. Property (ii) follows directly from the fact that $Ev_Z = \bigcup_{A\in Intr} Ev_A$, and for each $e \in Ev_Z$, $\alpha'_Z(e) = \alpha_A(e)$ where $A \in Intr$ is the unique principal such that $e \in Ev_A$. Clearly, these two properties imply that if γ is an authentication property then the attack must also appear at ξ in μ'. Indeed, if $\gamma \equiv \mathrm{auth}^{i,j,q}_{A,B}(\sigma)$ with honest A, then it follows from (i) that the antecedent $@_A[\mathrm{role}^i_A(\sigma)]$ of the main implication in γ still holds at μ' and ξ since $\mathrm{role}^i_A(\sigma) \in \mathcal{L}_A$ does not include communication subformulas. As for the consequent, if B is honest then $@_B[\mathsf{P}_\circ\ send(\sigma(M), A)]$ must also fail at μ' and ξ, again by using (i), given that $\mathsf{P}_\circ\ send(\sigma(M), A) \in \mathcal{L}_B$ does not have communication subformulas. If B is dishonest then the failure of $\bigvee_{C\in Intr} @_C[\mathsf{P}_\circ\ send(\sigma(M), A)]$ at μ and ξ implies the failure of $@_Z[\mathsf{P}_\circ\ send(\sigma(M), A)]$ at μ' and ξ, according to (ii). Note that

(iii) if $\mu, \xi \Vdash \bigvee_{A\in Intr} @_A[knows(M)]$ then $\mu', \xi \Vdash @_Z[knows(M)]$,

follows easily from (ii) and condition **(K)**. Hence, an attack on a secrecy property can be shown to appear at ξ in μ'. If $\gamma \equiv \mathrm{secr}_{\sigma(F)}(\sigma)$ with all roles played by honest principals then the antecedent $\bigwedge^j_{i=1} @_{A_i}[\mathsf{P}_\circ\ \mathrm{role}^i_{A_i}(\sigma)]$ of the main implication in γ still holds at μ' and ξ, according to (i). To show that the consequent $\bigwedge_{B\in Princ\setminus\{A_1,\ldots,A_j\}} \bigwedge_{N\in\sigma(F)} @_B[\neg\, knows(N)]$ also fails at μ' and ξ it now suffices to use either (i), for honest B, or (iii) for dishonest B. □

Fig. 5 provides a visual example of this transformation. Note that in this case we chose a linearization where *nonce*1 happened before *nonce*2, but any other possibility would be fine, as long as the initial causal restrictions are met, namely, *nonce*1 must occur after *spy*1, and *nonce*2 must precede the occurrence of *rec*. Note that the other initial causal restrictions, such as the fact that *send* must precede *spy*1, are automatically met since the channel is preserved.

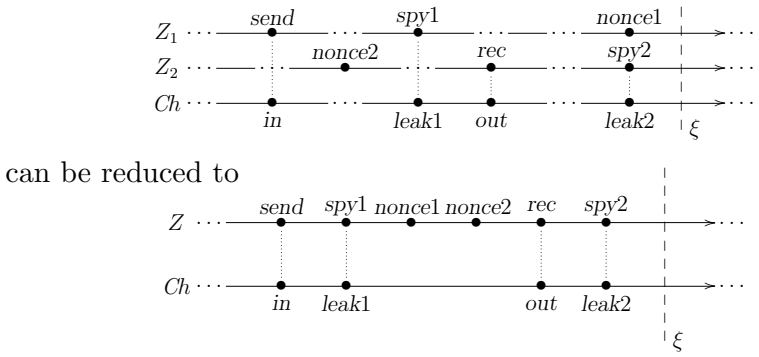

Fig. 5. The one-intruder reduction.

The one-intruder reduction is an intuitive and widely used simplification, but its proof can be enlightening. In fact, not only can the disjunctive view of the many dishonest principals be seen as a kind of "group intruder", but the translation also caters for the one intruder as controlling all of them, for which our characterization of authentication in the presence of many intruders was essential. The result is similar to part of the one obtained in [6]. There, however, the intruder was modeled as an abstract entity, with an obvious counterpart on the way security properties were expressed. Our result takes this same view from inside the system since we model the intruder as a concrete entity, namely one (or several) of the principals.

5.3 The predatory intruder

Among the possible one-intruder models of a given protocol, many will feature a rather passive intruder. Any attack that can happen under these circumstances should certainly also be achievable by a more effective intruder. In the following, we show that we can restrict attention to models where the intruder Z is relentlessly and effectively committed to his task, namely:

- he spies every message sent by an honest agent immediately after it arrives to the channel, and that is all the spying he does:

$$@_{Ch}[@_Z[spy(M)] \Leftrightarrow \mathsf{Y} \bigvee_{A \in Hon} @_A[\bigvee_{B' \in Name} send(M, B')]] \,;$$

- he never bothers receiving messages (he has already spied them):

$$@_Z[\neg\, rec(M)] \,;$$

- he only sends messages to honest agents, and he manages to send every message just immediately before the honest agent gets it:

$$@_Z[\neg\, send(M, Z')] \quad \text{and} \quad @_Z[send(M, A) \Rightarrow @_{Ch}[\mathsf{X}\ @_A[rec(M)]]] \,.$$

We call any one-intruder model fulfilling these requirements a *predatory intruder* protocol model. To show that this restriction is adequate, we need to be

able to transform every one-intruder model into an attack-preserving predatory intruder model. The transformation amounts to purging all the (possibly erratic) *old* interactions of the intruder, and introducing *new* timely interactions according to the predatory intruder requirements, while not changing anything from the point of view of honest agents.

Proposition 5.4 (The predatory intruder) *The restricted class of predatory intruder models is fully representative in the following sense: any attack on a protocol in a one-intruder model can be mapped to a corresponding attack in a predatory intruder model.*

Proof. Let $cp\Sigma = \langle \mathit{Hon}, \{Z\}, \mathit{Name}\rangle$ be a one-intruder protocol signature and assume that an attack on the security goal γ happens at ξ of $\mu = \langle \lambda, \alpha, \pi\rangle$. Consider the sets $\mathrm{Old} = Ev_Z \cap Ev_{Ch}$, $\mathrm{Succ} = \bigcup_{A\in \mathit{Hon}}\{e \in Ev_A \mid \alpha_A(e) = \mathit{send}(\_)\}$, $\mathrm{Orig}(M, B') = \{e \in Ev_{Ch} \mid \alpha_{Ch}(e) = \mathit{in}(M, B')\}$ with $M \in \mathit{Msg}$ and $B' \in \mathit{Name}$, $\mathrm{Pred} = \bigcup_{A\in \mathit{Hon}}\{e \in Ev_A \mid \alpha_A(e) = \mathit{rec}(M),\ \{e' \in \mathrm{Orig}(M, A) \mid e' \rightarrow^+_{Ch} e\} \subseteq \mathrm{Old}\}$, and $\mathrm{New} = \{s(e) \mid e \in \mathrm{Succ}\} \cup \{p(e) \mid e \in \mathrm{Pred}\}$. Define the model $\mu' = \langle \lambda', \alpha', \pi'\rangle$ as follows:

- $\mu'_A = \mu_A$ for every $A \in \mathit{Hon}$;
- $\lambda'_{Ch} = \langle Ev'_{Ch}, \rightarrow'_{Ch}\rangle$ with $Ev'_{Ch} = (Ev_{Ch} \setminus \mathrm{Old}) \cup \mathrm{New}$ and $\rightarrow'_{Ch}$ the successor relation obtained from $\rightarrow^*_{Ch}$ by letting $e \rightarrow'_{Ch} s(e)$ for every $e \in \mathrm{Succ}$, and $p(e) \rightarrow'_{Ch} e$ for every $e \in \mathrm{Pred}$, $\alpha'_{Ch}(e) = \alpha_{Ch}(e)$ for $e \in Ev_{Ch} \setminus \mathrm{Old}$, $\alpha'_{Ch}(s(e)) = \mathit{leak}$ and $\alpha'_{Ch}(p(e)) = \mathit{in}(M, A)$ if $\alpha_{Ch}(e) = \mathit{out}(M, A)$;
- $\lambda'_Z = \langle Ev'_Z, \rightarrow'_Z\rangle$ with $Ev'_Z = (Ev_Z \setminus \mathrm{Old}) \cup \mathrm{New}$ and $\rightarrow'_Z$ any successor relation compatible with $\rightarrow'_{Ch}$ on New that guarantees that every $e \in Ev_Z \setminus \mathrm{Old}$ with $\alpha_Z(e) = \mathit{nonce}(N)$ precedes any $p(e)$ with $\alpha_{Ch}(e) = \mathit{out}(M, A)$ and N occurring in M, $\alpha'_Z(e) = \alpha_Z(e)$ for $e \in Ev_Z \setminus \mathrm{Old}$, $\alpha'_Z(s(e)) = \mathit{spy}(M)$ if $\alpha_{Ch}(e) = \mathit{in}(M, B')$ and $\alpha'_Z(p(e)) = \mathit{send}(M, A)$ if $\alpha_{Ch}(e) = \mathit{out}(M, A)$, and $\pi'_Z(\emptyset) = \pi_Z(\emptyset)$.

It is straightforward to check that μ' is a predatory intruder model of the protocol. Take now the global configuration $\xi' \in \Xi'$ such that $\xi'|_A = \xi|_A$ for every $A \in \mathit{Hon}$, $\xi'|_{Ch} \setminus \mathrm{New} = \xi|_{Ch} \setminus \mathrm{Old}$ and $\xi'|_Z \setminus \mathrm{New} = \xi|_Z \setminus \mathrm{Old}$, plus $\xi' \cap \mathrm{New} = \{p(e) \mid e \in \xi \cap \mathrm{Pred}\} \cup \{s(e) \mid e \in \xi \cap \mathrm{Succ}\}$. We now show that:

(i) $\mu, \xi \Vdash @_A[\varphi]$ iff $\mu', \xi' \Vdash @_A[\varphi]$, for every $A \in \mathit{Hon}$ and every $\varphi \in \mathcal{L}_A$ that does not include communication subformulas; and

(ii) $\mu, \xi \Vdash @_Z[\mathsf{P}_\circ\, \mathit{send}(M, A)]$ if $\mu', \xi' \Vdash @_Z[\mathsf{P}_\circ\, \mathit{send}(M, A)]$, for every $A \in \mathit{Hon}$ and message M.

Property (i) follows from the fact that $\mu'_A = \mu_A$ for every $A \in \mathit{Hon}$. Property (ii) results from the fact that the only *send* actions of the predatory Z are on $p(e)$ events. Therefore, it must be the case that $e \in \xi \cap \mathrm{Pred}$ is an *out* event preceded by a corresponding *in* event $e' \in \xi$. Clearly, e' is an origination event for the message and so, by definition of Pred, $e' \in \mathrm{Old}$. Therefore $e' \in Ev_Z$ and $\alpha_Z(e')$ is the *send* action we were looking for. Using the two, we can show

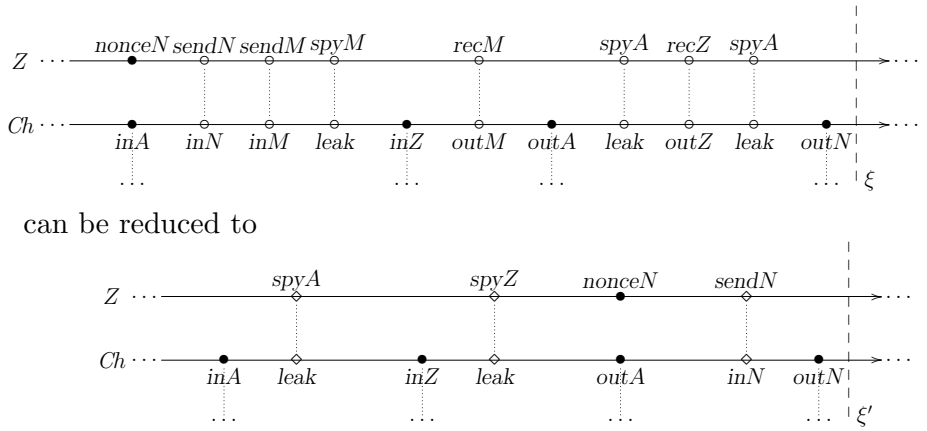

Fig. 6. The predatory intruder reduction.

that if γ is an authentication property then the attack also appears at ξ' in μ'. If we also prove:

(iii) if $\mu, \xi \Vdash @_Z[knows(M)]$ then $\mu', \xi' \Vdash @_Z[knows(M)]$,

then an attack can be shown to appear at ξ' in μ' also in the case that γ is a secrecy goal. Property (iii) follows from the facts that the initial knowledge of the predatory intruder is exactly the same, he does exactly the same *nonce* actions, and he spies every message at least as early as the original intruder received or spied it. □

Fig. 6 provides a visual example of the transformation. The *old* events are represented by ∘ in the first model, whereas the *new* events are represented by ⋄ in the second model. Clearly both *inA* and *inZ* are *successor* events, but only *outN* is a *predecessor* event since *outA* is preceded by *inA*, coming from an honest principal. Note also that *nonceN* could be ordered in other ways, but always before *sendN*.

The predatory intruder reduction is a first step towards formally justifying the linearization of distributed communication that underlies the inductive trace models of protocols (see [14] and also [2], for example). The remaining step, which we do not include here for brevity, involves abstracting away the communication channel by "replacing" it with the intruder: this amounts to identifying the two lines *Z* and *Ch* in the bottom half of Fig. 6. A nice side-effect of the predatory intruder reduction is that the sending actions of the intruder can really be bound by the possible shapes of messages that honest principals can receive in protocol roles, as is commonly assumed.

Corollary 5.5 *The restricted class of one-intruder models where the intruder only ever sends messages according to the protocol description is fully representative.*

6 Discussion

Communication and distribution are the essential ingredients of protocols. These are the main concepts underlying our logic. Through the choice of different signatures and axioms, we can define theories for formalizing and reasoning about different application domains, as shown here in the case of security protocol models and their properties. It is worth noting in this regard that many of the problems with security protocols arise out of problems with communication, as opposed to problems with the underlying cryptographic algorithms (which are abstracted away with the black-box, perfect cryptography approach). While some of the results presented here, mutatis mutandis, have already been shown using other formalisms, our logic provides a means for proving them in a general and uniform way, which provides a basis for further general investigations.

Event structures, which are the underlying models of our logic, are comparable with strand spaces, as explained in [8]. A comparison to other formalisms for reasoning about communicating processes can be found in [9]. It is worth emphasizing some of the advantages of our approach. To begin with, our approach provides not just a language for describing models (as in process algebras, for example) but also a logic for reasoning about them. Reasoning about local temporal properties of agents in a distributed system could also be performed in a linear temporal logic over linearizations of the distributed models. However, this would come at the price of readability and simplicity, as the locality dimension is lost. In contrast, our distributed logic is simple and robust in the sense that formulas are invariant with respect to different linearizations. We have taken advantage of this in the proofs of both the one-intruder and the predatory intruder reductions.

We have begun applying our logic to other metatheoretical investigations, such as developing appropriate partial-order techniques that may reduce the (potentially infinite) state-space exploration involved in model-checking protocol properties (cf. [2]). This is work in progress and the first results are promising. Further work is the application of our logic for reasoning about protocol composition, as well as the development of a calculus for the logic.

References

[1] A. Armando and L. Compagna. Abstraction-driven SAT-based Analysis of Security Protocols. In *Proc. SAT 2003*, LNCS 2919. Springer-Verlag, 2003. Available at http://www.avispa-project.org.

[2] D. Basin, S. Mödersheim, and L. Viganò. An On-The-Fly Model-Checker for Security Protocol Analysis. In *Proc. ESORICS'03*, LNCS 2808. Springer-Verlag, 2003. Available at http://www.avispa-project.org.

[3] C. Caleiro, L. Viganò, and D. Basin. Distributed Temporal Logic for Security

Protocol Analysis. In preparation, 2004.

[4] C. Caleiro, L. Viganò, and D. Basin. Towards a Metalogic for Security Protocol Analysis (extended abstract). In *Proc. Comblog'04*, 2004.

[5] Y. Chevalier and L. Vigneron. Automated Unbounded Verification of Security Protocols. In *Proc. CAV'02*, LNCS 2404. Springer-Verlag, 2002. Available at http://www.avispa-project.org.

[6] H. Comon-Lundh and V. Cortier. Security properties: two agents are sufficient. In *Proc. ESOP'2003*, LNCS 2618. Springer-Verlag, 2003.

[7] V. Cortier, J. Millen, and H. Rueß. Proving secrecy is easy enough. In *Proc. CSFW'01*. IEEE Computer Society, 2001.

[8] F. Crazzolara and G. Winskel. Events in security protocols. In *Proc. of CCS'01*. ACM Press, 2001.

[9] H.-D. Ehrich and C. Caleiro. Specifying communication in distributed information systems. *Acta Informatica*, 36:591–616, 2000.

[10] G. Lowe. Breaking and Fixing the Needham-Shroeder Public-Key Protocol Using FDR. In *Proc. TACAS'96*, LNCS 1055. Springer-Verlag, 1996.

[11] G. Lowe. A hierarchy of authentication specifications. In *Proc. CSFW'97*. IEEE Computer Society Press, 1997.

[12] J. Millen and H. Rueß. Protocol-independent secrecy. In *Proc. 2000 IEEE Symposium on Security and Privacy*. IEEE Computer Society, 2000.

[13] J. Millen and V. Shmatikov. Constraint solving for bounded-process cryptographic protocol analysis. In *Proc. CCS'01*. ACM Press, 2001.

[14] L. Paulson. The inductive approach to verifying cryptographic protocols. *Journal of Computer Security*, 6:85–128, 1998.

[15] P. Ryan, S. Schneider, M. Goldsmith, G. Lowe, and B. Roscoe. *Modelling and Analysis of Security Protocols*. Addison Wesley, 2000.

[16] D. Song, S. Berezin, and A. Perrig. Athena: a novel approach to efficient automatic security protocol analysis. *Journal of Computer Security*, 9:47–74, 2001.

[17] F. J. Thayer Fábrega, J. C. Herzog, and J. D. Guttman. Honest ideals on strand spaces. In *Proc. CSFW'98*. IEEE Computer Society, 1998.

[18] F. J. Thayer Fábrega, J. C. Herzog, and J. D. Guttman. Strand spaces: Proving security protocols correct. *Journal of Computer Security*, 7:191–230, 1999.

[19] G. Winskel. Event structures. In *Petri Nets: Applications and Relationships to Other Models of Concurrency*, LNCS 255. Springer-Verlag, 1987.

An Optimized Intruder Model for SAT-based Model-Checking of Security Protocols

Alessandro Armando [1] Luca Compagna [2]

AI-Lab, DIST – Università degli Studi di Genova,
Viale Causa 13, 16145 Genova, Italy

Abstract

In previous work we showed that automatic SAT-based model-checking techniques based on a reduction of protocol (in)security problems to a sequence of propositional satisfiability problems can be used to effectively find attacks on protocols. In this paper we present an optimized intruder model that may lead in many cases to shorter attacks which can be detected in our framework by generating smaller propositional formulae. The key idea is to assume that some of the abilities of the intruder have instantaneous effect, whereas in the previously adopted approach all the abilities of the intruder were modeled as state transitions. This required non trivial extensions to the SAT-reduction techniques which are formally described in the paper. Experimental results indicate the advantages of the proposed optimization.

Key words: Security Protocols, Bounded Model Checking, SAT, Rewriting.

1 Introduction

In the last decade we have witnessed a dramatic speed-up of SAT solvers: problems with thousands of variables are now solved routinely in milliseconds by state-of-the-art SAT solvers. This has led to breakthroughs in important areas such as planning and model-checking. Motivated by these results, in [1,2,3] we proposed reductions of protocol (in)security problems to satisfiability problems in propositional logic that can be used to effectively find attacks on protocols. We have developed a model-checker, SATMC, based on these ideas and experimental results obtained by running SATMC against security protocols drawn from the Clark-Jacob's library [9] confirm the effectiveness of the approach.

[1] Email: armando@dist.unige.it
[2] Email: compa@dist.unige.it

This is a preliminary version. The final version will be published in Electronic Notes in Theoretical Computer Science
URL: www.elsevier.nl/locate/entcs

In this paper we propose an optimized intruder model that leads in many cases to shorter attacks which can be detected in our framework by generating smaller propositional formulae. The key idea is to assume that some of the abilities of the intruder have instantaneous effect as opposed to the previously adopted approach in which all the abilities were modeled as state transitions. This required non trivial extensions to the SAT-reduction techniques which are formally described in the paper. We have implemented the proposed optimization within SATMC. Experimental results on protocols drawn from the Clark-Jacob's library clearly indicate the advantages of the proposed optimization.

Outline of the paper. We start in Section 2 by presenting, via a well-known authentication protocol, our model with particular emphasis to an optimized intruder model based on the concept of axiom. In Section 3 we define the notion of protocol insecurity problem with axioms. Section 4 is devoted to the formal description of the automatic compilation of protocol insecurity problems with axioms into a set of propositional satisfiability problems. Experimental results and some implementation detail are discussed in Section 5. We conclude in Section 6 with some final remarks and a discussion of the future work.

2 Modeling Security Protocols

Although the general verification problem—to prove that the security protocol satisfies the security guarantees for which it has been designed in a scenario with an arbitrary number of parallel sessions—is undecidable [12,13], for many protocols, verification can be reduced to verification of a bounded number of sessions. Moreover, even for those protocols that should be checked under a unbounded number of concurrent protocol executions, violations in their security requirements often exploit only a small number of parallel sessions. For these reasons, in many case of interest it is sufficient to consider a finite and small number of parallel sessions. For instance, all known attacks on the protocols in the Clark-Jacob's library involve at most two concurrent sessions.

We model the concurrent execution of a security protocol by means of a state transition system specified in the IF [4] declarative language based on multi-set rewriting which is particularly amenable to formal analysis [7,15]. States are represented by sets of atomic formulae of a sorted first order language called *facts* and transitions by means of *labeled rewrite rules* over sets of facts. In our setting, we assume that the network is controlled by means of the very general Dolev-Yao intruder [11]. [3] In this model the intruder has the ability to eavesdrop and divert messages as well as that to compose, decompose, encrypt, and decrypt messages (provided the encryption keys are known). Fi-

[3] It is worth pointing out that the model is not bound to the Dolev-Yao intruder, but, on the contrary, it is expressive enough to get generic intruder models suited for a variety of communication networks such as secure channels, wireless channels, etc.

nally, he can send those messages to other participants with a false identity. Besides this, we make the standard assumptions of *perfect cryptography* i.e. an encrypted message can be decrypted only by using the appropriate decryption key and of *strong typing* i.e. agents only accept type-correct messages and therefore type confusion is not allowed. [4]

In order to clarify our presentation, let us consider the example of the Needham-Schroeder Public-Key (NSPK) authentication protocol [16]. In the common Alice&Bob notation, the NSPK protocol looks like follow:

$$\begin{array}{llcll} (1) & A & \rightarrow & B & : \quad \{A, Na\}_{Kb} \\ (2) & B & \rightarrow & A & : \quad \{Na, Nb\}_{Ka} \\ (3) & A & \rightarrow & B & : \quad \{Nb\}_{Kb} \end{array}$$

where A and B are the roles involved in the protocol; Ka and Kb are the public keys of A and B, respectively; and Na and Nb are nonces [5] generated, respectively, by A and B. Notice that, the above high level protocol specification describes a kind of template $NSPK(A, B, Ka, Kb, Na, Nb)$ parametrized by some free variables [6] appearing in it. A ground instance of the security protocol template represents a session of the protocol. Successful execution of the NSPK protocol should convince both A and B that they have been talking to each other. The rationale is that only B and A could have formed the appropriate response to the message issued in (1) and in (2), respectively. In fact, a malicious agent I can deceit *bob* (an instance of B) into believing that he is talking with *alice* (instance of A) whereas he is talking with I. This is achieved by executing concurrently two sessions $NSPK(alice, I, ka, ki, na, ni)$ and $NSPK(alice, bob, ka, kb, na2, nb)$ of the protocol and using messages from one session to form messages in the other as illustrated by the following protocol trace:

$$\begin{array}{llcll} (1.1) & alice & \rightarrow & I & : \quad \{alice, na\}_{ki} \\ (2.1) & I(alice) & \rightarrow & bob & : \quad \{alice, na\}_{kb} \\ (2.2) & bob & \rightarrow & I(alice) & : \quad \{na, nb\}_{ka} \\ (1.2) & I & \rightarrow & alice & : \quad \{na, nb\}_{ka} \\ (1.3) & alice & \rightarrow & I & : \quad \{nb\}_{ki} \\ (2.3) & I(alice) & \rightarrow & bob & : \quad \{nb\}_{kb} \end{array}$$

[4] As pointed out in [14], in many case of interest, type-flaw attacks can be prevented by tagging the fields of a message with information indicating its intended type.

[5] *Nonces* are numbers randomly generated by principals and they are intended to be used *only once*.

[6] Free variables are indicated by means of capital letters excepted I that is used to indicate the malicious agent.

where $I(alice)$ indicates the intruder pretending to be *alice*. At the end of the above trace *bob* believes he has been talking with *alice*, but this is obviously not the case.

Let us describe in our setting the state transition system and the security requirement of the above example.

Facts. The facts useful to describe the NSPK protocol are the following:

- $ik(T)$, meaning that the intruder knows the message T;
- $fresh(N)$, meaning that the nonce N has not been used yet;
- $m(J, S, R, T)$, meaning that principal S has (supposedly) [7] sent message T to principal R at protocol step J; and
- $w(J, S, R, [T_1, \ldots, T_k], C)$, representing the state of execution of principal R at step J of session C; in particular it means that R knows the messages $T_1, \ldots, T_k$ at step J of session C, and—if $J \neq 0$—also that a message from S to R is awaited for step J of session C to be executed.

Initial State. The initial state of the system representing two concurrent sessions of the NSPK is: [8]

$$w(0, a, a, [a, i, ka, ka^{-1}, ki], 1) \tag{1}$$
$$\centerdot\, w(0, a, a, [a, b, ka, ka^{-1}, kb], 2) \centerdot w(1, b, a, [b, a, kb, kb^{-1}, ka], 2) \tag{2}$$
$$\centerdot\, fresh(nc(n1, 1)) \tag{3}$$
$$\centerdot\, fresh(nc(n1, 2)) \centerdot fresh(nc(n2, 2)) \tag{4}$$
$$\centerdot\, ik(i) \centerdot ik(a) \centerdot ik(b) \centerdot ik(ki) \centerdot ik(ki^{-1}) \centerdot ik(ka) \centerdot ik(kb) \tag{5}$$

The fact (1) represents the initial state of the honest agent a that plays the role of initiator in session 1 and knows at the beginning her identity, the identity of intruder (the agent she would like to talk with), her public and private keys, [9] and the intruder public key. Facts (2) represent the initial state of the honest agents a and b involved as initiator and responder, respectively, in session 2. Facts (3) and (4) state the initial freshness of the nonces $nc(n1, 1)$, $nc(n1, 2)$, and $nc(n2, 2)$. Notice that, since we bound the number of parallel sessions also the number of fresh terms to be taken into account in the protocol analyses can be computed and bounded in advance. Facts (5) represent the information initially known by the intruder (commonly, in a asymmetric cryptosystem, its identity, its public and private keys as well as the identities of honest agents

[7] As we will see, since the intruder may fake other principals' identity, the message might have been sent by the intruder.

[8] To improve readability we use the "$\centerdot$" operator as set constructor. For instance, we write "$x \centerdot y \centerdot z$" to denote the set $\{x, y, z\}$.

[9] Notice that with ka^{-1} we indicate the inverse of the public key ka.

and their public keys). Notice that, according to the standard close world assumption all the facts that do not occur in the set representing the initial state are considered false.

Goal Predicates. A security protocol is intended to enjoy specific security properties. In our example this property is the ability of authenticating the responder with the initiator and vice versa. A security property can be specified by providing goal predicates representing a set of "bad" states. For instance, it is immediate to see that any state in which b believes to have completed a session with a, while a has not started this session with him, represents a violation of the expected authentication property between b and a. This set of bad states can be easily represented by the goal predicate $w(1, a, b, [b, a, kb, kb^{-1}, ka], s(2)) \wedge w(0, a, a, [a, b, ka, ka^{-1}, kb], 2)$. [10]

Labeled Rewrite Rules. As mentioned above, labeled rewrite rules over sets of facts are used to specify how the transition system evolves. In particular, we distinguish between rules modeling the behavior of honest agents (so called protocol rules) and rules representing the Dolev-Yao intruder (so called intruder rules). These rules will be described in the next sections. Concerning the notation used, let $M1$ and $M2$ be messages, then the terms $\{M1\}_{M2}$ and $\langle M1, M2\rangle$ represent the asymmetric encryption of $M1$ using $M2$ as asymmetric key and the pairing of the $M1$ and $M2$, [11] respectively.

2.1 Modeling the behavior of Honest Participants

Honest participants strictly behave according to the protocol. The following rewrite rule models the activity of sending the first message of the NSPK protocol:

$$fresh(nc(n1, S)) \centerdot w(0, A, A, [A, B, Ka, Ka^{-1}, Kb], S) \xrightarrow{step_0(A,B,Ka,Kb,S)} w(2, B, A, [nc(n1, S), A, B, Ka, Ka^{-1}, Kb], S) \centerdot m(1, A, B, \{A, nc(n1, S)\}_{Kb})$$

[10] Notice that with $s(C)$ we indicate the next execution of session C.

[11] To improve readability we write $M1, M2$ in place of $\langle M1, M2\rangle$ where the pairing message can be easily evinced from the context.

Notice that, the nonce $nc(n1, S)$ is added to the knowledge of A for subsequent use. The receipt of the message and the reply of the responder is modeled by:

$$\begin{aligned} &fresh(nc(n2,S)) \centerdot m(1, A, B, \{A, Na\}_{Kb}) \\ &\quad \centerdot w(1, A, B, [B, A, Kb, Kb^{-1}, Ka], S) \xrightarrow{step_1(A,B,Ka,Kb,Na,S)} \\ &\quad\quad w(3, A, B, [Na, nc(n2, S), B, A, Kb, Kb^{-1}, Ka], S) \\ &\quad\quad\quad \centerdot m(2, B, A, \{Na, nc(n2, S)\}_{Ka}) \end{aligned}$$

The state of the responder is updated by increasing the protocol step and by extending the knowledge with the acquired informations namely the nonce sent by the initiator and the nonce freshly generated by the responder itself. The third step of the protocol is modeled by:

$$\begin{aligned} &m(2, B, A, \{Na, Nb\}_{Ka}) \\ &\quad \centerdot w(2, B, A, [Na, A, B, Ka, Ka^{-1}, Kb], S) \xrightarrow{step_2(A,B,Ka,Kb,Na,Nb,S)} \\ &\quad\quad w(0, A, A, [A, B, Ka, Ka^{-1}, Kb], s(S)) \\ &\quad\quad\quad \centerdot m(3, A, B, \{Nb\}_{Kb}) \end{aligned}$$

Notice that, after the application of this rule, A completes her part in the session S and she is eventually ready to start an other session $s(S)$ of the protocol with B. The final step of the protocol is modeled by:

$$\begin{aligned} &m(3, A, B, \{Na\}_{Kb}) \\ &\quad \centerdot w(3, A, B, [Na, Nb, B, A, Kb, Kb^{-1}, Ka], S) \xrightarrow{step_3(A,B,Ka,Kb,Na,Nb,S)} \\ &\quad\quad w(1, A, B, [B, A, Kb, Kb^{-1}, Ka], s(S)) \end{aligned}$$

After the application of this rule also B finishes his part in the session S and he is eventually ready to communicate with A in an other session $s(S)$.

2.2 Modeling the behavior of the Intruder

Contrary to honest agents that execute faithfully each statement specified by the protocol, the Dolev-Yao intruder has many degrees of freedom. By observing all the traffic in the network, it extends its knowledge and from such a knowledge it can compose and send fraudulent messages to honest participants. However, many of these messages can be of no interest for the analysis of the protocol. In fact, as previously mentioned, we focus on reachability problems with a finite number of parallel sessions in which honest agents only react to a message if, and only if, this message has the type and the pattern expected by the receiver. The set of such "interesting" messages is finite. For instance, the following partially instantiated rule modeling the ability of the Dolev-Yao intruder of pairing messages

$$ik(a) \centerdot ik(M) \xrightarrow{pairing(a,M)} ik(a) \centerdot ik(M) \centerdot ik(\langle a, M\rangle))$$

can be applied in the initial state of the NSPK by substituting M with a and in every successor state extending the knowledge of the intruder with messages of the form $\langle a, \langle a, \langle a, \ldots \rangle \rangle \rangle$. However, such messages are not "interesting" for the analysis of our example and, therefore, also the rules applied for composing them are useless. As a matter of fact, *diverting*, *impersonating* and *decomposing* rules are sufficient to model the Dolev-Yao intruder and it is easy to see that this optimization is correct as it preserves the existing attacks and does not introduce new ones.

Diverting Rules. The following rule models the ability of the intruder of diverting the information exchanged by the honest participants:

$$m(I, A, B, M) \xrightarrow{divert(A,B,I,M)} ik(M)$$

It states that, if a message has been sent on the communication channel, then the intruder can read its content and remove it from the channel.

Impersonating Rules. The observation that most of the messages generated by a strictly compliant Dolev-Yao intruder are rejected by the receiver as non-expected or ill-formed suggests to restrict these rules so that the intruder sends only messages matching the patterns expected by the receiver. For each protocol rule of the form

$$\ldots \;\centerdot\; m(I, A, B, M) \;\centerdot\; w(I, A, B, Knw, S) \xrightarrow{step_I(\ldots)} \ldots$$

and for each possible set of messages $\{M_{1,k}, \ldots, M_{j_k,k}\}$ (let m be the number of such sets, then $k = 1, \ldots, m$ and $j_k > 0$) from which the Dolev-Yao intruder would be able to build a message M' that matches (and complies with the type of) M, we add a new rule of the form

$$\begin{aligned} \ldots \centerdot\, w(I, A, B, Knw, S) \centerdot\, i(A) \centerdot\, i(B) \centerdot\, i(M_{1,k}) \centerdot \ldots \centerdot\, i(M_{j_k,k}) \\ \xrightarrow{impersonate_{I,k}(\ldots)} \ldots \centerdot\, m(I, A, B, M') \centerdot\, w(I, A, B, Knw, S) \\ \centerdot\, i(A) \centerdot\, i(B) \centerdot\, i(M_{1,k}) \centerdot \ldots \centerdot\, i(M_{j_k,k}) \centerdot\, i(M') \end{aligned}$$

This rule states that if agent B is waiting for a message M from A and the intruder is able to compose a message M' matching (and complying with the type of) M, then the intruder can impersonate A and send M'. This optimization, first introduced in [15], often reduces the number of rule instances in a dramatic way (for more details see [1]).

It is worth pointing out that, by applying the *step compression* optimization proposed in [5], it is possible to merge the above intruder rules with protocol rules reducing dramatically the dimension of the search space to be explored. See [1] for more details on the implementation of this optimization in our setting.

Decomposing Rules. In order to be able to apply either the impersonating rules or the step compressed rules, the intruder must be provided with the smallest set of rules modeling its ability of decomposing every sub-message of messages exchanged in the parallel sessions of the protocol. For instance, the rules useful to decompose the first message of the NSPK are the following:

$$ik(\{M\}_K) \cdot ik(K^{-1}) \xrightarrow{decrypt(K,M)} ik(\{M\}_K) \cdot ik(K^{-1}) \cdot ik(M)$$
$$ik(\langle M1, M2\rangle) \xrightarrow{decompose(M1,M2)} ik(M1) \cdot ik(M2)$$

The first rule states that if the intruder knows both a message encrypted with the key K and the decryption key K^{-1}, then the transition system can move in a state where the knowledge of the intruder is extended with the content M of the message. Similarly, the second rule states that if the intruder knows the pairing message $\langle M1, M2\rangle$, then the rule can be applied to reach a state in which the knowledge of the intruder is extended with the messages $M1$ and $M2$.

Especially for industrial-scale security protocols in which messages can have a complex structure, the number of decomposing rule instances to be applied in order to find out an attack can be significant. As we will see in Section 4, the size—in terms of atoms and clauses—of the generated propositional formulae increases linearly with the length of the attack. Since the complexity of the SAT problem grows exponentially according to the increasing of the atoms number, it is critical to reduce such a length. The optimization proposed in this paper is based on the concept of *axiom*. An axiom is a formula that states a relation between the facts of the transition system and that holds in each state of the transition system. [12] It turns out that axioms are particularly suited to represent relations between intruder knowledge facts. For instance, the axiom

$$ik(\{M\}_K) \wedge ik(K^{-1}) \supset ik(M)$$

states that, every time the intruder knows both a message encrypted with the key K and the decryption key K^{-1}, then it knows *instantaneously* also the content M of the message. The main difference between an axiom and a rule

[12] Notice that, if an axiom contains variables, then they are intended universally quantified.

is in the fact that the former must hold in every state of the transition system, while the latter is not forced to be executed even if all its preconditions are satisfied and it spends a transition for being executed. As a consequence by replacing decomposing rules with appropriate decomposing axioms we obtain a twofold benefit: the size of the propositional formulae generated can decrease as well as the number of transition steps to be applied for finding attacks. Notice that, given the set of intruder decomposing rules $\mathcal{DR} \subset \mathcal{R}$, it is straightforward to build the set of decomposing axioms $\mathcal{DA}$:

$$\mathcal{DA} = \{p_1 \wedge \cdots \wedge p_j \supset c \mid (p_1 \centerdot \ldots \centerdot p_j \xrightarrow{\ell} C) \in \mathcal{DR}, c \in C\}$$

It can be proved both that this optimization is correct as it preserves the existing attacks and does not introduce new ones, and that it leads to equal or shorter attacks.

3 Protocol Insecurity Problems with Axioms

The concepts presented in Section 2 can be recast into the concept of protocol insecurity problem with axioms. A *protocol insecurity problem with axioms* is a tuple $\Xi = \langle \mathcal{F}, \mathcal{L}, \mathcal{R}, \mathcal{A}, \mathcal{I}, \mathcal{G} \rangle$ where $\mathcal{F}$ is a set of atomic formulae of a sorted first-order language called *facts*, $\mathcal{L}$ is a set of function symbols called *rule labels*, $\mathcal{A}$ is a set of axioms of the form $p_1 \wedge \ldots \wedge p_j \supset c$, where $p_1, \ldots, p_j, c$ are in $\mathcal{F}$, and $\mathcal{R}$ is a set of rewrite rules of the form $L \xrightarrow{\ell} R$, where L and R are finite subsets of $\mathcal{F}$ such that the variables occurring in R occur also in L, and ℓ is an expression of the form $l(\underline{x})$ where $l \in \mathcal{L}$ and $\underline{x}$ is the vector of variables obtained by ordering lexicographically the variables occurring in L. The components $\mathcal{I}$ and $\mathcal{G}$ of a protocol insecurity problem with axioms are the initial states and a boolean formula representing the bad states of the protocol, respectively. In this setting, a state is represented by the set of facts $S \subseteq \mathcal{F}$ that are true in it. As a consequence, all the facts that are not in S are considered false (close world assumption). Moreover, all the states of Ξ must satisfy the set of axioms $\mathcal{A}$. Formally, $states(\Xi) = \{S \mid S \subseteq \mathcal{F}, S \models \mathcal{A}\}$.[13] Let S be a state and $(L \xrightarrow{\ell} R) \in \mathcal{R}$, if σ is a substitution such that $L\sigma \subseteq S$ and $S' = (S \setminus L\sigma) \cup R\sigma$ is such that $S' \models \mathcal{A}$, then one possible next state of S is S' and we indicate this with $S \overset{\ell\sigma}{\rightsquigarrow} S'$. We assume the rewrite rules are *deterministic* i.e. if $S \overset{\ell\sigma}{\rightsquigarrow} S'$ and $S \overset{\ell\sigma}{\rightsquigarrow} S''$, then $S' = S''$. A *solution to a protocol insecurity problem with axioms* Ξ, called *attack*, is a sequence of rules $\ell_1\sigma_1, \ldots, \ell_n\sigma_n$ such that $S_i \overset{\ell_i\sigma_i}{\rightsquigarrow} S_{i+1}$ for $i = 1, \ldots, n$ with $S_1 \subseteq \mathcal{I}$ and $S_n \models \mathcal{G}$. The *length* of an attack is the number of rules occurring in it.

It is convenient to relax the definition of the transition relation associated to a protocol insecurity problem with axioms by allowing parallel execution of

[13] Notice that, $S \models \mathcal{A}$ iff for each substitution σ and $(p_1 \wedge \cdots \wedge p_k \supset c)$ in $\mathcal{A}$, it holds that $S \models (p_1\sigma \wedge \cdots \wedge p_k\sigma) \supset c\sigma$.

rules, though preserving the interleaving semantic of the model. This means that every trace in which some rules are executed simultaneously can be linearized into a longer trace in which all the rules are executed sequentially. In order to guarantee the interleaving semantic of the model we define $\oplus$ to be the (commutative) relation of mutual exclusion (*mutex* for short) between rules. With the introduction of axioms in the concept of protocol insecurity problem, the mutex relation requires the construction of a directed graph $G_{\mathcal{A}} = \langle \mathcal{F}, E \rangle$ representing the dependencies between facts wrt the set of axioms. The basic idea is that for each substitution σ and $(p_1 \wedge \cdots \wedge p_j \supset c) \in \mathcal{A}$, the pairs $\langle c\sigma, p_1\sigma \rangle, \ldots, \langle c\sigma, p_j\sigma \rangle$ are in the set of edges E. A fact c is *dependent* from a fact p, denoted with $c \hookrightarrow_{\mathcal{A}} p$, iff there is in $G_{\mathcal{A}}$ a path from the node c to the node p. Let $dep_{\mathcal{A}}(c) = \{p \mid c \hookrightarrow_{\mathcal{A}} p\} \cup \{c\}$ be the set of facts from which c depends wrt $\mathcal{A}$ and, similarly, let $dep_{\mathcal{A}}(C) = \bigcup_{c \in C} dep_{\mathcal{A}}(c)$ be the set of facts from which the facts in C depend wrt $\mathcal{A}$. The mutex relation $\oplus$ is thus defined as follow: for each $L_1 \xrightarrow{\ell_1} R_1$, $L_2 \xrightarrow{\ell_2} R_2$ in $\mathcal{R}$ and for each pair of substitutions σ_1 and σ_2 such that $\ell_1\sigma_1 \neq \ell_2\sigma_2$, then $\ell_1\sigma_1 \oplus \ell_2\sigma_2$ iff $L_1\sigma_1 \cap dep_{\mathcal{A}}((L_2\sigma_2 \setminus R_2\sigma_2)) \neq \emptyset$ or $L_2\sigma_2 \cap dep_{\mathcal{A}}((L_1\sigma_1 \setminus R_1\sigma_1)) \neq \emptyset$. If $\ell_1 \oplus \ell_2$ we say that ℓ_1 and ℓ_2 are *conflicting* rule instances. The parallel execution of rules is thus allowed by the following relaxed definition of transition. Let S be a state, if there exist a set of rules $\{L_1 \xrightarrow{\ell_1} R_1, \ldots, L_m \xrightarrow{\ell_m} R_m\} \subseteq \mathcal{R}$ and a set of substitutions $\{\sigma_1, \ldots, \sigma_m\}$ such that, by defining $L = (\bigcup_{i=1}^{m} L_i\sigma_i)$ and $R = (\bigcup_{i=1}^{m} R_i\sigma_i)$, (i) for each $i, j = 1, \ldots, m$ with $i \neq j$, then $\ell_i\sigma_i \oplus \ell_j\sigma_j$ does not hold (non-conflicting rule instances), (ii) $L \cap R = \emptyset$, (iii) $L \subseteq S$, and (iv) $((S \setminus L) \cup R) \models \mathcal{A}$, then one possible next state of S is $S' = ((S \setminus L) \cup R)$. We indicate this with $S \overset{\Lambda}{\underset{\mathrm{P}}{\rightsquigarrow}} S'$ where $\Lambda = \{\ell_1\sigma_1, \ldots, \ell_m\sigma_m\}$. Similarly, the definition of solution to a protocol insecurity problem can be recast into the concept of *partial-order attack*. A partial-order attack to a protocol insecurity problem Ξ is a sequence of sets of rewrite rule instances $\Lambda_1, \ldots, \Lambda_n$ such that $S_i \overset{\Lambda_i}{\underset{\mathrm{P}}{\rightsquigarrow}} S_{i+1}$ for $i = 1, \ldots, n$ with $S_1 \subseteq \mathcal{I}$ and $S_n \models \mathcal{G}$. The length of a partial-order attack corresponds to the number of sets in the sequence. It can be proved that a partial-order attack to Ξ corresponds to a set of attacks to Ξ. Moreover, let $reach(S, \rightsquigarrow)$ and $reach(S, \underset{\mathrm{P}}{\rightsquigarrow})$ the set of states that can be reached from the state S by means of the transition relations $\rightsquigarrow$ and $\underset{\mathrm{P}}{\rightsquigarrow}$, respectively. It can be proved that $reach(S, \rightsquigarrow) = reach(S, \underset{\mathrm{P}}{\rightsquigarrow})$.

4 Automatic SAT-Compilation of Protocol Insecurity Problems with Axioms

Let $\Xi = \langle \mathcal{F}, \mathcal{L}, \mathcal{R}, \mathcal{A}, \mathcal{I}, \mathcal{G} \rangle$ be a protocol insecurity problem with axioms such that (i) the sets of facts, axioms, and rules are finite, (ii) the set of axioms $\mathcal{A}$ does not entail any propositional equivalence between facts (i.e. for all the facts $p_1, \ldots, p_j, c$ in $\mathcal{F}$, it holds that $\not\models_{\mathcal{A}} (p_1 \wedge \cdots \wedge p_j) \equiv c$), and let k be a

positive integer, then it is possible to build a propositional formula Φ^k_Ξ such that any model of Φ^k_Ξ corresponds to a partial-order attack of length k solution of Ξ. It is worth pointing out that the axioms modelling the ability of the intruder of decomposing messages satisfy the above requirements (i) and (ii).

In previous work we described how a protocol insecurity problems without axioms is compiled into a set of SAT formulae using two encoding techniques: the first belongs to the family of so-called *linear encodings* [1,2], the second is the more sophisticated *graphplan-based encoding* [3]. Let us see how to extend our approach to be able to deal with the axioms described in Sections 2 and 3. The basic idea does not change and consists into adding an additional time-index to the rules and facts to indicate the state at which the rule begins or the fact holds. Facts are thus indexed by 0 through k and rules by 0 through $k-1$. If p is a fact or an rule and i is an index in the appropriate range, then p^i is the corresponding time-indexed propositional variable. However, with the introduction of the axioms into the concept of protocol insecurity problem, significant alterations must to be done in the encodings schemes. In the rest of this section we will formally describe how to compile a protocol insecurity problem with axioms into SAT by using the linear encoding technique. Similar concepts apply also to the graphplan-based encoding technique. Notice also that the encoding technique we are going to present can be applied also to compile protocol insecurity problems without axioms into SAT simply by setting $\mathcal{A} = \emptyset$.

Similarly to the classic bounded model-checking approach [6], the propositional formula Φ^k_Ξ is defined by

$$\Phi^k_\Xi = I(\underline{f}^0) \wedge \bigwedge_{i=0}^{k-1} T_i(\underline{f}^i, \underline{\ell}^i, \underline{f}^{i+1}) \wedge G(\underline{f}^k)$$

where $\underline{f}$ and $\underline{\ell}$ are vectors of facts and rules respectively [14] and

- $I(\underline{f}^0)$ is a formula encoding the initial states $\mathcal{I}$;
- $G(\underline{f}^k)$ is an k-indexed formula encoding the goal states represented by $\mathcal{G}$;
- $T_i(\underline{f}^i, \underline{\ell}^i, \underline{f}^{i+1})$ is a formula encoding the transition relation between states reachable in i steps and states reachable in $i+1$ steps.

The main difference between our encoding techniques and those employed in other bounded model-checkers (e.g. NuSMV [8]) is in the way the formula that encodes the transition relation, i.e. $T_i(\underline{f}^i, \underline{\ell}^i, \underline{f}^{i+1})$, is generated.

By using the linear encoding technique, each component of Φ^k_Ξ is defined as follow:

- $I(\underline{f}^0)$ is a conjunction of the formulae f^0 if $f \in \mathcal{I}$ and $\neg f^0$ if $f \notin \mathcal{I}$;

[14] Let $\underline{p}$ be a vector of facts or rules and i be an index in the appropriate range, then $\underline{p}^i$ is the corresponding time-indexed vector of propositional variable.

- $G(\underline{f}^n)$ is obtained from $\mathcal{G}$ by replacing each fact f with f^n;
- $T_i(\underline{f}^i, \underline{\ell}^i, \underline{f}^{i+1})$ is equivalent to $T(\underline{f}^i, \underline{\ell}^i, \underline{f}^{i+1})$ (i.e. the formula encoding the transition relation is independent from the time step) and is a conjunction of the *Universal Formulae*, *Explanatory Frame Formulae*, *Axioms Formulae*, and *Conflict Exclusion Formulae*.

In order to improve readability, let us define $\tilde{\mathcal{R}}$ and $\tilde{\mathcal{A}}$ to be the sets of rule instances and axiom instances, respectively. Clearly, $\tilde{\mathcal{R}} = \{L\sigma \xrightarrow{\ell\sigma} R\sigma \mid L \xrightarrow{\ell} R \in \mathcal{R}, \sigma \text{ is a substitution}\}$ and $\tilde{\mathcal{A}} = \{p_1\sigma \wedge \cdots \wedge p_j\sigma \supset c\sigma \mid p_1 \wedge \cdots \wedge p_j \supset c \in \mathcal{A}, \sigma \text{ is a substitution}\}$.

Universal Formulae. They express how the transition relation evolves and they are defined as the conjunction of the following:

$$\ell^i \supset \bigwedge\{f^i \mid f \in L\}$$
$$\ell^i \supset \bigwedge\{f^{i+1} \mid f \in (R \setminus L)\}$$
$$\ell^i \supset \bigwedge\{\neg f^{i+1} \mid f \in (L \setminus R)\}$$

for each $L \xrightarrow{\ell} R \in \tilde{\mathcal{R}}$.

Explanatory Frame Formulae. They express the inertia of the transition system. By introducing the axioms, the definition of such formulae require to define the set $\hat{\mathcal{A}}$ of the *contraposed* axioms of $\mathcal{A}$. For instance, $\neg b \supset \neg a$ is the contraposed axiom of $a \supset b$. Given the set of axiom instances $\tilde{\mathcal{A}}$, the set of contraposed axiom instances $\hat{\mathcal{A}}$ is $\{\neg c \wedge p_1 \wedge \cdots \wedge p_{h-1} \wedge p_{h+1} \wedge \cdots \wedge p_j \supset \neg p_h \mid (p_1 \wedge \cdots \wedge p_j \supset c) \in \tilde{\mathcal{A}}, h = 1, \ldots, j\}$. The explanatory frame formulae are thus defined as the conjunction of the following:

$$(f^i \wedge \neg f^{i+1}) \supset \Bigg(\bigvee \Big\{\ell^i \mid (L \xrightarrow{\ell} R) \in \tilde{\mathcal{R}}, f \in (L \setminus R)\Big\} \vee \bigvee\{\neg p_1^{i+1} \wedge p_2^{i+1} \wedge \cdots \wedge p_j^{i+1} \mid (\neg p_1 \wedge p_2 \wedge \cdots \wedge p_j \supset \neg f) \in \hat{\mathcal{A}}\}\Bigg)$$

$$(\neg f^i \wedge f^{i+1}) \supset \Bigg(\bigvee \Big\{\ell^i \mid (L \xrightarrow{\ell} R) \in \tilde{\mathcal{R}}, f \in (R \setminus L)\Big\} \vee \bigvee \Big\{p_1^{i+1} \wedge \cdots \wedge p_j^{i+1} \mid (p_1 \wedge \cdots \wedge p_j \supset f) \in \tilde{\mathcal{A}}\Big\}\Bigg)$$

for all facts $f \in \mathcal{F}$. In order to translate the above formulae into CNF without incurring in a combinatorial explosion in the number of clauses it suffices to replace each conjunction $p_1 \wedge \cdots \wedge p_j$ with a new variable v and to add the formula $v^i \equiv (p_1^i \wedge \cdots \wedge p_j^i)$. It is immediate to see that in the worst case the

number of these new variables is in $O(|\tilde{\mathcal{A}}|)$.

Axioms Formulae. They express properties between the facts of the transition system. They are defined as the conjunction of $(p_1^i \wedge \cdots \wedge p_j^i) \supset c^i$ for each $(p_1 \wedge \cdots \wedge p_j \supset c) \in \tilde{\mathcal{A}}$.

Conflict Exclusion Formulae. They state what pair of rule instances cannot be executed in parallel for guaranteeing the interleaving semantic of the model. They are defined as the conjunction of $\neg(\ell_1^i \wedge \ell_2^i)$ for all $\ell_1 \neq \ell_2$ such that $\ell_1 \oplus \ell_2$.

It is immediate to see that the number of atoms in Φ_Ξ^k is in $O(k|\mathcal{F}| + k|\tilde{\mathcal{R}}| + k|\tilde{\mathcal{A}}|)$. Moreover the number of clauses generated by the Universal Formulae is in $O(kP_0|\tilde{\mathcal{R}}|)$ where P_0 is the maximal number of facts mentioned in an rule instance (usually a small number); the number of clauses generated by the Explanatory Frame Formulae is in $O(k|\mathcal{F}| + kR_0|\tilde{\mathcal{A}}|)$ where R_0 is the maximal number of facts mentioned in a precondition of an axiom instance (usually a small number); finally, the number of clauses generated by the Conflict Exclusion Formulae is in $O(k|\tilde{\mathcal{R}}|^2)$.

5 Implementation and Experimental Results

We have implemented the above ideas in SATMC, a SAT-based Model-Checker for security protocol analysis. Given a protocol insecurity problem with axioms Ξ, SATMC compiles it into a SAT formula Φ_Ξ^k using one of its encoding techniques for increasing values of k and the propositional formula generated at each step is fed to a state-of-the-art SAT solver (Chaff, SIM, and SATO are currently supported). As soon as a satisfiable formula is found, the corresponding model is translated back into a partial order attack which is reported to the user. Two encoding techniques are currently implemented in SATMC and both have been extended for supporting axioms: the first belongs to the family of so-called linear encodings, the second is the more sophisticated graphplan-based encoding.

We have run our tool against a selection of (flawed) security protocols drawn from the Clark/Jacob library [9]. For each protocol we have automatically generated, by means of a translator from IF [4] into the SATMC internal language, two corresponding protocol insecurity problems modeling a scenario with a bounded number of sessions in which the involved principals exchange messages on a channel controlled, respectively, by the Dolev-Yao intruder and by the optimized Dolev-Yao intruder. As explained in Section 2 the optimization consists in modeling the intruder ability of decomposing messages by means of axioms instead of rules.

Tables 1 and 2 report the results of our experiments obtained by applying the linear encoding and the graphplan-based encoding, respectively, on the

Table 1
Experimental data using the linear encoding

	DY			Optimized DY		
Protocol	**K**	**Atoms**	**Clauses**	**K**	**Atoms**	**Clauses**
KaoChow 2	9	530,726	1,804,005	7	414,536	1,489,121
KaoChow 3	9	995,323	5,736,662	7	776,805	4,590,268
NSCK	9	114,530	334,086	8	88,343	298,491
NSPK	7	6,612	33,326	4	3,714	19,242
NSPK-server	8	9,157	53,741	5	5,600	33,835
Woo-Lam M	6	481,394	2,498,382	5	409,114	2,133,265

Table 2
Experimental data using the Graphplan-based encoding

	DY			Optimized DY		
Protocol	**K**	**Atoms**	**Clauses**	**K**	**Atoms**	**Clauses**
KaoChow 2	9	726	3,065	7	458	1,784
KaoChow 3	9	990	5,019	7	587	2,606
NSCK	9	435	1,392	8	348	1,105
NSPK	7	411	1,249	4	199	549
NSPK-server	8	847	2,688	5	380	1,177
Woo-Lam M	6	481	1,518	5	358	1,137

protocol insecurity problem without the optimized intruder (**DY**) and with it (**Optimized DY**). Experiments have been carried out on a PC with a 1.4 GHz CPU and 1 GB of RAM. For each protocol and for each protocol insecurity problem with and without the optimized intruder we give the smallest value of k at which the attack is found (**K**), and the number of propositional variables (**Atoms**) and clauses (**Clauses**) in the SAT formula. [15] It is immediate to see that for both the applied encoding techniques, SATMC is able to find out up to 40% shorter attacks [16] by generating up to 50% smaller propositional formulae. These results confirm the effectiveness of the proposed optimized intruder model for the SAT-based model-checking approach and pave the way to the application of SATMC to protocols of industrial complexity where attacks can eventually require a considerable number of intruder knowledge inference

[15] Notice that, if the length of the attack on the protocol insecurity problems with and without the optimized intruder is the same, the results obtained are so comparable that we avoid to show them.

[16] Notice that for each security protocol analyzed, the attacks found on the corresponding protocol insecurity problem with and without the optimized intruder are identical except for the fact that in the case of the former we save all those rules executed for decomposing the knowledge of the intruder.

steps.

6 Conclusions and Perspectives

We have proposed an optimized intruder model for SAT-based model-checking of security protocols. We have shown that this optimization, based on the idea of modeling the decomposition of the intruder knowledge by means of axioms instead of rules, leads to the discovering of shorter attacks by generating smaller propositional formulae. We have enhanced our SAT-based model-checker (SATMC) to be able to analyse protocol insecurity problems extended with a set of axioms (without equivalence cycles). Experimental results obtained by running SATMC against a selection of security protocols drawn from the Clark-Jacob's library confirm the effectiveness of the proposed optimized intruder model and pave the way to its application to large protocols which arise in practical applications. As a matter of fact, messages exchanged in industrial-scale security protocols can have a complex structure and, therefore, attacks on such protocols can require a considerable number of intruder knowledge manipulations. Since the proposed optimization allows for decomposing the whole intruder knowledge instantaneously, we conjecture that it will be particularly successful on such protocols and experiments will be done in this direction.

Another interesting and challenging future direction that we would like to investigate concerns the extension of our SAT-reduction techniques for supporting the encoding of generic set of axioms also specifying equivalences between facts. This would be particularly useful to express algebraic equations that specify, for instance, special properties of cryptographic operators such as exponentiation in the Diffie-Hellman protocol [10]. It is worth pointing out that on such a protocol we have already obtained some preliminary results in our current setting. In fact, by partially specifying exponentiation by means of a set of axioms, we have been able to discover the well-known attack on the Diffie-Hellman protocol.

Acknowledgments

We are grateful to Cristina Frà for her contribution to the implementation of the encodings for supporting such an optimized intruder. Moreover we wish to thank the anonymous reviewers for their useful comments. This work was partially funded by EU Calculemus Training Network (HPRN-CT-2000-00102) and by FET Open EC Project "AVISPA: Automated Validation of Internet Security Protocols and Applications" (IST-2001-39252).

References

[1] A. Armando and L. Compagna. Automatic SAT-Compilation of Protocol Insecurity Problems via Reduction to Planning. In *Proceedings of FORTE 2002*, LNCS 2529, pages 210–225. Springer-Verlag, 2002.

[2] A. Armando and L. Compagna. Abstraction-driven SAT-based Analysis of Security Protocols. In *Proceedings of SAT 2003*, LNCS 2919. Springer-Verlag, 2003. Available at http://www.avispa-project.org.

[3] A. Armando, L. Compagna, and P. Ganty. SAT-based Model-Checking of Security Protocols using Planning Graph Analysis. In *Proceedings of FME'2003*, LNCS 2805. Springer-Verlag, 2003.

[4] Alessandro Armando, David Basin, Mehdi Bouallagui, Yannick Chevalier, Luca Compagna, Sebastian Mödersheim, Micha"el Rusinowitch, Mathieu Turuani, Luca Viganò, and Laurent Vigneron. The AVISS Security Protocol Analysis Tool. In *Proceedings of CAV'02*, LNCS 2404, pages 349–354. Springer-Verlag, 2002.

[5] D. Basin, S. Mödersheim, and L. Viganò. An On-The-Fly Model-Checker for Security Protocol Analysis. In Einar Snekkenes and Dieter Gollmann, editors, *Proceedings of ESORICS'03*, LNCS 2808, pages 253–270. Springer-Verlag, 2003. Available at http://www.avispa-project.org.

[6] A. Biere, A. Cimatti, E. Clarke, and Y. Zhu. Symbolic Model Checking without BDDs. In W. R. Cleaveland, editor, *Proceedings of TACAS'99*, LNCS 1579, pages 193–207, Berlin, 1999. Springer-Verlag.

[7] Iliano Cervesato, N. A. Durgin, Patrick Lincoln, John C. Mitchell, and Andre Scedrov. A meta-notation for protocol analysis. In *CSFW*, pages 55–69, 1999.

[8] A. Cimatti, E. Clarke, F. Giunchiglia, and M. Roveri. NuSMV: A New Symbolic Model Verifier. *LNCS 1633*, 1999.

[9] J. Clark and J. Jacob. A Survey of Authentication Protocol Literature: Version 1.0, 17 November 1997. Available at http://www.cs.york.ac.uk/~jac/papers/drareview.ps.gz.

[10] W. Diffie and M. E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976.

[11] D. Dolev and A. Yao. On the Security of Public-Key Protocols. *IEEE Transactions on Information Theory*, 2(29), 1983.

[12] N. Durgin, P. D. Lincoln, J. C. Mitchell, and A. Scedrov. Undecidability of Bounded Security Protocols. In *Proceedings of the FLOC'99 Workshop on Formal Methods and Security Protocols (FMSP'99)*, 1999.

[13] Shimon Even and Oded Goldreich. On the security of multi-party ping pong protocols. In *Proceedings of 24th IEEE Symposium on Foundations of Computer Science*. IEEE Computer Society, 1983.

[14] James Heather, Gavin Lowe, and Steve Schneider. How to prevent type flaw attacks on security protocols. In *Proceedings of The 13th Computer Security Foundations Workshop (CSFW'00)*. IEEE Computer Society Press, 2000.

[15] F. Jacquemard, M. Rusinowitch, and L. Vigneron. Compiling and Verifying Security Protocols. In M. Parigot and A. Voronkov, editors, *Proceedings of LPAR 2000*, LNCS 1955, pages 131–160. Springer-Verlag, 2000.

[16] R. M. Needham and M. D. Schroeder. Using Encryption for Authentication in Large Networks of Computers. Technical Report CSL-78-4, Xerox Palo Alto Research Center, Palo Alto, CA, USA, 1978. Reprinted June 1982.

Satisfiability of Dolev-Yao Constraints

Laurent Mazaré [1]

Laboratoire VERIMAG
Grenoble, France

Abstract

The secrecy problem, under certain hypothesis, is equivalent to satisfiability of constraints involving Dolev-Yao's operator $\vdash$. Making some restrictions over these constraints, their satisfiability has been proven to be NP-complete. However, to check opacity [11] or some modified versions of secrecy, there is a need to find similar results for a larger class of predicates. This paper starts to extend this decidability result to more general constraints allowing in particular inequalities and gives a simple decision procedure based on a rewriting system.

Key words: Security, Formal Verification, Dolev-Yao Constraints, Rewriting Systems, Opacity.

1 Introduction

During the last decade, verification of security protocols has been widely investigated. The majority of the studies focussed on demonstrating secrecy properties using formal methods (see for example [4], [6], [5] or [9]). These methods have lead to effective algorithms and so to concrete tools for verifying secrecy like those proposed by the EVA project [3] or the Avispa project [2]. The main result is that considering an active intruder and a bounded number of sessions in Dolev-Yao model (see [8]), secrecy is decidable and is an NP-complete problem. One possible way to demonstrate this result is to use constraints based on Dolev-Yao's operator $\vdash$. This has been done for example in [12] or in [14]. A Dolev-Yao constraint will be a conjunction of atomic constraints of the form $E \Vdash m$, where E is an environment (i.e. a finite set of messages), m is a message and both of them are potentially not closed. Thus, in this case, the secrecy problem is equivalent to satisfiability of a given constraint.

In this paper, after giving a formal definition of Dolev-Yao constraints, we will investigate their satisfiability. In a first part, we will restrict ourselves

[1] Email: laurent.mazare@imag.fr

This is a preliminary version. The final version will be published in Electronic Notes in Theoretical Computer Science
URL: www.elsevier.nl/locate/entcs

to "well-formed" constraints. These constraints will form an extension of the well-known set of constraints which satisfiability is exactly equivalent to secrecy in Dolev-Yao's model with a bounded number of sessions. This extension will add the possibility to use the $\neq$ operator. Thus, protocols which check that a received variable is not a given one (for example, a protocol verifying that a nonce has not already been used before) could be verified by checking satisfiability of this kind of constraint. Using a method based on term rewriting, satisfiability of such constraints will be proved to be decidable and we will provide a concrete decision algorithm. Moreover, one of the hypothesis over well-formed constraints could be removed. This will define quasi well-formed constraints which satisfiability will be proved decidable too. In addition to protocols checking inequalities, there are two main motivations when extending classical Dolev-Yao constraints. The first one, leading to the definition of well-formed, is to study opacity [11] instead of secrecy in the case of active intruders. Opacity in this case will be equivalent to satisfiability of a well-formed constraint. The criterion defining "quasi well-formed" constraints is useful when studying opacity too, so that we could model the unfolding of two parallel sessions. An other use of these constraints could be to model attacks performed by two distinct intruders that are not allowed to communicate. Application of these constraints to opacity will not be entirely detailed here and will be the object of a future paper.

The remainder of this paper is organized as follows. In section 2, we will briefly recall the usual definitions leading to Dolev-Yao constraints and proper definitions for well-formed and quasi well-formed will be given. Then, in section 3, we will use a rewrite system to prove that satisfiability of well-formed constraints is decidable. Section 4 will extend this result to the case of quasi well-formed constraint. In section 5, we will quickly explain why satisfiability of our constraints is NP-complete. Section 6 will show how these results could be applied to check opacity with an active intruder. And eventually, section 7 will conclude this paper.

2 Dolev-Yao Constraints

Let A, X and F be three infinite countable disjoint sets. A is the set of atomic messages usually written a, b and so on. X is the set of message variables (written x, y,...) and F is the set of functions (written f, g,...).

Definition 2.1 Let Σ be the signature $A \cup \{pair, encrypt\} \cup F$ where *pair* and *encrypt* are two binary functions. The atomic messages are supposed constant functions and each function f has a fixed arity $ar(f)$. Then a *message* is a first order term over Σ and the set of variables X, namely an element of $T(\Sigma, X)$. A message is said to be *closed* iff it is a closed term of $T(\Sigma, X)$, i.e. a term of $T(\Sigma)$.

In the rest of this paper, we will use the well-known notations $\langle m_1, m_2 \rangle =$

$pair(m_1, m_2)$ and $\{m_1\}_{m_2} = encrypt(m_1, m_2)$. The height of a message m could be easily defined recursively and will be noted $|m|$. The set of variables used in a message m is noted $var(m)$, this is not the usual definition of var as we will consider that $f(...)$ is "atomic":

$$\begin{aligned} var(a) &= \emptyset \\ var(x) &= \{x\} \\ var(f(...)) &= \emptyset \\ var(\langle m, n\rangle) &= var(m) \cup var(n) \\ var(\{m\}_n) &= var(m) \cup var(n) \end{aligned}$$

The substitutions σ from X to $T(\Sigma, X)$ are defined as usual. Application of σ to message m will be noted $m\sigma$ (instead of $\sigma(m)$). If σ is defined by $x\sigma = n$ and $y\sigma = y$ for any other variables y, then we could write $m[x\backslash n]$ instead of $m\sigma$. The domain of a substitution σ is the set $dom(\sigma)$ of variables such that $x\sigma \neq x$.

Using the pair operator, it is possible to introduce n-tuples. Then, we will use the t^n notation to denote tuples composed using n times the same message t. Formally, t^n is recursively defined by $t^1 = t$ and $t^{n+1} = \langle t, t^n\rangle$

We will use Dolev-Yao intruder's model [8]. The intruder controls the network: he could intercept any message, forge messages using its initial knowledge and previously intercepted messages, and send these messages to other agents usurping an agent's identity. To forge new messages, the intruder uses Dolev-Yao theory $\vdash$. If E is a finite set of messages (usually called an *environment*) and m is a message, $E \vdash m$ means that m could be deduced from E using the classical inferences. In this paper, we only consider symmetric cryptography. However, all of our results would still hold when considering public key cryptography but we will keep this hypothesis for simplicity's sake.

Dolev-Yao constraints are predicates built using classical logic operators and a new tertiary operator noted $\Vdash$. This operator is used in atomic predicates like $T \Vdash m[U]$ where T (environment), m (message) and U (environment) could be non closed. This predicates' intuition is that there exists a proof of $T \vdash m[U]$ which is defined as the classical $\vdash$ except for the decode rule.

$$\frac{T \vdash \{m\}_u[U] \qquad u \in U}{T \vdash m[U]}$$

The only keys allowed to decode a message are the keys in U and these keys are not to be proven deducible.

Definition 2.2 [Constraints] Dolev-Yao *constraints* are defined according to the following grammar:

$$C ::= \bot | \top | C \vee C | C \wedge C | C_A$$

$$C_A ::= T \Vdash m[U] | m \neq n$$

Where T and U are finite sets of messages, m and n are messages.

For a constraint C, the sets T appearing as left part of an atomic constraint $T \Vdash m[U]$ in C are called environments of C. The classical notion of model is used for operators like $\vee$ or $\neq$. We will extend it by saying that σ is a model of $T \Vdash m[U]$ written $\sigma \models T \Vdash m[U]$ iff $T\sigma$, $m\sigma$ and $U\sigma$ are closed and there exists a proof of $T\sigma \vdash m\sigma[U\sigma]$.

Definition 2.3 [Model] A substitution σ is a model for constraint C iff $C\sigma$ is closed and $\sigma \models C$ where $\sigma \models .$ is the smallest predicate verifying the classical inferences for $\top$, $\wedge$ and $\vee$, and the two following inferences:

$$\frac{m\sigma \neq n\sigma}{\sigma \models m \neq n}$$

$$\frac{T\sigma \vdash_T m\sigma[U\sigma]}{\sigma \models T \Vdash m[U]}$$

Using standard boolean rules, any constraint could be transformed to $C = \bigvee_i Co_i$ where $Co_i = \bigwedge C_A$. The Co_i constraints are called the conjunctions composing C. We will usually suppose that our constraints follow this form as it is needed to check that they are well-formed or quasi well-formed.

Definition 2.4 [Well-Formed Constraint] A constraint C is said to be *well formed* iff none of the environment is empty and for any conjunction Co composing C, the two following conditions hold:

- If $T \Vdash m[U]$ and $T' \Vdash m'[U']$ are in Co, then $T \subseteq T'$ or $T' \subseteq T$ (Environment Inclusion).
- If $T \Vdash m[U] \in Co$ and $x \in var(T)$, then there exists $T' \Vdash m'[U'] \in Co$ such that $x \in var(m')$, $U' \subseteq U$ and $T' \subsetneq T$ (Variable Introduction).

A constraint that only satisfies the Variable Introduction requirement and such that there exists a same closed message m in all its environment will be said *quasi well-formed*.

It is easy to remark that for conjunctions in a well-formed constraint, there exists a constraint $T_x \Vdash m_x[U_x]$ in Co such that T_x is minimal and $x \in var(m_x)$. This notation T_x will be used in the following. An immediate property about T_x is that $x \notin var(T_x)$. Moreover, we could notice that any well-formed constraint is quasi well-formed.

As we have now proper definitions for Dolev-Yao constraints, we will give a very classical example of constraint. To achieve this, it is practical to be able to formalize the classical Dolev-Yao constraint: message m is deducible from the set of message T written $T \Vdash m$. This constraint could be described using constraint with [] by quantifying on the order upon which keys are compromised. m will be deducible iff there exist k_1,...,k_n distinct keys in T or m such that k_1 is deducible without any decoding, k_2 deducible using k_1 only and so on. At the end, m needs to be deducible from T using only keys k_1 to k_n.

Proposition 2.5 *Let m be a message and T a finite set of messages. Then, for any substitution σ, the following equivalence holds.*

$$T\sigma \vdash m\sigma \Leftrightarrow \sigma \models \bigvee_{k_1,...,k_n \in keys(T,m)} (T \Vdash k_1[] \wedge T \Vdash k_2[k_1] \wedge ... \wedge T \Vdash m[k_1, ..., k_n])$$

This property could be extended to predicates with several $\Vdash$ operators. If environments of the predicate are ordered, it is possible to assume that keys compromised for an environment T are compromised for T' verifying $T \subseteq T'$. Let us call the environments $T_1 \subseteq T_2 \subseteq ...T_n$. Thus, we quantify on the previous order that gives us k_1,...,k_α and for each environment T_j on the value of i_j such that in T_j, keys k_1 to k_{i_j} are compromised (and so, we only have to consider $i_1 \leq i_2 \leq ... \leq i_n$).

Proposition 2.6 *For i between 1 and n, let m_i be a message and T_i a finite set of messages. If $\bigwedge_{1\leq i\leq n} T_i\sigma \Vdash m_i$ is well-formed and $T_1 \subseteq T_2 \subseteq ...T_n$, then for any substitution σ, the following equivalence holds.*

$$\begin{aligned}\bigwedge_{1\leq i\leq n} T_i\sigma \vdash m_i\sigma \Leftrightarrow \sigma \models \bigvee_{\substack{k_1,...,k_\alpha \in keys(T,m)\\ 1\leq i_1\leq i_2\leq ...\leq i_n\leq n}} & (T_1 \Vdash k_1[] \wedge ... \wedge T_1 \Vdash m_1[k_1, ..., k_{i_1}]) \\ & \wedge (T_2 \Vdash k_{i_1+1}[k_1, ..., k_{i_1}] \wedge ... \wedge T_2 \Vdash m_2[k_1, ..., k_{i_2}]) \\ & \wedge ... \\ & \wedge (T_n \Vdash k_{i_{n-1}+1}[k_1, ..., k_{i_{n-1}}] \wedge ... \wedge T_n \Vdash m_n[k_1, ..., k_{i_n}])\end{aligned}$$

This property gives the following result: secrecy is equivalent to satisfiability of a well-formed Dolev-Yao constraint.

An Example: Needham-Schroeder Constraint

Secrecy with a bounded number of sessions in Dolev-Yao model could be linked to satisfiability of a well-formed constraint. This is explained in [7] for example. The usual example of attack that could be discovered using formal methods is an attack on Needham-Schroeder protocol [13] found by Lowe in 1995 [10]. The constraint NS which satisfiability is equivalent to the existence of this attack is given below. As it describes an attack in Dolev-Yao model, the environments describe the knowledge of the intruder and thus verifies Environment Inclusion and Variable Introduction. That is why, this constraint is well-formed and its satisfiability could be verified. The two sessions considered here occur between A and the intruder C and between C usurping A's identity and B. To keep it as simple as possible, this example uses public key cryptography. Let E be the initial knowledge of the intruder C, $E = \{A, B, C, K_A, K_B, K_C, K_C^{-1}\}$.

$$\begin{aligned}NS = \quad & E, \{N_C, A\}_{K_C} \Vdash \{x, A\}_{K_B} \\ & \wedge E, \{N_C, A\}_{K_C}, \{x, N_B\}_{K_A} \Vdash \{N_C, y\}_{K_A} \\ & \wedge E, \{N_C, A\}_{K_C}, \{x, N_B\}_{K_A}, \{y\}_{K_C} \Vdash N_B\end{aligned}$$

3 Decidability for Well-Formed Constraints

In this section, we will provide an original decision procedure for satisfiability of well-formed constraints. For such constraints using only $\Vdash$, it is well known that the satisfiability problem is NP-complete: see for example [14], [12] or an extension to XOR in [7]. Our decision procedure will work using a rewriting system over constraints. We will have to prove that our system terminates, that satisfiability for our normal forms is decidable and eventually that our system is correct and complete, i.e. models of a constraint are exactly the models of this constraint once rewritten. Knowing that, we are able to provide a decision procedure for satisfiability which is to rewrite the constraint up to its normal form and then to check satisfiability on this normal form. Compared to other proofs of decidability of satisfiability for this kind of constraints (that are easy to find in the literature, see [12] as a starting point), our main objective is to provide a very simple, yet extensible, decision procedure.

Definition of the rewriting system needs some explanation. Our rewriting system is triggered by a condition. As we will check for termination without considering that condition, this will not cause any problem. The first step in our decision procedure is to test all possible unifications among sub-terms of the different environments and messages. This is done using a large quantification ($\vee$ operator). After that step, two different non-closed messages (that are sub-messages of the original constraint) are supposed to be instantiated with different closed messages (i.e. they cannot be unified). Using that, an atom a is deducible from T (using U) iff a appears in T (protected only by keys occurring in U). The same kind of idea is used for $\{\}$ whereas the case of $\langle\rangle$ is trivial.

Definition 3.1 The rewriting system $\rightarrow$ is defined over well-formed constraints by the three following rules that will only apply if for all $x \in var(T)$, $T_x \Vdash x[U_x]$ occurs in the same conjunction with $U_x \subseteq U$.

(1) $T \Vdash a[U] \rightarrow \top$
(2) $T \Vdash a[U] \rightarrow \bot$
(3) $T \Vdash f(m_1, ..., m_n)[U] \rightarrow \top$
(4) $T \Vdash f(m_1, ..., m_n)[U] \rightarrow \bot$
(5) $T \Vdash \langle m, n\rangle[U] \rightarrow T \Vdash m[U] \wedge T \Vdash n[U]$
(6) $T \Vdash \{m\}_n[U] \rightarrow \top$
(7) $T \Vdash \{m\}_n[U] \rightarrow T \Vdash m[U] \wedge T \Vdash n[U]$

- Rule 1 is applied iff a occurs in T protected only by keys that appear in U, else rule 2 is applied.
- Rule 3 is applied iff $f(m_1, ..., m_n)$ occurs in T protected only by keys that appear in U, else rule 4 is applied. Thus, rule 1 and 2 are useless if we consider atoms as constant functions.
- Rule 5 is the intuitive rule for pairs.

- Rule 6 is applied iff $\{m\}_n$ occurs in T protected only by keys that appear in U (this is the case where $\{m\}_n$ could be obtained using decompositions), else rule 7 is applied (in the other case, $\{m\}_n$ is obtained with a composition operation).

The condition expressed over T aims to force the execution order to follow $\subseteq$. Therefore if $T_1 \subseteq T_2$, then the pattern $T_1 \Vdash m_1$ will be entirely rewritten before processing $T_2 \Vdash m_2$.

Last remark, these rules are easy to extend to the case of public key cryptography. In this case, in the condition of application of the different rules, "protected only by keys that appear in U" is to be replaced by protected only by keys which inverses appear in U.

The first thing to check is that after applying $\rightarrow$ to a well-formed constraint, the result remains well-formed. This is expressed by the following property.

Proposition 3.2 *If C and C' are constraints such that $C \rightarrow C'$ and C is well-formed, then C' is well-formed.*

The second thing is to prove that our rewriting system terminates. Then, it will be possible to transform any constraint to a constraint in normal form that has exactly the same models. This is easy to show as when rewriting, right members of each constraint are discomposed or the constraint is rewritten to $\bot$ or $\top$. Thus, by taking a simple measure over constraint $s(C)$ defined by:

$$s(T_1 \Vdash m_1[U_1] \wedge ... \wedge T_n \Vdash m_n[U_n]) = \sum_{i=1}^{n} s(m_i)$$

This gives us $s(\bot) = s(\top) = 0$. And s is recursively defined over messages by:

$$\begin{aligned} s(a) &= 1 \\ s(f(...)) &= 1 \\ s(\langle m_1, m_2 \rangle) &= s(m_1) + s(m_2) + 1 \\ s(\{m_1\}_{m_2}) &= s(m_1) + s(m_2) + 1 \end{aligned}$$

Using an order based on this measure, the following property is immediate.

Proposition 3.3 *The rewriting system $\rightarrow$ terminates.*

After transforming a constraint to its normal form, we want to check that both have the same sets of models. Formally, if $C \rightarrow C'$ then for any substitution σ, σ is a model of C iff σ is a model of C'. This will prove that if the normal form is satisfiable then, the original constraint is satisfiable too and that the reciprocal is true.

Proposition 3.4 *The rewriting system $\rightarrow$ is correct and complete for well-formed constraints.*

Note that, to prove this result, the Variable Introduction hypothesis is used. Else, rules would not be correct and complete anymore. For example, if we

consider the following constraint:

$$a, \{b\}_a \Vdash x[a] \wedge a, \{b\}_a, x \Vdash b[]$$

This constraint would be rewritten to $\bot$ as b does not appear in $a, \{b\}_a, x$ unprotected. But, by using $x = b$, we know that this constraint is satisfiable. That is why, most of the rules are only correct and complete if the Variable Introduction hypothesis is respected.

Eventually, the last step is to show that satisfiability for our normal forms is easily decidable. First, normal forms of well-formed constraints are well-formed constraints. They have the form:

$$\bigvee \Big((T \Vdash x[U]) \wedge (\bigwedge m \neq n) \Big)$$

At this point, every normal form that is well-formed is satisfiable if the lowest environment T_0 is non empty (this is always true by definition of well-formed) and if the conjunction of inequalities is satisfiable. To prove that, let t be a message in T_0 (t always exists as T_0 is not empty). As the environment T_0 is closed (this could be seen by applying the Variable Introduction hypothesis to T_0), t is closed. Then, for every variable in an atomic constraint, we will use a message of the form t^n. It is clear that the conjunction of predicates using $\Vdash$ is satisfied. And by changing values of the different n, if the $\neq$ part is satisfiable, we will satisfy it by giving to n a value lower than the number of inequalities plus the number of variables. The intuition is that, given a number n, if a conjunction $m_1 \neq n_2$ is not satisfied, then it satisfied for any other number $n' \neq n$.

We will use the σ^k notation which signification $x\sigma^k$ is the tuple composed by k times $x\sigma$.

$$x\sigma^k = (x\sigma)^k = \langle x\sigma, ..., x\sigma \rangle$$

Proposition 3.5 (Solving Inequalities) *Let P be the constraint $m_1 \neq n_1 \wedge ... \wedge m_j \neq n_j$ where m_i and n_i are messages. If P is satisfiable, then for any substitution σ such that $P\sigma$ is closed and taking distinct values for any variables in $var(P)$, there exists an integer k such that $k \leq j + 1$ and σ^k is a model of P.*

Proof. The proof use the following lemma: let m and n be two messages, σ be a substitution such that $m\sigma$ and $n\sigma$ are closed, and that $x\sigma = y\sigma \Rightarrow x = y$. If there exists two different integers i and j such that $m\sigma^i = n\sigma^i$ and $m\sigma^j = n\sigma^j$, then we have $m = n$. □

This kind of property is very general and could be applied to other operators instead of $=$. For example, the same thing holds for the $\sim$ operator introduced in [11] and this will allow to prove decidability of satisfiability for a similar extension of the predicates defined in this paper. This decidability (and NP-completeness) will prove that the opacity problem when considering an active

intruder (Dolev-Yao's model) and a finite number of session is NP-complete. Eventually, satisfiability is equivalent to satisfiability of the $\neq$ part. This last satisfiability is easy to check using for example negation of this conjunction. This gives a constraint using only equalities. Then, by applying classical unification, it is possible to check that this final constraint is always satisfied (hence is rewritten to $\top$ using unification) or not (and thus that its opposite is satisfiable or not).

Finally, the main result of this paper is a consequence of all the former properties.

Theorem 3.6 (Main Theorem) *Satisfiability of well-formed constraints is decidable.*

This result is not really new, it could already be found in [1] for instance. However, the demonstration used here seems to be easier to adapt to other cases. In particular, having first order symbols like f, this theorem will apply quite directly to show decidability of opacity in a next section.

An Example: Needham-Schroeder Constraint

The former rewriting system has been implemented in a prototype using ocaml. However, it has been adapted so that it does not have to test all the possible unifications. This makes the method easier to understand but is very negative for performance. This is the reason why our algorithm does only make unification when it need them. For example, the rules 3 and 4 would be replaced by a rule trying all the possible unification of $f(..)$ and the reachable patterns starting with f in the environment. The advantage of this method is that by looking at the unification requested by the rewriting system, we obtain values for the different parameters and this algorithm does not have to look to all possible unifications but only to "plausible ones". In the formal proof and other works, the general first idea is to guess all possible equalities between sub-messages and to get rid of them using the most great unifier (see [7] on how to adapt this to exclusive or). It has been tested on the Needham-Schroeder constraint. Its result is of course that this constraint is satisfiable and the only possible variables' values are: $x = N_C$ and $y = N_B$. During this check, five possible unifications have been tested. Three of them involved two messages that could not be unified and so only two cases were explored. One quickly lead to satisfiability whereas the other one was not satisfiable. Adding to the initial constraint the inequality $y \neq N_B$, we immediately have that the constraint is not satisfiable anymore. This gives us a possible fix for this protocol: if agent A was able to determine the origin of nonce N_B (by replacing it by a pair nonce, identity of the nonce's creator), then it would not be possible to perform this attack anymore. This is the idea behind Lowe fixed version of this protocol appearing in [10]. This fix adds the agent's identity to the second message, hence allows to check if the identity of the nonce's creator is valid.

4 Quasi Well-Formed Constraints

The previous theorem could be extended to quasi well-formed constraints. There are at least two possible ways to demonstrate this.

- The first solution is to notice that the only place where we use the first hypothesis is to prove that satisfiability for normal forms is decidable. This is the reason why we added the last hypothesis for quasi well-formed constraints, thus there exists a closed message common to all the environments. The modifications to perform are to demonstrate the modified property:

  Proposition 4.1 *If C and C' are constraints such that $C \rightarrow C'$ and C is quasi well-formed, then C' is quasi well-formed.*

  Termination is proved for any kind of constraints so this will not be a problem. Correctness and completeness could be adapted too.

  Proposition 4.2 *The rewriting system $\rightarrow$ is correct and complete for quasi well-formed constraints.*

  Eventually, when considering the normal form, the proof does not change: using tuples of message m, it is possible to verify satisfiability as message m is deducible from any of the involved environments.
- The other way is to remark that a quasi well-formed conjunction is equivalent to a conjunction of well-formed conjunctions. In the initial conjunction, the atomic constraint $T \Vdash m[U]$ could be replaced by $T \Vdash m[U] \wedge \bigwedge T' \Vdash m'[U']$ where the conjunctions introducing the different variables in T are added and those introducing variables in the different m' (to formalize that, we would need a fix point). As the number of conjunctions in the initial constraint is finite, the results are well-formed and finite conjunctions. The rest of this demonstration will be shorter as it will use the results of the former section. This is why, these conjunctions could be rewritten. And so, we obtain an equivalent normal form which is a quasi well-formed constraint. As we have seen before, satisfiability of this constraint is rather easy to check. There remains the inequalities part to check but this could still be done using the method described above.

  Both of these proof skeletons allow us to prove the following theorem.

Theorem 4.3 *Satisfiability of quasi well-formed constraints is decidable.*

More than satisfiability, we provide a decision procedure that has been implemented and seems to perform quite well.

An Example: Needham-Schroeder Constraint

As said earlier, our main motivation for this extension is its application to opacity. However, there are other possible uses for it. The Environment Inclusion hypothesis held because the environments represented the intruder's knowledge. As the intruder could only learn new messages, this set could

only grow and the hypothesis is not restrictive. However, by suppressing this hypothesis, it is now possible to model distinct knowledge with the different environments. Concretely, this extension easily allows to model two intruders that are not able to communicate between each other. For example, let us consider a modified version of Dolev-Yao attack. There are two intruders C_1 and C_2 such that there is a session between A and C_1 and another one between C_2 (usurping A's identity) and B. We want to know if it is possible for one of the intruder to deduce one of the secret N_A or N_B. Let T_0 be the initial knowledge of our intruders $\{A, B, C_1, C_2, K_{C_1}, K_{C_2}, K_{C_1}^{-1}, K_{C_2}^{-1}\}$. Then, the constraint becomes:

$$
\begin{array}{ll}
 & T_0 \Vdash \{A, x\}_{K_B} \\
\wedge & T_0, \{A, N_A\}_{K_B} \Vdash \{x', y'\}_{K_A} \\
\wedge & (\ T_0, \{x, N_B\}_{K_A} \Vdash N_B \\
 & \vee\ T_0, \{A, N_A\}_{K_B}, \{y'\}_{K_{C_1}} \Vdash N_A)
\end{array}
$$

This constraint appears to be quasi well-formed but is not well-formed. By using our decision procedure, we know that this constraint is not satisfiable. Thus, the two intruders need to communicate so that the classical attack over Dolev-Yao's protocol could be performed.

5 NP-Completeness

We will now discuss the complexity of our approach. First, satisfiability of well-formed constraints is NP-hard (see for example [15] or [14]). This is the case for well-formed constraints that only involve $\Vdash$. And as all of these constraints are in the set of constraints that we are studying, our satisfiability's problem is NP-hard. To show that this satisfiability problem is in NP and thus NP-complete, we will rely on the results presented in [14]. The authors of this paper proved that given a satisfiable well-formed constraint without inequalities, there exists a model which size is polynomial in the size of the constraint. The size used here is the number of different sub-terms. The same result holds for our method. Its result is a substitution σ such that the size of $x\sigma$ is polynomial in the size of the constraint. When solving the inequalities part, the size remains polynomial: let $|m|_{DAG}$ be the number of distinct sub-terms in m. Then it is easy to prove that $|m^n|_{DAG} = |m|_{DAG} + n - 1$. So in the worst case, we only added the number of inequalities and variables to the size of our model. Thus the DAG size of our model remains polynomial in the size of the initial problem. That is why, we could conclude that satisfiability of well-formed constraints is NP-complete.

For quasi well-formed constraints, the exact same decision algorithm could be used. This proves that satisfiability for these constraints is also NP-complete.

6 Application to Opacity

In this section, we will give a quick look at how the former method could be used to prove that opacity is decidable when considering Dolev-Yao model. In other papers, we only studied the case of a passive intruder. Here, we will start to extend decidability results to the case of active intruders.

First, let us give an extended definition of $\vdash$ to the case of two environments and two messages $E, E \vdash n, n'$. This means that n and n' are deducible from E and E' by making the same operations on both environments. For example, $\{a, b\}, \{c, d\} \vdash \langle a, b\rangle, \langle c, d\rangle$ is true as both messages are obtained by pairing the first and second messages in the environments whereas $\{a, b\}, \{c, d\} \vdash \langle a, a\rangle, \langle c, d\rangle$ is false as the messages could not be obtained by the same operations. $E, E \vdash n, n'$ is defined on closed messages n, n' and closed environments E and E' by:

$$\frac{}{\{n_1, ..., n_k\}, \{n'_1, ..., n'_k\} \vdash n_i, n'_i} \quad \frac{E, E' \vdash n_1, n'_1 \quad E, E' \vdash n_2, n'_2}{E, E' \vdash \langle n_1, n_2\rangle, \langle n'_1, n'_2\rangle}$$

$$\frac{E, E' \vdash \langle n_1, n_2\rangle, \langle n'_1, n'_2\rangle}{E, E' \vdash n_1, n'_1} \quad \frac{E, E' \vdash \langle n_1, n_2\rangle, \langle n'_1, n'_2\rangle}{E, E' \vdash n_2, n'_2}$$

$$\frac{E, E' \vdash \{n_1\}_{n_2}, \{n'_1\}_{n'_2} \quad E, E' \vdash n_2, n'_2}{E, E' \vdash n_1, n'_1} \quad \frac{E, E' \vdash n_1, n'_1 \quad E, E' \vdash n_2, n'_2}{E, E' \vdash \{n_1\}_{n_2}, \{n'_1\}_{n'_2}}$$

Note that E and E' must have the same length (in fact, as these sets will be supposed similar, this will not be a problem). Moreover, as we want to perform the same operations on both environments, we need the environments to be ordered. They could be represented using lists.

The definition of opacity (given for example in [11]) is that an intruder is not able to distinguish a run where the property is satisfied from a run where it is not. To distinguish two messages, the intruder could decompose them, according to his knowledge but if he does not know the key k for example, he will not be able to make the difference between two different messages encoded by this key k. However, in this section, we will only consider a very simplified version of opacity. Let us consider a finite protocol P with a unique parameter v (called the vote) that could only take two values *yes* (protocol P_y) or *no* (protocol P_n). We will say that the value of v is not opaque (in fact, the property $v = yes$ is not opaque) iff the intruder is able to produce *yes* and *no* by doing the same operations on both protocols. The intruder has to perform the exact same sequence of actions: for example if in the first protocol, he sends a constant A then receives a message that is a pair and eventually sends the left part of the pair, he will have to do the same actions for the second protocol. This will be equivalent to satisfiability of the following constraint:

$$T_0, T_0 \Vdash m_0, m'_0 \wedge T_0; n_0, T_0; n'_0 \Vdash m_1, m'_1 \wedge ... \wedge T_n, T'_n \Vdash yes, no$$

Where $T_i = T_0; m_0; ...; m_{i-1}$. The messages m_i and m'_i are produced by the intruder and sent to protocols P_y and P_n. Then, the protocols replies are

messages n_i and n'_i. Both sessions are carried in the same manner and we ask that at the end, the intruder is able to tell on which session v was equal to *yes* and on which session v was equal to *no*.

To link our new constraint to Dolev-Yao constraints, let us introduce an explicit binary function f for our extended $\vdash$ operator. The constraint could be written

$$f(T_0, T_0) \Vdash f(m_0, m'_0) \wedge f(T_0; n_0, T_0; n'_0) \Vdash f(m_1, m'_1) \wedge ... \wedge f(T_n, T'_n) \Vdash f(yes, no)$$

Next, we will distribute the f function inside our constraint by using the following *distrib* operator recursively defined by:

$$distrib_f(f(\langle m, n\rangle, \langle m', n'\rangle)) = \langle distrib_f(f(m, m')), distrib_f(f(n, n'))\rangle$$

$$distrib_f(f(\{m\}_n, \{m'\}_{n'})) = \{distrib_f(f(m, m'))\}_{distrib_f(f(n,n'))}$$

Else we have $distrib_f(m) = m$.

Applying the $distrib_f$ function creates a constraint equivalent to the original one but which is a Dolev-Yao constraint. This is formalized in the following property which could be proved using induction over the proofs' structures. If E and E' are two environments, n and n' are two messages such that f does not appear in E, E', n or n', then the following equivalence is true:

$$E, E' \Vdash n, n' \Leftrightarrow distrib_f(f(E, E')) \Vdash distrib_f(f(n, n'))$$

So, opacity is equivalent to satisfiability of a Dolev-Yao constraint. Moreover, this constraint is, by construction, well-formed. Thus, opacity as defined before is decidable and we have an NP-complete algorithm to check satisfiability. However, we only studied a very restrictive version of opacity but we believe that this result could be extended to the general opacity problem.

7 Conclusion

In this paper, we gave a simple decision procedure to prove satisfiability of extended versions of usual Dolev-Yao constraints. Using a rewriting system, this algorithm is easy to implement. Moreover, we believe that this method could easily be used for other constraints derived from Dolev-Yao's one. This has been done through a prototype that performed well on simple examples. An immediate application of these new predicates is to check the secret in presence of two intruders that are not allowed to communicate. Then quasi well-formed constraint will be useful as they allow to describe two "branches" of environments that do not have to be ordered by set inclusion. Each of these branches will be described by a well-formed constraint. Further works include of course linking more precisely these constraints to the opacity problem in the case of an active intruder. However, there remains an open question: is satisfiability of general Dolev-Yao constraints decidable ? We did not manage to prove that in the general (no hypothesis at all) case.

Acknowledgment

The author wishes to thank Romain Janvier for his reading of a preliminary version of this paper and Hubert Comon for his DEA course "Cryptographic Protocols' Proofs": the decision method used here has been inspired by the content of this course during year 2003.

References

[1] R. M. Amadio and D. Lugiez. On the reachability problem in cryptographic protocols. In *International Conference on Concurrency Theory*, volume 1877 of *LNCS*, pages 380–394, 2000.

[2] The Avispa Project. http://www.avispa-project.org/, 1999.

[3] L. Bozga, Y. Lakhnech, and M. Périn. Abstract interpretation for secrecy using patterns. Technical report, EVA : http://www-eva.imag.fr/, 2002.

[4] E.M. Clarke, S. Jha, and W. Marrero. Using state space exploration and a natural deduction style message derivation engine to verify security protocols. In *IFIP Working Conference on Programming Concepts and Methods*, 1998.

[5] H. Comon-Lundh. and V. Cortier. Security properties: Two agents are sufficient. Technical report, LSV, 2002.

[6] H. Comon-Lundh and V. Cortier. New decidability results for fragments of first-order logic and application to cryptographic protocols. In *14th Int. Conf. Rewriting Techniques and Applications (RTA'2003)*, volume 2706 of *LNCS*, 2003.

[7] Hubert Comon-Lundh and Vitaly Shmatikov. Intruder deductions, constraint solving and insecurity decision in presence of exclusive or. In *Proceedings of the eighteenth annual IEEE symposium on Logic In Computer Science*. IEEE Computer Society Press, June 2003.

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

[9] Jean Goubault-Larrecq. A method for automatic cryptographic protocol verification. In *International Workshop on Formal Methods for Parallel Programming: Theory and Applications*, volume 1800 of *LNCS*, 2000.

[10] G. Lowe. An attack on the Needham-Schroeder public-key authentification protocol. *Information Processing Letters*, 56(3):131–133, 1995.

[11] L. Mazaré. Using unification for opacity properties. In *Proc. of the Workshop on Issues in the Theory of Security (WITS'04)*. To appear, 2004.

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

[13] R.M. Needham and M.D. Schroeder. Using encryption for authentication in large networks of computers. *Communications of the ACM*, 21(12):993–999, 1978.

[14] M. Rusinowitch and M. Turuani. Protocol insecurity with finite number of sessions is NP-complete. In *IEEE Computer Security Foundations Workshop*, 2001.

[15] Michaël Rusinowitch and Mathieu Turuani. Protocol insecurity with a finite number of sessions and composed keys is np-complete. *Theor. Comput. Sci.*, 299(1-3):451–475, 2003.

Attacking Group Multicast Key Management Protocols Using CORAL

Graham Steel [1,2] Alan Bundy [3]

School of Informatics,
University of Edinburgh,
Edinburgh, EH8 9LE, Scotland

Abstract

This paper describes the modelling of a two multicast group key management protocols in a first-order inductive model, and the discovery of previously unknown attacks on them by the automated inductive counterexample finder CORAL. These kinds of protocols had not been analysed in a scenario with an active intruder before. CORAL proved to be a suitable tool for a job because, unlike most automated tools for discovering attacks, it deals directly with an open-ended model where the number of agents and the roles they play are unbounded. Additionally, CORAL's model allows us to reason explicitly about lists of terms in a message, which proved to be essential for modelling the second protocol. In the course of the case studies, we also discuss other issues surrounding multicast protocol analysis, including identifying the goals of the protocol with respect to the intended trust model, modelling of the control conditions, which are considerably more complex than for standard two and three party protocols, and effective searching of the state space generated by the model, which has a much larger branching rate than for standard protocols.

Key words: Security protocol analysis, group multicast key management

1 Introduction

In terms of analysing the standard corpus of two and three party protocols given in [7], the field of cryptographic security protocol analysis can be said to be saturated. Much research attention has now turned to trying to widen the scope of the techniques, e.g. to group protocols, [14,27,24]. The term 'group protocol' refers

[1] The work reported in this paper was carried out while the first author was supported by the European 'Calculemus' scheme as a visiting researcher at the University of Karlsruhe, Germany, and the University of Genova, Italy.
[2] Email: graham.steel@ed.ac.uk
[3] Email: bundy@ed.ac.uk

This is a preliminary version. The final version will be published in Electronic Notes in Theoretical Computer Science
URL: www.elsevier.nl/locate/entcs

to a protocol (or suite of sub-protocols) for establishing a secure key between an unbounded number of agents. This may take the form of a single key-establishment run between the agents, or a series of requests to join and leave a group with associated key updates. Some protocols involve a trusted key server. Significant attacks on group protocols have appeared in the literature, but these have been discovered by patient pen and paper analysis, [22]. Attempts to extend automatic tools to analyse such protocols have had mixed results (see §2). Taghdiri and Jackson, [27], published an analysis of a protocol for group multicast key management proposed by Tanaka and Sato, [28]. They were able to find major flaws and proposed a new, improved version of the protocol. However, their model was rather weak: in particular, it did not include an active attacker. In this paper, we investigate their improved protocol using CORAL, [24], and show how it discovered two new attacks just as serious as the original ones. Additionally we look at the Iolus key management protocol, [18], which was analysed in Taghdiri's MSc thesis, [26], and thought to be secure. Again CORAL was able to find an attack.

To carry out the analysis, CORAL's model had to be developed to include state information not always included in the trace of messages exchanged, such as the current group key held by the key server, and the current time, expressed as the number of events that have taken place in the trace so far. The reasons for this will be made clear in the description of the case studies below.

In successfully modelling and attacking these protocols, we have given further evidence for the utility of the CORAL approach; though run times are typically slow, we can very quickly adapt the Horn-clause model to new kinds of protocol. Additionally, the attacks discovered are significantly longer than those typically found in the Clark-Jacob corpus, [7], and those found by CORAL before, [24]. To achieve really fast attack discovery times on these protocols, it should be possible to adapt purpose built protocol analysis tools, though significant work may be required. We examine this possibility in §5.

The paper is organised as follows: in §2, we review previous work on group protocol analysis. In §3 we describe the first case study, on Taghdiri and Jackson's improved version of the Tanaka-Sato protocol. The second case study, on the Iolus protocol, follows in §4. Throughout the case studies, we highlight some general issues about the modelling of multicast protocols. These points are summarised in §5, where we discuss the lessons learned and the problems that remain to be solved. §6 contains conclusions and plans for further work.

2 Background

The field of cryptographic security protocol analysis is now far too large to cover here. A recent survey can be found in [15]. In this section, we will briefly survey previous work specific to group protocol analysis.

Perhaps the first formal analysis of a group protocol was by Paulson. He investigated a recursive authentication protocol proposed by Bull and Otway, [6], as part of his development of the inductive method, [21]. Paulson was able to model

it in the general case, i.e. for any number of participants, and prove some security properties for arbitrarily-sized groups. Paulson's method uses a typed higher-order logic formalism, and poses security properties as conjectures about properties of the trace. These properties are then proved by induction on traces in the interactive theorem prover Isabelle/HOL. However, if there is a flaw in the protocol, Paulson's method provides no automated support for finding the attack.

CORAL, [24], has been used to discover attacks on the Asokan–Ginzboorg protocol for key establishment in an ad-hoc wireless network, [2]. CORAL uses a first-order version of Paulson's protocol model, and searches for counterexamples to the security property under consideration, i.e. it searches for attacks when the protocol is flawed. Inference is carried out by an adapted version of the first-order theorem prover SPASS, [29], using the refutation complete 'proof by consistency' strategy, [8]. CORAL searches the infinite model of the system directly, with no parameterisation with respect to sessions, roles, size of the group etc. In addition to standard subsumption and tautology checking, a small set of domain-specific reduction rules are used to prune the search space. The use of an inductive model allowed the specification of the Asokan–Ginzboorg protocol for a group of unbounded size, which lead to the discovery of distinct attacks on groups of size two and then three while investigating the same security property.

Several significant attacks on the CLIQUES protocol suite, [3], were found by Pereira and Quisquater as a result of a programme of manual analysis, [22]. The CLIQUES protocol suite contains a number of protocols for establishment of groups, and groups within groups, each with their own secure key. Extensive use is made of Diffie-Hellman style exponentiation, [10]. Pereira and Quisquater proposed a method for converting the problem of the intruder obtaining a particular term to the solution of a system of linear equations. Using this method, they were able to find weaknesses in every protocol they examined. Often, the attacks involved quite imaginative behaviour by the intruder, e.g. being accepted as a member of a group of size 4, and then using the values learnt in that key establishment session to force a group of size 3, intended to exclude him, to use a key that he knows. One clear lesson from Pereira and Quisquater's work is that the design of these kinds of protocols is extremely tricky. It is easy to understand how the designers of the CLIQUES protocols failed to see such attacks. This strengthens the case for the use of formal methods in their design.

Meadows used the NRL protocol analyser (NPA) to tackle the Group Domain of Interpretation (GDOI) protocol suite, as part of an effort to introduce formal methods to the design stage of protocol development, [16]. GDOI is a method for group key management involving a trusted key server. NPA could not handle the infinite data structures required for a general model of the protocol, so a concrete abstraction of the protocol was made, restricting phase two to the distribution of one 'security association key'. NPA was able to find a type flaw in the protocol, which was fixed in later versions of GDOI. However, Meadows' attempt to use NPA to rediscover the Pereira-Quisquater attacks on the CLIQUES suite was less successful, [14]. NPA was extended to handle the Diffie-Hellman exponentiation

operation, but could not find the attacks described in [22].

The designers of the protocol specification language CAPSL, [9], have recently turned their attention to group protocols in the development of an extended language, MuCAPSL, [17]. After translating the GDH.2 protocol, [25], to their intermediate language MuCIL they were able to discover a type flaw. However, their analysis requires the number of group members to be set in advance, which can prejudice the chances of discovering an attack. The designers of the HLSPL protocol language, [23], have also been working on extending their coverage to group protocols, though no results of this work were available at the time of writing this paper.

Taghdiri and Jackson, [27], reported results from the modelling of a multicast key management scheme proposed by Tanaka and Sato, [28]. This protocol was designed for a scenario where the group is highly dynamic, i.e. agents join and leave the group frequently. The idea was to minimise the number of key updates required by supplying keys to agents on demand. Taghdiri and Jackson formalised a model for the protocol in the Alloy specification language, [12], and used Alloy's SAT checker to search for counterexamples to desirable properties of the protocol. Several counterexamples were found, the most serious one indicating that current members of the group will accept as valid messages broadcast by ex-members of the group. Taghdiri and Jackson proposed an improved protocol. However, their formal protocol model differed from the norm established over the last 25 years in that no active attacker was included. In the rest of this paper, we explain how we modelled and analysed both Taghdiri and Jackson's improved protocol, and the Iolus key management protocol treated in Taghdiri's MSc thesis (and considered to be secure), [26], using our inductive counterexample discovery tool, CORAL. We discovered that neither of these protocols are secure in the presence of an active attacker, even if he is weaker than the Dolev-Yao intruder generally used in automated protocol analysis, [11].

3 The Tanaka–Sato/Taghdiri–Jackson Protocol

The protocol that Tanaka and Sato originally proposed, [28], was primarily concerned with minimising the burden of key updates in terms of network traffic and processor time. Two main design features were introduced for this purpose: the first was the division of the group into subgroups, each under the management of a key distribution server (KDS). The communication between the KDSs is assumed to be not only secure but also conducted under a reliable totally ordered multicast protocol (RTOMP). Taghdiri and Jackson, [27], modelled this by assuming that as soon as one KDS updates its key, all the other KDSs instantaneously update theirs, effectively reducing the model to a single server. In our model, we also restrict attention to a single server. The second design feature of the protocol is that agents retain a list of keys rather than just one key. They discard an old key as invalid t units of time after having received a more up-to-date key, where t is set with respect to the delay in the network. Keys are distributed only when an agent sends a request

to the server. An agent will make such a request when he wants to send a multicast message, or if he receives a message encrypted under a key he doesn't know. In both cases, he will send a message to the server giving the ID number of the newest key he has, and the server will send back all newer keys. Only the newest key is used for multicast broadcasting.

This retention of a list of keys was shown in Taghdiri and Jackson's analysis to lead to major security problems. The most serious attack involved members of the group accepting messages from a principal outside the group. A member of the group A can simply broadcast a message from inside the group, leave, and then broadcast a message using the same key. Though the group key has been updated as a result of A leaving, the other agents in the group will still accept the second message as valid as they all have the old key. To counter this, Taghdiri and Jackson suggested changes to the protocol. Each agent should retain only the most recent key he has received, and upon receiving a multicast message, should contact the server to confirm that it is encrypted under the newest key. This may result in some message loss, because delays in the network might mean that by the time a multicast message has been received and a key request sent to the server, the group key has changed, but this was reckoned to be acceptable compared to the potential security breach.

It is the improved version of the protocol we have modelled and analysed using CORAL. In doing so we were aiming to address one major oversight of the Taghdiri–Jackson analysis, namely the lack of an active intruder in their model. An active intruder of some kind has been assumed since the very first security protocol paper, [20]. His behaviour was formalised by Dolev and Yao, [11], and since then it has generally been accepted that the spy should also be able to pose as a legitimate agent, as for example in Lowe's famous attack, [13]. If anything, compared to a unicast protocol, it would seem even more likely that a multicast protocol would be subject to attack by an active intruder, as argued in [18]. There are inherently more opportunities for interception of traffic, and the 'crowd' of principals would typically make it easier for an intruder to pose as another legitimate principal. Such protocols should therefore be subjected to analysis under the full Dolev-Yao attacker model, as is standard for unicast protocols.

The Tanaka-Sato protocol assumes the existence of a unicast authentication protocol that allows the server to establish an individual key (IK) with a new member joining the group. This IK is used to encrypt all communication between that member and the server. We model the underlying authentication protocol by assuming the existence of a long-term key shared by each valid potential member of the group with the KDS. Since we are looking for attacks on the protocol rather than trying to verify it, we can easily justify this. We can simply take the attacks we discover and examine them to see if the specific way we implemented the authentication phase was exploited. The attacks described in this paper would be effective for any initial authentication protocol. Additionally, we make the standard assumption that the spy has access to a valid long-term key.

Here is a description of the improved version of the protocol as described by

Taghdiri and Jackson, and as modelled in this paper:

Joining the Group

1. $M_i \rightarrow S : \{\!|$ join $|\!\}_{K_{M_i}}$
2. $S \rightarrow M_i : \{\!| Ik_{M_i}, Gk(n) |\!\}_{K_{M_i}}$

In message 1, M_i wants to join the group, so sends a join request under his long-term key K_{M_i}. The server generates a fresh individual key , $Ik_M i$, and a new group key $Gk(n)$. Each group key has a unique ID number (n). The new individual key and group key are sent to the joining member in message 2.

Leaving the Group

1. $M_i \rightarrow S : \{\!|$ leave $|\!\}_{Ik_{M_i}}$
2. $S \rightarrow M_i : \{\!|$ ack.leave $|\!\}_{Ik_{M_i}}$

In message 1, M_i sends a request to leave encrypted under his individual key Ik. The server acknowledges the leave in message 2, and generates a new group key. The new group key is not distributed yet though. In fact, if another membership change occurs before a request for a key is received, it will never be distributed.

Sending a message

1. $M_i \rightarrow S : \{\!|$ send, $n |\!\}_{Ik_{M_i}}$
2. $S \rightarrow M_i : \{\!| n', Gk(n') |\!\}_{Ik_{M_i}}$
3. $M_i \rightarrow ALL : \{\!|$ message $|\!\}_{Gk(n')}$

In message 1, agent M_i signals to the server that he would like to send a message by sending what the protocol designers call a 'sequence request' message together with the ID number of the newest key he has, n. The server checks that M_i is in the group, and then sends back the newest key $Gk(n')$. If no joins or leaves have occurred since M_i last received a key, it may be that $n = n'$, but this will not be the case in general. In message 3, agent M_i broadcasts his message to the group.

Receiving a message

1. $M_j \rightarrow S : \{\!|$ read, $n |\!\}_{Ik_{M_j}}$
2. $S \rightarrow M_j : \{\!| Gk(n') |\!\}_{Ik_{M_j}}$

Suppose a multicast message has been broadcast, as in message 3 of the 'sending a message' fragment above. When another agent M_j receives the message, he first sends a request to the server for the newest key. He then receives the newest key $Gk(n')$, and will only accept the multicast message if it was encrypted under that key.

Commentary
The revised protocol as proposed by Taghdiri and Jackson contains some redundancy as a result of their security improvements. For example, there is no reason for the server to send the key to a new member when he joins, since he is required to ask for a key update whenever he sends or receives a multicast message. Additionally, the sequence number sent in the request for a key update before sending a message also seems redundant. Previously, the server would have used it to decide which keys to send back, but in the revised version, the server only ever sends back the most recent key. It would be better to replace this with a nonce, as we argue after presenting the attacks we discovered, in §3.2.

3.1 Modelling in CORAL

CORAL's model is a first-order version of Paulson's inductive model, [21]. We enforce strong typing on message elements by using a sorted signature, with unary functions acting as sort constraints. A protocol is modelled as the set of all possible traces, i.e. all possible sequences of messages sent by any number of honest users under the specification of the protocol and, additionally, faked messages sent by the intruder. A trace of messages is modelled as a list. Horn clauses define how a valid trace may be extended by honest agents as specified by the protocol. Further clauses model the knowledge the intruder can learn from previous messages in the trace, using the same $synth$ and $analz$ operators that Paulson uses. To search for attacks, we pose conjectures about security properties exactly as Paulson does. CORAL then searches for counterexamples, effectively by a backwards search, i.e. it goes from a state violating the security property backwards to see if there is a valid trace reaching that state. More details of our model and the operation of CORAL are available in [24].

A feature of our model was that all the information about the state of the system, i.e. the state and knowledge of all the principals involved, was stored in the trace and inferred from the trace each time it was needed. First-order rules can then be used to add an arbitrary number of messages to the trace in a single instant. This was particularly useful when we were modelling the Asokan–Ginzboorg group key agreement protocol, as it allowed us to create a general model for any group size, [24]. However, for the Taghdiri-Jackson protocol, this was not so helpful. Some information about the state of the system does not normally appear in the trace. For example, when an agent leaves the group, the server generates a new key, but this key does not appear in the trace. Additionally, our model follows Paulson's original design in that is does not include a 'gets' event to model message reception, such as Bella later introduced to Paulson's model, [5]. The only events in the trace are 'sent' events, and in the presence of a Dolev-Yao attacker, these messages may never be received by their intended recipients. This makes it hard to work out who is legitimately in the group at a particular time, which is vital when modelling the control conditions, i.e. tests that honest agents apply before sending a protocol message. Even with a 'gets' event, a lot of digging through message

trace would be required to determine group composition, and we would need to do this almost every time an agent sent a message, creating an enormous search problem. So, the model was changed to include some information about the state of the principals. The unary function $m()$ that was previously used to store just the message trace is now an arity 4 function storing the trace, a counter, the current group key stored by the server, and the composition of the group stored as a list of triples. The triples store the agents name, the individual key which he shares with the server for this session in the group, and the most recent group multicast key he has received. We define a boolean function $ingroup$ on these lists of triples that determine whether or not a particular agent is in the group. A further change is our modelling of freshness. We used to use the $parts$ operator as used by Paulson, but in our model for this protocol, we have a counter, and use this to model fresh values. Our motivation for this was that so many fresh values have to be created in a typical scenario, for individual keys, group keys and multicast messages, that our checking of the $parts$ literals would quickly slow down the search process. We model multicast messages as $hello(T)$, where T is the counter value when the message was sent, thus ensuring all (honestly sent) messages are unique.

Having chosen to use a counter-based model, we introduced a new pruning rule to CORAL: if the counter variable occurs in term X inside an antecedent literal $ingroup(X, Y, Z) = true$, then the clause is redundant, since this would require an agent at some point in the past to have joined the group and obtained a group key or individual key that is only available now. A similar check is applied to $member(X, Y) = true$ literals. This eliminates a lot of unreachable states from the search. This rule could be generally applied to backward searching tools using a tick based model.

As an illustration, in Figure 1, we give the clauses required for modelling the sub-protocol for the sending of multicast messages. Note that in a further change from our original model used in [24], we record the composition of the group at each point in time in the fourth argument of the $sent$ constructor. This is important for making conjectures about security properties later on. Note also that $ingroup$ is an arity 3 function, with the third argument returning the list of group members without the agent named in the first argument. This is used when agents leave the group or update their keys, as in the third clause in Figure 1. A further point to note is that we still infer state information about principals from the trace, for example to decide if they should be expecting a key update message in the third clause. Our new model is something of a hybrid between a Paulson style trace model and a state-based model like that used, for example, in [4].

3.2 Attacking the Protocol

In [22], Pereira and Quisquater attempt to lay down a list of desirable security properties for group protocols. They define *implicit key authentication*, that an outsider cannot learn the group key; two flavours of *perfect forward secrecy*, i.e. that the compromise of long-term keys does not compromise past session keys;

%% *SEND a message*
m(Trace,Group,Keysequence,Tick)=true ∧
ingroup(triple(principal(Mi),Ikey,key(Sq)),Group,Newgp)=true
→ m(cons(sent(Mi,server,encr(send(Sq),Ikey),Group),Trace),Group,
Keysequence,s(Tick))=true
%% *server gives key*
m(Trace,Group,Keysequence,Tick)=true ∧
ingroup(triple(principal(Mi),Ikey,Oldk),Group,Newgp)=true ∧
member(sent(X,server,encr(send(Sq),Ikey),Tgroup),Trace)=true
→ m(cons(sent(server,Mi,encr(pair(key(Keysequence),send(Sq)),Ikey),group),Trace),
Group,Keysequence,s(Tick))=true
%% *agent broadcasts his message, updates his key*
m(Trace,Group,Keysequence,Tick)=true ∧
ingroup(triple(principal(Mi),Ikey,Oldk),Group,Newgp)=true ∧
member(sent(X,Mi,encr(pair(key(Xk),send(Sq)),Ikey),Tg1),Trace)=true ∧
member(sent(Mi,server,encr(send(Sq),Ikey),Tg2),Trace)=true
→m(cons(sent(Mi,all,encr(hello(s(Tick)),key(Xk)),
cons(triple(principal(Mi),Ikey,key(Xk)),Newgp)),
Trace),cons(triple(principal(Mi),Ikey,key(Xk)),Newgp),Keysequence,s(Tick))=true

Fig. 1. Clauses for modelling the 'send' sub-protocol

and *resistance to known-key attacks*, i.e. that compromise of session keys does not lead to the loss of future session keys. However, the properties Taghdiri and Jackson found not to be satisfied by the original protocol design fall outside of this categorisation. Essentially, this is because we are analysing a protocol for managing a group key for an evolving group, not just establishing a key for a static one.

The property vital to a key management protocol is that throughout the evolution of the group, agents currently outside the group should not be accepted as group members by the agents inside the group. We could perhaps call this *multicast group authenticity*. For this protocol, this property has two flavours: the first, which Taghdiri and Jackson call 'outsider can't read', implies that no agent outside the group should be able to read a message sent by a member of the group. The second, which they call 'outsider can't send', implies that members of the group should not accept as valid a message sent from outside the group.

Posing security conjectures in an inductive formalism requires some thought. We must translate an abstract idea of authenticity into a property expressed in terms of messages in the trace. For group protocol properties, our trace includes the composition of the group at each point, which becomes important now. For the property 'outsider can't read', we need to express the fact that when our dishonest agent is indeed outside the group, and a message is sent by an honest player under a key Gk, the spy does not know this key. So, we posed the property as a conjecture to CORAL in this form:

```
% For honest agent Mj
eqagent(Mj,spy)=false ∧
% There is a trace containing the sequence of
% messages for Mj broadcasting to the group under Gk
m(cons(sent(Mj,all,encr(hello(Y),Gk),Xgroup),
   cons(sent(X,Mj,encr(pair(Gk,send(Sq2)),Ikey),Xgroup),
   cons(sent(Mj,server,encr(send(Sq2),Ikey),Xgroup),
   Trace))),Group,Keyseq,Tick)=true ∧
% and the spy is not in the group
ingroup(triple(principal(spy),X3,X2),Xgroup,Newgp)=false ∧
% but the spy has the key Gk
in(Gk,analz(Trace)=true →
```

This conjecture is negative, i.e. it states there should be *no* trace $Trace$ ending with the 3 messages specified in the first literal, with the spy outside the group, and with the message $hello(y)$ being sent under a key the spy knows ($analz(X)$ is the set of terms the spy can learn from a trace X). The three final messages had to be specified together because otherwise CORAL (correctly) finds a rather trivial attack where the spy leaves the group between the server sending a key update out to Mj and Mj broadcasting his message. Then he can read the message quite legitimately, since he was in the group when it was sent. Given the above conjecture, CORAL gives the counterexample in Figure 2. This is an attack on the protocol which hinges on the spy sending a replayed key update message in message 13. Since the key may or may not have changed since she last saw it, agent a will accept this key. The problem is that there is minimal freshness information sent in the request for a key (just the sequence number of the key an agent currently holds). Enclosing a fresh nonce inside the package sent to the server requesting a key update would blunt this attack.

Having discovered this attack, we realised that there is a similar one whereby a spy can send a message from outside the group and have it accepted by an agent inside the group, thus breaking the multicast group authenticity property, 'outsider can't send'. We gave an appropriate conjecture to CORAL for confirmation, and it discovered the counterexample in Figure 3. Like the previous attack, this is also a replay attack, in this case with the spy replaying message 8 in message 13, tricking agent a into thinking that message 11 came from a legitimate member of the group. Replay attacks are in general more serious in the context of a group key management protocol. A replay attack on a standard unicast protocol typically assumes that it would be possible for a spy to obtain a short-term key by cryptanalysis or some other means, and so it would constitute an attack on the protocol if he was able to force an agent to accept an old key. In the two attacks above, no cryptanalysis in necessary, since the spy can obtain some old keys by joining the group legitimately, and then leave before effecting the attack. However, even if we assume the spy does not have access to a valid long-term key, and so cannot join the group, these replay attacks are still dangerous. If the spy obtains a short-term group

1.	spy	$\rightarrow$	server	:	$\{\!\|\ spy \|\!\}_{longtermK(spy)}$
2.	server	$\rightarrow$	spy	:	$\{\!\|\ ik(1), Gk(1) \|\!\}_{longtermK(spy)}$
3.	a	$\rightarrow$	server	:	$\{\!\|\ a \|\!\}_{longtermK(a)}$
4.	server	$\rightarrow$	a	:	$\{\!\|\ ik(3), Gk(2) \|\!\}_{longtermK(a)}$
5.	spy	$\rightarrow$	server	:	$\{\!\|\ send(1) \|\!\}_{ik(1)}$
6.	server	$\rightarrow$	spy	:	$\{\!\|\ Gk(2), send(1) \|\!\}_{ik(1)}$
7.	a	$\rightarrow$	server	:	$\{\!\|\ send(2) \|\!\}_{ik(3)}$
8.	server	$\rightarrow$	a	:	$\{\!\|\ Gk(2), send(2) \|\!\}_{ik(3)}$
9.	a	$\rightarrow$	all	:	$\{\!\|\ hello(9) \|\!\}_{Gk(2)}$
10.	spy	$\rightarrow$	server	:	$\{\!\|\ leave \|\!\}_{ik(1)}$
11.	server	$\rightarrow$	spy	:	$\{\!\|\ ackleave \|\!\}_{ik(1)}$
12.	a	$\rightarrow$	server	:	$\{\!\|\ send(2) \|\!\}_{ik(3)}$
13.	spy	$\rightarrow$	a	:	$\{\!\|\ Gk(2), send(2) \|\!\}_{ik(3)}$
14.	a	$\rightarrow$	all	:	$\{\!\|\ hello(14) \|\!\}_{Gk(2)}$

Fig. 2. First attack on the Tanaka-Sato/Taghdiri-Jackson Protocol

key by cryptanalysis, he can effect an attack without joining the group.

The second attack can also be prevented by adding a fresh nonce to the request for a key, this time for reading a message, and including it in the reply from the server. A complete listing of the protocol model file is available at http://homepages.inf.ed.ac.uk/s9808756/tanaka-sato/.

4 The Iolus Protocol

The main difference between the Iolus protocol and the Taghdiri-Jackson version of the Tanaka-Sato protocol is that Iolus eagerly distributes new keys, whereas Tanaka-Sato distributes keys only on demand, i.e. as and when members of the group want to send or read messages.

Joining the Group

1.	M_i	$\rightarrow$	S	:	$\{\!\|\ \text{join} \|\!\}_{K_{M_i}}$
2.	S	$\rightarrow$	M_i	:	$\{\!\|\ Ik_{M_i}, Gk_{n'} \|\!\}_{K_{M_i}}$
3.	S	$\rightarrow$	ALL	:	$\{\!\|\ Gk_{n'} \|\!\}_{Gk_n}$

Members join the Iolus protocol in the same way as for Tanaka-Sato, i.e. by use of a pairwise authentication protocol that we model with the use of a long-term key.

1.	a	$\rightarrow$	server	:	$\{\!\|\ a\|\!\}_{longtermK(a)}$
2.	server	$\rightarrow$	a	:	$\{\!\|\ ik(1), Gk(1)\|\!\}_{longtermK(a)}$
3.	spy	$\rightarrow$	server	:	$\{\!\|\ spy\|\!\}_{longtermK(spy)}$
4.	server	$\rightarrow$	spy	:	$\{\!\|\ ik(3), Gk(2)\|\!\}_{longtermK(spy)}$
5.	spy	$\rightarrow$	server	:	$\{\!\|\ read\|\!\}_{ik(3)}$
6.	server	$\rightarrow$	spy	:	$\{\!\|\ Gk(2)\|\!\}_{ik(3)}$
7.	a	$\rightarrow$	server	:	$\{\!\|\ read\|\!\}_{ik(1)}$
8.	server	$\rightarrow$	a	:	$\{\!\|\ Gk(2)\|\!\}_{ik(1)}$
9.	spy	$\rightarrow$	server	:	$\{\!\|\ leave\|\!\}_{ik(3)}$
10.	server	$\rightarrow$	spy	:	$\{\!\|\ ackleave\|\!\}_{ik(3)}$
11.	spy	$\rightarrow$	all	:	$\{\!\|\ hello(12)\|\!\}_{Gk(2)}$
12.	a	$\rightarrow$	server	:	$\{\!\|\ read\|\!\}_{ik(1)}$
13.	spy	$\rightarrow$	a	:	$\{\!\|\ Gk(2)\|\!\}_{ik(1)}$

Fig. 3. Second attack on the Tanaka-Sato/Taghdiri-Jackson Protocol

---

The server generates a fresh individual key , $Ik_M i$, and a new group key with ID n', $Gk_{n'}$. In message 2, the group key is sent to the new member, and in message 3, it is sent to the old members of the group under the old group key, Gk_n.

Leaving the Group

1.	M_i	$\rightarrow$	S	:	$\{\!\|$ leave $\|\!\}_{Ik_{M_i}}$
2.	S	$\rightarrow$	ALL	:	$[\{\!\|\ Gk_{n'}\|\!\}_{Ik_{M_j}} \ldots] \forall j \neq i, M_j \in$group

When a member M_i leaves, a new key $Gk_{n'}$ is generated, and sent to each member in the form of a broadcast list. The list contains the new group key encrypted under the pairwise session key of each member still in the group (the key cannot be broadcast under the old group key, because this would give it away to the leaving member).

Sending a message

1.	M_i	$\rightarrow$	ALL	:	$\{\!\|$ message $\|\!\}_{Gk(n)}$

The message is simply broadcast under the current group key.

4.1 Modelling Iolus

No changes were required to the framework of the CORAL model to formalise the Iolus protocol. Though the protocol distributes new keys eagerly rather than lazily, the operations are similar to those used in the Tanaka-Sato protocol. The most complex part of the model concerns the second message of the 'leave' sub protocol. Here we must model the generation of an appropriate key update list for an arbitrary group. This is a straightforward task in our first-order model. We define a recursive function `rekey` that generates an appropriate rekeying message for a given group and given fresh group key. This function works correctly in all modes of instantiation, i.e. given a rekeying message it will return an appropriate group and key. This is important in our backwards search process. The use of an auxiliary `rekey` function is similar to our use of the `all_msg2s_received` function in our model of the Asokan–Ginzboorg protocol, [24]. Being able to make these kinds of recursive calculations seems (perhaps unsurprisingly) to be important in modelling group protocols, and our first-order logic model is well suited to them.

4.2 Attacking Iolus

Having posed security conjectures for the Tanaka-Sato/Taghdiri-Jackson protocol above, formulating an appropriate conjecture for the Iolus protocol was easier. Again, we are investigating multicast group authenticity, i.e. that those outside the group cannot successfully impersonate those inside the group, either by sending or receiving messages. However, since keys are supplied eagerly, we do not have the same division of this property into 'outside can't read' and 'outsider can't send' conjectures. Instead we have to look at the possible ways keys can be updated. The desirable property is: when a key update to key Gk is accepted by a member of a group, and that group does not contain the spy, then the spy should not know the key Gk. We formulate this property with respect to the updates sent after a group member leaves like this:

% *For honest agent Mj*
eqagent(Mj,spy)=false $\wedge$
% *Mj accepts an update to key Gk*
m(cons(sent(X,all,cons(encr(Gk,Ikey),Rest),
  cons(triple(Mj,Ikey,OldGk),Restgp)),Trace),
  Group,Xk,Tick)=true $\wedge$
% *the spy is not in the group*
ingroup(triple(principal(spy),Y,Z),Restgp,Newgp)=false $\wedge$
% *but he knows the key*
in(Gk,analz(Trace))=true $\rightarrow$

Again this is a negative conjecture suggesting that for the protocol to be secure, no trace should exist where the spy knows a key Gk accepted by group member Mj

when the spy is outside the group. CORAL finds the counterexample in Figure 4 [4]. Again this is a replay attack. In message 14, the spy replays a key update originally sent in message 11, while he was still in the group. So, the spy knows the group key $Gk(4)$ even though he is no longer a group member. Note for this attack to work, it is necessary for two honest agents to join the group as well as the spy, whereas only one was required for the previous attacks, and note further that CORAL has discovered this for itself - there is no pre-setting of the number of agents. Preventing this attack is not as straightforward as it was for the Tanaka-Sato/Taghdiri-Jackson protocol because the key updates are unsolicited, so there is no opportunity for the agents and the server to exchange a nonce. The only way to protect against replays would seem to be to include a timestamp inside the encrypted packages sent in key updates, both when agents join and leave the group. This would require all group members to have at least loosely synchronised clocks, and would further require a decision in advance about the lifetime of group keys and the expected amount of delay in the network. However, this limitation seems inescapable for such protocols. In the presence of a Dolev-Yao spy it is insufficient, for example, to include the last key in the key update message as freshness information, since the spy may prevent this message from being received, and then re-send it once he has left the group.

A complete listing of the Iolus protocol model file is available at `http://homepages.inf.ed.ac.uk/s9808756/iolus/`.

5 Lessons from the Case Studies

We have seen in these case studies the re-emergence in new protocols of oversights made by the designers of the first security protocols. As we argued in §3.2, replay attacks like the ones we discovered are even more serious in the context of a multicast key management protocol, and do not even require the spy to have full Dolev-Yao capabilities - he need only be able to replay an old message, he does not need to stop a message from being received or break messages apart. Our fix for the Iolus protocol requires timestamps to be used in the protocol, which is not wholly desirable, but seems the only solution for multicast protocols that eagerly update keys. For on-demand rekeying such as is used in the Tanaka-Sato protocol, a nonce exchange can be used.

In terms of automated group protocol analysis, we have seen that CORAL's inductive model, a first-order version of Paulson's, is well suited to modelling group key management protocols. The protocols involve an unbounded number of agents, each of whom may play a role in different sub-protocols an unbounded number of times. CORAL's inductive model handles this naturally. Again we have seen the value of a model that does not require us to pre-set the number of agents involved in a group - only two agents were required for the attacks on the Tanaka-

[4] In CORAL's inductive model, we have an arbitrary and unbounded number of agents, so the first agent is called a, the second $s(a)$, and so on.

1.	a	$\rightarrow$	server	:	$\{\!\|\ a\|\!\}_{longtermK(a)}$
2.	server	$\rightarrow$	all	:	$\{\!\|\ Gk(1)\|\!\}_{Gk(X8)}$
3.	server	$\rightarrow$	a	:	$\{\!\|\ ik(2), Gk(1)\|\!\}_{longtermK(a)}$
4.	spy	$\rightarrow$	server	:	$\{\!\|\ spy\|\!\}_{longtermK(spy)}$
5.	server	$\rightarrow$	all	:	$\{\!\|\ Gk(2)\|\!\}_{Gk(1)}$
6.	server	$\rightarrow$	spy	:	$\{\!\|\ ik(5), Gk(2)\|\!\}_{longtermK(spy)}$
7.	s(a)	$\rightarrow$	server	:	$\{\!\|\ s(a)\|\!\}_{longtermK(s(a))}$
8.	server	$\rightarrow$	all	:	$\{\!\|\ Gk(3)\|\!\}_{Gk(2)}$
9.	server	$\rightarrow$	s(a)	:	$\{\!\|\ ik(8), Gk(3)\|\!\}_{longtermK(s(a))}$
10.	a	$\rightarrow$	server	:	$\{\!\|\ leave\|\!\}_{ik(2)}$
11.	server	$\rightarrow$	all	:	$[\{\!\|\ Gk(4)\|\!\}_{ik(8)}, \{\!\|\ Gk(4)\|\!\}_{ik(5)}]$
12.	spy	$\rightarrow$	server	:	$\{\!\|\ leave\|\!\}_{ik(5)}$
13.	server	$\rightarrow$	all	:	$[\{\!\|\ Gk(5)\|\!\}_{ik(8)}]$
14.	spy	$\rightarrow$	all	:	$[\{\!\|\ Gk(4)\|\!\}_{ik(8)}, \{\!\|\ Gk(4)\|\!\}_{ik(5)}]$

Fig. 4. Attack on the Iolus Protocol

Sato/Taghdiri-Jackson protocol, but three are necessary for the Iolus protocol attack. CORAL discovered this for itself.

There were two weaker aspects to CORAL's performance: one was the difficulty of posing conjectures. For the first case study, it took several attempts to pose the security property in such a way that counterexamples really were attacks. This is particularly annoying when it takes several hours to get the counterexample. Some of the difficulty comes from the fact that is not trivial to translate the kinds of authenticity properties required of unicast protocols to group protocol situations. As we saw is §3.2, the properties outlined by Pereira and Quisquater for group key establishment protocols are also unsuitable for a dynamic group situation. For example, the property of perfect forward secrecy, where the compromise of long-term secrets does not lead to the compromise of past sessions, does not hold for either of the protocols analysed in this paper, since they both assume an authentication protocol using long-term secrets is used to set up the pairwise keys for communication with the server. Compromise of these would lead to the compromise of all session keys from the period when the compromised agent was in the group. The Iolus protocol quite trivially does not satisfy the property of resistance to known key attacks. New session keys are always distributed under old session keys, so the compromise of one will lead to the loss of all subsequent keys. It is clear the designers of these multicast key management protocols have not aimed to provide these properties. Instead, we have identified *multicast group authenticity* as the key

property, i.e. throughout the evolution of the group, keys used by group members must be known only to other group members. This proves to be quite a tricky property to express formally, and not just in CORAL, as we learnt from our experiments with the AVISPA SATMC tool (see below). Having analysed the first protocol, it was much easier to form the property required for the second. There has been some work on automating the process of formulating conjectures in the inductive model, [19], and this could perhaps be adapted to formulate properties for group key management protocols.

The second weaker aspect of CORAL's performance was the run times (up to 3.5 hours to find the second attack). This seems to be because the mixed trace based and state based nature of the model led CORAL to explore a lot of unreachable states. This we hope to address with a revised model storing more state information. We are working on an interface to the intermediate format used in the AVISPA project (see for example [4]). Thus we hope to take advantage of the efforts being made to extend HLSPL to group protocols to facilitate further testing of CORAL. With the help of L. Compagna, we have also been experimenting with one of the current AVISPA back-ends, the SATMC tool, [1], to see if these protocols can be modelled and the same attacks found. Some significant work will be needed to model the protocols in a completely general way, without restrictions on the number of agents and sub-protocol roles. Also, it seems that modelling key update in the Iolus protocol will require the ability to reason explicitly about lists. We have found that posing group security conjectures in the AVISPA intermediate format is also difficult. In particular, it is not straightforward to express multicast group authenticity. Not being able to reason about the order in which events took place, and who was in the group at the time, further complicates the issue. However, with appropriate extensions to the intermediate format and some more work, we expect be able to rediscover the attacks in much faster times.

6 Conclusions

We were pleased with the way CORAL performed on these protocols. Firstly, the use of an inductive model meant we didn't have to make fundamental changes to our modelling strategy to accommodate an open-ended protocol with an unbounded number of agents, joins, leaves, messages sent and received etc. Secondly, modelling at the first-order Horn clause level in a theorem prover meant making the adaptations required to store and manipulate a list of current group members was just an evening's work. Thirdly, CORAL was able to discover attacks requiring a long trace of messages to be sent, indicating it has scaled up well, despite exploring a model without any pre-setting of the number of agents, joins and leaves etc.

There are dozens of protocols for this scenario in the literature, most of which have received little or no formal attention. Our experience so far suggests that they are likely to be vulnerable to attack. Since the field of standard protocol analysis is already mature, this would seem to be a great opportunity for the automated reasoning/formal methods community to solve an outstanding problem

and prove the worth of their methods. Two pre-requisites for this are the extension of protocol specification languages to group protocols, which is already underway, and the establishment of a corpus of group protocols together with attacks found or verification results achieved. This we are beginning to do: see http://homepages.inf.ed.ac.uk/s9808756/group-protocol-corpus/.

References

[1] 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, *FME*, volume 2805 of *Lecture Notes in Computer Science*, pages 875–893. Springer, 2003.

[2] N. Asokan and P. Ginzboorg. Key-agreement in ad-hoc networks. *Computer Communications*, 23(17):1627–1637, 2000.

[3] G. Ateniese, M. Steiner, and G. Tsudik. New multiparty authentication services and key agreement protocols. *IEEE Journal on Selected Areas in Communications*, 18(4):628–639, April 2000.

[4] D. Basin, S. Mödersheim, and L. Viganò. An on-the-fly model-checker for security protocol analysis. In *Proceedings of the 2003 European Symposium on Research in Computer Security*, pages 253–270, 2003. Extended version available as Technical Report 404, ETH Zurich.

[5] G. Bella. Message reception in the inductive approach. Technical Report 460, Computer Laboratory, University of Cambridge, 1999.

[6] J. Bull and D. Otway. The authentication protocol. Technical Report DRA/CIS3/PROJ/CORBA/SC/1/CSM/436-04/0.5b, DERA, Malvern, UK, 1997.

[7] J. Clark and J. Jacob. A survey of authentication protocol literature: Version 1.0. Available via http://www.cs.york.ac.uk/jac/papers/drareview.ps.gz, 1997.

[8] H. Comon and R. Nieuwenhuis. Induction = I-Axiomatization + First-Order Consistency. *Information and Computation*, 159(1-2):151–186, May/June 2000.

[9] G. Denker and J. Millen. CAPSL integrated protocol environment. In *DARPA Information Survivability Conference and Exposition*, volume 1, pages 207–221, 2000.

[10] W. Diffie and M. Helman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976.

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

[12] D. Jackson. Alloy: a lightweight object modelling notation. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 11(2):256–290, 2002.

[13] G. Lowe. Breaking and fixing the Needham Schroeder public-key protocol using FDR. In *Proceedings of TACAS*, volume 1055, pages 147–166. Springer Verlag, 1996.

[14] C. Meadows. Extending formal cryptographic protocol analysis techniques for group protocols and low-level cryptographic primitives. In P. Degano, editor, *Proceedings of the First Workshop on Issues in the Theory of Security*, pages 87–92, Geneva, Switzerland, July 2000.

[15] C. Meadows. Formal methods for cryptographic protocol analysis: Emerging issues and trends. *IEEE Journal on Selected Areas in Communication*, 21(1):44–54, January 2003.

[16] C. Meadows and P. Syverson. Formalizing GDOI group key management requirements in NPATRL. In *ACM Conference on Computer and Communications Security 2001*, pages 235–244, 2001.

[17] J. Millen and G. Denker. MuCAPSL. In *DISCEX III, DARPA Information Survivability Conference and Exposition*, pages 238–249. EEE Computer Society, 2003.

[18] S. Mittra. Iolus: A framework for scalable secure multicasting. In *Proceedings of the ACM SIGCOMM '97 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, pages 277–288, Cannes, France, September 1997.

[19] R. Monroy and M. Carrillo. On automating the formulation of security goals under the inductive approach. In M. H. Hamza, editor, *Applied Informatics*, pages 1020–1025. IASTED/ACTA Press, 2003.

[20] R. Needham and M. Schroeder. Using encryption for authentication in large networks of computers. *Communications of the ACM*, 21(12):993–999, December 1978.

[21] L.C. Paulson. The Inductive Approach to Verifying Cryptographic Protocols. *Journal of Computer Security*, 6:85–128, 1998.

[22] O. Pereira and J.-J. Quisquater. Some attacks upon authenticated group key agreement protocols. *Journal of Computer Security*, 11(4):555–580, 2003. Special Issue: 14th Computer Security Foundations Workshop (CSFW14).

[23] AVISPA Project. Deliverable 2.1: The high-level protocol specification language. Available from `http://www.avispa-project.org/delivs/2.1`.

[24] G. Steel, A. Bundy, and M. Maidl. Attacking a protocol for group key agreement by refuting incorrect inductive conjectures. In D. Basin and M. Rusinowitch, editors, *Proceedings of the International Joint Conference on Automated Reasoning*, Cork, Ireland, July 2004. To appear.

[25] M. Steiner, G. Tsudik, and M. Waidner. Diffie-Hellman key distribution extended to group communication. In *Proc. 3rd ACM Conference on Computer and Communications Security (CCS' 96)*, pages 31–37, 1996.

[26] M. Taghdiri. Lightweight modelling and automatic analysis of multicast key management schemes. Master's thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, December 2002.

[27] M. Taghdiri and D. Jackson. A lightweight formal analysis of a multicast key management scheme. In *Proceedings of Formal Techniques of Networked and Distributed Systems - FORTE 2003*, LNCS, pages 240–256, Berlin, 2003. Springer.

[28] S. Tanaka and F. Sato. A key distribution and rekeying framework with totally ordered multicast protocols. In *Proceedings of the 15th International Conference on Information Networking*, pages 831–838, 2001.

[29] C. Weidenbach et al. System description: SPASS version 1.0.0. In H. Ganzinger, editor, *Automated Deduction – CADE-16, 16th International Conference on Automated Deduction*, LNAI 1632, pages 378–382, Trento, Italy, July 1999. Springer-Verlag.

The ASW Protocol Revisited: A Unified View [1]

Paul Hankes Drielsma [2] Sebastian Mödersheim [3]

Department of Computer Science, ETH Zurich, Switzerland

Abstract

We revisit the analysis of the ASW contract signing protocol and use a unified view of the protocol as a whole as a basis to reason about the protocol and its objectives. This line of reasoning yields a simpler and clearer model of agents and protocol objectives which is within the scope of standard security analysis methods, as it does not require fairness constraints and uses only standard authentication and secrecy properties. We also analyse this model for finitely and infinitely many sessions of the protocol using the automated analysis tools OFMC and its extension OFMC-FP.

Key words: Contract Signing, Fair Exchange, Automated Protocol Analysis

1 Introduction

Contract signing protocols like the ASW protocol presented in [1] allow their users to digitally sign contracts without having to meet and sign a document or exchange it via standard mail, which can be very helpful in everyday communication in the business world.

When considering the formal analysis of such protocols, the difficulty arises that they are out of scope of many existing protocol analysis methods: although the act of signing and exchanging messages is standard for these methods, it is difficult to integrate the objectives which contract signing protocols aim to fulfil and the special assumptions upon which they rely.

We present a unified view of the ASW protocol in which the subprotocols are seen as a single protocol with different possible execution paths. While this view is implicit in the protocol models built, for instance, in [5,14], we

[1] This work was partially supported by the FET Open Project IST-2001-39252 and the BBW Project 02.0431, "AVISPA: Automated Validation of Internet Security Protocols and Applications".

[2] Email: drielsma@inf.ethz.ch

[3] Email: moedersheim@inf.ethz.ch

This is a preliminary version. The final version will be published in Electronic Notes in Theoretical Computer Science
URL: www.elsevier.nl/locate/entcs

explicitly describe and reason about the protocol and its objectives based on this high level view, which yields a simpler, more intuitive understanding of the ASW contract signing protocol.

In particular, our model is simpler than other approaches in two respects. First, as a consequence of our view, one does not have to distinguish between an intruder and dishonest and corrupt participants (or even different degrees of corruption), as it is necessary in several other models. Second, adopting this view allows us to reason about the objectives of such a protocol in a simple yet powerful way. We demonstrate, for instance, how several of the security objectives identified by the designers of the protocol can in fact be expressed as standard secrecy and authentication properties, thus "opening the door" to a variety of existing automated protocol analysis tools.

We then apply our unified view concretely, constructing a model of the ASW protocol and formally analysing it using the tools we have developed in our group, the On-the-Fly Model-Checker OFMC and its abstraction-based extension OFMC-FP. Both were developed in the context of the AVISPA project (http://www.avispa-project.org), which offers a toolset for the automated analysis of security protocols and applications. Using OFMC, we can verify the protocol for finitely many sessions (that is, executions of the protocol). Beyond this, we also perform an analysis with OFMC-FP, verifying the protocol for infinitely many sessions.

In the analysis, the tools report an attack on the ASW protocol which results from a subtlety in the specification of the objectives. Adapting these, we were able to verify that the protocol does ensure a slightly weaker objective that still implies the main fair exchange objective.

We observe that, even given the simpler understanding of the protocol that our approach affords, the design of a formal model for automatic analysis presents several challenges. After introducing these, we briefly discuss the results of our automated analysis for both finitely and infinitely many protocol sessions.

2 Background

The ASW protocol, presented by Asokan, Shoup, and Waidner in [1], is an optimistic fair exchange protocol for contract signing intended to enable two parties to commit themselves to a previously agreed upon contractual text. A trusted third party (T3P) is involved *only* if dispute resolution is required (hence the term *optimistic*, which differentiates this approach from others in which an online trusted party is involved in every exchange). In resolving disputes, the T3P issues either a *replacement contract* asserting that he recognises the contract in question as valid, or an *abort token* asserting that he has never issued, and will never issue, a replacement contract. An important requirement of the protocol is that the intruder cannot block messages between an honest agent and the T3P forever.

Exchange subprotocol:

1. $O \to R: \quad me_1 = Sig_O(V_O, V_R, T, text, h(N_O))$
2. $R \to O: \quad me_2 = Sig_R(me_1, h(N_R))$
3. $O \to R: \quad N_O$
4. $R \to O: \quad N_R$

Abort subprotocol:

1. $O \to T: \quad ma_1 = Sig_O(aborted, me_1)$
2. $T \to O: \quad ma_2 = \mathsf{if}\ resolved(me_1)\ \mathsf{then}\ Sig_T(me_1, me_2)$
   $\mathsf{else}\ Sig_T(aborted, ma_1)\ ;\ aborted(ma_1) = true$

Resolve subprotocol:

1. $O \to T: \quad mr_1 = me_1, me_2$
2. $T \to O: \quad mr_2 = \mathsf{if}\ aborted(me_1)\ \mathsf{then}\ Sig_T(aborted, me_1)$
   $\mathsf{else}\ Sig_T(me_1, me_2)\ ;\ resolved(me_1) = true$

Fig. 1. The Subprotocols of ASW

2.1 Protocol Objectives

The objectives that such a protocol is supposed to fulfil are manifold. We discuss here the security objectives identified by the designers and later refer to these informal descriptions in the discussion of our verification. Note that [1] refers to "fairness" in the sense of "fair exchange," but we adopt this latter term to avoid confusion with the notion of fairness constraints as understood by the model checking community, which we will use later.

Though [1] presents a framework for the fair exchange of arbitrary items, we consider only the application of this framework to contract signing. We therefore describe the objectives that follow in a manner specialised to our purposes.

1. First and perhaps foremost is the notion of *fair exchange*, which intuitively means that, at the end of a protocol execution, either both parties possess valid contracts, or neither does. In particular, we require that if one agent ends up with only an abort token, then the other cannot be in possession of a valid contract.
2. *Effectiveness* means that, if two honest agents P and Q have finished the protocol and never chose to abandon the current protocol run, then each indeed has a valid contract.
3. The protocol also provides guarantees of *timely completion*: more specifically, the originator and responder of a protocol run can be sure of completion within a finite amount of time.

4. The objective of *non-repudiability*, in the contract signing case, means that the contract contains an implicit proof of the agents' acceptance of the contractual text.
5. *Third party verifiability* dictates that, if the trusted third party should be corrupt and behave in such a way as to compromise fairness of the exchange for one of the protocol participants, then this corrupt behaviour can be proven to an external verifier.

The requirements for fair exchange are often stated in terms of liveness properties of the form "if one agent has a valid contract, then the other either has one as well or is in the position to eventually obtain one." In general, liveness properties are problematic for a variety of verification approaches, in particular those involving infinite state-spaces. In this case, one often approximates liveness properties via safety properties, i.e. if the protocol satisfies the safety property, then it also satisfies the liveness property that was approximated, as it is for instance done in [14]. In §3, we similarly identify appropriate safety properties to check; as we will show, however, from our unified view of the protocol we can directly obtain appropriate safety properties by a simple meta-reasoning.

2.2 Explanation of the protocol

The protocol, shown in Fig. 1, consists of three subprotocols: *exchange*, *abort*, and *resolve*. The former involves only the two protocol participants, the originator O and the responder R, while the latter two are only executed if the trusted third party T is called upon to resolve a dispute. Our notational conventions are as follows: $Sig_O(M)$ denotes the digital signature of message M by agent O, whose public key for signature verification is V_O. The contractual text we call *text*. During the protocol, each party generates a nonce, which we write N_O and N_R for the originator and responder, respectively. Finally, the function h is a cryptographic hash function which is assumed to be collision resistant. We note that the protocol defines two kinds of valid contracts: either the *standard contract* as it is obtained by the exchange subprotocol, or a *replacement contract* issued by the T3P, and both hold equal validity.

The Exchange Subprotocol: If both participants are honest and in the absence of network failures or intruder intervention, after execution of the exchange subprotocol, both will be in possession of a valid standard contract.

Both originator and responder generate nonces N_O and N_R which are called their respective *secret commitments* to the contract. Given these, they compute their so-called *public commitments* by hashing these values, yielding $h(N_O)$ and $h(N_R)$, respectively. The protocol then proceeds in two rounds: in the first, each party expresses his public commitment to the agreed-upon contract but does not disclose his secret commitment. In the second round, they then exchange their respective secret commitments. Each party can then hash this latter and thus verify that the purported secret commitment he receives

indeed corresponds to the public commitment from the first protocol stage. At the end of this exchange, each party is in possession of a valid standard contract of the form me_1, me_2, N_O, N_R.

The Abort Subprotocol: If O does not receive R's reply me_2 within an acceptable time frame (where the definition of "acceptable" is left entirely up to O), he may abort the protocol by invoking the trusted third party. He sends a signed abort request ma_1 indicating that he wishes to abort the exchange.

The T3P is assumed to maintain a permanent database of contracts for which he has been called upon to arbitrate. If he has already asserted the validity of the contract (indicated by $resolved(me_1)$), then he sends the originator a *replacement contract* of the form $Sig_T(me_1, me_2)$. Otherwise, he replies with a so-called *abort token*, signing the originator's abort request and adding an entry in his database of aborted contracts. Such a token does not render an existing contract invalid, but rather serves merely as a promise from the T3P that he has not previously resolved the contract in question and will not do so in the future.

The Resolve Subprotocol: The resolve subprotocol is analogous to the abort but can be invoked by either participant. The parties will request resolution of a contract from the T3P if they do not receive the secret commitment nonce of the other party within a reasonable amount of time. A resolution request includes both messages from the first stage of the exchange subprotocol, me_1 and me_2. If the T3P has already issued an abort token for the contract in question (indicated by $aborted(me_1)$), he replies in kind with an abort token. Otherwise, he issues a replacement contract and indicates in his database that he has resolved the contract.

2.3 The Intruder Model

We adopt the standard intruder model of Dolev and Yao [8] in which the intruder has complete control over the network but cannot break cryptography. In addition, the intruder can play as a normal protocol participant, acting as either the originator or the responder, but not as the T3P.

As we will discuss in more detail in §3, such an intruder model already subsumes the possibility of compromised or dishonest agents that collaborate with the intruder, and we want to show that the interests of honest agents are always ensured, even in protocol runs with the intruder.

3 The Unified View

The key idea behind this paper is to view, and reason about, ASW's subprotocols not in isolation, but rather as *one* protocol. This view is implicit in the construction of the protocol and accordingly also in the models of the protocol built by [5,14]. We explicitly exploit this view to reason about properties of the protocol. More specifically, we consider the abort and resolve subprotocols

$exchange_1.\ O \to R : me_1$

---

if *timeout* then	$abort_1.\ O \to T : ma_1$
	$abort_2.\ T \to O : ma_2$ (*abort token* or *replacement contract*)
else	

$exchange_2.\ R \to O : me_2$

$exchange_3.\ O \to R : N_O$

---

if *timeout* then	$resolve_1.\ O \to T : mr_1$
	$resolve_2.\ T \to O : mr_2$ (*abort token* or *replacement contract*)
else	

$exchange_4.\ R \to O : N_R$

Fig. 2. Originator role under our unified view of the protocol.

to be *part* of the main exchange protocol. The originator role, for instance, looks then as shown in Fig. 2 (the responder role is similar), where *timeout* represents the event that the agent playing O did not receive a reply to his last message within a reasonable amount of time. We avoid specifying the concrete amount of time after which the timeout shall occur: it may be just a few seconds or a full hour—important for the security of the protocol is only that there *is* such a timeout, so the agent will not wait for an answer forever. Fig. 3 illustrates the internal states of an agent playing the originator role: after sending his initial message, he is in the state in which he waits for a reply until the timeout. If the timeout occurs, then he tries to abort the protocol and thus waits for the answer of the trusted third party (which can be either a replacement contract or the signed abort token). Otherwise (if he receives a reply in time), he carries on with the regular protocol execution and sends his nonce, arriving in a state similar to the one he was in after sending the first message: either there is a reply within the allotted time or he contacts the trusted third party.

This model, though abstract, is thus a faithful representation of a real implementation of the protocol, as agents indeed protect themselves with such internal timeouts. Note that this is related to the possibility of abuse in this contract signing protocol: when the originator has made the first step, the responder has the freedom to either accept the contract by sending the second message, or to reject it by ignoring it. In particular, a dishonest responder could abuse the originator-signed part of the contract in negotiations with other agents (for instance, by soliciting more advantageous contracts from competitors). Note that, unlike for instance the similar GJM [9] protocol, the

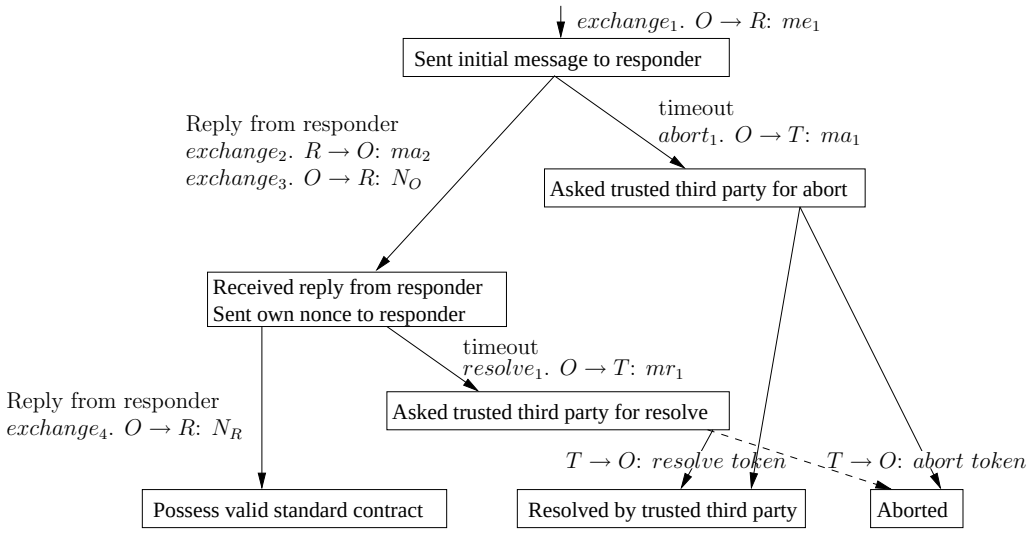

Fig. 3. A state transition view of the originator role. The dashed line represents a transition that can never occur if the trusted server is honest, as we will show below as part of our analysis.

ASW protocol has no means to prevent this abuse by special cryptographic primitives such as private contract signatures. Thus, the timeout is the only way to narrow the window of the originator's vulnerability to abuse attacks as argued in [6].

Although we assume that the intruder can control the entire network according to the Dolev-Yao model, the protocol requires that he cannot block messages between an honest agent and the T3P forever. One could intuitively imagine this situation as follows: all network connections could crash, but an honest agent can still transmit the necessary messages over other media (e.g. ordinary mail) to the T3P and this process cannot take forever.

One could say that we thus have two kinds of fairness assumptions: first, an honest agent will not wait forever for an answer from the other party, and second, the "emergency" protocols with the T3P will eventually succeed. Looking once again at Fig. 3, we can interpret this combination of fairness constraints as the guarantee that an honest agent playing the originator role (and a similar guarantee holds for the responder role) will not stay forever in any of the *intermediate states* (the states of the figure with an outgoing arrow), but will eventually reach one of the three *final states*, (i) where he received the responder's nonce [4] and thus now possesses a valid standard contract, (ii) where he received a valid replacement contract from the trusted third party, or (iii) where he received an abort token.

Thus, the two fairness constraints (timeout and guaranteed reply from the T3P) are sufficient to conclude that every honest agent playing either the

[4] If the responder sends a secret commitment that, when hashed, does not correspond with the public commitment, then this is treated as if he had not sent any message at all (which will probably result in a timeout).

originator or the responder role will eventually end up with either a valid (standard or replacement) contract or an abort token. Roughly speaking, if the agent receives a valid contract, then his interests are ensured, but if he receives an abort token, then it remains to show that nobody else can obtain a valid contract.

This gives us a fresh view on the protocol, as with a simple meta-argumentation we can now go from a model with fairness constraints to a state-reachability property in an infinite state transition system without fairness constraints.

The idea is essentially that we need only to check that if an honest agent reaches his final state of the protocol execution, then the guarantees he should obtain through the protocol are indeed satisfied. In other words, we do not need to consider the guarantees of agents in their intermediate states, since they will eventually reach their final state and we thus spare ourselves any considerations of the form "if the agent can eventually reach a certain state" in the properties we check.

The encoding of the objectives as safety properties is the basis for the deployment of automatic and semi-automatic methods for infinite-state analysis. Also in finite-state analysis, the restriction to safety properties is often essential, e.g. [14] use a similar argumentation that checking the protocol with fairness conditions can be reduced to checking properties of "terminal states" of agents.

4 Encoding of the Objectives

We now want to contrast two models: on the one hand, the model with the fairness constraints described above (i.e. that the agent will eventually get the timeout and the reply from the trusted third party), and on the other hand a model without fairness constraints.

In the model without fairness, the state transitions of the honest agents as shown in Fig. 3 are interpreted as follows: there is no timeout and no guaranteed replies, thus an agent can remain in any intermediate state forever. An agent's local state transition system is thus non-deterministic, as in the states where an honest agent waits for the reply of the other party, he can at any time (i.e. without timeout) begin the abort or resolve protocol, as appropriate.

It follows immediately that, if there is a violation of a safety property in the model with fairness, then there is also a violation in the model without fairness. This shows that our approach is sound in the sense that if we can prove properties in the model without fairness constraints, then they must also hold in the model with fairness constraints. The challenge is to find appropriate safety properties that indeed hold without the fairness constraints and that imply the safety and liveness properties of §2.1 that we wish to check.

We now review the objectives laid out in §2.1 under the new view of the

protocol and show how to encode those objectives that we wish to check as safety properties.

Let us begin with objective (3.), *timely completion*, which means that an honest agent will always eventually reach a valid standard or replacement contract or an abort token. This objective is a direct consequence of the model with fairness constraints, as discussed, and thus does not need to be explicitly checked.

Objective (1.), *fair exchange*, is the main objective of the protocol, namely that either both parties obtain a contract or neither does. We decompose this objective into the following two:

1a. If an honest agent receives an abort token, then nobody (except the trusted third party) can ever obtain a valid standard or replacement contract.

1b. If an agent A (who is not necessarily honest) has obtained a valid standard or replacement contract signed by an honest agent B, then B also possess a valid contract or can obtain one from the trusted third party.

Objective (1a.) is the main objective of our analysis, as it reflects the basic guarantee linked with the abort token. The nice aspect of this objective is that it refers to the final state of an honest agent (which will not subsequently change), not to an intermediate state. It is thus possible to check in the model without fairness that in all states where an agent has reached a final state with an abort token, nobody except the T3P can generate a valid standard or replacement contract matching that abort token. The inability to generate these messages can be expressed in standard protocol analysis approaches by secrecy properties. We can thus reduce the main objective of the protocol to a standard property in protocol analysis (though there is a technical difficulty in the direct application of tools as we will discuss below).

Objective (1b.) is a consequence of objectives (1a.) and (3.): if an agent A possesses a valid contract signed by an honest agent B, then by (3.) B will also eventually reach either an abort token or a valid contract, and by (1a.), if he gets an abort token, then A cannot possess a valid contract, which is a contradiction. Thus B will eventually obtain a valid contract.

Note that the circumstance in which the intruder or a dishonest agent playing the role of the originator can obtain both a valid contract and an abort token (by performing a normal run with an honest agent and asking the T3P for an abort) is not a violation of the objectives above: the abort token only guarantees that the T3P has never and will never resolve this contract but does not render an existing contract invalid.

We now turn to the objective (2.), *effectiveness*. In the view of our model, where every honest agent will eventually reach a final state, effectiveness means that when an honest agent A receives an abort token for a session with an honest agent B, then A or B must have chosen to abort the contract. Since the abort token from the T3P contains the signature of the agent A or B according

to the protocol, it contains the implicit proof that either A or B indeed wants to abort the protocol run. Effectiveness is thus an implicit guarantee due to the form of the abort token, and will therefore not be considered in the later analysis.

Objective (4.), *non-repudiability*, can also be seen as a consequence of the message formats, since in a valid standard or replacement contract, the signatures of both parties are contained and we can thus assume that they agree with the contract text. However, we are also interested in a further analysis, namely an analysis of the authentication properties (or agreement properties in [10]) of ASW. Our analysis will include checks for replay and for confusions of nonces. Such standard authentication properties do not rely on fairness and are thus straightforward to check.

Finally, objective (5.), *third-party verifiability*, is not relevant in our setting, as we assume that the T3P is always honest.

To summarise, we have showed that several objectives of the protocol are direct consequences of its structure, assumptions, and message formats. In essence, an honest agent will receive either an abort token or a valid contract. In the latter case, his interests are ensured, while it remains to show that in the case of an abort token, his interests are also ensured. This amounts to checking that, if an honest agent obtains an abort token, then the valid contract remains secret. Thus due to our view and the meta-reasoning about the protocol we have obtained a model that falls within the realm of standard automated protocol analysis approaches (which often support only secrecy and authentication properties), and we have avoided fairness issues completely. As we will see in the following section, the analysis even of this simplified model is challenging.

Let us conclude this section with a remark on the intruder model. Several approaches distinguish between the intruder and dishonest or corrupted agents (with various degrees of corruption). One of the reasons for this distinction is that the security properties of a protocol usually only hold for sessions between honest agents. In particular, the intruder can play, under his real name, the role of the initiator or the responder in a session with an honest agent; in such a session no security properties are ensured for this honest agent, while this session should not jeopardise the security of other sessions between honest agents.

Due to our simplified view, the main security property that we have to check for ASW, namely (1a.) that the secrecy of the valid contract once an honest agent has received an abort token, should also hold in the case that the other agent is dishonest. (But it is not necessarily ensured that this dishonest agent also has the same security guarantee.) This means that we need not distinguish between various kinds of corruption of dishonest agents.

5 Results

In the previous section, we have developed a unified view of the ASW protocol and a formulation of safety properties. We now apply these ideas concretely, formally specifying and then automatically analysing the protocol using two tools that we have developed in our group.

The first tool is the On-the-Fly Model-Checker (OFMC) which is based on a symbolic representation of the intruder, called the *lazy intruder* [4]. For termination, it requires a bound on the number of sessions that can be performed, but does not require other restrictions, e.g. on the complexity of messages. This is similar to the finite-state analysis of [14].

The second tool OFMC-FP is an extension of OFMC with an abstract fixed-point computation of the reachable states when there is no bound on the number of protocol runs that can be executed, however the complexity of messages is bounded in this method. OFMC-FP is still in a preliminary state at the time of writing: in particular, the user must manually design the employed abstractions.

Both OFMC and OFMC-FP were developed in the context of the AVISPA project and are based on the specification languages developed in this project. The user specifies protocols using the High-Level Protocol Specification Language (HLPSL [2]); these specifications are then automatically translated into the low-level Intermediate Format (IF [3]) which is the input language for automated analysis tools. The first task is thus to specify our view of the ASW protocol in HLPSL.

5.1 Specification

The construction of a formal model of ASW presents three major challenges:

Firstly, an aspect of the protocol that is difficult to model is the database of aborted and resolved contracts maintained by the T3P. Many existing protocol specification languages cannot express this, however HLPSL and IF include the necessary constructs (i.e. finite sets of messages) to model such a database. Moreover, the database cannot be integrated directly in infinite state verification approaches that use abstraction.

Secondly, when using OFMC, we bound the number of sessions of the honest agents to obtain a finite state-space. However, there is no a priori bound on the number of steps that the T3P can perform: in particular, the intruder can exchange an unbounded number of messages with the T3P. In the finite-session analysis with OFMC, we therefore also bound the number of requests from the intruder that the T3P can process.

Finally, although we have reduced the main problem of our analysis to a secrecy question, a further subtlety arises. When an abort token containing an initial message me_1 is issued, we must check for the secrecy of any valid contract that contains me_1. We thus do not state the secrecy of only *one* par-

$e_1.\ I \to R : me_1$	$e_1'.\ I \to R : me_1$
$e_2.\ R \to I : me_2$	$e_2'.\ R \to I : me_2'$
$e_3.\ I \to R : N_I$	Intruder stops communication
$e_4.\ R \to I : N_R$	
$a_1.\ I \to T : ma_1$	$r_1.\ R \to T : \{me_1, me_2'\}$
$a_2.\ T \to I : \mathit{abort\ token}$	$r_2.\ T \to R : \mathit{abort\ token}$

Fig. 4. An attack returned by OFMC: The intruder (denoted as I) aborts a contract that has already been exchanged. In a subsequent run with the same responder, the responder is then unable to resolve the protocol.

ticular message, but of a pattern of messages. Such a feature is currently not supported by HLPSL and IF (or most other protocol specification languages). As a simple way around the problem we specify an honest agent that acts as an observer and flags an error state appropriately.

5.2 Bounded-session Analysis with OFMC

Bounding the number of sessions and the number of requests from the intruder that the T3P processes, we can now directly check whether the transition of the referee ever fires, as well as check authentication properties.

OFMC discovers several authentication problems that were already identified in [14]. First, the protocol does not provide strong authentication (injective agreement as defined by Lowe in [10]), as it has no explicit protection against replay: if the intruder listens to a session of two honest agents, he can replay the exchange protocol with the responder any number of times and obtain valid contracts, each with a fresh responder nonce. However, we think one should assume an implicit replay-protection as part of the contract, e.g. transactions are usually identified by unique transaction numbers. In this scenario, we thus check that the protocol provides weak authentication (also called non-injective agreement, [10]). Weak authentication with respect to the contractual text and the nonces N_0 and N_R is also violated, and OFMC returns an attack trace resulting from the same authentication problems reported in [14]. Finally, weak authentication with respect to only the contractual text is verified by OFMC for various finite test scenarios.

When turning to the secrecy properties we have formulated, OFMC reported the attack displayed in Fig. 4. Assume the intruder I acting as the protocol originator and an honest responder R have completed a run of the exchange subprotocol without involving the T3P (steps e_1 through e_4). Each generated a secret commitment (N_I and N_R, respectively) and ends up with a valid contract in the standard form. Assume now that the intruder issues an abort request for this same contract to the T3P. This latter, having never

before resolved the contract in question, will respond with a valid abort token. The intruder now starts a second session with R, replaying the first message me_1 from the previous session. R generates a new nonce $N_R{}'$ and replies in good faith with the second message $me_2{}'$, including the hash of this new nonce. The intruder, however, does not reply with his nonce but rather ignores R. In turn, R will time out and request resolution of the contract from the T3P, who will respond with an abort token, since the originator message me_1 in question has already been aborted by I. Upon completion of the second protocol session, R therefore has an abort token, while I has a valid contract that corresponds to this abort token (in the sense that it contains the same initial message me_1). Of course, R himself also possesses this contract, having exchanged it with I in the first protocol session.

Formally this violates the objective (1a.): an honest agent has an abort token, while somebody else (the intruder) has a valid contract. It is not really a problem, since the honest agent itself also has this valid contract. In particular, the original objective (1.) is not violated, since both agents indeed possess valid contracts for the same contractual text. This is somewhat surprising, as we now see that there can be situations in which an honest agent indeed possesses both a valid contract and an abort token. Note that objective (1a.) is also considered by [14], who reported problems in the relation of nonces and contracts but did not detect that (1a.) can be violated. The same authors report in [13] an analogous attack on GJM, a similar contract signing protocol [9]. We note also that the improvement of the protocol that they suggest to address the authentication problems described above does not prevent this situation.

We have therefore relaxed objective (1a.) to the following weaker objective (1a′.): "if an honest agent has an abort token, then he also possesses a valid contract or nobody else can obtain one." Note that this property together with (1b.) still implies fair exchange (1.). For this weakened objective, OFMC detected no further attacks.

We also wish to note that an additional check on the responder side for replay of public commitments would prevent this attack.

5.3 Unbounded-session Analysis with OFMC-FP

We have also analysed the protocol using OFMC-FP, which employs a novel abstraction-based fixed-point computation. It was necessary to extend the existing OFMC-FP technique to allow for the integration of the server and its database of contracts. The technique is not completely automatic as the user must himself specify an appropriate abstraction.

OFMC-FP can also detect attacks, but due to the abstraction they may not be possible in our initial concrete model; however, if the security is proven for the abstract model, then this also holds for the concrete model. This is similar to other abstraction-based verification approaches like [7].

Using the OFMC-FP technique, we have established the verification results of OFMC for an unbounded number of participants, sessions, and transitions of the T3P; only the complexity of messages is bounded in this method. In particular, we have first shown that weak authentication on only the contractual text holds. Further, OFMC-FP reveals that a dishonest initiator can obtain a valid standard contract which has also been aborted by the server, which subsumes the violation of (1a.) already reported above. Also, we established that in all such situations, the other party also obtained a valid standard contract and thus verified the weakened property (1a′.). We note, however, that as part of the analysis, we have found that, for property (1a′.) to hold, an important prerequisite is the fact that an honest agent playing in the originator role can never obtain an abort token from the T3P as a reply to a resolve request, explaining the dashed line in Fig. 3.

6 Related Work and Conclusion

A substantial body of literature exists on the analysis of contract signing protocols, in particular ASW and the similar GJM protocol.

Shmatikov and Mitchell undertake an analysis (with a bounded number of sessions) of both the ASW and the GJM protocol using the finite-state model-checker Murφ [12,13,14]. Their approach is closest to ours, namely they also follow the principal idea to reduce the problem of fairness properties to safety properties. While they also implicitly employ the unified view of the protocol, they do not use it explicitly to perform meta-reasoning about the protocol. Another difference is that they also distinguish the intruder from dishonest agents (with varying degrees of corruption). Moreover, they check abuse-freeness for the GJM protocol (while ASW is not designed to ensure abuse-freeness).

Das and Dill [7] were the first to describe the automated analysis of a contract signing protocol, GJM, for an unbounded number of session using abstractions and the model-checker Murφ. Similar to our analysis, they focus on the property of fair exchange.

Kremer and Raskin [11] focus on abuse-freeness and argue that, under certain assumptions, even the ASW protocol is abuse free (while this was not one of the original objectives of the protocol designers). To appropriately model strategies of malicious agents and strategic advantages over other agents, they use a game theoretic method and alternating transition systems. They perform an automated analysis using the model-checker Mocha; note that they do not consider multiple runs of the protocol in parallel and adopt the strong typing assumption. [5]

There are several works (which do not focus on automated analysis) on

[5] The strong typing assumption is a stronger restriction than bounding the message size as it is done by OFMC-FP; no similar restriction is necessary for OFMC.

reasoning about such protocols and their guarantees, in particular optimism and fairness [5,6]. The employed models are considerably more detailed than ours in that they explicitly use time-outs and distinguish intruder and (different kinds of) dishonest agents. Also, here a similar view to ours is often taken, though to our knowledge not been used to reason about the protocol and its objectives.

We adopt this unified view and use it explicitly to reason about the protocol's objectives and thereby reduce several of them to standard authentication and secrecy properties which are easily digestible by many automated analysis tools for which protocols like ASW would previously have been out of scope.

Yet, as described in §5.1, even under the unified view, the specification and analysis with existing protocol analysis tools is challenging, in particular this holds for the modelling of the trusted third party that maintains a data-base of aborted and resolved contracts.

Our analysis demonstrated the same authentication failures discussed in [14] and also revealed that a guarantee that one might intuitively expect of the protocol, objective (1a.), is in fact violated by the attack we present. We can, however, show that, beyond these problems, the protocol is secure.

While we have focused on ASW in our work to date, we are optimistic that the benefits offered by the adoption of such a unified view will be applicable to similar protocols as well. In general, the meta-reasoning we perform regarding security objectives can help not only to better understand the objectives of a given protocol, but, as we have seen, can also identify potential ways in which seemingly complicated objectives can be reduced to more standard notions such as authentication and secrecy. In this way, we hope to extend the applicability of existing methods for the formal analysis of security protocols.

References

[1] N. Asokan, V. Shoup, and M. Waidner. Asynchronous protocols for optimistic fair exchange. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, pages 86–99, 1998.

[2] AVISPA. Deliverable 2.1: The High-Level Protocol Specification Language. Available at www.avispa-project.org/delivs/2.1/, 2003.

[3] AVISPA. Deliverable 2.3: The Intermediate Format. Available at www.avispa-project.org/delivs/2.3, 2003.

[4] D. Basin, S. Mödersheim, and L. Viganò. 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, 2003. Available at www.avispa-project.org.

[5] R. Chadha, M. Kanovich, and A. Scedrov. Inductive methods and contract-signing protocols. In *P. Samarati, editor, Proceedings, 8th ACM Conference on*

Computer and Communications Security, pages 176–185, New York, November 2001. ACM Press.

[6] R. Chadha, J. C. Mitchell, A. Scedrov, and V. Shmatikov. Contract signing, optimism, and advantage. In *CONCUR: 14th International Conference on Concurrency Theory*. LNCS, Springer-Verlag, 2003.

[7] S. Das and D. L. Dill. Successive approximation of abstract transition relations. In *Proceedings of the 16th Annual IEEE Symposium on Logic in Computer Science (LICS-01)*, pages 51–60, Los Alamitos, CA, June 16–19 2001. IEEE Computer Society.

[8] D. Dolev and A. Yao. On the Security of Public-Key Protocols. *IEEE Transactions on Information Theory*, 2(29), 1983.

[9] J. A. Garay, M. Jakobsson, and P. MacKenzie. Abuse-free optimistic contract signing. In *Proc. 19th International Advances in Cryptology Conference – CRYPTO '99*, pages 449–466, 1999.

[10] G. Lowe. A hierarchy of authentication specifications. In *Proceedings of the 10th IEEE Computer Security Foundations Workshop (CSFW'97)*, pages 31–43. IEEE Computer Society Press, 1997.

[11] J. Raskin and S. Kremer. Game analysis of abuse-free contract signing. In *15th IEEE Computer Security Foundations Workshop*, pages 206–220. IEEE Computer Society Press, June 2002.

[12] V. Shmatikov and J. C. Mitchell. Analysis of a fair exchange protocol. In *Proceedings of the 1999 FLoC Workshop on Formal Methods and Security Protocols*, Trento, Italy, 1999.

[13] V. Shmatikov and J. C. Mitchell. Analysis of abuse-free contract signing. In *Proceedings of the 4th International Conference on Financial Cryptography (FinCrypto '00)*, 2001.

[14] V. Shmatikov and J. C. Mitchell. Finite-state analysis of two contract signing protocols. *Theoretical Computer Science*, 283(2):419–450, June 2002.

Part II

Short Presentations

Web Services Security: a preliminary study using Casper and FDR

E. Kleiner and A.W. Roscoe*
Oxford University Computing Laboratory

Abstract

Web Services is an important new XML-based architecture in which security is increasingly important. The WS-Security specification defines mechanisms for securing the SOAP messages. We show how those messages can be mapped to Casper notation and therefore be analysed with FDR. We show two attacks on proposed protocols and lastly discuss informally some ramifications of the use of the WS-Security specification.

1 Introduction

Web Services is an XML-based architecture that has been developed in order to make the coupling between distributed components looser. In the last few years, with the growth of the popularity and importance of the Web Services architecture, more and more standards have been defined for extending the functionality and for dealing with different concerns. Due to its growing importance, Web Services requires rigorous security.

A common way for achieving it is relying on a secure transport layer, typically SSL as was studied and analysed in [3]. Apart from the fact that this technique provides security only in a secure channel (and not in files or databases), it does not correspond with the WS architecture in which the intermediaries can manipulate the message on its way. Once using a secure transport layer intermediaries are not able to control the messages.

A more suitable way is using the WS-Security specification [1] that by using the XML-signature [7] and XML-encryption [8] specifications, deals with and defines standards and ways of securing SOAP messages [6] without relying on a secure transport layer. In effect it creates a new sphere for cryptographic protocols in terms of design and implementation.

The theoretical community has been very successful in the last decade in developing methods for analysing cryptographic protocols. One of these, based upon Hoare's CSP [10] is Casper [15], supported by the FDR refinement checker [17]. This approach has proved to be very successful for modelling security protocols: firstly in finding attacks (e.g. [14] and [16]) and more recently providing general proofs (e.g. [4].) This paper reports the preliminary results of an exercise in applying Casper to check Web services

*{Eldar.Kleiner, Bill.Roscoe}@comlab.ox.ac.uk

protocols secured by the WS-Security specifications. The only comparable work we are aware of is that of [2].

Our paper is structured as follows. In Section 2 we give short overview of Web Services. In Section 3 we indicate how the syntax of a SOAP message that uses the WS-Security may be transformed into Casper input. In Section 4 we show some results on the examples from [13], in particular, demonstrating two related attacks. In section 5 we discuss some reflections of using WS-Security. Finally we conclude and give the outline of our planned work in this area.

We recognise that our work is at an early stage, but given our success (see 4) in finding attacks on a proposed standard protocols, we feel it is appropriate to publish this version.

2 Web Services Background

2.1 Motivation

In IT today one often faces the need to integrate different computing systems within an organization, perhaps even running on different platforms, for consolidated decision making or central monitoring.

The integration task is frequently challenging, largely due to the fact that traditional application are statically bound, namely the parameters, objects' types and the programming language are agreed upon during their individual designs. *Service Oriented Architecture* is a concept which addresses this issue.

2.2 SOA - Service Oriented Architecture

> "SOA is an approach to build distributed systems that deliver application functionality as services to end user applications or to build other services"(IBM).

A service is a package of functions the do not depend on the context or state of other services. Each service can publish its functionality, while other services are capable of discovering and binding dynamically to this functionality.

Web Services adopt the SOA concept using XML-based message layer (called SOAP) and can use any transport layer such as HTTP [18] and SMTP [12]. The main significance of Web Services technology is that is has been embraced by the entire industry.

2.3 SOAP - Simple Object Access Protocol

SOAP was proposed originally by Microsoft and DevelopMentor to provide a way to package information using XML for exchange between different computing systems. Today it is a W3C recommendation.

A SOAP message consists of two main parts:

- `Header` is an open element for extra application requirement. It is used by other specifications for expanding SOAP. For example, it can contain information about routing, context or security.

- `Body` contains the main applicative payload and can also include an optional fault element.

In addition, the SOAP specification defines how to exercise intermediaries and RPC conventions for allowing a client to call a remote function using the SOAP mechanism.

3 Modelling WS-Security protocols

In common with most modern work on cryptographic protocols, we study the WSS ones using the Dolev-Yao model [9]. In this model the network is controlled by the intruder who has the following abilities when attacking a set T of trusted agents. (i) overhearing all the messages flowing through the network, (ii) intercepting messages, (iii) faking messages based on what he knows limited only by cryptography, and (iv) behaving as any agent outside of T. This model is particularly appropriate in the Internet context of Web Services, since there is no sense in which users have any control over the medium which connects them. We claim that in this model the syntax of the SOAP message has relatively[1] little effect on the security of the protocol and therefore an abstracted view of the protocol (in the way we will present here) taken that it encapsulates all the security elements, provides an accurate model.

We construct a mapping ϕ from SOAP messages to Casper input, such that if a WS-security protocol contains the messages $m_1, m_2..., m_n$ then,

1. If an attack is found on $\phi(m_1), \phi(m_2), \ldots, \phi(m_n)$ then a corresponding attack can be reproduced on $m_1, m_2, \ldots, m_n$.

2. If an attack exists on $m_1, m_2, \ldots, m_n$ then it also exists on $\phi(m_1), \phi(m_2), \ldots, \phi(m_n)$

We suggest that if the intruder possesses $\phi(m)$, then he can create (perhaps with some reasonable guesses) a message m' which has the same security behaviour as m. Furthermore, the security checks performed during the execution of the original protocol are equivalent to those in the abstracted protocol.

The first property can be achieved by defining ϕ^{-1}. In case an attack is found in the model then ϕ^{-1} can be used to map it back to the original protocol space. We indicate briefly how ϕ^{-1} can be constructed.

More important of the above properties is (2), since we definitely do not want to generate a false "proof" of correctness using the translation.

3.1 Constructing ϕ

In this preliminary study we address relatively simple WS-Security protocols; in particular we will assume for now that the messages contain only a security header and a body. Furthermore, we will also assume that encryption can be performed only in the body. In future work we will extend ϕ such that it will support more complex messages.

For clarity, we will use an example taken from [13] to demonstrate the way ϕ works. For convenience, we have removed the `Namespaces` from the message (creates no ambiguity since we are dealing here only with security related information items). In addition,

[1] An interesting way in which SOAP can help will be discussed later, but in fact this just increases the faithfulness of the translation described below.

since the `TimeStamp` element is ignored in the proposed protocol we have also removed it. The following message is the first of a protocol. We will see the rest of the protocol, and its purpose, later.

In the following the values `BV1`, ..., `BV6` denote boolean strings holding data (signatures, encryptions, etc).

```
<Envelope>
  <Header>
    <Security mustUnderstand="1">
      <BinarySecurityToken ValueType="x509v3" Id="myCert"> BV1
      </BinarySecurityToken>
      <Signature>
        <SignedInfo>
          <CanonicalizationMethod Algorithm=.... />
          <SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig\#rsa-sha1"/>
          <Reference URI="#body">
            <Transforms>
              <Transform Algorithm=.... />
            </Transforms>
            <DigestMethod Algorithm=... />
            <DigestValue> BV2 </DigestValue>
          </Reference>
        </SignedInfo>
        <SignatureValue> BV3 </SignatureValue>
        <KeyInfo>
          <SecurityTokenReference>
            <Reference URI="#myCert" />
          </SecurityTokenReference>
        </KeyInfo>
      </Signature>
      <EncryptedKey>
        <EncryptedMethod Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-1_5"/>
        <KeyInfo>
          <SecurityTokenReference>
            <KeyIdentifier ValueType="X509v3"> BV4
            </KeyIdentifier>
          </SecurityTokenReference>
        </KeyInfo>
        <CipherData>
          <CipherValue> BV5 </CipherValue>
        </CipherData>
        <ReferenceList>
          <DataReference URI="#enc" />
        </ReferenceList>
      </EncryptedKey>
    </Security>
  </Header>
  <Body  Id="body">
    <EncryptedData Id="enc" Type="http://www.w3.org/2001/04/xmlenc#content">
      <EncryptedMethod Algorithm="http://www.w3.org/2001/04/xmlenc#tripledes-cbc" />
      <CipherData>
        <CipherValue> BV6 </CipherValue>
```

```
      </CipherData>
    </EncryptedData>
  </Body>
</Envelope>
```

Message: M

The syntax and content of message M are built from BV1-BV6 using the structuring mechanisms of XML complying to the Web Services Security specification.

The crucial observation we make is that anyone who possesses the values BV1...BV6 in fact possesses the entire message from the point of view of security, since all of the structuring mechanisms and other values such as URL's are essentially public and guessable. In particular an intruder who possesses these values can create the whole message. We conclude that, from the point of view of maintaining security, it is only these values that can matter.

Furthermore, careful inspection reveals that BV1 and BV4 are not really secret at all and we would expect any attacker to possess them since they are public certificates. So in fact the only "novel" parts of M are (BV2, BV3, BV5, BV6). The rest of the syntax simply puts them in context and indicates what they are.

What we will therefore do is demonstrate a mapping ϕ which reduces M to a representation of this quadruple, presenting them in the form used by Casper.

We now define the function ϕ. However the definition of ϕ we present in this paper does not cover all structures defined by [7, 1, 8] (for example we do not deal here with all the Security and KeyInfo optional children defined by [7]). Nevertheless, our definition is adequate for the protocols presented in this paper. We will provide a complete definition in a later paper.

We adopt the following general principles:

- ϕ can be applied to a valid, parsed XML document (we use the XML infoset recommendation [5])
- ϕ is a mapping that works on an element information item as a function of its children.

Note that ϕ might ignore children when they are not important security-wise.

Since ϕ cannot comprehend some of the elements' content (e.g. binary data) it uses the function ψ that "knows" the protocol specification to retrieve the necessary information that ϕ needs. The automated version of ϕ will have to get as an input some information about the protocol specification so that ψ will be able to work as described later. In addition, since ϕ might return different outputs for the same input depending on its context, ψ is also used to retrieve the context. More details are given later in this paper.

3.1.1 Envelope element

The result of applying ϕ to the Envelope element is the Envelope element's children list where ϕ is applied to every element in the list. In the Example.

$$\phi(M) = \phi(\langle Header\rangle ... \langle /Header\rangle), \phi(\langle Body\rangle ... \langle /Body\rangle).$$

The inverse of ϕ in this case is obvious. The original message can be constructed by putting the elements from the sequence one after the other inside an envelope. Note that different messages can be semantically equivalent because the order of the element does not necessarily change the semantic of the message. One can observe that the abstracted messages will be semantically equivalent as well.

3.1.2 `Header` element

In a similar way ϕ creates a sequence of the children of the `Header` element.

$$\phi(\langle Header\rangle ... \langle /Header\rangle) = \phi(Child_1), \phi(Child_2), ..., \phi(Child_n)$$

3.1.3 `Security` element

ϕ is defined very similarly when it is applied to the `Security` element, except for a slight change which is swapping the signature and the encryption elements. The reason for this swap is that [1] specifies that if the designer of the message wants to sign an element after encrypting it, the `Signature` element should be followed by the `Encryption` element and vice versa (i.e. the order of the operation is right to left.) Since ϕ works from left to right we need to swap their places to preserve the semantics.

$$\phi(\langle Security\rangle ... \langle /Security\rangle) = \phi(\langle BinarySecurityToken\rangle ... \langle /BinarySecurityToken\rangle),$$
$$\phi(\langle EncryptedKey\rangle ... \langle /EncryptedKey\rangle), \phi(\langle Signature\rangle ... \langle /Signature\rangle)$$

Again, the inverse of ϕ can be constructed similarly to the way it was constructed in 3.1.1.

3.1.4 `BinarySecurityToken` element

The `BinarySecurityToken` contains binary data. The `ValueType` attribute indicates what is encoded (e.g. X.509 certificate or Kerberos ticket.) In the example the `BinarySecurityToken` contains X.509 certificate. In this case, since the certificate is public, there is no need to model it. Therefore, the ϕ function will not return a value in this case.

When the X.509 `BinarySecurityToken` is referred to by another element, ϕ returns the public or the secret key corresponding to the certificate.

$$\phi(Ref) = \{A, PK(A)\}_{SK(CA)}$$
$$\phi(Ref, ENC) = PK(A)$$
$$\phi(Ref, SIG) = SK(A)$$

Here, *Ref* is the value of the `Id` attribute of the `BinarySecurityToken` and $A = \psi(\langle BinarySecurityToken...\rangle ... \langle /BinarySecurityToken\rangle)$.

ψ is responsible for returning the identity of the holder of the certificate. When automating ϕ and ψ, ψ will not be designed to extrapolate the identity of the certificate holder out of the binary data. Instead will create (with user assistance if necessary) a symbolic name for a typical agent in this role of the protocol.

3.1.5 `Signature` element

The ϕ function creates an abstracted signature of the `Signature` element in the following way.

$$\phi(\langle Signature\rangle ... \langle /Signature\rangle) = \{\phi(\langle Reference ...\rangle ... \langle /Reference\rangle),$$
$$\phi(\langle Reference ...\rangle ... \langle /Reference\rangle) ...\}_{\phi(\langle KeyInfo\rangle ... \langle /KeyInfo\rangle, SIG)}$$

3.1.6 `Reference` element

We are distinguishing between two cases. The first is when the `Reference` is obliged to contain the `DigestMethod` element (when it appears in the signature element by [7])

$$\phi(\langle Reference\ \ URI = "Ref"\rangle ... \langle /Reference\rangle) = \phi(DigestMethod)(\phi(Ref))$$

In the rest of the cases,

$$\phi(\langle Reference\ \ URI = "Ref"/\rangle) = \phi(Ref)$$
$$\phi(\langle Reference\ \ URI = "Ref"/\rangle, ENC) = \phi(Ref, ENC)$$
$$\phi(\langle Reference\ \ URI = "Ref"/\rangle, SIG) = \phi(Ref, SIG)$$

3.1.7 `DigestMethod` element

In this case ϕ returns the name of the abstracted hash function. See [15] for details of abstracting hash functions. In our example:

$$\phi(\langle DigestMethod\ \ algorithm = \text{“}http://www.w3.org/2000/09/xmldsig\#sha1"/\rangle) = sha1$$

3.1.8 `KeyInfo` element

The ϕ function returns the abstracted key that the `KeyInfo` represents. Here are two examples of possible scenarios:

$\phi(\langle KeyInfo\rangle ... \langle /KeyInfo\rangle, T) = \phi(\langle SecurityTokenReference\rangle ... \langle /SecurityTokenReference\rangle, T)$
or
$\phi(\langle KeyInfo\rangle ... \langle /KeyInfo\rangle, T) = \phi(\langle KeyName\rangle ... \langle /KeyName\rangle, T)$

3.1.9 `SecurityTokenReference` element

The `SecurityTokenReference` provides a way for referencing a security token by direct reference, key identifier, key names and embedded references. ϕ works as follows:

$$\phi(\langle SecurityTokenReference\rangle ... \langle /SecurityTokenReference\rangle, T) = \phi(Child, T)$$

3.1.10 `KeyName` element

`KeyName` contains a string that uniquely defines a cryptographic key. ϕ returns an abstracted version of the key that the `KeyName` element stands for. Once ϕ is automated, the user will have to supply the abstracted key. For example, if the `KeyName` contains a “distinguished name” of an X.509 certificate, the user will have to supply the identity of the certificate holder.

- If the string defines a symmetric key, then

$$\phi(\langle KeyName\rangle ... \langle /KeyName\rangle, T) = K$$

  Where $K = \psi(\langle KeyName\rangle ... \langle /KeyName\rangle)$

- If the string defines an asymmetric key then,

$$\phi(\langle KeyName\rangle ... \langle /KeyName\rangle, SIG) = SK(A)$$
$$\phi(\langle KeyName\rangle ... \langle /KeyName\rangle, ENC) = PK(A)$$

  Where $A = \psi(\langle KeyName\rangle ... \langle /KeyName\rangle)$ is the identity of the certificate holder.

3.1.11 `KeyIdentifier` element

The `KeyIdentifier` is a value that uniquely identifies a security element (cryptographic key in our example). The value type and the way to process it can vary and is defined by the designer and not by [1]. In our example the `KeyIdentifier` contains an X.509 certificate of the responder.

$$\phi(\langle KeyIdentifier\rangle ... \langle /KeyIdentifier\rangle, SIG) = SK(A)$$
$$\phi(\langle KeyIdentifier\rangle ... \langle /KeyIdentifier\rangle, ENC) = PK(A)$$

where $A = \psi(\langle KeyIdentifier\rangle ... \langle /KeyIdentifier\rangle)$ is the identity of the certificate holder.

3.1.12 `ReferenceList` element

ϕ is defined over `ReferenceList` element in the following way,

$$\phi(\langle ReferenceList\rangle ... \langle /ReferenceList\rangle) = \phi(\langle DataReference .../\rangle) ... \phi(\langle DataReference .../\rangle)$$
$$\phi(\langle ReferenceList\rangle ... \langle /ReferenceList\rangle, A) = \phi(\langle DataReference .../\rangle, A) ... \phi(\langle DataReference .../\rangle, A)$$

3.1.13 `DataReference` element

When ϕ gets the `DataReference` element as an input, it does not return a value, instead it creates a context for ϕ that can be retrieved later by ψ (see 3.1.14 for example.)

$$\phi(\langle DataReference\ \ URI = Ref/\rangle) = Context(Ref)$$
$$\phi(\langle DataReference\ \ URI = Ref/\rangle, A) = Context(Ref, A)$$

where *Ref* is the context name and A is the set of values of the context.

3.1.14 `EncryptedKey` element

The `EncryptedKey` element provides a way to encrypt a symmetric key with the recipient's public key. Once again, when automating the mapping, the user will need to supply the abstraction of the key that this element holds. We will mark this key K. In this case,

$$\phi(\langle EncryptedKey\rangle ... \langle /EncryptedKey\rangle) = \phi(\langle ReferenceList\rangle ... \langle /ReferenceList\rangle, \{K\}),$$
$$\{K\}_{\phi(\langle KeyInfo\rangle ... \langle /KeyInfo\rangle, ENC)}$$

where $K = \psi(\langle EncryptedKey\rangle ... \langle /EncryptedKey\rangle)$.

3.1.15 `EncryptedData` element

The value of ϕ of the `EncryptedData` element depends on the context of ϕ. The context is retrieved using ψ and affect ϕ in the following way:

- If $\psi(Ref)$ is not defined (in case the context *Ref* does not exist), then $\phi(\langle EncryptedData\ Id = Ref\rangle ... \langle /EncryptedData\rangle)$ is also not defined.
- If $\psi(Ref)$ is defined and the `EncryptedData` element has a `KeyInfo` element child then, $\phi(\langle EncryptedData\ Id = Ref\rangle ... \langle /EncryptedData\rangle) = \phi(\langle KeyInfo\rangle ... \langle /KeyInfo\rangle, ENC)$
- If $\psi(Ref)$ is defined and the EncryptedData element does not have a `KeyInfo` element child then, $\phi(\langle EncryptedData\ Id = Ref\rangle ... \langle /EncryptedData\rangle) = A$ where A is the set of values of the context.

3.1.16 `Body` element

The body is abstracted in the following way:

- If the `Body` element has no `EncryptedData` element child then, $\phi(Body) = Body$
- If ϕ of the `EncryptedData` element is defined and it is the only child of the `Body` element then, $\phi(Body) = \{Body\}\{\phi(\langle EncryptedData...\rangle ... \langle /EncryptedData\rangle)\}$
- If ϕ of the `EncryptedData` element is not defined and it is the only child of the `Body` element then, $\phi(Body) = Body$
- If `EncryptedData` is one of the elements in the body then the designer will have to supply the applicative structure of the body and only the relevant part will be dealt with. More details will be given in future work.

3.2 Example

We will now demonstrate the complete derivation of $\phi(M)$.

$\phi(M)$
$\Rightarrow \phi(\langle$Header: /cvs/AVISPA-FET-EU-02/shared/ARSPA/cameraready/KleinerRoscoe/Pre.tex,v 1.1 2004/06/07 10:56:41 uid525 Exp $\rangle ... \langle$/Header$\rangle), \phi(\langle$Body$\rangle ... \langle$/Body$\rangle)$

$\Rightarrow \phi(\langle$Security$\rangle ... \langle$/Security$\rangle), \phi(\langle$Body$\rangle ... \langle$/Body$\rangle)$

$\Rightarrow \phi(\langle$BinarySecurityToken$\rangle ... \langle$/BinarySecurityToken$\rangle)$, $\phi(\langle$EncryptedKey$\rangle ... \langle$/EncryptedKey$\rangle)$, $\phi(\langle$Signature$\rangle ... \langle$/Signature$\rangle)$, $\phi(\langle$Body$\rangle ... \langle$/Body$\rangle)$

$\Rightarrow \phi(\langle$EncryptedKey$\rangle ... \langle/$EncryptedKey$\rangle)$, $\phi(\langle$Signature$\rangle ... \langle/$Signature$\rangle)$, $\phi(\langle$Body$\rangle ... \langle/$Body$\rangle)$

$\Rightarrow$ According to the protocol specification [13], ψ of the EncryptedKey element is a random generated symmetric key. We will give it the conventional name K. $\Rightarrow$
$\phi(\langle$ReferenceList$\rangle ... \langle/$ReferenceList$\rangle$,{K}), $\{K\}_{\phi(\langle \text{KeyInfo} \rangle ... \langle/\text{KeyInfo}\rangle,\text{ENC})}$,
$\phi(\langle$Signature$\rangle ... \langle/$Signature$\rangle)$, $\phi(\langle$Body$\rangle ... \langle/$Body$\rangle)$

$\Rightarrow \phi(\langle$DataReference URI=#enc /$\rangle$,{K}), $\{K\}_{\phi(\langle \text{KeyInfo} \rangle ... \langle/\text{KeyInfo}\rangle,\text{ENC})}$,
$\phi(\langle$Signature$\rangle ... \langle/$Signature$\rangle)$, $\phi(\langle$Body$\rangle ... \langle/$Body$\rangle)$

$\Rightarrow$ Context(enc,{K}), $\{K\}_{\phi(\langle \text{KeyInfo} \rangle ... \langle/\text{KeyInfo}\rangle,\text{ENC})}$, $\phi(\langle$Signature$\rangle ... \langle/$Signature$\rangle)$,
$\phi(\langle$Body$\rangle ... \langle/$Body$\rangle)$

$\Rightarrow$ Context(enc,{K}), $\{K\}_{\phi(\langle \text{SecurityTokenReference} \rangle ... \langle/\text{SecurityTokenReference}\rangle,\text{ENC})}$,
$\phi(\langle$Signature$\rangle ... \langle/$Signature$\rangle)$, $\phi(\langle$Body$\rangle ... \langle/$Body$\rangle)$

$\Rightarrow$ Context(enc,{K}), $\{K\}_{\phi(\langle \text{KeyIdentifier} \rangle ... \langle/\text{KeyIdentifier}\rangle,\text{ENC})}$, $\phi(\langle$Signature$\rangle ... \langle/$Signature$\rangle)$,
$\phi(\langle$Body$\rangle ... \langle/$Body$\rangle)$

$\Rightarrow$ According to the designer's specification, ψ of the KeyIdentifier element is the receiver's X.509 certificate. We will give the receiver the conventional name B. $\Rightarrow$
Context(enc,{K}), $\{K\}_{PK(B)}$, $\{\phi(\langle$Reference URI=#body$\rangle ... \langle/$Reference$\rangle)\}$,
$\{\phi(\langle$KeyInfo$\rangle ... \langle/$KeyInfo$\rangle)$,SIG$\}$, $\phi(\langle$Body$\rangle ... \langle/$Body$\rangle)$

$\Rightarrow$ Context(enc,{K}), $\{K\}_{PK(B)}$, $\{\phi(\langle$DigestMethod.../$\rangle)(\phi(\text{body}))\}_{\phi(\langle \text{KeyInfo} \rangle ... \langle/\text{KeyInfo}\rangle,\text{SIG})}$,
$\phi(\langle$Body$\rangle ... \langle/$Body$\rangle)$

$\Rightarrow$ Context(enc,{K}), $\{K\}_{PK(B)}$, $\{\text{sha1}(\phi(\text{body}))\}_{\phi(\langle \text{KeyInfo} \rangle ... \langle/\text{KeyInfo}\rangle,\text{SIG})}$, $\phi(\langle$Body$\rangle ... \langle/$Body$\rangle)$

$\Rightarrow$ Context(enc,{K}), $\{K\}_{PK(B)}$,
$\{\text{sha1}(\{\text{Body}\}_{\phi(\langle \text{EncryptedData Id=“enc”} \rangle ... \langle/\text{EncryptedData}\rangle)})\}_{\phi(\langle \text{KeyInfo} \rangle ... \langle/\text{KeyInfo}\rangle,\text{SIG})}$,
$\phi(\langle$Body$\rangle ... \langle/$Body$\rangle)$

$\Rightarrow$ Context(enc,{K}), $\{K\}_{PK(B)}$, $\{\text{sha1}(\{\text{Body}\}_K)\}_{\phi(\langle \text{KeyInfo} \rangle ... \langle/\text{KeyInfo}\rangle,\text{SIG})}$, $\phi(\langle$Body$\rangle ... \langle/$Body$\rangle)$

$\Rightarrow$
Context(enc,{K}), $\{K\}_{PK(B)}$, $\{\text{sha1}(\{\text{Body}\}_K)\}_{\phi(\langle \text{SecurityTokenReference} \rangle ... \langle/\text{SecurityTokenReference}\rangle,\text{SIG})}$,
$\phi(\langle$Body$\rangle ... \langle/$Body$\rangle)$

$\Rightarrow$ Context(enc,{K}), $\{K\}_{PK(B)}$, $\{\text{sha1}(\{\text{Body}\}_K)\}_{\phi(\langle \text{Reference URI=\#myCert.../} \rangle,\text{SIG})}$,
$\phi(\langle$Body$\rangle ... \langle/$Body$\rangle)$

$\Rightarrow$ The Reference element points to the BinarySecurityToken element. According to [13], the holder of this certificate is the sender. We will give him the conventional name A$\Rightarrow$ Context(enc,{K}), $\{K\}_{PK(B)}$, $\{\text{sha1}(\{\text{Body}\}_K)\}_{SK(A)}$, $\phi(\langle$Body$\rangle ... \langle/$Body$\rangle)$

$\Rightarrow$ Context(enc,{K}), $\{K\}_{PK(B)}$, $\{sha1(\{Body\}_K)\}_{SK(A)}$, $\{Body\}_K$

$\Rightarrow$ $\{K\}_{PK(B)}$, $\{sha1(\{Body\}_K)\}_{SK(A)}$, $\{Body\}_K$

4 Results of analyzing example protocols

We now quote message `M'`, taken from [13] with slight changes: namely, we removed the Namespaces and the `TimeStamp` element since it was ignored.

```
<Envelope>
  <Header>
    <Security mustUnderstand="1">
      <Signature>
        <SignedInfo>
          <CanonicalizationMethod Algorithm=.... />
          <SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig\#rsa-sha1"/>
          <Reference URI="#body">
            <Transforms>
              <Transform Algorithm=.... />
            </Transforms>
            <DigestMethod Algorithm=... />
            <DigestValue>... </DigestValue>
          </Reference>
        </SignedInfo>
        <SignatureValue>...</SignatureValue>
        <KeyInfo>
          <SecurityTokenReference>
            <KeyIdentifier ValueType=X509v3>...
            </KeyIdentifier>
          </SecurityTokenReference>
        </KeyInfo>
      </Signature>
      <BinarySecurityToken ValueType="x509v3" Id="myCert"> ...
      </BinarySecurityToken>
      <EncryptedKey>
        <EncryptedMethod Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-1_5"/>
        <KeyInfo>
          <SecurityTokenReference>
            <Reference URI="#myCert" />
          </SecurityTokenReference>
        </KeyInfo>
        <CipherData>
          <CipherValue>...</CipherValue>
        </CipherData>
        <ReferenceList>
          <DataReference URI="#enc" />
        </ReferenceList>
      </EncryptedKey>
    </Security>
  </Header>
  <Body  Id="body">
```

```
    <EncryptedData Id="enc" Type="http://www.w3.org/2001/04/xmlenc#content">
      <EncryptedMethod Algorithm="http://www.w3.org/2001/04/xmlenc#tripledes-cbc" />
      <CipherData>
        <CipherValue>...</CipherValue>
      </CipherData>
    </EncryptedData>
  </Body>
</Envelope>
                                    Message: M'
```

In scenario #6 in [13] the following protocol is proposed.

1. A→B: `M`
2. B→A: `M'`

After applying ϕ to both of the messages we get the following protocol.

1. MSG 1 .A→B : $\{\text{K}\}_{\text{PK(B)}}$, $\{\text{sha1}(\{\text{Body}\}_{\text{K}})\}_{\text{SK(A)}}$, $\{\text{Body}\}_{\text{K}}$
2. MSG 2. B→A : $\{\text{K2}\}_{\text{PK(A)}}$, $\{\text{sha1}(\{\text{Body2}\}_{\text{K2}})\}_{\text{SK(B)}}$, $\{\text{Body2}\}_{\text{K2}}$

Here A is an initiator who seeks to transfer some applicative data in Body to B and receive a response in Body2. A sends his certificate, a key K that he freshly generated signed with B's public key, the Body encrypted with K and a signature of the encrypted Body. When B receives the message he checks the certificate in order to make sure that A is an authorized user, he decrypts the second element using his private key to get K, ensures that the signature is correct and then use K to decrypt the last element to get Body. B then applicatively creates Body2 and sends it back in MSG 2.
The author of [13] claim that the response (MSG 2) is authenticated.[2]
We checked this protocol with the following Casper authentication specification:

$$Agreement(B, A, [Body2])$$

This specifies that if A thinks he has successfully completed a run of the protocol with B, then B has previously been running the protocol, apparently with A, and B was the one who sent *Body2* to A. Using FDR the following authentication attack was found.

1. MSG 1. I → Bob : $\{\text{K}\}_{\text{PK(Bob)}}$, $\{\text{sha1}(\{\text{Body}\}_{\text{K}})\}_{\text{SK(I)}}$, $\{\text{Body}\}_{\text{K}}$
2. MSG 2. Bob → I : $\{\text{K2}\}_{\text{PK(I)}}$, $\{\text{sha1}(\{\text{Body2}\}_{\text{K2}})\}_{\text{SK(Bob)}}$, $\{\text{Body2}\}_{\text{K2}}$
3. MSG 1. Alice → I_{Bob} : $\{\text{K3}\}_{\text{PK(Bob)}}$, $\{\text{sha1}(\{\text{Body3}\}_{\text{K3}})\}_{\text{SK(Alice)}}$, $\{\text{Body3}\}_{\text{K3}}$
4. MSG 2. I_{Bob} → Alice : $\{\text{K2}\}_{\text{PK(Alice)}}$, $\{\text{sha1}(\{\text{Body2}\}_{\text{K2}})\}_{\text{SK(Bob)}}$, $\{\text{Body2}\}_{\text{K2}}$

On receiving the final MSG 2 (step 4), Alice believes she has completed a run of the protocol with Bob. The truth is that Bob was never actually running the protocol with her. The Intruder successfully impersonated him: there is therefore no reasonable sense in which Bob's response to Alice is authenticated.

[2]The author emphasizes that the scenarios in the paper have not been extensively vetted for attacks.

This attack relies on a previous run of the protocol (steps 1,2) which may take place long before steps 3,4. The intruder uses an old MSG 2 to attack the protocol in the second run by encrypting K2 with the public key of Alice, replacing the certificate to the one of Alice and sending it in step 4 as a message from Bob. An inverse attack in which the intruder is the initiator of the protocol in step 3 is also feasible.

We also analysed scenario #7 in [13] in which FDR found a similar attack. The protocol after applying ϕ looks like this:

1. MSG 1. A→B : $\{\text{sha1}(\{\text{A,PK(A)}\}_{\text{SK(CA)}}), \text{sha1(Body)}\}_{\text{SK(A)}}, \{\text{K}\}_{\text{PK(B)}}, \{\text{Body}\}_{\text{K}}$
2. MSG 2. B→A : $\{\text{sha1(Body2)}\}_{\text{SK(B)}}, \{\text{K2}\}_{\text{PK(A)}}, \{\text{Body2}\}_{\text{K2}}$

We again used Casper to create the CSP model of the proposed protocol. FDR then found a similar attack:

1. MSG 1. I→Bob : $\{\text{sha1}(\{\text{I,PK(I)}\}_{\text{SK(CA)}}), \text{sha1(Body1)}\}_{\text{SK(I)}}, \{\text{K1}\}_{\text{PK(Bob)}}, \{\text{Body1}\}_{\text{K1}}$
2. MSG 2. Bob→I : $\{\text{sha1(Body2)}\}_{\text{SK(Bob)}}, \{\text{K2}\}_{\text{PK(I)}}, \{\text{Body2}\}_{\text{K2}}$
3. MSG 1. Alice→I_{Bob} : $\{\text{sha1}(\{\text{Alice,PK(Alice)}\}_{\text{SK(CA)}}), \text{sha1(Body3)}\}_{\text{SK(Alice)}}, \{\text{K3}\}_{\text{PK(Bob)}}, \{\text{Body3}\}_{\text{K3}}$
4. MSG 2. I_{Bob}→Alice : $\{\text{sha1(Body2)}\}_{\text{SK(Bob)}}, \{\text{K2}\}_{\text{PK(Alice)}}, \{\text{Body2}\}_{\text{K2}}$

The vulnerability of both the protocols is caused by the fact that neither of the messages in the first protocol and the second message in the second protocol are bound correctly to the sender. The intruder can use this fact for re-sending those messages, pretending they came from him. A simple solution to correcting the flaw is adding the sender identity to the signature (his certificate for example.) This prevents the above attacks because then the intruder will not be able to produce a valid MSG 2 (in step 4) since Alice will be expecting it to contain Bob's identity signed in the message.

We analysed the fixed protocols and FDR then failed to find an attack. However they are safe only for a single run. When Bob receives a message from Alice he still has no guarantee that the message was recently sent by her. The protocol can be strengthened further by using nonces or timestamps to prevent the intruder re-sending messages.

Those attacks are analogous to many found on protocols outside the WebServices area. For example, the attack on the Needham-Schroeder Public Key protocol described in [14] takes advantage of a similar flaw (the flaw in this protocol is that the identity of the responder is not bound to the message.)

5 Reflections on WS-Security

It appears that at present SOAP Message Security is used for the purpose of setting the parts of messages which convey actual security in context, namely allowing the receiver to see details of what the bit strings constituting signatures, encryptions, hashes etc are meant to be. We have seen that because this formatting is public it is as easily created by an intruder.

We indicated earlier that there is an interesting way in which SOAP can assist security. This is because there are two examples of possible forms of attack that the Web Services Security mechanism provides extra strength against, both of which have been studied in the literature of cryptographic protocols.

- *Type-flaw* attacks, where an intruder persuades a legitimate user that a value is of a different type to what it really is, and exploits this in an attack. So for example the intruder might persuade agent A to sign a value which A thinks is a nonce or a hash value, but which in fact has other structures that the intruder can then use to impersonate A in some run with (say) B. The avoidance of this type of problem (see [11]) for example, depends on the nature of the cryptography involved, how it is used – for example whenever one signs something one adds the signer's interpretation of what it is signing – and protocol design. In XML-signature, the tag name of the element that is signed provides information about the type of the element. Similar things are true about values that are encrypted or hashed. Therefore, SOAP messages that are protected by the Web Services specification are relatively resistant to type-flaw attacks, with two caveats:
  - In order for the above mechanism to work, the signed/hashed/encrypted object must contain a sufficient description of the type so as not to leave any ambiguity.
  - This mechanism only prevents type confusion by trusted agents of data which is received by them protected by a cryptographic device they can understand.
- *Protocol interference*, where information gleaned from running one protocol can be used to attack another. To avoid this type of attack the best approach is to use the principle of *explicitness* within the encryptions and signed data, and ensure hash functions are context dependent. This would mean that instead of encrypting, signing or hashing X, one does the respective thing to (T, X) where the tag T includes information such as which protocol this was done for, and which field in what message. By signing/encrypting/hashing a value in a SOAP context we have the machinery here to create excellent *de facto* tags.

We emphasise that the SOAP mechanism do not in general prevent these forms of attack, but they do provide some protection implicitly. We hope it might be possible to strengthen the SOAP standard so as to guarantee absence of these forms of attack. This will be subject of further work.

6 Conclusion

We have demonstrated that the security of SOAP based protocols is essentially the same problem as the traditional form of cryptographic protocol. We have demonstrated the correspondence for a few protocols and presented preliminary work on the general mapping.

The main advantages of this translation are firstly clarity in the sense that the abstract descriptions are far more concise, and secondly the availability of tools such as Casper to analyse the WS-Security protocols.

The usefulness of our method was immediately shown by the attacks we discovered. It seems clear to us that our methods or something very similar need to be adopted in the Web Services community when security is a concern.

We note that because our methods give a translation to Casper notation where the whole area of protocol verification is well understood, they do not suffer from the restriction to only authentication specifications reported in [2].

7 Future work

In the future we will present the complete formalised mapping between the Web Services notation and Casper input.

We expect that most or all of this process can be automated, so that in effect one can simply input a WS-Security protocol to an extended Casper and obtain a model that we can input to FDR to test its security.

As far as we know, *intermediaries* is a poorly understood area in Web Services Security. We are interested in "internalising" potential intermediaries in the style of [4] and believe we then be able to model and check protocols with arbitrary number of intermediaries.

Of course the question remains of how we can draw inferences about general implementations from checks of small examples. The fact that we have translated WS-Security into equivalent Casper input means we can expect the body results that already exists for standard protocols to apply here as well.

Lastly, we would like to look into SAML and extend ϕ such that we will be able to reason about protocols that contain messages with SAML assertions.

Acknowledgements

We would like to thank Philippa Hopcroft and Gavin Lows for their suggestions and Hal Lockhart for his help and the information he shared with us, in particular [13].

References

[1] A. Nadalin, C. Kaler, P. Hallam-Baker and R. Monzillo. "Web Services Security: SOAP Message Security 1.0 (WS-Security 2004)". Oasis, 2004. http://www.oasis-open.org/committees/download.php/5941/oasis-200401-wss-soap-message-security-1.0.pdf.

[2] Karthikeyan Bhargavan, Cedric Fournet, and Andrew D. Gordon. A semantics for web services authentication. In *Proceedings of the 31st ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 198–209. ACM Press, 2004.

[3] P. Broadfoot and G. Lowe. On distributed security transactions that use secure transport protocols. In *the 16th IEEE Computer Security Foundations Workshop, pages 141-154, IEEE Computer Society Press*, 2003.

[4] P.J Broadfoot and A.W. Roscoe. Internalising agents in CSP protocol models. Workshop on Issues in the Theory of Security (WITS '02), Protland Oregon, USA, 2002.

[5] J. Cowan and R. Tobin. "XML Information Set". W3C Recommendation, 2001. http://www.w3.org/TR/2001/REC-xml-infoset-20011024.

[6] D. Box, D. Ehnebuske, G. Kakivaya, A. Layman, N. Mendelsohn, H. Nielsen S. Thatte and D. Winer. "Simple Object Access Protocol (SOAP) 1.1". W3C Note, 2000. http://www.w3.org/TR/2000/NOTE-SOAP-20000508.

[7] D. Eastlake, J. Reagle, D. Solo, M. Bartel, J. Boyer, B. Fox, B. LaMacchia and E. Simon. "XML-Signature Syntax and Processing". W3C Recommendation, 2002. http://www.w3.org/TR/xmldsig-core/.

[8] D. Eastlake, J. Reagle, T. Imamura, B. Dillaway and E. Simon. "XML Encryption Syntax and Processing". W3C Recommendation, 2001. http://www.w3.org/TR/xmlenc-core/.

[9] D. Dolev and A.C. Yao. On the security of public-key protocols. *Communications of the ACM*, 29(8):198–208, August 1983.

[10] C.A.R. Hoare. *Communicating Sequential Processes*. Prentice Hall, 1985.

[11] J. Heather, G. Lowe and S. Schneider. How to prevent Type Flaw Attacks on Security Protocols. *Proceeding of 13th IEEE Computer Security Foundations Workshop*, pages 255–268, 2000.

[12] J. Klensin . "Simple Mail Transfer Protocol". IETF, 2001. http://www.ietf.org/rfc/rfc2821.txt.

[13] H. Lockhart. "Web Services Security: Interop 2 Scenarios Working Draft 06". Oasis, 2003. http://lists.oasis-open.org/archives/wss/200310/pdf00000.pdf.

[14] G. Lowe. An attack on the Needham-Schroeder Public-Key authentication protocol. *Information Processing Letters*, 1995.

[15] G. Lowe. Casper: A compiler for the analysis of security protocols. *Journal of Computer Security*, 6:53–84, 1998.

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

[17] Formal Systems (Europe) LTD. *Failure-Divergences Refinement FDR2 Manual*, 1997.

[18] R. Fielding, J. Gettys, J. C. Mogul, H. Frystyk and T. Berners-Lee. "RFC 2616: Hypertext Transfer Protocol – HTTP/1.1". IETF, 1997. http://www.ietf.org/rfc/rfc2616.txt.

A logic-based approach to reasoning with beliefs about trust

Fariba Sadri[1] Francesca Toni[1,2]

1: Department of Computing, Imperial College London, UK
2: Dipartimento di Informatica, Università di Pisa, Italy
{fs,ft}@doc.ic.ac.uk,
http://doc.ic.ac.uk/∼{fs,ft}

Abstract

We adopt an existing general-purpose, logic-based framework for knowledge representation and reasoning to endow agents interacting within multi-agent systems with the capability to build up beliefs about the trustworthiness of other agents and to reason with these beliefs, e.g. to favour trustworthy agents when choosing who to negotiate with regarding a task or a resource. The framework is that of abductive logic programming. We illustrate our approach by means of several examples.

1 Introduction

The ever-growing use of the internet and of agents deployed in it and roaming over it poses the problem of dealing with possibly malicious agents and ensuring that they do as little harm as possible. Instead of specifying and enforcing *security protocols* to isolate such agents, in this paper we consider the orthogonal problem of allowing such possibly malicious agents to participate in interactions with the other agents, but endow agents with the capability to prevent the interactions to be harmful, by allowing them to reason about the trustworhiness of other agents. Following [8, 9], we adopt a distributed approach to *trust*, whereby we assume no central control upon the trustworthiness of agents. Instead, agents are equipped with trust policies, owned by the agents themselves and thus private to them. Such policies represent local beliefs and knowledge about the trustworthiness of other agents. Typically, trust beliefs emerge from the interactions amongst agents in the context of concrete scenaria that the agents participate in. Then, agents can make use of these beliefs when interacting with other agents, e.g. with the aim of reaching fruitful deals. Although trust beliefs are local to the agents, they might be shared amongst agents by means of communication, thus trading privacy for the benefit of collaboration towards an improved trust model. In any case, different agents may have different approaches to trust, and some agents might decide not to share their beliefs with others. Moreover, the same agent may have different approaches to trust according to the society it is in, its goals, plans etc, e.g. an agent may be trusted in one setting and not in another.

We represent trust policies within the framework of abductive logic programming, a general-purpose framework for knowledge representation and reasoning, equipped with

an operational (theorem-proving-based) counterpart which is provably correct with respect to the semantics of the framework. This choice allows representing and reasoning with trust policies, and thus paves the way towards proving properties of the policies, e.g. such as those proposed in [2]. Thus, our approach can be seen as a qualitative approach to trust, based upon direct reasoning via theorem-proving techniques (upon which abductive logic programming builds), and thus differs from quantitative approaches at the methodological level.

Our approach is closely related to the approach of [7], which also uses theorem-proving techniques (the DLVD system) for reasoning with trust (and delegation) policies specified by means of Datalog programs. However, the language of abductive logic programming is more expressive than that of datalog programs, and thus our approach can be seen as an extension of that of [7] in terms of the class of trust policies that it can represent. Moreover, our approach to modelling trust policies can be directly imported to design (abductive logic) agents that are provably conformant to the policies, for examples agents such as those of [13, 16, 10].

The remainder of the paper is organised as follows. In section 2 we give the necessary background on abductive logic programming. In section 3 we give several examples of trust policies within abductive logic programming. In section 4 we show how trust policies can be used directly by abductive logic agents, so that their behaviour is conformant to the policies. In section 5 we conclude.

The paper describes preliminary, ongoing work.

2 Abductive logic programming

Abductive logic programming is a general-purpose knowledge representation and reasoning framework that can be used for a number of applications and tasks [11, 13, 1, 14, 15, 17, 16, 19, 20]. It relies upon a symbolic, logic-based representation of beliefs, for any given domain of application, via *abductive logic programs*, and the execution of logic-based reasoning engines, called *abductive proof procedures*, for reasoning with such representations.

2.1 Abductive logic programs

An abductive logic program consists of

- A **logic program**, P, namely a set of if-rules of the form $Head \leftarrow Body$, where $Head$ is an atom, $Body$ is either $true$ (in which case we write the if-rule simply as $Head$) or a conjunction of (negations of) atoms. All variables in $Head$ and $Body$ are implicitly universally quantified from the outside. P is understood as a (possibly incomplete) set of beliefs.

- A set of **integrity constraints**, I, namely if-then-rules of the form $Condition \Rightarrow Conclusion$, where $Condition$ is either $true$ (in which case we write the if-then-rule simply as $Conclusion$) or a conjunction of (negations of) atoms, $Conclusion$ is $false$ or a conjunction of atoms, all variables in $Condition$ are implicitly universally quantified from the outside and all variables in $Conclusion$ but not in

Condition are implicitly existentially quantified on the right of $\Rightarrow$. The integrity constraints are understood as properties that must be "satisfied" by any extension of the logic program by means of atoms of abducible predicates, in the same way integrity constraints in databases are understood as properties that must be "satisfied" in every database state.

- A set of **abducible atoms**, Ab, namely atoms whose predicate occurs in the logic program and the integrity constraints but not in the $Head$ of any if-rule. Abducible atoms can be used to "complete" the (beliefs held within the) logic program, subject to the satisfaction of the integrity constraints.

When abductive logic programming is used to represent agents, $Ab = Act \cup Obs$, such that $Act \cap Obs = \emptyset$, where Act represents actions that the agent can execute and Obs represents phenomena that the agent can observe. In general, the abducibles in Act can be either physical or communicative actions, namely utterances that the agent can perform. In this paper, communicative actions will be of the form:

$tell(Utterer, Receiver, Content, Time)$, for example

$tell(fran, maria, \text{"}I\ will\ deliver\ the\ goods\ by\ May\ 15\text{"}, May\ 10)$,

and physical actions will be of the form

$do(Actor, Action, Time)$, for example

$do(fran, \text{"}deliver\ the\ goods\text{"}, 12\ May)$.

The first (communicative) action is an observable for *maria* (and thus in Obs) and, if the system allows for overhearing, also for all other agents. The second (physical) action is an observable for *maria* (and thus in Obs) and any other agent. Observations in Obs may also include events happening in the world in which agents are situated.

Given an abductive logic program (P, I, Ab) and a **query** (i.e. a possibly empty conjunction of literals) Q, the task of an *abductive proof procedure* is to compute a so-called **explanation** for Q, namely a pair (E, θ) consisting of a (possibly empty) set E of ground abducible atoms and a (possibly empty) variable substitution θ for the variables in Q, such that $P \cup E$ *entails* $\exists Q\,\theta$ and $P \cup E$ *satisfies* I. Various notions of entailment and satisfaction can be adopted, for example entailment and satisfaction could be entailment and consistency, respectively, in first-order, classical logic. The provision of such notions is beyond the scope of this paper, see [11] for a detailed discussion.

2.2 Abductive proof procedures

We will omit the full description of any concrete abductive proof procedures (for some such procedures see [11, 6, 14]). However, note that typically abductive proof procedures interleave *backward reasoning* (also called *unfolding*) with if-rules in the logic program (reduction of $Head$ to $Body$) and *forward reasoning* (also called *propagation*) with if-then-rules in the integrity constraints (addition of $Conclusion$ to $Condition$ if the latter holds). Backward reasoning with the if-rules allows to generate the kernel of candidate explanations, whereas forward reasoning with the if-then-rules allows to rule out unwanted candidate explanations and expand the kernel of promising ones. Indeed, some kernel explanations might allow to explain the query but might fail to take into account the integrity constraints, whereas others might need to be augmented in order to do so. The interleaving terminates when no more backward or forward reasoning

can be performed in a branch in the tree of candidate explanations: the explanation can then be extracted from the branch. For example, given an abductive logic program (representing the beliefs of an agent *maria*)

$$
\begin{array}{ll}
P\text{: } have(R,T,T') \leftarrow & friend(X), \\
& tell(maria, X, \text{“}give\ me\ R\ by\ T-5\text{”}, T'), \\
& do(X, \text{“}deliver\ R\text{”}, T-5) \\
\quad friend(fran) & \\
I\text{: } do(X, \text{“}deliver\ R\text{”}, T) \Rightarrow tell(maria, X, \text{“}thanks\text{”}, T+1) & \\
Ab: \ Act = \{tell(maria, X, C, T)\} \cup Obs = \{do(X, A, T)\} &
\end{array}
$$

and a query that *maria* has the goal, on 10 May, to have some *goods* by 17 May, and that *maria* observes that the agent $fran$ delivers the *goods* on 12 May:

$$Q: \ have(goods, 17\,May, 10\,May), do(fran, \text{“}deliver\ goods\text{”}, 12\,May)$$

then an abductive proof procedure (such as IFF [6]) generates the following tree (consisting of a single branch with nodes $N_1, \ldots, N_4$) to search for an explanation for Q:

$$
\begin{array}{ll}
N_1\text{: } & have(goods, 17\,May,\ 10\,May),\ do(fran, \text{“}deliver\ goods\text{”}, 12\,May), I \\
N_2\text{: } & friend(X),\ tell(maria, X, \text{“}give\ me\ goods\ by\ 12\,May\text{”}, 10\,May), \\
& do(X, \text{“}deliver\ goods\text{”}, 12\,May),\ do(fran, \text{“}deliver\ goods\text{”}, 12\,May), I \\
N_3\text{: } & tell(maria, fran, \text{“}give\ me\ goods\ by\ 12\,May\text{”}, 10\,May), \\
& do(fran, \text{“}deliver\ goods\text{”}, 12\,May),\ do(fran, \text{“}deliver\ goods\text{”}, 12\,May), I \\
N_4\text{: } & tell(maria, fran, \text{“}give\ me\ goods\ by\ 12\,May\text{”}, 10\,May), \\
& do(fran, \text{“}deliver\ goods\text{”}, 12\,May),\ tell(maria, fran, \text{“}thanks\text{”}, 13\,May), I
\end{array}
$$

whereby N_1 is the initial goal (consisting of Q and I), N_2 is obtained from N_1 by unfolding the atom in the predicate *have*, N_3 is obtained from N_2 by unfolding the atom in the predicate $friend$, and N_4 is obtained from N_3 by factoring the two atoms in the predicate *tell* and by propagating with I. $(E = N_4 \setminus I, \{\})$ is the computed explanation for Q.

In the remainder of this paper, we illustrate how abductive logic programs can be used to represent beliefs about the trustworthiness of other agents (that we refer to as *trust policies*) as well as to make use of these beliefs when interacting with agents. Then, abductive proof procedures can be used to endow agents with the capability to reason with trust-related information. We will assume the use of an abductive proof procedure that can deal with constraint predicates (such as $\leq$), e.g. the CIFF proof procedure [3].

We will assume that the query posed to the abductive proof procedure consists of any *goals* and *observations* (i.e. abducibles in *Obs*) by the agent, and that the procedure is executed within an *observe–reason–act* cycle that allows the agent to be alert to the world in which it is situated and react to it as well as reason and devise *plans* for its goals [13, 16, 10]. Note that goals of the agent can be seen as conjunctions of literals, and plans as sets of actions, which can be represented as abducibles in abductive logic programming.

The cycle interleaves the execution of the abductive proof procedure (*reason*) for the computation of explanations for observations and goals, the perception of inputs from

the world (*observe*), to be added dynamically to the observation or goal to be explained, and the selection of actions (i.e. abducibles in *Act* in explanations) to be performed in the world (*act*). For example, in the concrete example given earlier, the second conjunct in the query Q will be typically added to Q after the action *tell* by *maria* has been executed, and the generation of the explanation is interleaved with the observation of the world and actions by other agents in it.

3 Representing trust policies

Beliefs of one agent about the trustworthiness of other agents may be expressed by means of rules within the logic program of the abductive logic program of the given agent. Belief in the trustworthiness of a given agent can be *static*, for example in the case of one agent trusting another at all time under all circumstances. For example, in the abductive logic program of agent *maria*, the logic program may contain the following if-rule

$$trust(maria, anna, T)$$

representing that *maria* always trusts agent *anna*.

Alternatively such beliefs can be *dynamic*, and evolve through interaction between agents. For example when agents exchange promises of granting each other's requests for goods or services the history of how well such promises have been kept can be used to decide how trustworthy an agent is. Similarly, when agents trade information, the history of information received from an agent and how reliable it has been can help the receiving agent decide a measure of trustworthiness.

In addition to the above, trust can be *context-dependent*. Often in real life we may trust someone for one thing but not for another. This too can either be static or could evolve through interaction.

Trust may be defined not for individuals but rather in respect to the *roles* they may have. For example we may trust the police in case of fear of some criminal attack.

In the remainder of this section we show through examples how these different considerations about trust can be formalised as abductive logic programs.

3.1 Dynamic trust policies

Let us consider *maria*'s knowledge base again. Suppose *maria* trusts all her friends who are honest. This can be represented as

$$trust(maria, X, T) \leftarrow friend(maria, X, T), honest(X, T)$$

The logic program could also contain rules defining $friend$, for example

$$friend(maria, john, T)$$
$$friend(maria, anna, T)$$
$$friend(X, Y, T) \leftarrow friend(X, Z, T), friend(Z, Y, T)$$

$$friend(maria, X, T) \leftarrow do(X, lendMoney(maria), T'), T' < T$$

The first two rules say that John and Anna are always friends of Maria. The third rule says that any friend of a friend is a friend.The fourth rule says that anyone who has lent money to Maria is her friend. All atoms in the predicate *honest* can be designated as abducible. In this way *maria* gives benefit of the doubt about agents' honesty in the absence of any information violating *maria*'s integrity constraints constraining the predicate *honest*, for example

$$honest(X, T) \wedge in\_prison(X, T') \wedge T' \leq T \Rightarrow false$$

So *maria* may trust *john* to begin with but if she learns that he is in prison or has been in prison in the past then she will change her mind.

Given that *maria* is equipped with an abductive logic program including all the above rules, any abductive proof procedure is capable of computing $trust(maria, john, 10)$ and $trust(maria, anna, 10)$, conditional on $honest(john, 10)$ and $honest(anna, 10)$, respectively. Now suppose two things happen at time 12. Firstly that *maria* is lent money by *jim* and that she learns that *john* has been in prison at time 1, i.e. *maria* makes the observations $do(jim, lendMoney(maria), 12)$ and $in\_prison(john, 1)$. Then *maria* can conclude abductively $trust(maria, anna, 13)$, (conditional on $honest(anna, 13)$),and $trust(maria, jim, 13)$, unconditionally. She cannot conclude $trust(maria, john, 13)$.

Beliefs about trust, as given by the abductive logic programs of agents, can be acquired and can evolve as the agents interact. An example of such acquisition is given by the following if-then-rule amongst the integrity constraints in the abductive logic program of *maria*:

$$trust(maria, X, T), tell(X, maria, \text{“}ok,\ I\ will\ give\ you\ G\ at\ time\ T''\text{”}, T'),$$
$$not\ do(X, \text{“}deliver\ G\ to\ maria\text{”}, T''), T \geq T', T \geq T'' \Rightarrow false$$

namely, if an agent X does not keep its promises to *maria*, then *maria* will not trust him. Here the acquisition of beliefs about trust is a result of communication. This rule is an example of revising policy taking into account evidence arising from direct interaction with agents. Direct evidence supporting (rather than denying) trust arising from interaction can also be represented. For example the following rule

$$\begin{aligned} trust(maria, X, T) \leftarrow\ & tell(X, maria, \text{“}ok,\ I\ will\ give\ you\ G1\ at\ time\ T1\text{”}, T'), \\ & tell(X, maria, \text{“}ok,\ I\ will\ give\ you\ G2\ at\ time\ T2\text{”}, T''), \\ & do(X, \text{“}deliver\ G1\ to\ maria\text{”}, T1), \\ & do(X, \text{“}deliver\ G2\ to\ maria\text{”}, T2), G1 \neq G2 \end{aligned}$$

states that Maria trusts anyone who has kept his word at least twice about delivering some goods to her.

An additional example of use of interaction to establish trust in an agent is the following if-rule within the abductive logic program of agent *maria*:

$$\begin{aligned} trust(maria, X, T) \leftarrow\ & tell(maria, X, \text{“}please,\ do\ Task\ at\ time\ T1\text{”}, T') \\ & do(X, Task, T1), T1 < T \end{aligned}$$

stating that *maria* would trust any agent who has performed a task that *maria* has requested earlier.

Indirect evidence (namely arising from communication with third parties) can be represented too. An example of use of indirect evidence is

$$trust(maria, X, T) \leftarrow tell(Y, maria, ok(X), T'), T' \leq T, trust(maria, Y, T)$$

3.2 Context-dependent trust policies

It may be advantageous to define trust with respect to specific tasks or objectives. In analogy with real life we may trust our friends to tell us the truth but we may not trust all of them to fix our plumbing or to cut our hair. Also we may trust someone to co-operate with us on a given task, even though they may not be our friend, but because we know that the successful completion of the task would help them towards their own goals. The following are some (self-explanatory) examples of context-dependent if-rules defining trust:

$$\begin{array}{l} trust(maria, X, T, Task) \leftarrow expert(X, Task, T) \\ trust(maria, X, T, Task) \leftarrow has\_a\_goal(X, G), helps(Task, G) \\ trust(maria, X, T, Task) \leftarrow tell(Y, maria, recommended(X, Task), T1), T1 \leq T, \\ \qquad\qquad\qquad\qquad\quad trust(Maria, Y, T) \\ trust(maria, X, T, Task) \leftarrow tell(maria, X, request(Task', T'), T1), T1 \leq T, \\ \qquad\qquad\qquad\qquad\quad do(X, Task', T'), similar(Task', Task) \end{array}$$

The following are some if-then-rules constraining trust:

$$\begin{array}{l} trust(maria, X, T, providing\_info), newspaper(X) \Rightarrow false \\ trust(maria, jim, T, Task), \\ not\,offer(maria, jim, payment(Task), T'), T' \leq T \Rightarrow false \end{array}$$

Namely, *maria* does not trust newspapers as far as providing information (but may trust them e.g. as far as giving wheather forecasts), and *maria* does not trust *jim* for tasks for which he has not been offered payment.

3.3 Role based trust policies

Agents may be trusted on specific tasks according to their roles. For example:

$$trust(maria, X, T, advise(Issue)) \leftarrow authorised(X, give\_advice(Issue), T)$$

where

$$\begin{array}{l} authorised(X, give\_advice(msc\_course), T) \leftarrow msc\_director(X, T) \\ authorised(X, give\_advice(hotel\_booking(Hotel)), T) \leftarrow receptionist(X, Hotel, T) \end{array}$$

Namely, *maria* trusts any agent on giving advice on any issue, if they are authorised to do so. Authorisation depends on the role they play. For example an MSc Course

Director is authorised to give advice on the MSc course, and a hotel receptionist is authorised to give advice about bookings at his hotel.

4 Using beliefs about trust

Beliefs about the trustworthiness of agents, as computed by abductive proof procedures, might be incorporated within rules of behaviour that can be used to specify private communication policies of agents. Such rules of behaviour can be expressed by means of integrity constraints [1, 14, 15, 17, 16, 20]. As an example, in the simple case where no infomation about trust is used, the abductive logic program of *maria* may contain the following integrity constraints (if-then-rules)

$$\begin{aligned} &tell(X, maria, \text{“}give\ me\ goods\ G\text{”}, T), have(maria, G, T), not\ need(maria, G, T) \Rightarrow \\ &\qquad tell(maria, X, \text{“}OK,\ I\ will\ give\ you\ G\ at\ T + 10\text{”}, T + 5), \\ &\qquad do(maria, \text{“}deliver\ G\ to\ X\text{”}, T + 10) \\ &tell(X, maria, \text{“}give\ me\ goods\ G\text{”}, T), not\ have(maria, G, T) \Rightarrow \\ &\qquad tell(maria, X, \text{“}sorry,\ I\ cannot\ give\ you\ G\text{”}, T + 5) \\ &tell(X, maria, \text{“}give\ me\ goods\ G\text{”}, T), need(maria, G, T) \Rightarrow \\ &\qquad tell(maria, X, \text{“}sorry,\ I\ cannot\ give\ you\ G\text{”}, T + 5) \end{aligned}$$

The first if-then-rule sanctions that *maria* will accept to give any goods to any agent that asks for them, given that *maria* has the goods and does not need it. In such a case the rule says that *maria* will reply 5 time points after the time the request is made. At the time of reply, *maria* sets a time for the delivery (the time of the request + 10 in the given example, but other choices would of course be possible), and gets ready for performing the delivery of the goods at that time.

The last two if-then-rules sanction that *maria* will refuse the request if she has not got the requested goods or if she needs it herself. In case of refusal, the reply could have been more informative, indicating the reasons for the refusal.

Then, assuming that the logic program of *maria* includes the additional beliefs

$have(maria, cds)$ $have(maria, torch)$ $need(maria, torch)$

a request for *cds* from *nicolas*, say, would result in an acceptance of the request by *maria* as dictated by the execution of an abductive proof procedure, whereas a request for the torch from anyone would result in a refusal.

4.1 Private communication policies - Determining how to respond to queries/requests

Now consider a richer setting where resources are exchanged or given away for periods of time. Such a scenario has been discussed in [18] and has the advantage that resources can be shared amongst agents.

Suppose agent *maria* has goods g and needs it for the period of time $[10, 20]$ for a task that she has scheduled to do during this time. Now if another agent jim asks *maria* for g for the period $[5, 8]$, say, then *maria* can grant this request provided jim gives g back to *maria* by time 10. Agent *maria*, therefore, might propose such an arrangement only if *maria* trusts jim (to give back the goods and on time). Such policies can be

generalised as follows:

$$tell(X, maria, \text{“}give\ me\ goods\ G\text{”}, [T1, T2], T),$$
$$have(maria, G, T), not\ need(maria, G, [T1, T2]),$$
$$need(maria, G, [T3, T4]), trust(maria, X, T), T3 > T2 \Rightarrow$$
$$tell(maria, X, \text{“}OK,\ I\ give\ you\ G\ from\ T1\ to\ T2,\ but\ I\ need\ G\ back\ by\ T3\text{”}, T+5),$$
$$do(maria, \text{“}deliver\ G\ to\ X\text{”}, T1-1)$$

$$tell(X, maria, \text{“}give\ me\ goods\ G\text{”}, [T1, T2], T),$$
$$need(maria, G, [T3, T4]), T3 > T2, not\ trust(maria, X, T) \Rightarrow$$
$$tell(maria, X, \text{“}sorry,\ I\ cannot\ give\ you\ G\text{”}, T+5)$$

These examples show that beliefs about trustworthiness of agents can be reasoned upon and used to regulate individual behaviour and inter-agent interaction in a principled way. The second rule states that *maria* will not agree to give anything away to someone she does not trust if she needs to get the resource back later.

4.2 Private communication policies - Deciding who to contact for one's information or resource requirements

In the following examples we have dropped the time for ease of reading.

$$need(maria, G), not\ have(maria, G) \Rightarrow ask\_for(maria, G)$$
$$ask\_for(maria, G) \leftarrow trust(maria, A), tell(maria, A, \text{“}give\ me\ goods\ G\text{”})$$

Namely, *maria* will ask some agent she trusts for some goods she needs.

A context-dependent trust example would be the following. Suppose *maria* knows two ways of getting some action done, one by doing it herself, and the other by asking someone else to do it for her, i.e.

$$happens(Action, T) \leftarrow executed(maria, Action, T)$$
$$happens(Action, T) \leftarrow requests(maria, Agent, Action, T, T'),$$
$$executed(Agent, Action, T)$$

Now it might be wise when *maria* asks some other agent to perform an action for her that she trusts that agent in doing that action, i.e.

$$requests(maria, X, Action, T, T') \leftarrow trust(maria, X, T', Action),$$
$$tell(maria, X, \text{“}do\ Action\ for\ me\ at\ time\ T\text{”})$$

4.3 Private communication policies - In answering other agents' queries about trust

The following examples show some possible use of trust beliefs about agents in answering requests for information by other agents on the trustworthiness of agents:

$$tell(X, maria, ask(trust(Y), T), trust(maria, Y, T) \Rightarrow$$

$$tell(maria, X, \text{“}yes\, I\, trust\, Y'', T+5)$$
$$tell(X, maria, ask(trust(Y), T), not\, trust(maria, Y, T) \Rightarrow$$
$$tell(maria, X, \text{“}no\, I\, do\, not\, trust\, Y'', T+5)$$

5 Concluding remarks

We have shown how an existing framework based on abductive logic programming for multi-agent systems can be used to incorporate trust without any need to change its knowledge presentation or reasoning principles. We have grounded this in several examples covering various aspects of trust. These include static trust, where information about trust of specific agents is incorporated at design time, dynamic trust, where decisions about how trustworthy an agent is can evolve from interactions with that agent or from information about him gleaned from other agents, context-dependent trust, where an agent can be trusted in some contexts or for certain tasks, but not necessarily for others, and role-based trust, where decisions about trust is based on the roles of agents rather than on their detailed individual identities.

We have shown further through examples how the concept of trust can be used within the agents' private communication policies in deciding who to contact for their requirements and how to answer other agents' requests for resources or for information. In all these cases the knowledge representation is based on the abductive framework of logic programs, integrity constraints and abducible atoms, and the reasoning is based on abductive proof procedures such as the IFF [6] and CIFF [3].

A system of abductive logic agents such as the KGP agents [10] can adopt trust polices directly as part of the knowledge base of the agents. In this way they can use information about trust in their interactions with other agents. Moreover, given the formal logic-based specification of such a system, it is possible to prove formally the conformance of the agents to their trust policies. It is also possible to represent public protocols shared by all the agents in a system (society) that incorporate trust in a similar way to representing private policies. Conformance to such public policies can also be verified a priori or at run time [4].

One of the limitations of our work, and subject of future work, is that we do not as yet incorporate any conflict resolution to cater for cases where we have conflicting information about the trustworthiness of an agent. Such conflicting information could come in answer to an agent's queries to other agents, or in how some agent A itself judges another agent B compared to its reputation, namely how others judge B. It is easy to incorporate some simple form of conflict resolution, for example one in which we always prefer the latest information, or one in which we always prefer our own assessment rather than what others tell us. It would be interesting to see how we can incorporate a more sophisticated and more dynamic form of conflict resolution. It is worth noting that although we do not deal with conflict resolution, our approach is capable of checking at design time whether or not the set of rules and integrity constraints in the knolwedge base of agents is satisfiable. This would be straightforward in many abductive proof procedure, and in particular in CIFF [3, 5].

It would be interesting to see whether our approach could be used to deal with security as well as trust. This would amount to investigate whether security protocols

could be represented in abductive logic programming, possibly following [4], and whether the execution of abductive proof procedures could help detect and possibly prevent attacks against the protocols.

Finally, further work would be required to see whether any existing computational counterparts to abductive logic programming, e.g. [12] or [3, 5], could scale up to the needs or real-world trust-based applications.

Acknowledgements

This research was supported by the EU-funded project SOCS, IST-2001-32530, within the Global Computing programme. The second author was also supported by the Italian programme "Rientro dei cervelli".

References

[1] P. Dell'Acqua, F. Sadri, and F. Toni. Combining introspection and communication with rationality and reactivity in agents. In J. Dix, L. Fariñas del Cerro, and U. Furbach, editors, *Logics in Artificial Intelligence, European Workshop, JELIA'98, Dagstuhl, Germany, October 12–15, 1998, Proceedings*, volume 1489 of *Lecture Notes in Computer Science*, pages 17–32. Springer-Verlag, October 1998.

[2] R. Demolombe. Reasoning about trust: a formal logical framework. In *Proceedings Second International Conference on Trust Management: iTrust*, volume LNCS 2995, pages 291–303. Springer Verlag, 2004.

[3] U. Endriss, P. Mancarella, F. Sadri, G. Terrreni, and F. Toni. The CIFF proof procedure for abductive logic programming with constraints. Technical report, SOCS, 2004.

[4] U. Endriss, N. Maudet, F. Sadri, and F. Toni. Aspects of Protocol Conformance in InterAgent Dialogue. In J. S. Rosenschein, T. Sandholm, M. Wooldridge, and M. Yokoo, editors, *Proceedings of the Second International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS-2003)*, pages 982–983, Melbourne, Victoria, July 14–18 2003. ACM Press.

[5] Ulle Endriss, Paolo Mancarella, Fariba Sadri, Giacomo Terreni, and Francesca Toni. Abductive logic programming with ciff: Implementation and applications. In *Proceedings of the Convegno Italiano di Logica Computazionale (CILC-2004)*. University of Parma, June 2004. To appear.

[6] T. H. Fung and R. A. Kowalski. The IFF proof procedure for abductive logic programming. *Journal of Logic Programming*, 33(2):151–165, November 1997.

[7] P. Giorgini, F. Masacci, J. Mylopoulos, and N. Zannone. Requirements engineering meets trust management. In *Proceedings Second International Conference on Trust Management: iTrust*, volume LNCS 2995, pages 176–190. Springer Verlag, 2004.

[8] L. Kagal, T.W. Finin, and A. Joshi. Trust-based security in pervasive computing environments. *IEEE Communications*, December 2001.

[9] L. Kagal, T.W. Finin, and A. Joshi. A policy based approach to security for the semantic web. In *Proceedings International Semantic Web Conference*, pages 402–418, 2003.

[10] A. Kakas, P. Mancarella, F. Sadri, K. Stathis, and F. Toni. The KGP model of agency. In *Proceedings ECAI*, 2004. To appear.

[11] A. C. Kakas, R. A. Kowalski, and F. Toni. The role of abduction in logic programming. In D. M. Gabbay, C. J. Hogger, and J. A. Robinson, editors, *Handbook of Logic in Artificial Intelligence and Logic Programming*, volume 5, pages 235–324. Oxford University Press, 1998.

[12] A. C. Kakas, B. van Nuffelen, and M. Denecker. A-System: Problem solving through abduction. In B. Nebel, editor, *Proceedings of the 17th International Joint Conference on Artificial Intelligence*, pages 591–596, Seattle, Washington, USA, August 2001. Morgan Kaufmann Publishers.

[13] R. A. Kowalski and F. Sadri. From logic programming towards multi-agent systems. *Annals of Mathematics and Artificial Intelligence*, 25(3/4):391–419, 1999.

[14] F. Sadri and F. Toni. Abduction with negation as failure for active and reactive rules. In E. Lamma and P. Mello, editors, *AI*IA'99: Advances in Artificial Intelligence, Proceedings of the 6th Congress of the Italian Association for Artificial Intelligence, Bologna*, number 1792 in Lecture Notes in Artificial Intelligence, pages 49–60. Springer-Verlag, 2000.

[15] F. Sadri, F. Toni, and P. Torroni. Logic agents, dialogues and negotiation: an abductive approach. In *Proceedings AISB'01 Convention, York, UK*, March 2001.

[16] F. Sadri, F. Toni, and P. Torroni. An abductive logic programming architecture for negotiating agents. In S. Greco and N. Leone, editors, *Proceedings of the 8th European Conference on Logics in Artificial Intelligence (JELIA)*, volume 2424 of *Lecture Notes in Computer Science*, pages 419–431. Springer-Verlag, September 2002.

[17] F. Sadri, F. Toni, and P. Torroni. Dialogues for negotiation: agent varieties and dialogue sequences. In *Intelligent Agents VIII: 8th International Workshop, ATAL 2001, Seattle, WA, USA, Revised Papers*, volume 2333 of *Lecture Notes in Artificial Intelligence*, pages 405–421. Springer-Verlag, 2002.

[18] F. Sadri, F. Toni, and P. Torroni. Minimally intrusive negotiating agents for resource sharing. In G. Gottlob and T. Walsh, editors, *Proceedings of the 18th International Joint Conference on Artificial Intelligence*. Morgan Kaufmann Publishers, August 2003.

[19] F. Toni. Automated information management via abductive logic agents. *Journal of Telematics and Informatics*, 18(1), 2001. Special issue on LocalNets: Environments for Community-based Interactive Systems.

[20] F. Toni and K. Stathis. Access-as-you-need: a computational logic framework for flexible resource access in artificial societies. In *Proceedings of the Third International Workshop on Engineering Societies in the Agents World (ESAW'02)*, Lecture Notes in Artificial Intelligence. Springer-Verlag, 2002.

Author Index