Analysis and Synthesis of Authentication and Authorisation Mechanisms Against Subterfuge

by

Hongbin Zhou, B.E., M.E.

THESIS

Presented to the Faculty of Science
National University of Ireland
Cork

for the Degree of

Doctor of Philosophy

September 2007

Abstract

Modern networks are intended to support collaborations between large numbers of autonomous entities. A coalition provides a virtual space across a network that allows its members to interact in a transparent manner. Coalitions may be formed for various purposes, such as resource sharing, information exchange, and highly structured environments in which businesses and applications are governed according to regulation and contract.

Malicious entities may exist inside and/or outside of a coalition. Such entities attempt to use coalition services without legitimate permissions. In order to defend against attacks from malicious entities, a variety of security mechanisms are needed. For example, authentication protocols provide ways to withstand attempts at impersonation; Trust Management/authorisation systems are used to ensure that only authorised entities may perform requested actions. However, many existing security mechanisms are designed in ad-hoc manners. Their design follows best practice, and withstand only classes of known malicious behaviour. Yet effective attacks on these mechanisms are often based on other unexpected behavior. Due to the lack of systematic design approaches, designing well behaved security mechanisms is a challenging task.

We are interested in the systematic verification and synthesis of security mechanisms. We first explore the analysis and synthesis of authentication mechanisms. A logic for verifying authentication protocols, and an automated tool for synthesising authentication protocols from their specifications is proposed. We draw comparisons between delegation chains and authentication-protocol message exchanges. They are both composed of a number of ordered messages that are exchanged between principals. We argue that protocol-like flaws may also affect authorisation environments. As a result, we observe the problem of *authorisation subterfuge*, whereby, a principal receiving a permission in one domain, can misuse the permission in another domain via some unexpected circuitous but apparently authorised route. Authorisation subterfuge is similar to a combination of freshness attack and parallel session attack on security protocols. Consequently, a logic for analysing subterfuge in delegation schemes, and a subterfuge-free authorisation language, are proposed.

To Ting

Acknowledgements

First and foremost, I want to thank Dr. Simon Foley, my supervisor, for his guidance, patience, and encouragement over the last five years. His guidance focused on my research directions and also on my daily life. I could not integrate into the peaceful city of Cork without his help. His patience was reflected in our discussions of research problems, and also in bearing with my initial poor english skills and my delayed work after a serious operation. I can never thank him enough for his help. If I am a better person after my studies it is largely because of him.

I also want to thank my external examiner Dr. Babak Friozabadi and Dr. John Herbert as my internal examiner. They provided a rigorous, but enjoyable, examination. Their interest and enthusiasm are greatly appreciated.

I am thankful to the Department of Computer Science in UCC. Special thanks to Prof. John Morrison, Niamh Power and Ann Hawes. I also want to acknowledge the help and friendship shown to me by the members of the Centre for Unified Computing especially: Adarsh, Barry, Brian, Chenqi, James, Keith, Max, Padraig, Philip and Tom. Special mention must go to Barry Mulcahy and Thomas Quillinan, with whom I have shared the same roof in the past few years.

I also need to thank my many friends in Cork, Zheng Cui, Jia Fan, Chunlong Huang, Albert Kwan, Wei Lan, Jing Rao, Lizhe Wang, Kaixue Wang, Guanhong Wu, Xueyan Wu, Zhenyong Yan and Ruizhi Zheng. I owe a great debt to you all. Special thanks must goes to Kuaixue, Xueyan, their mam, and their young son, Andy. They kindly provided me with a very comfortable room, delicious food and great fun during my viva preparation.

Last, but never least, thanks to my parents for their unconditional support and love. I am especially grateful to my dear wife, Ting Yao, for her love at all times. She is always emotionally supportive and she helped improve the presentation of this dissertation.

Contents

List of Figures

List of Tables

Part I

Introduction

Chapter 1

Introduction

Subterfuge is a deceptive behaviour with the goal of evading the intended controls of a security mechanism. The design of a security mechanism can be evaluated by answering the question "Is the security mechanism sufficient to prevent subterfuge from malicious principals?" This dissertation introduces and explores the notion of subterfuge in coalitions. Subterfuge provides a new and unifying perspective for understanding authentication and authorisation. Understanding subterfuge provides a new approach to developing secure coalitions. This dissertation first investigates the design of authentication protocols and decentralised authorisation mechanisms based on the analysis of authentication subterfuge and authorisation subterfuge. Then, a decentralised framework is developed to support the establishment and the management of secure coalitions.

The remainder of this chapter is organised as follows. We give a general context to our thesis in Section 1.1. Section 1.2 provides our definition of subterfuge. Section 1.3 describes the design of authentication protocols, and outlines our contributions in the area. The design of decentralised authorisation mechanisms is discussed in Sections 1.4 and 1.5. In Section 1.4, we introduce the design of authorisation languages, and address the research problem of authorisation subterfuge. In Section 1.5, we focus on the design of secure coalitions. We restate our contributions of this dissertation in Section 1.6, and give an overview of the thesis in Section 1.7.

1.1 Security in Coalitions

Modern network environments provide practical mechanisms to interconnect principals. Each of these principals can be a computer, a process, a person, an organisation, and so forth. These interconnected principals may collaborate with each other to form coalitions. A *coalition* provides a virtual space across a network that allows its members to interact in a transparent manner.

Coalitions may be formed for a wide range of purposes, for example, resource sharing, business transactions, information exchange, and so forth. A coalition's purpose should be accomplished only via a sequence of legitimate actions that are performed by the involved coalition members. For example, a business transaction should be completed only by necessary business partners; sensitive contract information should be exchanged only between contract signers.

However, malicious principals may exist inside and/or outside of a coalition. Such principals attempt subterfuge in order to accomplish the coalition goals of other principals. For example, obtaining sensitive contract information of other principals, completing a business transaction on behalf of another principal, and so forth.

Security mechanisms are designed to defend against subterfuge from malicious principals. A security mechanism ensures that coalition purposes are accomplished properly. Here, "accomplished properly" means that coalition purposes are accomplished via a sequence of legitimate actions, and each of these actions is legitimately performed by a coalition member.

Designing well behaved security mechanisms is a challenging task, since security mechanisms often fail to fulfil their goals for various reasons. One possible explanation for this is that many existing security mechanisms are designed in an ad-hoc manner. Their design follows best practice based on the experiences of their designers, and prevent only classes of known malicious behaviour. However, effective subterfuge is often based on other unexpected behaviour. Another possible explanation for this is that many existing security mechanisms are designed to work properly only for coalitions that satisfy a number of specified assumptions, but they are used for coalitions that do not satisfy all of these specified assumptions.

We argue that the design of a security mechanism can be evaluated by answering the question "Is the security mechanism sufficient to prevent subterfuge from malicious principals?" In order to answer this question, both security mechanism synthesis approaches and security mechanism analysis approaches are used in literature.

Security mechanism synthesis approaches attempt to generate well behaved security mechanisms in a systematic manner. These well behaved security mechanisms should be able to defeat a variety of malicious attempts. These malicious attempts are unified under the notion of subterfuge in this dissertation. Thus, these well behaved security mechanisms should be *subterfuge-free* security mechanisms. When a synthesis approach is provided, anyone may generate security mechanisms by following the steps that are specified in such an approach. A security mechanism generated in a systematic manner is sufficient to prevent known subterfuge vulnerabilities. It does not rely on the experiences of its designers. This is unlike security mechanisms that are designed by experience.

Security mechanism analysis approaches attempt to analyse existing security mechanisms under specified assumptions. Such an analysis approach specifies a number of possible subterfuge vulnerabilities, and verifies whether a given security mechanism may prevent those subterfuge scenarios under specified assumptions.

In this dissertation, we are interested in the design of three types of security mechanism: authentication protocols, authorisation languages, and coalition frameworks. Both analysis and synthesis approaches are proposed and used for evaluating the design of these types of security mechanisms.

1.2 Subterfuge Definition

A security mechanism allows a number of specified principals to accomplish a given task. These specified principals perform a sequence of legitimate actions to accomplish the task. The intended controls of a security mechanism are based on the deduction of these legitimate actions. Malicious principals are principals that are not allowed to accomplish this task by the security mechanism. They intend to accomplish this task by misrepresenting the true nature of the deduction via a sequence of deceptive actions. *Subterfuge* are these deceptive actions that are performed by malicious principals with the intention of evading the intended controls of a security mechanism.

1.3 Security Protocols

A number of security mechanisms are categorised as security protocols [3, 4, 89, 90, 93, 119] that are widely used for a variety of purposes, such as authentication, cryptographic key distribution, contract signing, delegation, and so forth. Well-designed security protocols should be able to withstand malicious attempts at impersonation. In other words, a round of a well-designed security protocol should be completed only by its legitimate participants. In this section, we introduce the design of security protocols.

A security protocol is a process composed of a number of ordered messages that are exchanged between two or more principals in order to achieve a number of purposes. Protocol messages may contain secret information that is deducible and usable only by certain protocol participants. When a protocol participant receives a message, it should be able to determine who sent the message, when the message was issued, and its purpose. Protocol participants use this information together with assumptions about the secret information to make decisions on how to respond. For example, protocol participants do not respond to forged messages. At the end of a round of that protocol, one or more participants can deduce their expected security related conclusions. The following example illustrates a simple

(flawed) authentication protocol for bank transactions.

Example 1 In order to start a transaction with bank B, customer A is willing to identify herself to bank B by sending her own account $acct_A$ together with her password pwd_A to bank B. However, her password is a secret shared only between bank B and herself, and should not be sent to B as a plaintext. If pwd_A is sent as a plaintext, malicious principals may obtain pwd_A by eavesdropping on the conversation between A and B, and imitate A in the future. In order to keep pwd_A as a secret only shared between A and B, a simple security protocol is demonstrated in Figure 1.1.

This simple security protocol contains two messages. First, bank B sends a message to initiate a protocol round with A. When A receives the first message, she sends the second message. The contents of the second message are her bank account $acct_A$ and password pwd_A. This message is encrypted by B's cryptographic key kb. The reason for this is to ensure that only B may understand the contents of this message. When B receives the second message, he deciphers that message and obtains $acct_A$ and pwd_A. Then, B compares $acct_A$ and pwd_A with A's bank account and password stored in the bank's database. If they are the same, A is identified by B. △

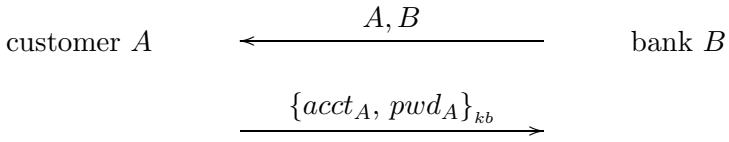
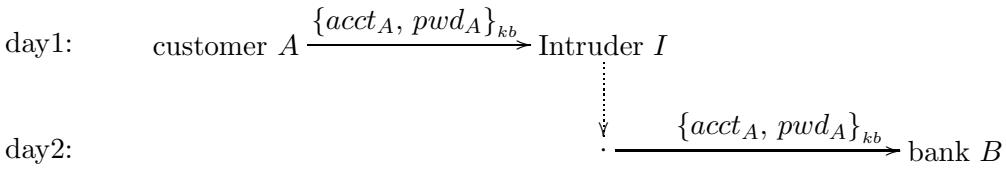


Figure 1.1: A Simple Protocol Example

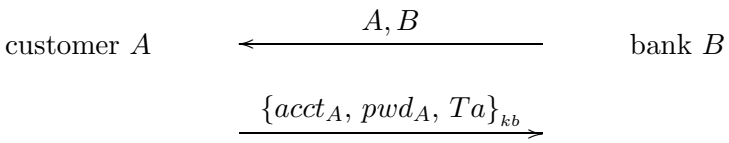
Authentication Subterfuge

A large number of security protocols are designed by experience. Security protocols that are designed by experience often contain subtle flaws that are difficult to find. Many security protocols have been demonstrated as flawed protocols long after they were proposed. For example, the Needham Schroeder mutual authentication protocol [89] was published in 1978. Seventeen years later, Lowe demonstrated a flaw in the protocol when its protocol assumptions are subtly changed [94]. This protocol flaw can be interpreted as a subterfuge vulnerability that allows malicious principals to achieve some protocol purposes by misrepresenting the true nature of the deduction via a sequence of deceptive messages. The next example demonstrates flaws in the simple bank protocol.

Example 2 As shown in Figure 1.2(a), malicious principal I may intercept the second protocol message for further use. B cannot distinguish the second message of the current



(a) Attacking the Simple Protocol



(b) A Revised Simple Protocol

Figure 1.2: Attacking and Fixing the Simple Protocol

protocol round from the second messages of previous protocol rounds. Thus, on day2, I may send this message that is intercepted in day1 in order to impersonate principal A. It is a possible way to prevent this kind of security threat by adding a timestamp Ta in the second message. The revised protocol is demonstrated in Figure 1.2(b). Bank B only responds to recently generated messages in this revised protocol. △

In order to detect whether these proposed security protocols may withstand attempts at impersonation, a variety of approaches [41, 27, 56, 103, 97, 116, 81] to the specification and analysis of security protocols were developed in the past. These approaches are very useful for discovering subtle and previously unknown flaws in given security protocols. For example, a flaw of the Andrew Secure RPC Protocol was found by Burrows, Abadi and Needham using the BAN logic [27].

These protocol analysis approaches may not be directly used as protocol synthesis approaches that generate well behaved security protocols in a systematic manner. The reason for this is that protocol analysis approaches are used after obtaining a protocol, but protocol synthesis approaches are used to obtain a protocol.

Without protocol synthesis approaches, designing well behaved security protocols remains a challenging task. Without protocol synthesis approaches, the design of existing security protocols is based mainly on the experience of security protocol designers. Security protocols that are designed by experience often contain subtle flaws that are difficult to find. For example, flaws of the Andrew Secure RPC Protocol, the Needham Schroeder symmetric key protocol, and the Otway-Rees protocol were discovered long after they were first proposed [34].

Researchers have recognised this problem, and developed a relatively small number of protocol synthesis approaches [10, 28, 59, 57, 104, 35, 32, 100]. Existing protocol synthesis approaches adopt the forward verification strategy from protocol analysis approaches. These synthesis approaches typically carry out a forward search for candidate protocols. Forward search means that possible protocol message sequences are extended by the order of protocol execution. A message may be appended to a candidate message sequence only when the message may be generated from a set of known assumptions. When a message sequence upholds all of the protocol goals, the message sequence is a candidate protocol. However, the message generating processes in these forward searching techniques are brute force, in the sense that whether a message is useful to generate a valid protocol is indeterminable. A very large protocol space must be searched in its entirety in order to obtain a valid protocol.

A result of our research is the development of a systematic approach to automatically generating security protocols in a backward manner from given protocol goals. Unlike the forward search approaches, message sequences are obtained by the converse order of protocol execution. A message is added as the first message of a message sequence when the message is required to achieve a given goal. When a message sequence can be generated from given assumptions and upholds all of the protocol goals, the message sequence is a candidate protocol. A prototype generator has been built that performs well in the automatic generation of authentication and key exchange protocols.

1.4 Authorisation Mechanisms

While security protocols mainly focus on determining the genuine identities of principals, authorisation mechanisms focus on determining whether an authenticated principal is permitted to access defined resources in defined ways. In this dissertation, we consider authorisation mechanisms for coalition schemes [62, 36, 43, 19, 38] which focus on implementing distributed authorisation. Cryptographic authorisation certificates bind authorisations to public keys and facilitate a decentralised authorisation approach.

1.4.1 Trust Management

Trust Management [62, 36, 43, 19, 38] is an approach to constructing and interpreting the trust relationships among public-keys that are used to mediate access control. Authorisation certificates are used to specify delegation of authorisation among public keys. Determining authorisation in these systems typically involves determining whether the available certificates can prove that the key that signed a request is authorised for the requested action. The following example demonstrates a delegation of authorisation among three principals using certificates.

Example 3 Figure 1.3 illustrates a chain of delegations. Certificate $C_1 = \langle BankA, Alice\rangle_t$ reflecting permission t for accessing *Alice*'s bank account is delegated from the certificate signer $BankA$ to *Alice*. Since her funds in $BankA$ are small, *Alice* decides to delegate this authority to her mobile $MobileA$ by signing $C_2 = \langle Alice, MobileA\rangle_t$. *Alice* believes that this is a reasonable strategy as, on the basis of her view of the world, Trust Management certificate reduction gives a indirect delegation $\langle BankA, Alice, MobileA\rangle_t$ (and no more).

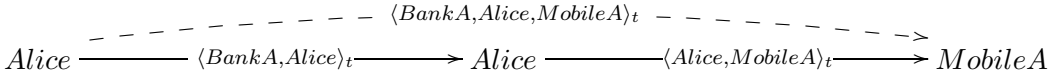


Figure 1.3: A Delegation Example

△

However, principals in Trust Management mechanisms are often ordinary users and familiar with only a limited number of resources. No principal has a complete picture of all users and resources that are available; any principal that makes authorisation/delegation decisions, does so based on its incomplete view of users and resources that are available to itself. Malicious principals may have opportunities to obtain permissions that they should not obtain in Trust Management systems. Example 4 demonstrates that when *Alice* delegates permission t from $BankA$ to $MobileA$ in Example 3, $MobileA$ may also obtain a similar permission t from another bank $BankB$.

Example 4 Continuing Example 3, suppose that *Alice* also has a bank account in $BankB$ and is unaware of the existence of the certificate $C_3 = \langle BankB, Alice\rangle_t$. *Alice* signs certificate $C_2 = \langle Alice, MobileA\rangle_t$, as recommended, believing that the resulting certificate chain (with $BankA \rightarrow Alice \rightarrow MobileA$) provides the appropriate permission t for $MobileA$ (as an account holder of $BankA$). Unknown to *Alice*, $MobileA$ could use certificate C_3 to provide an alternative certificate chain $BankB \rightarrow Alice \rightarrow MobileA$ as proof of authorisation for t. When $MobileA$ sends a request for permission t to $BankB$, $BankB$ verifies the certificates provided by $MobileA$, and then, allows $MobileA$ to perform the requested action. However, *Alice*, who delegates t to $MobileA$, does not intend to allow $MobileA$ perform the requested action for $BankB$ in which *Alice* keeps a large amount of her money.

△

The reason for this is that binding of public keys to principals does not work as traditional authentication mechanisms in the sense of withstanding malicious attempts at impersonation. Consequently, such Trust Management mechanisms are evolved only from traditional authorisation mechanisms that do not consider malicious attempts at impersonation. For example, even when malicious principals do not hold a permission, they may

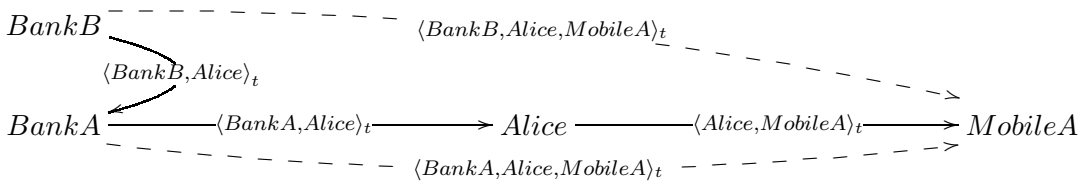


Figure 1.4: A Delegation Subterfuge Example

freely delegate the permission to ordinary users. When the resource owner cannot be determined for these ordinary users, then these ordinary users, as intermediate principals of a delegation chain,

When the resource owner is undecidable for these ordinary users, these ordinary users, as intermediate principals of a delegation chain, may be misled to inadvertently delegate un-intended permission to other malicious recipients.

1.4.2 Authorisation Subterfuge

This may result in *authorisation subterfuge* [51], whereby, a principal receiving a permission in one domain, can somehow misuse the permission in another domain via some unexpected circuitous but apparently authorised route. Subterfuge on authorisation mechanisms is similar to certain attacks on security protocols. They are both malicious attempts at impersonation. We argue that these protocol-like flaws in authorisation schemes should be analysed using security-protocol style analysis techniques. Unfortunately, prior to our work, the problem of authorisation subterfuge was not reported in literature. Consequently, to our knowledge, no existing approach addresses a particular delegation scheme may withstand attempts at authorisation subterfuge.

In this dissertation, a logic is proposed to provide a systematic way of determining whether a particular delegation scheme is sufficiently robust to be able to withstand attempts at subterfuge. We also propose a logic-based language that prevents authorisation subterfuge in large scale distributed collaborations among autonomous principals in this dissertation.

1.5 Security Mechanisms for Coalitions

Existing security mechanisms for coalitions have been designed to support both authentication and authorization of coalition participants. For example, when a customer intends to withdraw some money from a bank, the bank first verifies customer's bank account via an authentication mechanism. Once the customer's bank account is verified, the bank authorises the customer to withdraw money via an authorisation mechanism.

1.5.1 Centralised Security Mechanisms

Most traditional coalitions are similar to the above bank example in terms of security mechanisms. Such a centralised security mechanism is composed of an authentication mechanism and an authorisation mechanism. Both the authentication mechanism and authorisation mechanism rely on a "super" security administrator who controls all coalition users and permissions, and makes sure that coalition permissions are assigned to appropriate users.

The strategy of first determining who the user is and then whether that user is authorised provides best security practise for traditional coalitions. Part of the reason for this is that the "super" security administrator is familiar with all of the resources that are available and he/she makes sure that users get the appropriate permissions; no more and no less. Both coalition users and permissions are defined and controlled by the "super" security administrator and are only meaningful within the coalition. When a principal sends a request in order to perform a permission, the "super" security administrator verifies who the principal is and whether the principal is an appropriate user for its request permission. The opportunity to subvert the intentions of a good administrator is usually small.

1.5.2 Distributed Security Mechanisms

The above traditional strategy is not suitable for many kinds of coalitions. For example, several autonomous principals come together and establish a coalition to share resources. Each principal has its own security policies for sharing its resources and cooperating with others. In this scenario, the concern may be that a "super" administrator can arbitrarily authorise principals outside of the coalition. Thus, it is preferable that a coalition security mechanism should not rely on the notion of a "super" security administrator. Without a "super" security administrator, the "first authenticate, then authorise" strategy is not applicable.

In another scenario, the original security administrator of a large coalition may not be familiar with every coalition user or potential coalition user. Consequently, it is hard to assign appropriate permissions to those users or potential users. One may consider that roles can be used to help structuring and understanding coalition users. However, a role can be viewed as a special permission that is a group of several simple permissions. Assigning roles to unfamiliar principals takes the same risk as assigning permissions to unfamiliar users. In order to assign appropriate permissions only between familiar users, a delegation scheme can be introduced. In authorisation mechanisms that support delegation, particular coalition users may be directly or indirectly authorised to act as a security administrator for certain permissions. When a user is assigned a permission from one of these particular users, the user obtains that permission as assigned directly by the original security administrator.

Since permissions are not controlled centrally by the original security administrator, it is possible that the original security administrator does not know whether a user is assigned certain permission. Thus, it is impossible to have a user who controls and knows all coalition users and permissions as the "super" security administrator. Without a "super" security administrator, the "first authenticate, then authorise" strategy is also inapplicable here.

1.5.3 Secure Coalition Establishment

The design of secure coalition establishment is a further challenge. The "super" security administrator in traditional coalition frameworks [13, 53, 55, 87, 98] and top-level permission holders in modern coalitions, are appointed outside of the security mechanisms of the coalitions, and must be accepted by all coalition participants before a coalition can be established. Regulations concerning the resources under the control of the coalition administrator, should be carefully issued by the administrator, and well understood by all participants in advance. Different coalitions may require different establishment and regulations, and thus a high degree of expertise is required for an administrator to properly form and manage a coalition. We believe that coalition establishment should not be done in this ad-hoc manner, rather, it should be formalised as an integral part of the coalition framework.

In this dissertation, a formal framework for establishing subterfuge-free secure coalitions is proposed. With this framework, a coalition can be dynamically formed in a fully distributed manner without relying on a "super" security administrator, and the framework can be used to merge and spawn coalitions.

1.6 Contributions

The contributions contained within this dissertation are as follows.

1. The main contribution of this dissertation is based on the problem of subterfuge in coalitions. Subterfuge provides a unified way to think about coalition security mechanisms, such as authentication mechanisms and authorisation mechanisms.

2. We propose an automatic security protocol generator that uses logic-based synthesis rules to guide it in a backward search for suitable protocols from protocol goals. The approach taken is unlike existing automatic protocol generators which typically carry out a forward search for candidate protocols from the protocol assumptions. A prototype generator has been built that performs well in the automatic generation of authentication and key exchange protocols.

3. The problem of authorisation subterfuge is described in detail. A logic, Subterfuge Logic, is proposed to provide a systematic way of determining whether a particular delegation scheme is sufficiently robust to be able to withstand attempts at authorisation subterfuge. This logic is the first approach to analyse authorisation subterfuge. We argue that this logic is more appropriate to analyse decentralised authorisation mechanisms than existing analysis approaches.

4. We propose a logic-based language Distributed Authorisation Language (DAL) to support distributed delegation in large scale collaborations. DAL is a simple, yet expressive language. On the other hand, DAL can be used to prevent authorisation subterfuge without requiring pre-agreed global naming services.

5. Using DAL, a formal framework for establishing secure collaborations is proposed. With this framework, a collaboration can be dynamically formed in a fully distributed manner without relying on a "super" security administrator, and the framework can be used to merge and spawn collaborations.

Early versions of the results in this dissertation have published in peer-reviewed publications as follows.

- Hongbin Zhou, Simon N. Foley. A Framework for Establishing Decentralized Secure Coalitions. Proceedings of IEEE Computer Security Foundations Workshop, Venice, Italy, July 2006, IEEE CS Press.

- Hongbin Zhou, Simon N. Foley. A Logic for Analysing Subterfuge in Delegation Chain. Workshop on Formal Aspects in Security and Trust (FAST2005), Newcastle upon Tyne, UK, July 18-19, 2005.

- Simon N. Foley, Hongbin Zhou. Authorisation Subterfuge by Delegation in Decentralised Networks, In the 13th International Security Protocols Workshop, Cambridge, UK, April, 2005. Lecture Notes in Computer Science, Springer Verlag.

- Hongbin Zhou, Simon N. Foley. A Collaborative Approach to Autonomic Security Protocols, in Proceedings of New Security Paradigms Workshop 2004(NSPW2004), pages 9-16, Nova Scotia, Canada, Sept. 20-23, 2004.

- Hongbin Zhou, Simon N. Foley. Fast Automatic Synthesis of Security Protocols using Backward Search, In Proceedings of the 2003 ACM Workshop on Formal Methods in Security Engineering (FMSE'03), pages 1-10, Washington DC, October 2003.

- Simon N. Foley, Hongbin Zhou. Towards an Architecture for Autonomic Security Protocols, In Proceedings of the 11th International Security Protocols Workshop, Cambridge, UK. April 2-4, 2003. Lecture Notes in Computer Science, Springer Verlag.

1.7 Layout of Dissertation

The remainder of the dissertation is organised as follows.

Part II examines the background information and current research discussed in this dissertation. In particular, Chapter 2 examines the existing research on security protocols. The background information on authorisation mechanisms is investigated in Chapter 3. Finally, Chapter 4 gives the background information on secure coalition frameworks.

Part III contains the primary contribution provided by this dissertation. Specifically, Chapter 5 presents the BSW-ZF logic, a BAN-like belief logic. While its inference rules are similar to those of existing authentication logics, its heuristic rules are used to guide the protocol synthesis. The Automatic Security Protocol Builder (ASPB) based on the BSW-ZF logic is described in Chapter 6. In Chapter 7 we discuss the problem of authorisation subterfuge, a protocol-like flaw in authorisation frameworks. The Subterfuge Logic for verifying subterfuge in authorisation frameworks are also proposed in this chapter. Chapter 8 describes a subterfuge-free authorisation language, the Distributed Authorisation Language (DAL). A decentralised framework to support decentralised coalitions that have been specified using DAL is described in Chapter 9.

Part IV (Chapter 10) concludes the dissertation and shows directions for possible future work.

Part II

Background

Chapter 2

Security Protocols

The work described in this dissertation is related to a number of research areas, including security protocols, authorisation mechanisms and collaboration frameworks. In this chapter, we describe some relevant existing research on security protocols. This chapter begins with the basic notions that are used in security protocols. Then, existing approaches on attacking and analysing security protocols are reviewed. Finally, this chapter concludes by presenting existing approaches on designing security protocols.

2.1 Understanding Authentication Protocols

In the previous chapter, we characterized a security protocol as composed of a number of ordered messages that are exchanged between principals. Each of these principals can be a computer, a process, a person, an organisation, and so forth. Principals use information in their received messages, together with their assumptions about secret information to make decisions on how to respond. At the end of a round of a security protocol, one or more involved principals may draw certain conclusions, such as authentication, key exchange, and so forth. For example, the following protocol [100] is composed of three messages in order to implement mutual authentication between principal A and principal B.

$$\begin{aligned} &Message\ 1 \quad A \rightarrow B: \quad A, Na, \\ &Message\ 2 \quad B \rightarrow A: \quad \{Na, Nb, B\}_{K_{ab}}, \\ &Message\ 3 \quad A \rightarrow B: \quad Nb. \end{aligned}$$

In computer security, *authentication*, also known as identification or identity verification, is the process of attempting to verify identities of communication participants. It allows a principal (the authenticator) to assure that the identity of another principal (the

authenticatee) is as claimed, thereby preventing impersonation. Here, mutual authentication between A and B means that both A and B may ensure the participation of the other principal after a protocol round.

The first message is an intelligible message that is sent from A to B. An *intelligible message* is a plaintext that may be generated and understood by any principals, including malicious principals. For example, the first message (A, Na) can be generated and understood by any principal if principal identifier A and a fresh nonce Na are given. Generally, a fresh nonce is a random number that is generated by a participant of the current protocol round, and is intended to be used once and only in the current protocol round.

When principal B receives $Message$ 1, B generates the second message by using a cryptographic algorithm and a symmetric key K_{ab} that is shared only between A and B. Some cryptography-related concepts will be introduced in Section 2.1.1. For the moment, we only need to know that the second message is an unintelligible message that may be generated and understood only by protocol participants A and B.

Since the content of $Message$ 2 is understandable only by A and B, and other principals are not able to obtain Nb in the second message, $Message$ 3 may be generated only by A (assuming that B does not respond to his own generated message). The mutual authentication between A and B is satisfied, since the current protocol round may be accomplished only by A and B.

2.1.1 Preliminary Concepts on Cryptography

As shown in the above protocol, in order to ensure that only certain principals can understand and use the messages that they should receive in a given protocol round, cryptographic algorithms are used in security protocols. In order to prevent intruders (malicious principals) obtaining the intelligible message M by monitoring the network, the message sender P uses both a cryptographic algorithm and a key K to convert M to a form C that is unintelligible to anyone that is monitoring the network before transmission over the network. The conversion process is called *encryption*. The unintelligible form is called *ciphertext*. When C is received by the intended receiver Q, a reverse process uses the second key K^{-1} to recover the origin message M. The reverse process is called *decryption*. The complete process is illustrated in Figure 2.1.

The cryptographic algorithms used in encryption and decryption are generally public knowledge. The reason for this is that in the 19th century, Auguste Kerckhoffs [65] argued that a cryptosystem should be secure even if everything about the system, except the key, is public knowledge. This is widely accepted by cryptographers. By restricting who may know the encryption and decryption keys, the ability of encryption and the ability to determine

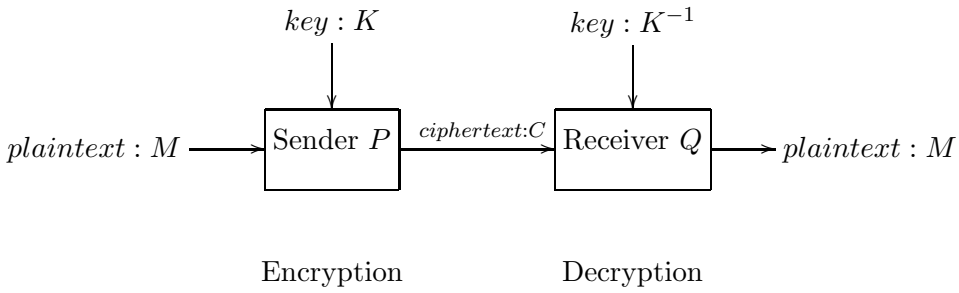


Figure 2.1: Encryption and Decryption

the plaintext from a ciphertext are restricted only to necessary principals.

Cryptographic algorithms are classified as symmetric key algorithms and public key (also called asymmetric key) algorithms. With symmetric key algorithms, such as DES [1], AES [2], and RC4 (designed by Rivest), the key K for encryption and the corresponding key K^{-1} for decryption are the same, or may be deduced from one another. Unlike symmetric key algorithms, two different but mathematically related keys, a public key and a private key, are used in public key algorithms, such as Diffie-Hellman [39] and RSA [102]. However, the calculation of the private key from the public key is infeasible. Typically, the public key is used for encryption, and may be freely distributed, while its paired private key is used for decryption, and must remain private. Public key algorithms are also used to implement digital signature schemes. In digital signature schemes, the secret key is used to process the message (or a hash of the message, or both), and the matching public key is used with the message to check the validity of the signature.

2.1.2 The Categories of Security Protocols

Security protocols can be categorised by various criteria as follows.

- According to the use of cryptographic algorithms, security protocols are categorised as symmetric key protocols, public key protocols, and hybrid protocols (in which both symmetric and public key algorithms are used);
- according to the involvement of Trusted Third Parties, security protocols are categorised as with Trusted Third Party protocols and without Trusted Third Party protocols;
- according to the number of protocol messages, security protocols are categorised as one-pass, two-pass, three-pass, and so on;
- according to the different protocol goals, security protocols can be categorised as one-way authentication, mutual authentication, key agreement, non-repudiation protocols,

and so forth.

These criteria can also be used jointly to categorise security protocols more precisely. For example, when all above criteria are used, the above protocol may be described as a three-pass mutual authentication protocol without Trusted Third Party using symmetric keys. In this dissertation, security protocols are categorised by these jointly used criteria.

2.2 Attacking Security Protocols

As illustrated in Section 1.3, the proper use of cryptographic algorithms is insufficient to guarantee proper completion of a protocol round. In this section, we demonstrate some classes of protocol attack.

2.2.1 Implementation Independent Attacks

Implementation independent attacks are the attacking strategies that rely on breaking cryptographic algorithms used in security protocols. In this section, we demonstrate freshness attack and parallel session attack. Interested readers are referred to the survey [34] in order to understand implementation-independent attacks in more detail.

Freshness Attack

Consider the following security protocol that is a variation of the protocol used by the bank B in order to authenticate customer A in the previous chapter.

$$\begin{aligned} Message\ 1 \quad & A \rightarrow B: \quad A, Na, \\ Message\ 2 \quad & B \rightarrow A: \quad \{A, Na\}_{K_{ab}}, \\ Message\ 3 \quad & A \rightarrow B: \quad \{A, pwd\}_{K_{ab}}. \end{aligned}$$

Customer A indicates that she wishes to start a round of this protocol with bank B by sending her own identifier A and a fresh nonce Na to B. When B receives $Message$ 1, he encrypts $Message$ 1 using the symmetric key K_{ab} shared between A and B, and sends it to A. Since only A and B know the key K_{ab}, A is sure that $Message$ 2 is sent by B. Then, A sends her identifier together with her password pwd to B. To avoid a malicious principals from obtaining pwd by monitoring the network, A encrypts the message by K_{ab}. When B receives $Message$ 3, B authenticates A by checking whether the message contains the correct password of customer A.

In fact, an intruder I may complete some protocol rounds with B, and convince B into believing that the initiator of those protocol rounds is A. The following attack works when an intruder I intercepts $Message\ 3$ from a previous round of this protocol. Here, $I(A)$ means that the intruder I represents another principal, a regular principal A, in the message exchange.

$$\begin{aligned} Message\ 1' \quad & I(A) \rightarrow B: \quad A, Na' \\ Message\ 2' \quad & B \rightarrow I(A): \quad \{A, Na'\}_{K_{ab}}, \\ Message\ 3' \quad & I(A) \rightarrow B: \quad \{A, pwd\}_{K_{ab}}. \end{aligned}$$

The above attack is a freshness attack, whereby, "when a message (or message component) from a previous run of a protocol is recorded by an intruder and replayed as a message component in the current run of the protocol" [34].

Parallel Session Attack

Consider the following mutual authentication protocol that corresponds to the original Needham-Schroeder Public Key protocol [89].

$$\begin{aligned} Message\ 1 \quad & A \rightarrow B: \quad \{A, Na\}_{K_b}, \\ Message\ 2 \quad & B \rightarrow A: \quad \{Na, Nb\}_{K_a}, \\ Message\ 3 \quad & A \rightarrow B: \quad \{Nb\}_{K_b}, \end{aligned}$$

Here, principal A first uses B's public key K_b to encrypt her identifier A and a fresh nonce Na. This is a nonce challenge to authenticate B, since B is the only principal (other than A) that may obtain Na by decrypting $Message\ 1$. When A receives the correct responding message $Message\ 2$ that contains Na, she can confirm that the other protocol participant is B. In order to provide a nonce challenge to A, B also encrypts a fresh nonce Nb in $Message\ 2$. When B receives the correct responding message $Message\ 3$ that contains Nb, he ensures that the other protocol participant is A.

However, the following scenario indicates that an intruder I may also complete a protocol round with B, and convince B to believe that the initiator of the protocol round is A. The

attack works by starting another protocol run in response to the initial challenge.

$$
\begin{array}{lll}
Message\ 1 & A \rightarrow I: & \{A, Na\}_{K_I}, \\
Message\ 1' & I(A) \rightarrow B: & \{A, Na\}_{K_b}, \\
Message\ 2' & B \rightarrow I(A): & \{Na, Nb\}_{K_a}, \\
Message\ 2 & I \rightarrow A: & \{Na, Nb\}_{K_a}, \\
Message\ 3 & A \rightarrow I: & \{Nb\}_{K_I}, \\
Message\ 3' & I(A) \rightarrow B: & \{Nb\}_{K_b}.
\end{array}
$$

The above attack is a parallel session attack, whereby, "when two or more protocol runs are executed concurrently and messages from one are used to form messages in another." [34].

2.2.2 Implementation Dependent Attacks

Implementation dependent attacks are attacking strategies that rely on concrete protocol implementation, including type flaw attacks, and so forth.

A type flaw attack occurs, when a field in a message that was originally intended to have one type is subsequently misinterpreted as having another type by the message recipient [110]. For example, suppose both nonces and keys are represented at the concrete level of a security protocol as bit sequences of the same length. When a principal B receives message (A, Na) in which the second component Na is originally intended to be a nonce, he accepts the message as (A, K_{ab}), and misinterprets the second component as a session key K_{ab}. If Na is available publicly, then an intruder can listen to the conversation between A and B now. Type flaw attacks can be prevented if all protocol message components are well tagged with the information that indicates its intended type [61].

If attacks on a protocol may occur only when particular cryptographic algorithms are used in the protocol, these attacks are implementation dependent attacks. Implementation dependent attacks can be prevented if proper cryptographic algorithms are used.

Here, we do not discuss these attacking strategies that are based on concrete implementation in detail. The reason for this is that the automatic security protocol generator described in this dissertation generates only abstract protocol specifications. The concrete implementation of these security protocols is a further research topic, and is not considered in this dissertation. Readers who are interested in protocol attacks based on concrete implementation are referred to Clark and Jacob's survey [34].

2.3 Specifying and Analysing Protocols

Designing well behaved security protocols is a challenging task since protocols often contain subtle flaws that are difficult to find. For example, the above Needham-Schroeder Public Key protocol [89] was proposed in 1978, and was assumed to be flawless for seventeen years. However, a flaw was discovered by Gavin Lowe [76] when its protocol assumptions are changed [94]. The protocol attacks in the previous section demonstrate that malicious principals may impersonate regular protocol participants in flawed security protocols even if perfect cryptographic algorithms are used in these protocols. In order to discover subtle flaws in security protocols, many approaches [41, 27, 56, 103, 82, 97, 116] for specifying and analysing security protocols have been developed in literature. In this section, we outline the basic approaches for specifying and analysing security protocols.

2.3.1 Specifying Protocols

Properly modeling malicious behavior is pre-requisite to analysing security protocols. However, since malicious principals are considered to be omnipotent in these early formal efforts, they are very difficult to be modelled precisely.

The Dolev-Yao Threat Model

Dolev and Yao [41] formulate a threat model to precisely represent the possible behavior of malicious principals. The Dolev-Yao threat model is well accepted as the standard threat model in protocol analysing approaches. In the Dolev-Yao threat model, malicious principals can be an individual or a group of attackers that have the complete control of the entire network. They exist between any message senders and receivers as shown in Figure 2.2.

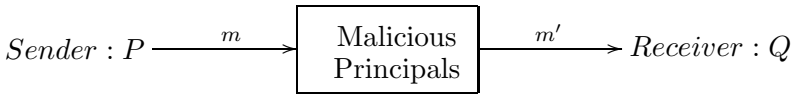


Figure 2.2: The Dolev-Yao Model

- Malicious principals are legitimate users of the network as well. They can participate in protocol rounds as legitimate protocol participants.
- When any message is sent through the network, malicious principals can not only eavesdrop passively, but can also intercept, alter, or re-route the message actively.
- They can send messages to any principal by impersonating any other principal.

However, malicious principals are not powerful enough to solve hard computational problems. This means that malicious principals do not have the following abilities:

- they cannot guess a random number that is chosen from a sufficiently large space;
- without the correct cryptographic key, malicious principals cannot compute plaintext from given ciphertext, and cannot create valid ciphertext from given plaintext, with respect to the perfect encryption algorithm;
- they cannot find the private component, such as the private key that matches a given public key.

Variant Threat Models

The Dolev-Yao threat model may express all possible powers of attackers. However, attackers in some specialized scenarios [11, 14, 26] are not as powerful as the attackers of the Dolev-Yao threat model. For example, messages are broadcast in a wireless communication environment. It is reasonable to assume that an attacker can eavesdrop any message or prevent the delivery of any message by some form of jamming. However, it is debatable to suppose that an attacker can do both at the same time.

Creese et al. proposed a number of variant threat models [37]. Each threat model variant satisfies one or more of the following restrictions that are placed on the powers of the Dolev-Yao attacker:

- the attacker cannot intercept messages on a given channel. It means that the attacker can eavesdrop a message or prevent its delivery, but cannot do both at the same time;
- the attacker cannot generate and send messages on a given channel at will;
- the attacker cannot eavesdrop messages on a given channel.

2.3.2 Analysing Protocols

Many mathematical models [27, 56, 103, 82, 97, 116] for analysing security protocols have been developed based on proper threat models. For a given security protocol, each of these models intend to determine whether it is possible for an intruder to complete a round of the given security protocol under given assumptions. A round of a well behaved security protocol should be completed only by involved principals, and these involved principals should achieve their goals under given protocol assumptions after the protocol round. If intruders can complete a round of a security protocol by impersonating other principals, the protocol is flawed. These mathematical models should be able to uncover protocol flaws or prove that a given protocol is flawless.

The BAN Logic

The first influential approach to protocol analysis in the literature is the BAN logic, a belief logic proposed by Burrows, Abadi, and Needham [27]. The BAN logic provides a formal language to represent logical statements. For example, statement $P \models X$ means that principal P believes that statement X is true; statement $P \mid\sim X$ means that P has sent a message containing X. This notation helps reveal hidden assumptions and flaws in protocols.

The BAN logic also defines a number of inference rules that are used during protocol message exchanges to change the beliefs of protocol participants. The logic applies inference rules on the protocol assumptions at each protocol step to reason about the beliefs that can be achieved by honest principals in the final state. For the sake of demonstration, we consider the following most commonly used inference rule.

Message meaning

$$\frac{P \models P \overset{K}{\longleftrightarrow} Q, P \triangleleft \{X\}_K}{P \models Q \mid\sim X}$$

That is, "if P believes that the key K is shared with Q and sees the message X encrypted under key K, then P believes that Q once said X." [27] This rule implicitly encodes the limit of the ability of a Dolev-Yao attacker that "without the correct cryptographic key, malicious principals cannot compute plaintext from given ciphertext, and cannot create valid ciphertext from given plaintext, with respect to the perfect encryption algorithm".

The BAN logic is easy to understand and use, and it has been highly successful in determining many protocol flaws, required assumptions, and so forth. However, the BAN logic is restricted in many aspects. First, the logic does not provide a systematic way to idealise protocol messages into logic statements. If a protocol message is idealised into an improper logic statement, improper beliefs are derived in the final statements. Consequently, a flawed protocol may be considered as a well behaved protocol [91, 24]. Second, the logic inference rules are not precise. For example, the logic can not represent specified trust relationship between principals. Third, the logic cannot represent some malicious behavior in the Delov-Yao threat model, such as intercept and re-route messages. Therefore, the logic fails to uncover flaws caused by these types of malicious behavior.

Due to the above mentioned restrictions, a large number of belief logics and knowledge logics, such as GNY logic [56], SVO logic [115] have been proposed by refining the protocol formalisation and redefining inference rules. However, the formalisation of belief logics is still somewhat ambiguous; the determination of protocol assumptions and the definition of inference rules are still done in an ad hoc manner. Belief logics are considered to be unable to prove the correctness of security protocols [80].

Strand Spaces

Thayer, Herzog, and Guttman [116] propose the strand space model to prove the correctness of security protocols. The use of graph theory makes the strand space model very intuitive.

In strand spaces, a protocol and its security environment are decomposed to a set of *strands* that represent the legitimate behavior of protocol participants and the malicious behavior of other principals. The behavior of malicious principals is modeled by penetrator strands. These malicious principals have the same power as in the Delov-Yao threat model. A *bundle* (a graph structure) is constructed by causal interactions (possible message exchanges) among all principals. The correctness of a protocol is proven if the bundle does not contain any penetrator strand. The proofs of the strand space model are reliable, because the model has a clear semantics, and a simple yet explicit model of possible behavior.

Compared with BAN-like logics, a significant advantage of the strand space model is that the strand space model does not idealise protocols into specific forms. Consequently, the strand space model avoids problems caused by improper protocol idealisation. In order to avoid these problems in the belief logics, Syverson [113] provides a semantics to the BAN logic based on strand spaces. Another advantage is that the Dolev-Yao threat model is explicitly expressed in strand space. Consequently, strand space may uncover all of the possible flaws caused by the Dolev-Yao attackers.

Automated Tools

Both the BAN logic and strand space are designed for manually analysing security protocols. Only experienced security experts can properly use these models to specify and analyse security protocols. In order to simplify the process of human analysis, a number of analysis approaches, such as Isabelle-based [95], Interrogator [85], the NRL Analyser [82], FDR-based [76], and Athena [107] attempt to provide automated analysis of protocols. These automated tools require that candidate protocols are described in specialised languages. These tools help understand and exclude protocols at the initial research stages.

These automated tools can be categorised into theorem proving tools and model-checking tools, and they can be used complementarily, because a protocol can only be correct (flawless) or incorrect(flawed).

Theorem proving tools attempt to prove the correctness of those protocols. With theorem proving tools, such as Isabelle [95], the protocol running environment is formalised as a set of traces that may communicate with each other. Then, a higher-order logic is used to state and prove theorems that are protocol goals. The main drawback of theorem provers is that analysis can be time consuming and provides limited support for uncovering

attacks when protocols are flawed. On the other hand, existing theorem provers are not fully automated. Considerable expertise is also required as they require a degree of manual direction.

Model-checking tools attempt to disprove the correctness of candidate protocols. This type of automatic tools, such as the NRL Analyser [82], FDR [76], and Athena [107] provide fully automatic support for protocol analysis. They simulate malicious behavior to determine whether protocols may withhold the attacks. Disproving the correctness of a candidate protocol is based on the discovery of attacking paths on the candidate protocol. Reprensenting an attacking path on the candidate protocol intuitively points out its corresponding protocol flaws. If attacks cannot be found then the correctness of protocol is considered to be proved. For example, using strand space, Song [107] develops an automated tool Athena that is effective to determine the correctness of a protocol without manual guidance.

However, failure to discover any flaw from a protocol does not guarantee that the protocol is correct (flawless). For example, when flaws of the original Woo-Lam protocol were discovered, a number of variations of the original protocol were proposed by modifying some protocol messages, and claimed to be flawless. However, flaws in these variations have been discovered subsequently [77].

2.4 Synthesising Protocols

Many approaches [41, 27, 56, 103, 97, 116] for verifying properties of security protocols have been developed in literature, however, little work has been carried out on systematic approaches to the design and development of security protocols.

2.4.1 Manual and Semi-automatic Design Approaches

Early design approaches do not provide a systematic way to construct security protocols. They only intend to help designers avoid classes of known protocol flaws. For example, Abadi and Needham [8] set out ten principles as informal guidelines for protocol design. The principles are neither necessary nor sufficient: designers can not necessarily design new protocols by obeying only these principles. Syverson [112] presents exceptions on some of the design principles by providing a number of examples. These examples show that some security protocols do not meet the principles but they are well behaved.

A number of formal design approaches [10, 28, 59, 57, 104] for security protocols have been proposed. They provide systematic ways to construct security protocols. However, these approaches are still not fully automated, and considerable expertise is required for

their use. Therefore, only experienced security experts can properly use these models to specify and design security protocols.

Gong and Syverson [57] present a novel methodology to facilitate the design and analysis of secure protocols. They introduce a novel notion of a fail-stop protocol, which automatically halts in response to any active attack. Protocol analysis in this methdology is reduced to the analysis of passive attacks. Following this methodology, the concerns of protocol design are also restricted to passive attacks, and based on well-defined practice. This is unlike general protocol analysis mechanisms. The complexity of general protocol analysis mechanisms is increased, when they intend to deal with every newly discovered attack. Gong and Syverson suggest that security protocols should be fail-stop. However, it may not be practical for some particular environments. Keromytis and Smith [66] present a generic method to create efficient fail-stop security protocols.

The Simple/BSW logic [28] is a BAN-like logic that uses the notion of channels with various access restrictions to abstract cryptographic keys. In particular, based on the logic for protocol analysis, they construct synthesis rules to guide the protocol designer in the manual systematic calculation of a protocol from its goals. However, the BSW inference rules cannot deal with authentication protocols based on message secrecy and key agreement protocols. The synthesis rules in [28] are based on the inference rules of the BSW logic. This limitation of the original BSW logic implies that only certain classes of authentication protocols can be synthesised. For example, authentication protocols based on message secrecy and key agreement protocols may not be synthesised within the BSW logic. Alves-Foss and Soule [10] describes a similar approach that generates weakest-precondition of a protocol based on the protocol operations and desired goals, although they do not actually use their results to derive security protocols.

Guttman [59] proposes a manual protocol design methodology that is based on authentication tests [60]. For different goals, individual subprotocols are generated that are combined together to form the final protocol. However, it is a manual design process and relies on the skill of the protocol designer.

Saidi [104] proposes a semi-automatic design tool based on BAN logic. It provides several commands for human intervention of the generation process. While it may simplify the protocol design process, it does not fully automatically generate protocols, and still relies on the skill of the protocol designer.

2.4.2 Automatic Design Approaches

We are interested in the automatic generation of a protocol from its protocol goals and assumptions. The automatic generation of security protocols does not rely on the experience of protocol designers. When a protocol requirement, that includes desired protocol assumptions and goals, are specified, automatic design tools may automatically generate one or more security protocols that satisfy the protocol requirement.

Existing research includes Clark and Jacob's evolutionary search [35], their successor [32], Perrig and Song's Automatic Protocol Generator (APG) [100], and ASPB described in this dissertation. All of these approaches are intuitive, easy to prove, and automatically search a large space of candidate protocols that is far larger than could be considered via a manual design.

The Evolutionary Search Approaches

Clark and Jacob propose the evolutionary search approach [35] that is the first full automatic protocol generating tool. The evolutionary search approach is based on a genetic algorithm, a fitness function, and the BAN logic.

In order to generate a protocol, the evolutionary search approach requires a set of BAN-style assumptions and intended goals. The assumptions are initial beliefs held by protocol participants. The protocol goals are beliefs that should be held after a round of the generated protocol.

The evolutionary search approach starts from a set of assumptions. A genetic algorithm is used to generate protocol message sequences in a forward manner. A fitness function is used to guide the search over the space of feasible protocols by evaluating the fitness of the current generated message sequence. This guided search approach allows increasingly fit protocols to be evolved, eventually leading to valid protocols that meet all the intended goals. The original inference rules of the BAN logic are used to systematically test whether candidate protocols uphold all protocol goals.

The evolutionary search approach is a forward search approach that generates protocol message sequence in a forward manner. Figure 2.3 demonstrates a protocol synthesis step of evolutionary search. Before step i, a sequence of messages $\text{message}_1, \ldots, \text{message}_{i-1}$ constructs an incompleted candidate protocol, and a set of beliefs $\text{belief}_1, \ldots, \text{belief}_j$ are derived from the incompleted candidate protocol and the initial protocol assumptions using the BAN logic. At step i, message_i is generated based on the fitness function. Message_i, together with known beliefs, are used to derive new derivable beliefs within the BAN logic. After step i, message_i is appended to the incomplete protocol, and the set of known beliefs is expanded to contain both the already known beliefs before step i and the new derived beliefs

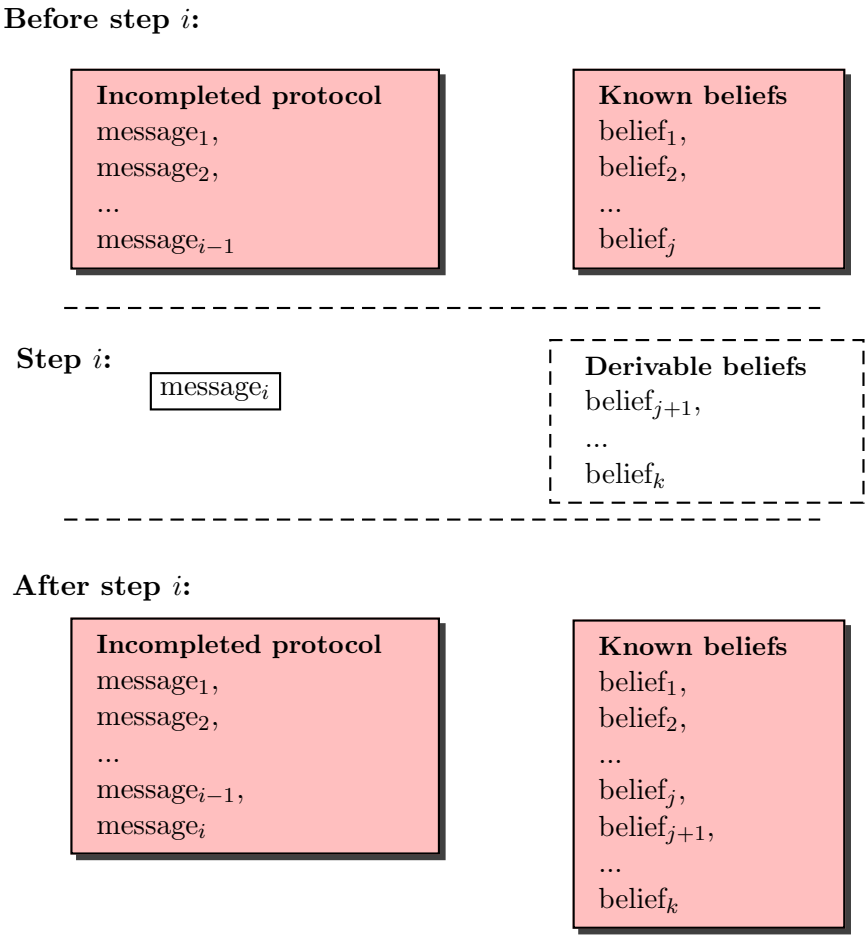


Figure 2.3: A Protocol Synthesis Step of Evolutionary Search

at step i. If the current set of known beliefs meet all the protocol goals, the message sequence message$_1$, ..., message$_i$ is a complete candidate protocol, and the search is terminated. Otherwise, protocol synthesis steps of evolutionary search are repeated.

Unlike the original evolutionary search approach [35] based on the BAN logic, their successor [32] is based on the SVO logic [115], which is more suitable for describing and analysing key agreement protocols.

However, these evolutionary search approaches have several limitations. First, some valid protocols are missed by both of these approaches. The reason for this is that both of these approaches do not search the entire space of feasible protocols. The part of the protocol space, which will be searched in an execution round, is determined by genetic parameters that are arbitrarily chosen by users. Second, providing an accurate fitness function for a protocol is very difficult, and in cases potentially impossible [35, 32]. For example, it is hard to determine how many messages and which messages are required to

construct a complete candidate protocol that meet all the protocol goals after step i in Figure 2.3. Third, protocols that are generated by evolutionary search are only proved to be valid within the corresponding belief logic. Since belief logics are unable to prove the correctness of security protocols, further correctness proof of these protocols are desired.

APG

Perrig and Song propose another automatic protocol generating tool APG [100]. APG is composed of an automatic protocol generator and an automatic security protocol checker Athena [107].

In order to generate a usable protocol, a set of security requirements and a set of desired security properties are provided. The security requirements are defined as a metric function, which defines the cost of the protocol primitives, such as protocol operations and components. The cost of a protocol is the sum of the costs of all the protocol primitives. For example, given the cost to send a nonce or a principal name and the cost to generate a fresh nonce are 1, and the cost to encrypt a message with a symmetric key is 3, the cost of the following protocol is 10.

$$
\begin{aligned}
Message\ 1 \quad & A \rightarrow B: \quad A, Na, \\
Message\ 2 \quad & B \rightarrow A: \quad \{Na, Nb, A\}_{K_{ab}}, \\
Message\ 3 \quad & A \rightarrow B: \quad Nb.
\end{aligned}
$$

The automatic protocol generator generates candidate protocols with respect to the order of increasing cost on the metric function. Some syntactical restriction rules are used to discard most severely flawed protocols in the generation process. After these generated candidate protocols are sorted by their costs, the automatic protocol checker Athena [107] examines whether these protocols satisfy the desired security properties. If a candidate protocol satisfies all the desired security properties, the generating process stops. This candidate protocol is the minimal cost valid protocol.

Since the protocols generated by APG have been verified with a powerful protocol checker Athena, the correctness of the generated protocols is guaranteed[1]. Another advantage is that APG always generates a minimal cost correct protocol, since protocols are generated with respect to the order of increasing cost.

[1]To the extent that the protocol validation approach can make such a guarantee.

ASPB

In this dissertation, we develop an automated synthesis tool ASPB to automatically generate security protocols.

ASPB is based on the BSW-ZF logic, an extension of of the BSW logic. By extending the BSW logic to the richer BSW-ZF logic, the proposed heuristic rules can synthesis a wider range of authentication and key-exchange protocols, including authentication protocols based on message secrecy and key agreement protocols. Additionally, the BSW-ZF heuristic rules have temporal order built-in so that the ordering of protocol messages can be done automatically. This is unlike the synthesis rules of the origin BSW logic that do not consider the temporal order of subgoals. When two subgoals in a synthesis rule correspond to messages exchanged between principals, then the proper order of these messages is not specified within the BSW logic. The BSW synthesis rules rely on the user to manually order the synthesised protocol steps to work effectively and are therefore insufficient to automatically generate security protocols that require a number of ordered messages.

Chapter 3

Authorisation

Beyond the design of security protocols, this dissertation is also concerned with the design of authorisation mechanisms. In computer security, authorisation ensures that only authorised principals may access data, computer programs, computer devices and other functionality provided by computer applications. Authorisation is generally implemented using access control and delegation.

This chapter starts from the basic concepts in authorisation, including access control and delegation. This section also introduces the notion of authorization subterfuge and how it relates to open and closed systems. Then, we review existing language-based approaches, including the SRC logic, Binder, and the delegation logic in Section 3.3. Our thesis concerns subterfuge in trust management and, therefore, this chapter focuses on related approaches.

3.1 Access Control

The basic access control model is a four-tuple: (S, O, A, M), where S is the set of subjects, O is the set of objects, A is the set of actions, and M is an access control function. A *subject* is an active entity, such as a person or process; a *object* is a passive entity, such as a system or file; an access control function is a matrix that maps each tuple (s, o, a) $\in S \times O \times A$ to an authorisation decision $\in \{ture, false\}$, where *true* means that the access request is permitted; $false$ means that the access request is denied.

Example 5 Table 3.1 illustrates a simple access control function of a system. The first column on the left indicates that the system has two subjects, Alice and Bob; the first row on the top indicates that the system has three objects, FileA, FileB, DirectoryC. The remaining cells display the actions that subjects may take on the objects. For example, Alice has access right *read* to FileB, but does not have access right *write* to FileB. When Alice sends the access request (Alice, FileB, *read*) to the system, the access control function

of the system returns authorisation decision *true*. However, if Alice sends the access request (Alice, FileB, *write*) to the system, the access control function of the system will return authorisation decision *false*.

	FileA	FileB	DirectoryC
Alice	*read*, *write*	*read*	
Bob	*read*	*read*, *write*	*write*

Table 3.1: A Simple Access Control Function

△

Access control can be categorised as mandatory access control, discretionary access control, and role-based access control.

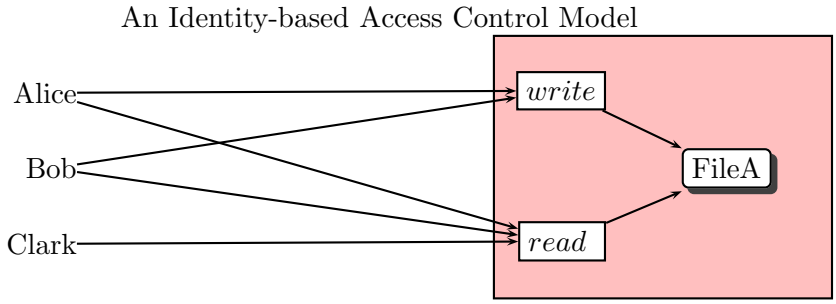
3.1.1 Mandatory Access Control

Mandatory access control is an access control policy determined centrally by system administrators instead of by object owners. In other words, the most important feature of mandatory access control is that users, who create resources, do not control these resources. The system security policy set by the administrator entirely determines the access rights granted. A user may not grant less restrictive access to their resources than the administrator specifies. For example, in Multilevel Security, system administrators assign a classification label to each subject and object. The classification label of a subject specifies the security level that the subject is trusted to access. The classification label of an object specifies the security level that is required to access that object. The security levels are partially ordered. For example, the security levels, top-secret, secret, confidential, and unclassified have the order top-secret > secret > confidential > unclassified. In order to access a given object, the subject must have a security level equal to or higher than the requested object.

Many different types of mandatory access control models are proposed in literature, such as Bell LaPadula (BLP) [16], Biba [18], Clark-Wilson [33], Chinese Wall [25]. Currently, mandatory access control is used only in systems that process highly sensitive data, such as classified government and military information. However, mandatory access control is not particularly suited to be used in many scenarios where sensitive information is not classified in a particular order. For example, in a company or an organisation, access control policies are derived from general regulations that cannot be easily classified in security levels. Role-based access control is proposed to model access control policies in such scenarios.

3.1.2 Discretionary Access Control

Discretionary access control(DAC) is an access control policy determined by the subject that is the owner of the object. Lampson [71] proposed a number of abstract models that are considered as early DAC work. The owner decides who is allowed to access the object and what kind of permission they should have. The controls are discretionary in the sense that a subject who holds an access permission may also delegate the permission to others. Since the access policy is determined by the object owner in discretionary access control, every object in a system is required to have an owner. Access control lists provide a flexible method to implement the access control function in discretionary access control. For example, Figure 3.1 demonstrates an identity based access control strategy.



Identity-Permission Map

Alice	*read*, *write*	FileA
Bob	*read*, *write*	FileA
Clark	*read*	FileA

Figure 3.1: An Access Control List Example

3.1.3 Role-based Access Control

Role-based Access Control(RBAC) [105] is an access control policy that associates users and permissions with roles within an organisation. Roles are created to perform various functions. Users obtain permissions to perform certain function by holding membership of the permission's corresponding role. Since users are not assigned permissions directly, but only acquire permissions through their roles, and it is not necessary for the system to know the full identity information of all users. For example, a request can be allowed as long as the permission requester can prove that it is associated with the proper role. In this way, RBAC helps to simplify management of individual user and their permissions. For example, Figure 3.2 demonstrates a role based access control strategy for the system in Figure 3.1.

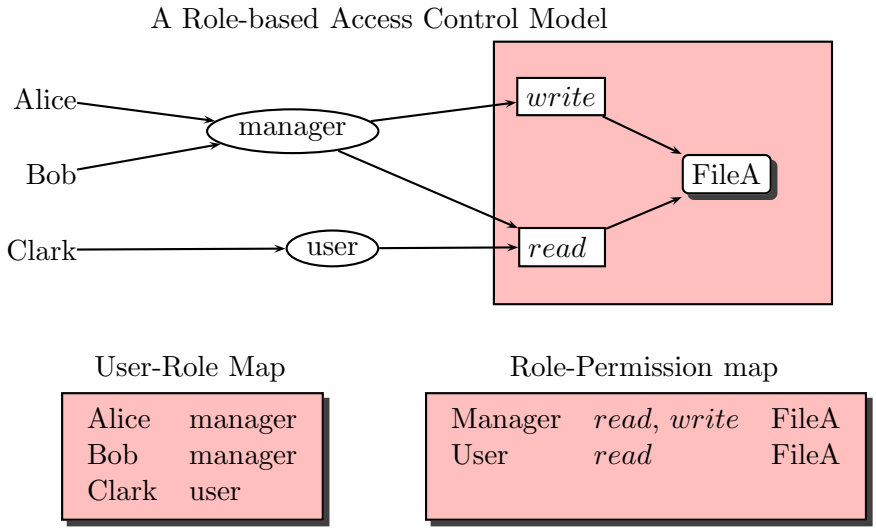


Figure 3.2: A Role-based Access Control Example

3.2 Delegation

This section considers *delegation* that is the operation of giving other principals or groups permissions to perform certain operations. In a delegation operation, the principal who gives permissions to others is the *delegator*; principals who receive permissions are *delegatees*.

Example 6 Assume Alice is the authority of permission T in Figure 3.3. The delegation operation between Alice and Bob means that Alice allows Bob to assign T to other users by issuing "Bob may assign T". In this delegation operation, Alice is the delegator, and Bob is the delegatee. △

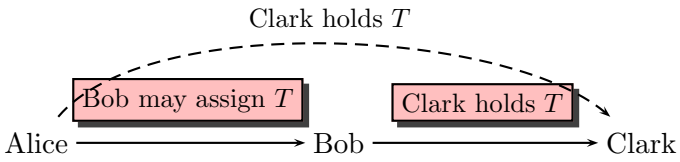


Figure 3.3: A Delegation Example

3.2.1 Direct and Indirect Delegation

If a principal obtains a permission directly from another principal, the delegation between them is a *direct delegation*. Otherwise, it is an *indirect delegation*. An indirect delegation is obtained via a chain of direct delegations.

Example 7 (Continuing Example 6) Figure 3.3 illustrates a chain of delegations. If Alice allows Bob to assign T to other users by issuing "Bob may assign T", and Bob delegates T to Clark, then Clark obtains T from Alice indirectly. The delegation between Bob and Clark is a direct delegation. The delegation between Alice and Clark is an indirect delegation. △

Indirect delegation is inapplicable in collaborations that are based on centralised access control. In such a collaboration, all of the delegation operations are centrally controlled by a single security administrator. The unique security administrator is assumed familiar with all of the collaborating users and all of the available collaboration resources. The security administrator assigns the appropriate permissions to every user directly by itself; any user, other than the administrator, may not perform permission assigning operations.

In many collaborations that are established between large numbers of autonomous principals, the above centralised authorisation strategy is ineffective, since nobody is familiar with all of the users within the system and all of the resources that are available. In order to implement authorisation in such collaborations properly, decentralised authorisation approaches use indirect delegation based on trust relationships among collaboration users. For example, the security administrator of a collaboration allows a number of trusted users to assign permissions to other collaboration users. A trusted user may assign permissions to other users based on the trust relationships between the trusted user and other collaboration users.

3.2.2 The Categories of Delegation Operations

Various categories of delegation operations are implemented in order to satisfy different purposes in flexible distributed environments. For example, if the delegatees of a delegation operation are identities, the delegation operation is an *identity-based delegation*. Similarly, if the delegatees of delegation operation are roles, the delegation operation is a *role-based delegation*. If a principal authorises anyone who satisfies one or more conditions to do a particular action, it is a *conditional delegation*. Delegation in literature supports threshold structures as well. A *threshold structure* means that at least a given number of principals are required to perform an operation. If all of the possible principal identities are listed in a threshold structure, the threshold structure is a *static threshold structure*. Otherwise, it is a *dynamic threshold structure*.

3.2.3 Closed and Open Delegation

This dissertation explores a particular delegation problem in distributed environments which we refer to as *permission name conflict*: when the same permission name is used to specify different operations in different collaborations, principals may not correctly specify the

current use of that permission name. For example, two different collaborations use the same permission name $\langle Alice\ buy * \leq \$500 \rangle$ to specify that *Alice* is allowed to buy anything less than $500 for the current collaboration. When *Alice* uses this permission, nobody can specify which collaboration *Alice* is buying goods for.

This is similar to the problem of name conflict in distributed environments, whereby, when the same name is used to identify different objects in different conditions, principals may not correctly determine which object is identified by the name. Many practical mechanisms solve this name conflict problem using global name providers, for example CORBA [58], the X.500 naming architecture [120], Enterprise Java Beans (EJB) [83] and Secure WebCom Names [101]. The use of global name providers may solve permission name conflict in closed systems. In a closed system, all of the collaborations are effectively coordinated by the same security administrator. The security administrator has a complete picture of the entire name schema for all of the resources and services that are available within those collaborations. The security administrator assigns every permission name with a unique meaning. In this way, a permission may be safely delegated in a closed system. We regard delegations in closed systems as *closed delegations*.

However, the use of global name providers may not solve the problem of permission name conflict in open systems. In an open system, collaborations are coordinated by different security administrators; no individual has a complete picture of the entire name schema for all of the resources and services that are available. Since global name providers are not collaboration security administrators, they only provide each name with a unique meaning. Principals from different collaborations may still use arbitrary names to represent their own resources, based on its incomplete view of the world.

This can result in *authorisation subterfuge* [124], whereby, in a poorly designed authorisation system, delegation chains that are used by principals to prove authorisation may not actually reflect the original intention of all of the participants in the chain. For example, the cross-domain delegation that is used by the the payment systems [22, 23, 52] based on Keynote, are vulnerable to authorisation subterfuge if care is not taken to properly identify the 'permissions' indicating the payment authorisations when multiple banks and/or provisioning agents are possible. Authorisation subterfuge will be discussed in Chapter 7 in detail.

We regard delegations in truly open systems as *open delegations*, whereby, a permission can be safely delegated in a decentralised way from one collaboration to another without ambiguity or subterfuge. To our knowledge, there is no existing authorisation mechanism that supports true open delegation. We believe that without the proper underlying support, open delegation is unreliable.

3.3 Language-Based Approaches for authorisation

Language-based approaches for authorisation aim to provide languages that support various access-control policies and various delegations in distributed systems. In this section, we review some of the most influential languages-based approaches [72, 67, 17, 20, 19, 43].

3.3.1 The SRC logic

In an early study, Lampson et al. [72] developed the SRC logic for authentication and access control in distributed systems. The basic concepts of the SRC logic are principals and statements. Principals can be basic named entities, such as users, machines, groups, roles, and so forth. Principals can also be compound principals, such as principals in roles, principals on behalf of other principals. Statements can be primitive statements, such as request, assertion, and so forth. Statements can also be composed of principals and/or statements. For example, given principals A and B, statements s and s', compound statements can be $A \mid\!\sim s$ (A says s), $s \wedge s'$ (s and s'), $s \rightarrow s'$ (s implies s), $A \Rightarrow B$ (A speak for B), and so forth. The "speak for" formula is used to represent delegation of authority. The statement $A \Rightarrow B$ is interpreted to mean that A gets all authority from B.

A number of axioms are used in the SRC logic to reason about a principal's authority by deducing statements. Some axioms are inherited from propositional logic, such as axioms $s \wedge s' \rightarrow s$, $(A \mid\!\sim s \wedge A \mid\!\sim (s \rightarrow s')) \rightarrow (A \mid\!\sim s)$, and so forth. A novel category of the SRC axioms are the *handoff* axioms. They introduce new facts about $\Rightarrow$. The basic handoff axiom is $A \mid\!\sim (B \Rightarrow A) \rightarrow (B \Rightarrow A)$. It means that B speaks for A when A says so. Using this axiom, A can delegate all its authority to another principal B.

As a language for authentication and access control, the SRC logic first formally modelled many traditional security mechanisms in centralised or closed distributed systems. However, it is limited in many respects to support open distributed systems in modern network environments. First, the SRC logic does not have re-delegation control mechanisms. Every delegation can be freely re-delegated to any other principal. However, organisations in open distributed systems prefer to provide delegation in a controlled way. For example, certain sensitive permissions may only be delegated to certain qualified principals; re-delegating these permissions to other principals is not allowed. Second, the SRC logic does not directly support threshold structures. Without this support, a delegation to a threshold of principals can only be implemented by an conjunction of many delegation statements and each statement delegates the same permission to a conjunction of principals. This kind of complex statement is difficult to implement and manage in practice. Third, SRC does not provide a formal interpretation for primitive statements.

3.3.2 Binder

Detreville [38] proposed a general and flexible logic-based security language, Binder. Binder is an extension of datalog, which is a restricted subset of the well-known Prolog logic programming language.

The Binder logic deduction is based on statements. A primitive statement in Binder is a function-like formula, which can be defined by any principal. For example, statement $employee(Alice, ComA)$ represents that $Alice$ is an employee of $ComA$. Given a term t and a primitive statement s, both s and $t \mid\sim s$(t says s) are atomic statements. Given atomic statements s_1, s_2, ..., s_n, both s_1 and $(s_2, \ldots, s_n) \rightarrow s_1$($s_2, \ldots, s_n$ implies s_1), are clause statements. Given clause statements s_i and s_j, an axiom $(s_i \wedge (s_i \rightarrow s_j)) \rightarrow s_j$ is used in Binder to reason about principal authorities by deducing statements. Delegation and trust between principals can be defined arbitrarily by principals.

Binder is well-designed and expressive. However, it is limited in many respects. First, since Binder allows ordinary users to define their own statements and policies arbitrarily, the flexibility of Binder makes it easy to mis-specify requirements. Second, similar to the SRC logic, Binder does not have re-delegation control mechanisms and does not directly support threshold structures.

3.3.3 Delegation Logic

Li et al. proposed another logic-based language, Delegation Logic [75], that supports authorisation with delegation in large-scale distributed systems.

Delegation logic [75] can represent a wide range of policies, credentials and requests in distributed authorisation. For example, Delegation Logic supports threshold structures. With this support, a delegation to a threshold of principals can be implemented by a simple delegation statement in practice.

While expressive, delegation logic is a very complex language and with 30 definitions. It can be quite challenging to write policies accurately. For example, delegation depth in Delegation logic is designed to relate just to `delegation statement`; however, a principal can bypass this and implement deeper delegation using the `if-statement` [121].

3.3.4 Trust Management Systems

Trust Management [62, 36, 43, 19] is an approach to constructing and interpreting the trust relationships among public-keys that are used to mediate access control. In trust management systems, public keys are used to represent principals; authorisation certificates are used to specify delegation of authorisation among public keys. Each certificate delegates

several specified permissions from a delegator to a number of delegatees. Determining authorisation in these systems typically involves determining whether the available certificates can prove that the key that signed a request is authorised for the requested action.

Trust Management systems are distinguishable from the SRC logic, Binder, and Delegation logic. The reason for this is that principals can only be represented using public keys in Trust Management systems. Contrasted with SRC, Binder and Delegation Logic, where principals can be represented by both global names and public keys.

A global name is a simple string that may not sign certificates as public keys. Therefore, a global name may not be verified as a public key.

These trust management languages are very expressive. Using these languages, we may express various (simple or complex) permissions and a wide range of delegation operations, for example identity-based delegation, role-based delegation, conditional delegation and threshold structures. On the other hand, all of these trust management languages allow principals to freely describe permissions. While principals may use arbitrary names to represent their own permissions, two or more permissions for different resources may be syntactically the same. As described in Section 3.2, this may result in authorisation subterfuge.

PolicyMaker and KeyNote

PolicyMaker [21] and its successor Keynote [19] first introduced the concept of trust management systems. They use policies and credentials to describe authorisation of particular actions. Both policies and credentials are written in a specific human-understandable assertion language to describe the conditions under which a principal authorises actions to other principals. Policies are issued by the local authority of actions that is the trusted root for granting these actions. Credentials are issued by principals to express delegation between principals. When there is a request for actions, a set of credentials and policies related to the requested actions are received by the local authority of the requested actions, the local compliance-checking engine provides *proof-of-compliance*, whereby a request complies with the local security policy if a set of credentials may prove that the requester holds the permission.

SPKI/SDSI

SPKI/SDSI [43, 36] is a trust management language merged from two independent approaches SPKI(Simple Public Key Infrastructure) and SDSI(Simple Distributed Security Infrastructure). It aims to provide a standard form of authorisation certificates. SPKI/SDSI has two kind of certificates, name certificates and authorisation certificates.

SPKI/SDSI name certificates are inherited from the original SDSI to provide local names as a consistent scheme for naming keys relative to one another. The reason for providing local names is that SPKI/SDSI [43] relies on the cryptographic argument that a public key provides a globally unique identifier that can be used to refer to its owner in some way, but public keys are not particularly meaningful to users. For example, the local name that Alice uses for Bob is (Alice's Verisign's Bob), which refers to Bob's public key as certified by the Versign that Alice knows. By binding local names to public keys with name certificates, principals may delegate their authorisation to others beyond their locality through a chain of local relationships.

A SPKI/SDSI name certificate is a four-tuple (K, A, S, V), where K specifies the certificate issuer's signature key, A is defined by the issuer as the local name for the subject S, and V is a validity specification to indicate the certificate's valid period. For example, $(K_{Bob}, \texttt{student}, K_{Alice}, V)$ means that Bob uses the local name `student` to indicate *Alice* in a valid period V. Given two name certificates $(K_1, A_1, K_2 B_1, V_1)$ and (K_2, B_1, K_3, V_2), we conclude $(K_1, A_1, K_3, V_1 \cap V_2)$ using four-tuple reduction rules.

SPKI/SDSI authorisation certificates are inherited from SPKI to delegate permissions between principals. A SPKI/SDSI authorisation certificate is a five-tuple (K, S, T, d, V), where K specifies the certificate issuer's signature key; tag T is the authorisation delegated from the issuer K to subject S; d is the delegation bit (0/1); and V is a validity specification to indicate the certificate's valid period. If the delegation bit is 1, the subject of this certificate is allowed to re-delegate tag T to others. If it is 0, the subject can not re-delegate T to others. Authorisation tags are expressed by s-expressions that is a LISP-like parenthesized expression. For example, when Alice is authorising Bob to perform the operations specified in T by signing authorisation certificate $\langle K_{Alice}, K_{Bob}, T, 0, V \rangle$, Bob is not allowed to delegate this authority to anyone else. For the sake of simplicity we do not consider the SPKI/SDSI validity period in the rest of this dissertation. We assume that all credentials are within their validity periods.

Given two authorisation certificates $(K_1, S_1, 1, T_1)$ and (S_1, S_2, d, T_2), if the subject of the first certificate is the issuer of the second certificate, and the delegation bit of the first certificate is 1, then SPKI/SDSI reduction rules conclude $(K_1, S_2, d, (T_1 \cap T_2))$, whereby $T_1 \cap T_2$ means that if T_1 (or T_2) is a subset permission of T_2 (or T_1), then T_1 (or T_2) is the result of this operation. If subject S_2 is someone's local name. SPKI/SDSI use four-tuple reduction rules to resolve S_2 to public keys of related principals.

RT

RT [74] is a family of role-based trust management languages that inherits local name spaces and linked local names from SPKI/SDSI. RT reinterprets the concepts of local name in SPKI to local roles, and parameterised local roles. By parameterising local roles, traditional local roles (such as student, manager) and traditional permissions are unified into the concept of RT local roles, and the problem of permission delegation and role assignment in traditional trust mangement systems are unified into one problem in RT, RT's role assignment. Since the logical correctness of the SPKI/SDSI name scheme has been proved in the literature [6, 31], the correctness of compliance checking in RT is automatically achieved. In addition, RT also provides the support of various threshold structures, and a credential chain discovery algorithm to locate and retrieve credentials that are not available locally.

Constrained Delegation

Constrained Delegation [15] provides a delegation language that focuses on transferring authority among organisations in a flexible and controllable way. Constraint structures are used to control the shape of a delegation chain. A constraint structure is a chain of group names. Only members of the last group of a constraint structure may be allowed to access a given resource. Members of other groups of the constraint structure may only be allowed to delegate permissions. In this way, a service provider may separate resource management from resource access, and decide only principals in a specified group may access its resource.

Chapter 4

Coalition Frameworks

In addition to the design of subterfuge-free security protocols and authorisation mechanisms, this dissertation is concerned with the design of secure coalition frameworks. Principals may authenticate each other using security protocols and delegate authority to other principals using authorisation mechanisms. However, neither security protocols nor authorisation mechanisms provide direct support for organising principals and cooperation controls across networks. Secure cooperation and sharing of resources across networks are supported by coalition frameworks.

This chapter begins with the concepts of coalition frameworks. This section also describes a general model for the requirements for coalition supporting frameworks. These requirements can be used to understand the existing approaches in a consistent/uniform way. Then, coalition security features are discussed in Section 4.2. Section 4.3 reviews existing coalition frameworks in the literature.

4.1 Understanding Coalitions

With the rapid growth of the Internet, information services and applications are migrating from centralised systems to distributed network-based systems. New frameworks are required to support secure cooperation and sharing of resources across networks. Sharing and cooperation controls are defined in terms of *coalitions*, and are also defined as *groups* [54, 55, 72, 108, 109] or *Virtual Organizations (VOs)* [53, 64, 87] in the literature. A coalition provides a virtual space across a network that allows its members to interact in a transparent manner.

A coalition may be established for various purposes. These range from simple spaces used by individuals to share resources and exchange information, to highly structured environments in which businesses and applications operate and may be governed according

to regulation and contract (security policy). For example, a coalition might provide a virtual space that supports the normal operational working relationship between an employee and employer. Another coalition might provide the necessary operational structure and regulation for a business-to-business relationship between organizations. Such a coalition provides a framework for forming further coalitions for business transactions; access to resources, and so forth, that constitute the procedures in the agreed relationship. At a system level, coalitions can be used to manage the relationships between its resources. For example, a distributed application may be thought of as forming a coalition between its execution components and the system resources that are available for it to use.

Coalitions may spawn further coalitions and coalitions may come-together and/or merge. Different coalitions may have their own security policies and structures. A coalition's internal regulations should not influence other coalitions or be influenced by other coalitions by reason of the language's expressive limitation, even if it's a part of another coalition. On the other hand, one coalition may be influenced by another coalition when it is necessary.

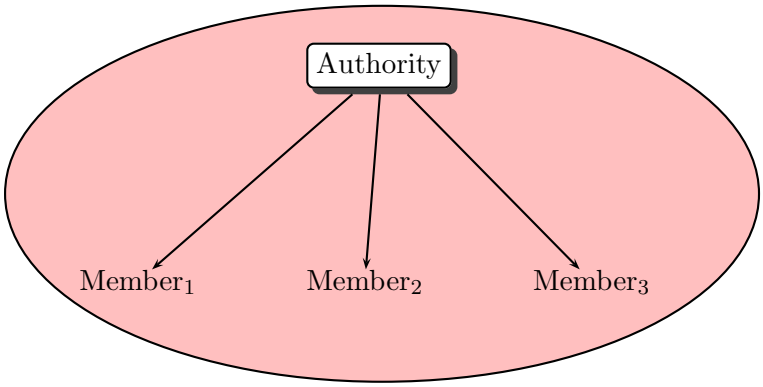


Figure 4.1: An Example for Coalition Structure

Figure 4.1 demonstrates a coalition that is constituted by a coalition authority who manages the coalition and three coalition members who provide and/or use coalition services. A coalition member can be a service provider and/or a service user. A service provider provides services to other coalition members. A service user uses services that are provided by service providers.

Figure 4.2 demonstrates the working process of an example coalition. Before a coalition user may use a coalition service, he/she should obtain a permission from a service authority who makes access decisions in step (1). Then, the user sends a request to the service provider who hosts the service in step (2). If the service provider believes that the client is authorised by a service authority to use the service in step (3), the request is allowed in step (4).

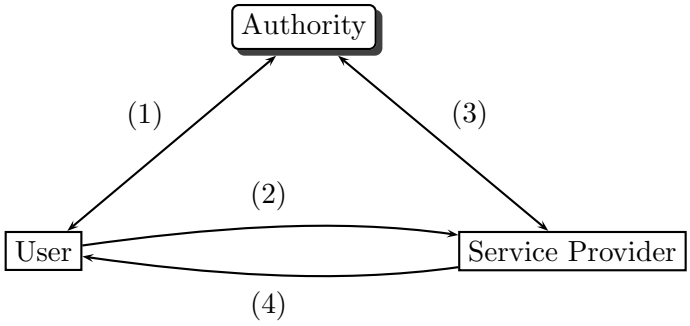


Figure 4.2: An Example for Coalition Operation

In order to manage a coalition, two kinds of access control structure are used. Principal-Member control structures are used to affiliate principals's global identifiers with the local name within the coalition. Member-Permission control structures are used to map coalition members to their permissions. For example, Figure 4.3 demonstrates these kinds of coalition access control structures.

Alice	manager
Bob	manager
Clark	user

(a) Control Structure: Principal-Member

Manager	*read*, *write*	FileA
User	*read*	FileA

(b) Control Structure: Member-Permission

Figure 4.3: An Example for Coalition Access Control Structures

4.2 Coalition Security Features

In order to achieve the purposes of a specified coalition, the coalition designers should consider many security aspects, such as coalition membership management, coalition authorisation mechanisms, the form of coalition administration, and so forth. In this section, we consider these security aspects in brief.

4.2.1 Membership Management

A coalition is constituted by its members. These coalition members are a subset of all principals over the entire network. In order to assign proper permissions only to the members of a coalition, it is important to explicitly distinguish its members from other principals.

4.2.2 Regulation-based Authorisation

DEFINITION **4.2.1** *Regulation based authorisation is a mechanism that uses regulations to govern the relationship between authorised principals and specified permissions.*

Generally, a regulation indicates that one or more principals may use a specified service. The mechanism requires the client to acquire such regulations before using a service and presenting them at the time of using the service.

This mechanism is unlike the traditional access control. In a traditional access control mechanism, all the information needed for computing a client's access rights is internally available in the service provider. In this traditional scenario, the service provider is also the service authority. However, a coalition is a virtual space for its members to cooperate. In order to cooperatively use services from different members within a coalition, some sort of coalition governors, who are not service providers, are necessary to be elected as the authorities of these services. All the information that is necessary for computing an access decision for using a service is in regulations issued by the authorities. These regulations are available outside of the service providers.

4.2.3 The Form of Administration

Coalitions can be categorised as centralised coalitions and decentralised coalitions.

DEFINITION **4.2.2** *A centralised coalition relies on a "super" security administrator.*

The "super" security administrator is familiar with all of the resources that are available and he/she makes sure that users get the appropriate permissions; no more and no less. Both coalition users and permissions are defined and controlled by the "super" security administrator. When a principal sends a request in order to perform a permission, the "super" security administrator verifies who the principal is and whether the principal is an appropriate user for its request permission. The opportunity to subvert the intentions of a good administrator is usually small.

However, the problem of single point failure is unavoidable in a centralised coalition. The reason for this is that all operations of a centralised coalition rely upon its one and only security administrator. The compromise of the security administrator results in the

compromise of the entire coalition. On the other hand, the security administrator has unlimited authority for the corresponding coalition. A "super" security administrator may not be suitable for many kinds of coalitions. For example, several autonomous principals may come together and establish a coalition to share resources. Each principal has its own security policies for sharing its resources and cooperating with others. In this scenario, the concern may be the fairness among principals, whereby no principal has advantages over any other in the coalition. This fair coalition does not allow any "super" administrator who can arbitrarily authorise any principal of the coalition. Furthermore, centralised coalitions that have a single security administrator do not scale. A "super" security administrator is inapplicable in many coalitions that are established among large numbers of principals. In such coalitions, it is possible that nobody is familiar with all users and resources within the coalition.

Decentralised coalitions are proposed in order to avoid the above problems. The operations of a decentralised coalition are controlled by a group of security administrators. There are two primary forms of decentralised coalitions.

DEFINITION **4.2.3** *A decentralised coalition in the first form has two or more security administrators. Each of these administrators has the same authority within the coalition.*

This kind of decentralised coalition is not subject to the problem of single point failure. The reason for this is that when one of these administrators fails, other security administrators may ensure that legitimate operations are allowed.

DEFINITION **4.2.4** *A decentralised coalition in the second form has two or more security administrators as decentralised coalitions in the first form. However, each administrator in the second form of decentralised coalition controls different authority within the coalition.*

This is unlike every administrator having the same authority within the first form of decentralised coalition. This kind of decentralised coalitions allows autonomous principals to share resources. Each principal can be one of the coalition administrators that is the only authority for sharing its own resources to other coalition members. On the other hand, the failure of an administrator affects only authorities that are controlled by this administrator. Other authorities of the coalition may still work properly.

4.2.4 Coalition Structures

A coalition can be structured in a top-down or bottom-up manner. Both top-down coalitions and bottom-up coalitions classify their members and/or regulations into different levels.

DEFINITION **4.2.5** *In a top-down coalition, all low level regulations are created after obtaining all high level regulations, and must be set according to high level regulations. This means that a low level regulation may not conflict with any high level regulation.*

For example, when a high level regulation defines that only coalition managers may access a file, a low level regulation may not allow other coalition members to access the specified file. Such top-down coalitions are not suitable for many scenarios. For example, a number of organisations intend to come together and establish a coalition. These organisations already have their own structures and regulations before establishing such a coalition. These regulations may conflict with each other since they are used separately. While these conflicting regulations may be considered as low-level regulations of the expected coalition, they may not result in agreed high-level regulations of the expected coalition. Therefore, a top-down coalition, which requires agreement in regulations, is not appropriate in this kind of scenario. Bottom-up coalitions can be used in the above scenarios.

DEFINITION **4.2.6** *In the above bottom-up coalition, a low level regulation may conflict with a high level regulation. When a low level regulation conflicts with a high level regulation, the applicable scope of the low level regulation is restricted.*

For example, when a number of organisations establish a coalition, existing regulations of these organisations can be considered as low level regulations in the forming coalition, and the newly defined regulations can be considered as high level regulations in the forming coalition. When a low level regulation conflicts with newly defined high level regulations in the forming coalition, an organisation may obey low level regulations within its own organisation, and obey high level regulations within the forming coalition.

4.2.5 Coalition Cooperation

The criteria to distinguish closed coalitions and open coalitions is whether a specified coalition may cooperate with other coalitions.

DEFINITION **4.2.7** *If a coalition may properly represent itself when it cooperates with other coalitions, it is an open coalition. Otherwise, it is a closed coalition.*

A closed coalition only defines and allows possible operations within the coalition. Such a closed coalition may not cooperate with principals outside the coalition. The reason for this is that a closed coalition does not have any global unified identifier to represent itself. It is impossible to distinguish a given coalition from other coalitions in current networked environments.

In order to implement cooperations among coalitions, open coalitions are proposed. An open coalition uses a global identifier to represent itself. When cooperating with other coalitions, the members and permissions of an open coalition are bound to its global identifier. Therefore, the members and permissions of an open coalition can be recognised.

4.2.6 Dynamic Establishment

Before forming or joining a coalition, the parties/principals involved may need to decide whether it would be appropriate and/or safe for them to establish the coalition. This decision is based not just on the security mechanisms and protocols that a secure coalition may provide, but also by negotiating agreeable trust relationships between the parties. This negotiation must determine to what extent the members of a coalition can be trusted and this is based on a number of attributes.

Definition **4.2.8** *Coalition establishment frameworks are mechanisms that provide templates to establish coalitions that solve the above problems.*

The parties/principals involved may also need to decide which category of a coalition is desired. For example, the coalition could be a centralised coalition or a decentralised coalition. When the category of a coalition is decided, a suitable coalition management framework is used to provide a template to manage a specified category of coalitions.

4.3 Coalition Frameworks

A *coalition framework* provides a systematic approach to establishing and managing a category of coalitions that have a number of the above features. Many coalition frameworks [53, 55, 87, 72] have been proposed to establish and manage coalitions. In this section, we review many influencing frameworks for establishing and managing coalitions.

4.3.1 The SRC Frameworks

At an early study, Lampson et al. [72] used the SRC logic to describe three frameworks SRC1, SRC2, and SRC3 for managing group membership. These frameworks are the origin of many other important frameworks.

In the SRC frameworks, a group is a virtual space that may not speak for itself. Instead, other principals speak for the group; they are its members. In order to speak for a group, a principal must become a member of that group. This is done by gaining membership from the certificate authority of the group.

SRC1

In the first framework SRC1, a group is indicated by a group name, and does not have its own public key. The certificate authority manages the group membership by listing all group members and the group name in a single membership certificate. The membership certificate is signed by the private key of the certificate authority and issued to all group members. Other than the certificate authority, any other group member may not admit or expel group members by modifying the membership certificate. When a principal is listed in a membership certificate, a principal may prove that it is a member of the listed group by presenting the membership certificate. For any principal who receives the group membership list certificate, it may distinguish whether a principal is a member of the group or not.

Figure 4.4 demonstrates how David joins group G. Membership certificate Cert_1 is issued by the authority of group G and held by all members of group G. In order to join group G, David sends a request to certificate authority of group G in step (1). If David is allowed to join G, the authority generates and issues cert_2 to all group members in step (2).

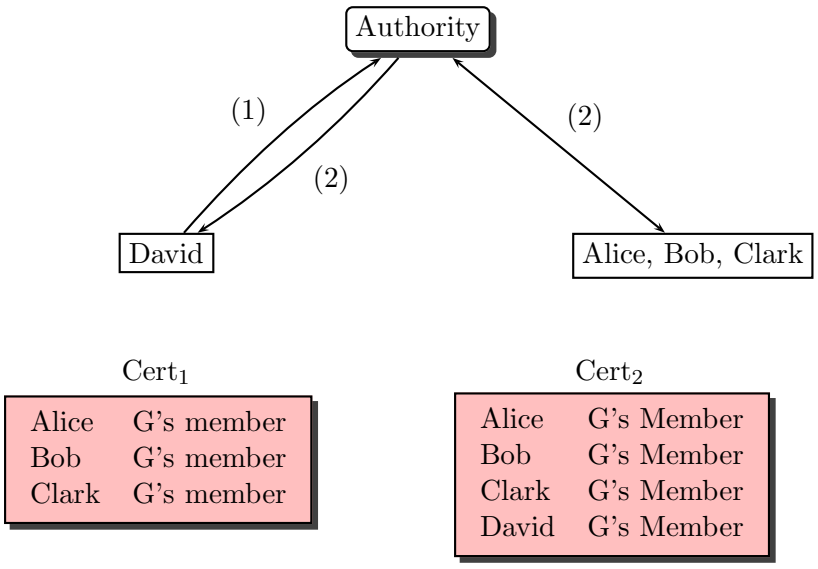


Figure 4.4: The SRC1 Model

However, if the group has a large number of members, the group membership list certificate is large. Group maintenance is difficult: gaining or losing a member results in the group having to update and re-issue the entire membership list certificate to all related principals. On the other hand, a group is controlled by a single certificate authority. Thus it is subject to the problem of single point failure. Furthermore, groups may not merge or

spawn further groups. The reason for this is that a group is only represent by a group name that may not uniquely identify the group over the network. Without a global unique identity, it is impossible to distinguish a given group from other groups. Therefore, cross-group cooperation is not possible.

SRC2

The second framework SRC2 is similar to SRC1. The only difference is that SRC2 separates the single membership certificate into a number of individual membership certificates. Each individual membership certificate only lists a principal and a group name to indicate the principal is a member of the listed group.

In this approach, gaining or losing a member does not require the update and re-issue of a certificate for the entire group member lists. To lose a member, a revocation certificate should be sent to all principals. However, it is impossible to prove whether a principal is not a member of a group without negotiating with the certificate authority.

SRC3

The third framework SRC3 assumes that a group has a group key. The certificate authority issues a membership certificate that lists a group key and a group name. The group key is then obtained by group members. A group member may speak for the group by issuing certificates signed by the corresponding group key.

However, a dishonest or compromised group member may leak the group key to others. There is no way to distinguish who uses the group key. To lose a member, the group key should be replaced by a new one, and sent to all current members in a secure way. For a large group, it's not practical. On the other hand, all of the group permissions are bound to the group membership. It is impossible to delegate an individual permission to a non-member or assign different permissions to different group members.

Table 4.1: Summary: Security Features of the SRC frameworks

Security Features	SRC1	SRC2	SRC3
membership management	√	√	√
regulation based authorisation	√	√	√
the form of administration	centralised	centralised	centralised
coalition structure	top-down	top-down	top-down
coalition cooperation	×	×	√
dynamic establishment	×	×	×

√: This feature is supported.
×: This feature is not supported or specified

4.3.2 Enclaves

Gong presented a centralized framework, enclaves [55], to support short-term groups of principals across a network. Enclaves is similar to SRC3. Comparing with SRC3, Enclaves provides details on the establishing process, and it is fully implemented.

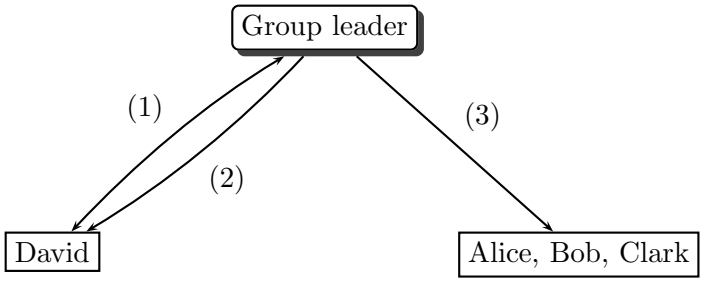


Figure 4.5: The Enclaves Model

The group leader (the same as certificate authority in SRC3) plays the key role in a group. It controls the group session keys used to provide secure communication between principals. Figure 4.5 demonstrates how an enclave group works. In order to admit a new member, the group leader and a principal mutually authenticate each other in step (1). Once authenticated, the group session key is distributed to new members (2). When a member leaves the group, the group session key is revoked and updated. The new group key is distributed only to all current group members in step (3). The leaving member may no longer access the group. The group leader maintains a role list of group members which is communicated to all members. To limit authority, a group member may only invoke predefined group communication operations though a predefined security protocol interface.

However, in a distributed environment, maintaining a large group with a high membership turnover becomes problematic. Since the session key is a symmetric key, a member may not prove its membership to non-members[1]. Another issue is that the compromise of the group leader's key results in the compromise of the entire group.

4.3.3 Virtual Private Network

Virtual Private Network (VPN) [54] is proposed for supporting secure cooperation from different physical locations of the same company. Its basic structure should be regarded as

[1]Enclaves focus on providing secure communication among members in the same group. Therefore, proving a principal's membership to non-members is not a design objective of enclaves.

similar to SRC3 and enclaves. It supports symmetric key and public key authentication. The routers or the AAA (authentication, authorization and accounting) servers are the security administrators. They keep all user information about keys and identities. In a VPN, there could be many routers, each assigned a subnet, making it scalable. However, it focuses on the authentication of individual users to join a group, and it does not address permissions and cooperation.

Table 4.2: Summary: Security Features of Enclaves and VPN

Security Features	Enclaves	VPN
membership management	$\surd$	$\surd$
regulation based authorisation	$\surd$	$\times$
the form of administration	centralised	centralised
coalition structure	top-down	top-down
coalition cooperation	$\times$	$\times$
dynamic establishment	$\times$	$\times$

$\surd$: This feature is supported.
$\times$: This feature is not supported or specified

4.3.4 The Ellison-Dohrmann Model

Ellison and Dohrmann [42] proposed a model based on SPKI name certificates for access control of mobile computing platforms.

A group has a group leader that controls all permissions of a group. The group leader represents the corresponding group using its local name. The group leader may directly admit group members by issuing name certificates that relate its local name for the group with the public keys of principals. For example, SPKI name certificate (K_G, G, K_A) means that principal K_A is a member of the leader K_G's group G. The group leader may also delegate the permission "admitting members" to other principals. For example, in order to allow a principal K_A to admit members, the group leader K_G defines a large random number n, which will be used as K_A's local name for indicating K_G's membership. Then, K_G issues certificate $(K_G, G, K_A.n)$ to K_A which means that if K_A accepts a principal as K_A's n, the principal also becomes K_G's group G's member.

This approach separates the "admit member" duty and delegate to others. It may be decentralized to admit members. However, no one can have the entire membership list, which means there is no way to prove non-membership. The roles of group members can not be distinguished. This approach is robust enough in the paper's scenario. The paper's scenario assumes that face-to-face verification about the certificate content is used when a person issues a name certificate to another. However, we found the following subterfuge problem, which limited this approach'e usage. Without a face-to-face verification, malicious

principal K_I may join K_G's G by hoaxing K_A. First, K_I intercepts certificate $(K_G, n, K_A.n$ and issues $K_I, G, K_A.n$ by using the same random number n. When K_A issues (K_A, n, K_C) to admit K_C as a member of K_I's group G. However, K_C may use (K_A, n, K_C) and $(K_G, n, K_A.n$ to prove that it is admitted by K_A as a member of K_G's group G.

4.3.5 The Mäki-Aura Model

Mäki and Aura [79, 13] presented a distributed security architecture based on groups and public-key certification for access control in ad-hoc networks.

A principal establishes a new group by generating a group identifier that is a signature key. At the moment, the principal is the only member of the group. Since the group identifier is a signature key, it is used to sign certificates to admit new members and verify membership. The principal is a group leader because he/she owns the group key. A group leader may admit a group member by issuing a member certificate. A group leader may also appoint another group leader by issuing a leader certificate. When a principal is appointed as a group leader, he/she can use its own signature key to sign certificates for admitting members and appointing leaders.

All leaders have the same authority as the original leader. As a consequence, a group can survive the leaving of a group leader. In this way, the model does not rely on centralised administration. However, as in [42], the membership list is distributed across leaders, and the resignation of a leader can result in an incomplete view of the group membership. Several revocation methods are supported: group reconstitution (replace the original group key and reissue all certificates) and short membership validity time (members must renew their membership regularly).

This model supports group spawning. To spawn a subgroup, a group leader generates a signature key for identifying a subgroup first. Then a subgroup certificate is issued to indicate that the subgroup key is the group identifier for a subgroup of the origin group. Now, the subgroup key can be used to admit members and appoint leaders for the subgroup.

By defining different group roles, this model may delegate different rights to different types of members. However, this model has some limitations. Firstly, the compromise of a group leader key results in the compromise of the entire group. Secondly, regardless of whether the other group leaders agree, a group leader can use its own full authority to issue group credentials. Once a principal has obtained the group leader role, it can not be stopped from issuing legal or illegal credentials.

Table 4.3: Summary: Security Features of the ED Model and the MA Model

Security Features	Ellison-Dohrmann	Mäki-Aura
membership management	√	√
regulation based authorisation	×	√
the form of administration	decentralised	decentralised
coalition structure	top-down	top-down
coalition cooperation	×	×
dynamic establishment	×	×

√: This feature is supported.
×: This feature is not supported or specified

4.3.6 Security Frameworks in GRIDs

GRIDs provide collaborative computing infrastructures that allow collections of users and institutions from different regions to work together under certain prearranged rules. Many architectures for GRID security are proposed, such as GSI [53], CAS [98], VOMS [9], PERMIS[30], and Akenti [117].

The GRID Security Infrastructure

The GRID Security Infrastructure (GSI) [53] is a groundwork of GRID security frameworks. GSI uses a modified version of the X.509 Public Key Infrastructure (PKI) [62]. GSI uses centralised security administrator CA to identify and manage grid users or services. The CA has complete and absolute control over the grid that she manages. Its user information is stored in the grid mapfile, which is a map file for mapping the principal's global identifiers to local identifiers within the current grid.

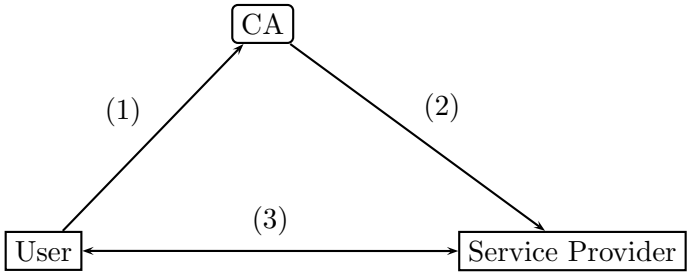


Figure 4.6: The GSI Model

Any user must register and sign-on to the grid before using it. As shown in Figure 4.6, a user, who wants to join a grid, sends the X.509 certificate which proves its global identifier to the CA (step (1)). The CA verifies this X.509 certificate, and assigns a local name to this

user. At the meantime, CA updates, signs and re-issues the grid mapfile to all local service providers (step (2)). After receiving the new grid mapfile, Service providers maintain the current grid user list. Since a service provider controls its own service, she maps the user's local names in the grid to its local name space, and manages the access control policies for grid users as local users. Now, a grid user may access a grid service by negotiating with the corresponding service provider in step (3).

Since a service provider (not the CA) decides whether a grid user may use a specified service, GSI does not manage services at the coalition level. On the other hand, GSI does not support roles, subgroups in the current implementation. When gaining or losing a user, the mapping file should be updated and sent to all service providers. The service providers should update its local policies based on the mapping file. Therefore, managing a large number of users that are not categorised by subgroups and roles is an arduous challenge to both CA and resource providers. Different grids may use different architectures and, therefore, cross-GRID authorisation must be done manually: dynamic establishment of virtual organisations is not possible.

Community Authorisation Service

Based on GSI, Pearlman et al. [98] describe a Community Authorisation Service (CAS) for solving the above problem. Similar to enclaves, a CAS server serves as the grid permission authority. As shown in Figure 4.7, the rights for accessing grid services are delegated from service providers to the CAS server in step (1). When a user wishes to join a grid, it authenticates itself to the CAS server using the same approach in GSI. Then, the CAS server does not only map a user's global identity to the grid's local identity, but also maps the user's local identity to suitable permissions. In this way, a CAS server manages both the grid membership list and the grid access control list. The grid mapfile contains not only the columns for user's public keys, global identifiers and their local identities in the GRID, but also the "permission" column, which means CAS also delegates the grid permissions to its local identities. In a service provider's local access control list, only the CAS server's identity is mapped to a local identity. Therefore, a service provider does not need to know and manage the whole grid user list in its local user list. When a user wants to access a grid service, she must login to the CAS server in step (2). The CAS server then verifies the user's identity, and signs a suitable authorisation certificate based on the grid mapfile. This authorisation certificate is issued to the user in step (3). After that, the user uses its membership certificate from the CAS server to prove its identity, and uses the authorisation certificate from the CAS server to prove that it is authorised to access a service provider's service in step (4). The service provider responds in step (5).

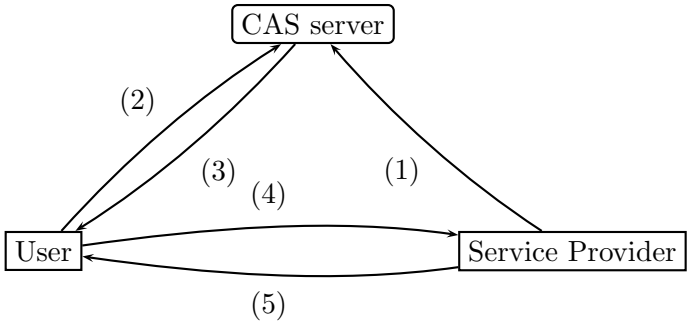


Figure 4.7: The CAS Model

In the CAS model the grid policy determines whether a user may access a grid service. After delegating the right to the CAS server, a local administrator has no right to stop the CAS server delegating rights to an unexpected principal, who is a grid user, but an unaccepted user by the service provider's local policy. Like GSI, CAS does not support roles or subgroups. Managing a large number of users without subgroups and roles is still an arduous challenge for the CA. A coalition is still a centralised identity-based system. The CAS server does not support the automatic establishment of new grids.

Virtual Organization Management Service

Virtual Organization Management Service(VOMS) [9] is another security solution based on GSI. Unlike GSI and CAS, VOMS is a role-based security architecture. Similar to GSI, a VOMS server also maintains a grid mapfile. However, a VOMS mapfile does not only contain global and grid local identities of its members, but also their roles within the grid. It means that VOMS supports group and roles in a grid. On the other hand, unlike mapfiles in CAS, a VOMS mapfile does not contain the permission column. The reason for this is that the grid service is not controlled by the VOMS server, but remains the right to the service providers. The service providers may authorise a group of grid users certain permission once. Generally, the work flow in VOMS is the same as in GSI. However, since the access control is based on roles and groups, managing a service provider's local permission is much easier than GSI and CAS.

PERMIS and Akenti

PERMIS[30] and Akenti [117] are two further models. While PERMIS supports grid roles, Akenti does not support grid roles. Their grid architectures and mapfiles are similar to CAS. However, their working processes are different from above models. As shown in Figure 4.8,

Table 4.4: Summary: Security Features of GSI, CAS and VOMS

Security Features	GSI	CAS	VOMS
membership management	√	√	√
regulation based authorisation	×	√	×
the form of administration	centralised	centralised	centralised
coalition structure	top-down	top-down	top-down
coalition cooperation	×	×	×
dynamic establishment	×	×	×

√: This feature is supported.
×: This feature is not supported or specified

when a user wants to access a service, it first sends a request, together with useful certificates obtained by the user, to the service provider in step (1). The service provider checks the certificate first. After the provider's local verification, the request, and related certificates are sent to the grid CA in step (2). The CA verifies the request by local policies and received certificates and makes the final decision. Then, the decision is sent to the service provider in step (3). At the last step, the service provider responds to the user.

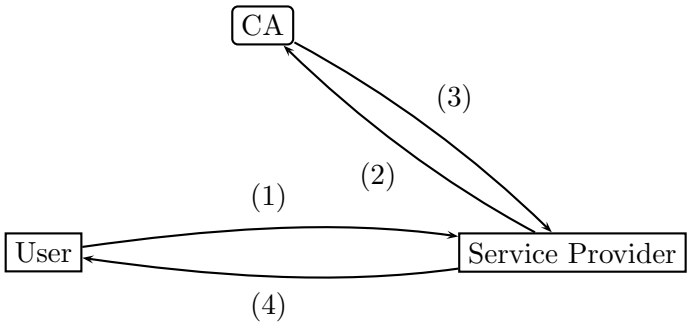


Figure 4.8: PERMIS and Akenti

In PERMIS and Akenti, the grid CA does not directly deal with requests from grid users. Instead, the service provider deals with them the first place. Only valid requests are passed to the grid CA. CA's workload can be significantly decreased when a great lot of invalid requests are received. This helps a CA to survive from DoS attacks.

Contractual Access Control Model

Firozabadi et al. [47] proposed a framework, Contractual Access Control Model, for sharing resources among coalitions. Unlike other GRID security frameworks, CACM does not rely on a grid security administrator. Cooperation among coalition members is regulated by a particular type of contract that is agreed by corresponding users and service providers.

As shown in Figure 4.9, after negotiating with a user, a service provider defines and signs a contract in step (1). The contract is a coalition security policy that defines a sequence of agreed obligations of the service provider that can be provided to a user. Each of these obligations specifies the type of service that can be used over a given time period. Different obligations specify the different amount of service that are available over different time periods. In this way, a user may freely choose the time period and the amount of service to use according to the contract provided by the service provider. Then a user may send a request for using the service in step (2).

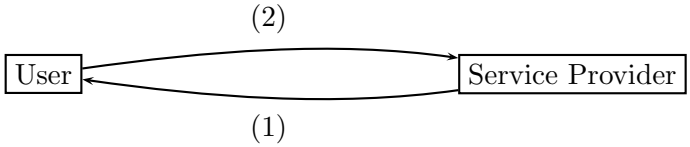


Figure 4.9: Contractual Access Control Model

In CACM, a coalition is a decentralised identity-based system. A service provider determines whether a user may access its service according to its own decision, and the decision may violate its local policy when it is necessary. On the other hand, the CACM supports cooperation only among identifiable principals. An identifiable principal should have a signature key for verifying its identity by other users. How an newly established coalition obtains its signature key and who controls this signature key is not specified. Therefore, whether a multi-level coalition can be established in this approach is still in question.

Table 4.5: Summary: Security Features of PERMIS, Akenti and CACM

Security Features	PERMIS	Akenti	CACM
membership management	√	√	×
regulation based authorisation	√	√	√
the form of administration	centralised	centralised	decentralised
coalition structure	top-down	top-down	bottom-up
coalition cooperation	×	×	√
dynamic establishment	×	×	√

√: This feature is supported.
×: This feature is not supported or specified

4.3.7 Other Approaches

Law-governed interaction (LGI) [87] is a top-down coalition management framework. As we discussed in Section 4.2.4, a top-down coalition, which requires agreement in regulations, is not appropriate in many scenarios.

Trust Management-based approaches provide finer-grained security and greater scope for forming coalitions. However, frameworks for dynamically establishing coalitions have not been proposed.

Dynamic Administrative Coalitions (DAC) [50] is a decentralised coalition management framework that introduces distributed administration workflows based on condensed graphs. However, since DAC is based on Keynote[19], cross-coalition delegation in DAC is vulnerable to authorisation subterfuge if care is not taken.

Part III

The Design of Security Mechanisms

Chapter 5

The BSW-ZF logic

In this chapter, we present the BSW-ZF logic, a BAN-like belief logic. The BSW-ZF logic improves and extends the BSW logic [28] to support reasoning about message secrecy, fresh channels, and 'holding' statements. While the inference rules of the BSW-ZF logic are used to verify some properties of security protocols, the heuristic rules of BSW-ZF logic are used in ASPB as the core technique to guide the automatic backward search for candidate subprotocols from their goals. Section 5.1 introduces the notation of the BSW-ZF logic. The inference rules of the BSW-ZF logic are given in Section 5.2. Section 5.3 describes how the inference rules are used to verify security protocols. Section 5.4 discusses our rationale for extending the BSW logic. Section 5.5 presents the heuristic rules of the BSW-ZF logic, and describes how these rules are used to guide the 'calculation' (in the sense of [40]) of a protocol from its goals. How these rules are used to guide an *automated* search for protocols is described in the next chapter.

5.1 Notation

The BSW-ZF logic uses abstract channels similar to the Spi Calculus [7] to represent keyed communication between principals. The capability to write into (e.g., using the encryption key) and to read from (e.g., using the decryption key) a channel C is denoted by $w(C)$ and $r(C)$, respectively. The formula $P \ni r(C)$ means that principal P has the capability to receive messages from channel C, and correspondingly, $P \ni w(C)$ means that principal P has the capability to send messages to channel C.

Let $\sigma(X)$ denote the set of principals that share a secret X. The secret X can be a temporary secret that is held during several protocol steps, or a long term secret. We assume that the secret X is never leaked to any principal outside $\sigma(X)$, other than to those principals that are trusted by the members of $\sigma(X)$. For example, $P \models (\sigma(X) = \{P, Q\})$ means that P believes X is a secret shared between P, Q, and any third parties who (trusted

by P and/or Q) may be privy to X. In this case, it is assumed that a trusted principal will neither use X as a proof of identity nor as a channel to communicate with. With this assumption, only two principals, P and Q, may use X as a proof of identity or as a channel to communicate with. When either of them receives X in a message, the principal may determine whether the message was sent by itself or by the other party. Thus, the message origin can be safely determined. Also note that this assumption implies that principals trusted by other protocol participants will not attack the current protocol.

The set of principals that can receive and can send messages via a channel C is denoted by its reader set $\sigma(r(C))$ and its writer set $\sigma(w(C))$, respectively. For example, if Ω represents the set of all principals, then $\sigma(r(C)) = \Omega$ and $\sigma(w(C)) = \{P\}$ represents an authentic channel, whereby any principal can authenticate messages signed by the private key of principal P.

In the following, P,Q range over principals; C represents a channel; X represents a message which can be data or formulae or both; ϕ represents a formula. Data, such as principal identities, nonces, and read and write channel properties, are atomic messages that may be held by principals. The BSW-ZF logic uses the following basic formulae.

$P \triangleleft X$**:** Principal P sees message X. Someone has sent X via a channel that P can read.

$P \triangleleft C(X)$**:** P sees X on channel C. Someone has sent a message X via channel C. If P can not read C then P can not discover the contents of X.

$P \mid\sim X$**:** P once said X. P sent a message containing X at some point in the past. We do not know exactly when the message was sent.

$P \parallel\sim X$**:** P says X. P sent X in the current run of the protocol.

$\sharp(X)$**:** Message X is fresh. X has never been said before the current run of the protocol. This is usually true for messages containing fresh nonces or messages sent using a fresh session key (channel).

$P \models \phi$**:** P believes that ϕ is true. It does not mean that ϕ is actually true, rather, P believes it.

$P \ni X$**:** P holds X, and therefore, may freely combine X with other messages and send the resulting combinations out to other principals. Unlike $P \triangleleft X$ requires that X is obtained by P during a round of protocol, $P \ni X$ can be obtained as an assumption before a round of protocol.

$P \mapsto X$**:** P generates X. This formula represents the origin of message X. The message X is used to represent constants or the atomic messages that are generated by the principal. Examples include nonces and principal identifiers.

The logic also uses the conventional logic operators $\wedge$ (conjunction), $\vee$ (disjunction) and $\rightarrow$ (implication) from propositional logic and some basic notation from set theory, such as $=$ (equality), and $\cup$ (union).

Example 8 Consider a mutual authentication requirement between principals A and B. We require that A believes that B speaks with A, and vice versa. These goals may be expressed as follows.

$$G_1: \quad A \models (B \mid\sim A)$$
$$G_2: \quad B \models (A \mid\sim B)$$

We assume that A and B share symmetric keys (abstracted as channels Cas and Cbs, respectively) with a third party S. Assumptions are defined as follows.

$$A \ni r(Cas); \qquad A \ni w(Cas);$$
$$S \ni r(Cas); \qquad S \ni w(Cas);$$
$$B \ni r(Cbs); \qquad B \ni w(Cbs);$$
$$S \ni r(Cbs); \qquad S \ni w(Cbs);$$
$$A \models (\sigma(w(Cas)) = \{A, S\}); \qquad A \models (\sigma(r(Cas)) = \{A, S\});$$
$$A \models (\sigma(r(Cbs)) = \{B, S\}); \qquad S \models (\sigma(w(Cas)) = \{A, S\});$$
$$S \models (\sigma(r(Cas)) = \{A, S\}); \qquad B \models (\sigma(w(Cbs)) = \{B, S\});$$
$$B \models (\sigma(r(Cbs)) = \{B, S\}); \qquad B \models (\sigma(r(Cas)) = \{A, S\});$$
$$S \models (\sigma(w(Cbs)) = \{B, S\}); \qquad S \models (\sigma(r(Cbs)) = \{B, S\});$$

We express the public channel, to which everyone may read from and write to, by the term Cp. In this way, all messages are exchanged via some channel. Assumptions about this channel are defined as follows.

$$A \ni r(Cp); \qquad A \ni w(Cp);$$
$$B \ni r(Cp); \qquad B \ni w(Cp);$$
$$S \ni r(Cp); \qquad S \ni w(Cp);$$

Further assumptions are that both A and B trust S as a trusted third party:

$$\begin{aligned} &A \models ((S \mid\!\sim \phi_1) \rightarrow (S \models \phi_1)) && \text{[honesty]} \\ &A \models ((S \models (B \mid\sim \phi_2)) \rightarrow (B \mid\sim \phi_2)) && \text{[competent]} \\ &B \models ((S \mid\!\sim \phi_1) \rightarrow (S \models \phi_1)) && \text{[honesty]} \\ &B \models ((S \models (A \mid\sim \phi_2)) \rightarrow (A \mid\sim \phi_2)) && \text{[competent]} \end{aligned}$$

for arbitrary ϕ_1, ϕ_2. These formulae reflect the beliefs of A (and B) that S is honest and that S is competent in deciding whether B (and A) at some time in the past said some message.

Further assumptions include $A \models \sharp(Na)$ and $A \mapsto Na$. These reflect that Na is a fresh nonce that is generated by A. B has similar beliefs: $B \models \sharp(Nb)$ and $B \mapsto Nb$. These reflect that Nb is a fresh nonce that is generated by B.

Assumption $A \ni A$ means that A holds its own identity. It reflects that a regular protocol participant freely uses its own identity. Identities of other principals are used only when the corresponding principals demand it. In other words, regular protocol participants do not masquerade as others. For the same reason, assumption that B holds its own identity is represented by $B \ni B$. △

Syvernson and Cervesato [114] proposed six forms of authentication goals for the SVO logic [115]. We categorise these forms of authentication goals into authentication goals and key-agreement goals in the BSW-ZF logic. Authentication goals in the BSW-ZF logic only consider the ping authentication goal "a principal P wants to know whether an interlocutor Q is alive", and the entity authentication goal "P's interlocutor Q said something relevant to their present conversation".

In the original paper that describes the BSW logic [28], it is noted that the formula $A \models (B \mid\!\sim Na)$ does not properly capture the entity authentication goal that A authenticates B. This is due to an attack based on the fact that B cannot judge whether Na is from A, and therefore, says Na to another principal I [28]. For example, we consider the following one-way authentication protocol.

$$\begin{aligned} &Message\ 1 \quad A \rightarrow B: \quad A, Na, \\ &Message\ 2 \quad B \rightarrow A: \quad \{Na\}_{K_b} \end{aligned}$$

A initiates a round of this protocol by sending its own identity A and its fresh nonce Na to B. A expects that B will sign this nonce by using B's signature key K_b, and send it back to A in Message 2. Because no one may hold B's signature key other than B, then A may

safely believe that B says Na recently. While this protocol achieves the ping authentication goal, it does not achieve the entity authentication goal. This is because B's response does not necessarily mean that B is responding to A's nonce challenge. In the following attack B considers the nonce challenge is initiated by another principal I, and B intends to respond the nonce challenge to I. Therefore, the formula $A \models (B \mid\!\sim Na)$ does not mean that A is authenticating B.

$$
\begin{array}{lll}
Message\ 1(\alpha) & A \rightarrow I(B): & A, Na, \\
Message\ 1(\beta) & I \rightarrow B: & I, Na, \\
Message\ 2(\beta) & B \rightarrow I: & \{Na\}_{K_b}, \\
Message\ 2(\alpha) & I(B) \rightarrow A: & \{Na\}_{K_b}
\end{array}
$$

It is proposed in [28] that the authentication goal should be expressed as $A \models (B \mid\!\sim (A, Na))$. This means that A believes that B says the nonce Na to A [28]. However, while this kind of goal correctly captures the fresh nonce challenge, the authentication goal of the following protocol (a variation of Yahalom protocol [27]), which relies on a fresh session key (channel) K_{ab}, cannot be expressed in this form.

$$
\begin{array}{lll}
Message\ 1 & A \rightarrow B: & A, Na, \\
Message\ 2 & B \rightarrow S: & B, \{A, Na, Nb\}_{K_{bs}}, \\
Message\ 3 & S \rightarrow A: & \{B, Na, Nb, K_{ab}\}_{K_{as}}, \{A, Nb, K_{ab}\}_{K_{bs}}, \\
Message\ 4 & A \rightarrow B: & \{Nb\}_{K_{ab}}, \{A, Nb, K_{ab}\}_{K_{bs}}, \\
Message\ 5 & B \rightarrow A: & \{A\}_{K_{ab}}.
\end{array}
$$

The latter components of $Message\ 3$ are amended to contain B's nonce Nb. When B receives this component, B may be sure that it is in the current round of this protocol. $Message\ 5$ extends this modified Yahalom protocol. When A receives $\{A\}_{K_{ab}}$, A is sure that B received the fresh session key K_{ab} and is talking to A.

The BSW-ZF formula $A \models (B \mid\!\sim A)$, that represents an authentication goal of the above variation of the Yahalom protocol, captures these two authentication goals. The formula represents the situation that A may distinguish whether the responder B is alive (ping authentication [114]) by whether B says something, and whether B is participating in their present conversation (entity authentication [114]) by B says (to) A.

The other various forms of authentication in the SVO logic [114] are considered as key-agreement goals in our BSW-ZF logic as follows. Note that the key k is represented by

channel C in the following BSW-ZF formulae.

Secure key establishment indicates "that a principal P believes that he has a good key k to communicate with a counterpart Q" [114]. This goal is formalised by the BSW-ZF formula:

$$P \models (\sigma(r(C)) = \sigma(w(C)) = \{P, Q\})$$

Key freshness requires that "principal P believes a key k (represented as abstract channel C) to be fresh" [114]. This goal is formalised by the BSW-ZF formula:

$$P \models \sharp(w(C))$$

Mutual understanding of shared keys "applies to situations where a principal P can establish that an interlocutor Q has sent a key k as an unconfirmed secret between the two of them (from Q's point of view)" [114]. This is formalised by the following formula:

$$P \models (Q \mid\sim (\sigma(r(C)) = \sigma(w(C)) = \{P, Q\}))$$

Key confirmation describes "scenarios in which principal P believes that an interlocutor Q has proved to have received and successfully processed a previously unconfirmed secret key k between the two of them" [114]. This is formalised by the following BSW-ZF formula:

$$P \models (\sigma(w(C)) = \{P, Q\} \wedge Q \mid\sim C(X))$$

Note that the logic inference rules are required to fully understand these goals.

5.2 Inference rules

Seeing rules

S1 If P receives a message X via a channel C, and P can read this channel, then P can see the message.

$$\frac{P \triangleleft C(X), P \ni r(C)}{P \triangleleft X}$$

S2 If P receives a compound message (X, Y), then P can also see parts of the message, (that is X and Y).

$$\frac{P \triangleleft (X, Y)}{P \triangleleft X, P \triangleleft Y}$$

Freshness rules

F1 If P believes that another principal Q once said a message X and P believes that X is fresh, then P believes that Q says X.

$$\frac{P \models (Q \mid\sim X), P \models \sharp(X)}{P \models (Q \mid\sim X)}$$

F2 If P believes that another principal Q says a compound message (X, Y), then P believes that Q also says parts of the message, (i.e. X and Y).

$$\frac{P \models Q \mid\sim (X, Y)}{P \models Q \mid\sim X, P \models Q \mid\sim Y}$$

F3 If X is a part of a compound message $(X,\ Y)$, and P believes that X is fresh, then P believes that the whole message $(X,\ Y)$ is fresh.

$$\frac{P \models \sharp(X)}{P \models \sharp(X, Y)}$$

F4 If P believes that the capability to write into channel C is fresh, then P also believes the capability to read from that channel is fresh, and vice versa.

$$\frac{P \models \sharp(w(C))}{P \models \sharp(r(C))} \qquad \text{and} \qquad \frac{P \models \sharp(r(C))}{P \models \sharp(w(C))}$$

In the BSW-ZF logic, fresh channels may represent fresh session keys. The **F4** rules reflect the fact that when one believes the encryption (decryption) key is fresh, then one must also believe that the corresponding decryption (encryption) key is fresh. This is typically true when using conventional cryptography. In this dissertation, we do not consider cryptography algorithms, such as the identity-based cryptography [106], that conflict with the above assumption.

Interpretation rules

I1 If P believes that he receives a message via C, and P believes that only Q and himself may send messages to C, then P believes that the message was said into channel C by Q, and he also believes that Q has the capability to write to channel C.

$$\frac{P \triangleleft C(X), P \ni r(C), P \models (\sigma(w(C)) = \{P, Q\})}{P \models (Q \mid\sim X), P \models (Q \mid\sim C(X)), P \models (Q \ni w(C))}$$

I2 If P believes that he receives a message via C, and P believes that only Q may send messages to C, then he believes that the message was said into channel C by Q, and he also believes that Q has the capability to write to channel C.

$$\frac{P \triangleleft C(X), P \ni r(C), P \models (\sigma(w(C)) = \{Q\})}{P \models (Q \mid\sim X), P \models (Q \mid\sim C(X)), P \models (Q \ni w(C))}$$

I3 If P believes that he received a message X via C, and that X is a secret shared between P and Q, then he believes that the message was said into channel C by Q, and he also believes that Q has the capability to write to channel C.

$$\frac{P \triangleleft C(X), P \ni r(C), P \models (\sigma(X) = \{P, Q\})}{P \models (Q \mid\sim X), P \models (Q \mid\sim C(X)), P \models (Q \ni w(C))}$$

Note that, in the BSW-ZF logic, we use the conventional assumption that a principal may recognize its own generated messages within a round of a protocol. Therefore, when P sees X from C, P may tell whether X was sent by itself or by Q.

Furthermore, note that each of the above rules **I1** − **I3** may be rewritten, each in turn, as three simpler rules, whereby, each has the same assumptions and only one conclusion formula. However, for the sake of simplicity and space, the rules are presented in a more compact form.

Rationality rules

R1 This is the well-known K axiom [70] of modal logic: if P believes ϕ_1 implies ϕ_2, and believes that ϕ_1 is true, then he believes that ϕ_2 is true.

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

Holding rules

H1 If P holds messages X and Y, then P holds the compound message (X, Y).

$$\frac{P \ni X, P \ni Y}{P \ni (X, Y)}$$

H2 If P generates message X, then P holds X.

$$\frac{P \mapsto X}{P \ni X}$$

H3 If P sees message X, then P holds X.

$$\frac{P \triangleleft X}{P \ni X}$$

Message Secrecy rules

M1 If P believes that X is a secret between P and Q, then he believes that the compound message (X, Y) is also a secret between P and Q.

$$\frac{P \models (\sigma(X) = \{P, Q\})}{P \models (\sigma(X, Y) = \{P, Q\})}$$

M2 If P believes that X is a secret within principal set Set_1 and the principal set Set_2 in which the principals may read from channel C, then P believes that the message X is a secret among Set_1 and Set_2. when X is sent to channel C. Trusted principals are omitted, because of the assumption that a trusted principal will neither use X as a proof of identity nor as a channel to communicate with.

$$\frac{P \models (\sigma(X) = Set_1), P \models (\sigma(r(C)) = Set_2), Q \triangleleft C(X)}{P \models (\sigma(X) = Set_1 \cup Set_2)}$$

5.3 Analysing Protocols

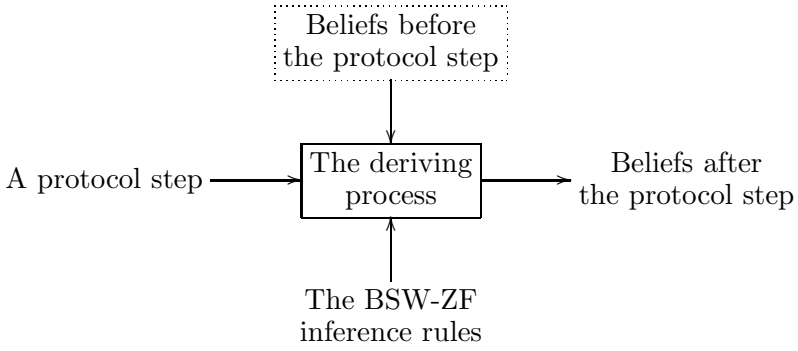


Figure 5.1: Belief Changing after a Protocol Step in Verification

Before a round of a security protocol, each principal holds a number of beliefs (assumptions) about nonces, channels, and their trust relationship. Each protocol step represents that a specified message is sent from one principal to another principal. After a protocol step, a number of new beliefs are derived by applying all of the BSW-ZF inference rules to their current beliefs and the current protocol message. At the same time, a number of previous beliefs are out of date. For example, a secret shared within a principal set before a

protocol step may be discovered by other principals in the current protocol step. Principals update their beliefs according to the derivation. The diagram in Figure 5.1 describes how beliefs are changed in a protocol step.

The protocol verification process verifies whether all of the protocol goals can be derived from the initial assumptions after a round of the protocol. If all of the protocol goals are derived after a round of the security protocol, then it is a valid protocol within the BSW-ZF logic. The diagram in Figure 5.2 describes the protocol verification process.

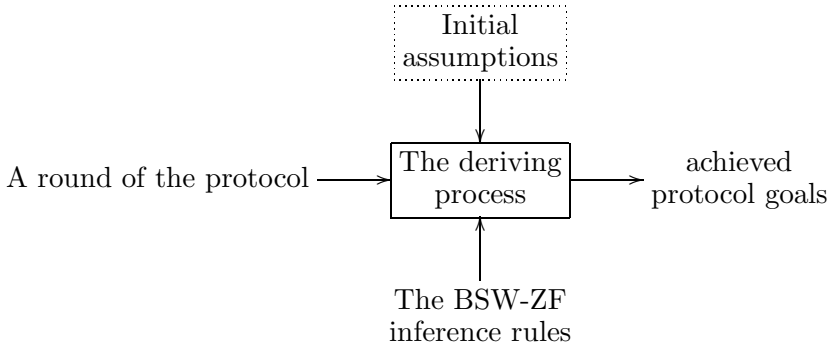


Figure 5.2: The Protocol Verification Diagram

Example 9 Given the assumptions in Example 8, the inference rules of the BSW-ZF logic is used to verify whether a mutual authentication protocol meets the authentication goals $A \models (B \mid\!\sim A)$ and $B \models (A \mid\!\sim B)$. The idealised protocol is described as following.

$$
\begin{aligned}
&B \triangleleft A, Na,\\
&S \triangleleft Cbs(A, Na, Nb),\\
&A \triangleleft Cas(B \mid\!\sim (A, Na, Nb)),\\
&B \triangleleft Cp(B \mid\!\sim Nb)
\end{aligned}
$$

For the purpose of easy reading, only necessary deductive results are listed in Figure 5.3 and Figure 5.4. In these figures, each line is composed by the line number, a BSW-ZF formula, and a commentary. The commentary interprets the current formula as a protocol step, an assumption, or a deduction. If the current formula is a deduction, then the commentary lists the line numbers of its premises and the applied inference rule. △

5.4 Discussion

Much of the notation of the BSW-ZF logic is taken from the BSW logic [28].

1	$A \triangleleft Cas(B \mid\sim (A, Na, Nb))$	protocol step
2	$A \ni r(Cas)$	assumption
3	$A \models (\sigma(w(Cas)) = \{A, S\})$	assumption
4	$A \models (S \mid\sim (B \mid\sim (A, Na, Nb)))$	1, 2, 3, using **I1**
5	$A \models \sharp(Na)$	Assumption
6	$A \models \sharp(A, Na, Nb)$	5, using **F3**
7	$A \models \sharp(B \mid\sim (A, Na, Nb))$	5, using **F3**
8	$A \models (S \mid\mid\sim (B \mid\sim (A, Na, Nb)))$	4, 7, using **I1**
9	$A \models (S \mid\mid\sim (B \mid\sim \phi_2) \rightarrow S \models (B \mid\sim \phi_2))$	Assumption
10	$A \models (S \models (B \mid\sim (A, Na, Nb)))$	8, 9, using **R1**
11	$A \models ((S \models (B \mid\sim \phi)) \rightarrow (B \mid\sim \phi))$	Assumption
12	$A \models (B \mid\sim (A, Na, Nb))$	10, 11, using **R1**
13	$A \models (B \mid\mid\sim (A, Na, Nb))$	6, 12, using **F1**
14	$A \models (B \mid\mid\sim A)$	13, using **F2**

Figure 5.3: Verifying $A \models (B \mid\mid\sim A)$ for Example 8

The BSW logic does not provide the notion of message holding "$\ni$", which we adopt from the GNY logic [56]. The set of principals that can receive and can send messages via a channel C is directly represented in the BSW logic by its reader set which we denote as $r_{BSW}(C)$ and its writer set which we denote as $w_{BSW}(C)$, respectively.

The BSW logic does not provide a notion for the representation of cryptographic keys as objects that may be held by principals or exchanged between principals. Key-agreement protocols require that principal P obtains a new key from the current round of protocol.

In the BSW logic, it means that an assumption $P \in w_{BSW}(C)$ is not held before a round of protocol, but it should be obtained after several protocol steps. However, while beliefs about keys, such as $P \models (w_{BSW}(C) = \{P, Q\})$, and $P \models (P \in w_{BSW}(C))$, may be exchanged among principals in the original BSW logic, principals may not derive new assumptions, such as $P \in w_{BSW}(C)$. This means that principals may not obtain new keys in the original BSW logic. As a result, while the original BSW logic can verify simple ping/entity authentication goals, it cannot be used to verify and synthesise the other key-establishment goals in Chapter 5.1. In the BSW-ZF logic, $w(C)$ and $r(C)$ represent encryption and decryption key objects, respectively. The corresponding principal sets are

1	$B \models\!\equiv (\sigma(Nb) = \{B\})$	assumption
2	$B \models\!\equiv (\sigma(B \mid\!\sim Nb) = \{A, B\})$	1, using **M1**
3	$S \triangleleft Cbs(A, Na, Nb)$	protocol step
4	$B \models\!\equiv (\sigma(r(Cbs)) = \{B, S\})$	assumption
5	$B \models\!\equiv (\sigma(B \mid\!\sim Nb) = \{B\})$	2–4, using **M2**
6	$A \triangleleft Cas(B \mid\!\sim (A, Na, Nb))$	protocol step
7	$B \models\!\equiv (\sigma(r(Cas)) = \{A, S\})$	assumption
8	$B \models\!\equiv (\sigma(B \mid\!\sim Nb) = \{A, B\})$	5–7, using **M2**
9	$B \triangleleft Cp(B \mid\!\sim Nb)$	protocol step
10	$B \ni r(Cp)$	assumption
11	$B \models\!\equiv (A \mid\!\sim (B \mid\!\sim Nb))$	8–10, using **I3**
12	$B \models\!\equiv\!\sharp(Nb)$	Assumption
13	$B \models\!\equiv\!\sharp(B \mid\!\sim Nb)$	12, using **F3**
14	$B \models\!\equiv (A \mid\!\!\sim (B \mid\!\sim Nb))$	11, 13, using **F1**
15	$B \models\!\equiv (A \mid\!\!\sim B)$	14, using **F2**

Figure 5.4: Verifying $B \models\!\equiv (A \mid\!\!\sim B)$ for Example 8

represented by $\sigma(r(C))$ and $\sigma(w(C))$ in the BSW-ZF logic. When $r(C)$ (or $w(C)$) is sent in a protocol message, the message receiver may hold $r(C)$ (or $w(C)$). Therefore, the BSW-ZF logic can be used to verify and synthesise key-agreement protocols.

The BSW logic cannot be used to reason about message secrecy. Message secrecy is dealt with by many belief logics such as GNY [56] and SVO [115] by providing axioms for reasoning about the secret contents of a message. In our logic, $\sigma(X)$ is used in axioms, such as **I3**, **M1** and **M2**, for reasoning about message secrecy. For example, the proof for the authentication goal $B \models\!\equiv (A \mid\!\!\sim B)$ in Example 2 is based on message secrecy. This verification cannot be done within the original BSW logic.

In the BAN logic [27], principals are treated as trustworthy, and in the GNY logic [56], fixed axioms are used for reasoning about the trustworthiness of a principal. In both of the BSW logic and the BSW-ZF logic, the rules about the trustworthiness of a principal are expressed as formulae as part of the assumptions of the protocol. This is illustrated in

Example 1 by the following formulae.

$$A \models ((S \mid\!\sim \phi_1) \rightarrow (A \models \phi_1))$$
$$A \models ((S \mid\!\sim (B \mid\!\sim \phi_2)) \rightarrow (B \mid\!\sim \phi_2))$$
$$B \models ((S \mid\!\sim \phi_1) \rightarrow (B \models \phi_1))$$
$$B \models ((S \mid\!\sim (A \mid\!\sim \phi_2)) \rightarrow (A \mid\!\sim \phi_2))$$

Rules **S1, S2, F1, F3, R1**, and the first part of **I1, I2** are based on the BSW logic. The axioms of the BSW-ZF logic are more compact, yet as expressive as the BSW logic.

The original BSW logic [28] has a formula $P \triangleleft X \mid C$ which defines that P sees message X via channel C. The inference rule $\mathbf{I1_{BSW}}$ in the BSW logic is defined as

$$\frac{P \models (w_{BSW}(C) = \mathcal{W})}{P \models ((P \triangleleft X \mid C) \rightarrow \bigvee_{\forall Q_i \in \mathcal{W} \setminus \{P\}} (Q_i \mid\!\sim X))}.$$

We refine it by observing the following. The rule $\mathbf{S1'_{BSW}}$

$$\frac{P \triangleleft C(X), P \in r_{BSW}(C)}{P \models (P \triangleleft X \mid C)}$$

is a part of $\mathbf{S1_{BSW}}$ in [28]. Since this is the only axiom in the logic that allows $P \models (P \triangleleft X \mid C)$ to be deduced, then apart from the rule $\mathbf{S1'_{BSW}}$, formula $P \models (P \triangleleft X \mid C)$ appears only in the interpretation rule $\mathbf{I1_{BSW}}$ of the original BSW logic. By replacing the formula $P \models (P \triangleleft X \mid C)$ by $P \triangleleft C(X)$ and $P \in r_{BSW}(C)$ in all inference rules, then the formula $P \models (P \triangleleft X \mid C)$ and the rule $\mathbf{S1'_{BSW}}$ can be safely removed from the logic without any loss of expressiveness.

The following rule is easily obtained by variable substitution from the rationality rule $\mathbf{R1_{BSW}}$.

$$\frac{P \models (P \triangleleft X \mid C), P \models ((P \triangleleft X \mid C) \rightarrow \bigvee_{\forall Q_i \in \mathcal{W} \setminus \{P\}} (Q_i \mid\!\sim X))}{P \models \bigvee_{\forall Q_i \in \mathcal{W} \setminus \{P\}} (Q_i \mid\!\sim X)}$$

By combining the three rules above, the modified interpretation rules **I1** and **I2** are obtained (when only P and Q, or only Q is in the principal set $\mathcal{W}$) and $\mathbf{S1'_{BSW}}$ can be safely removed from the logic.

5.5 The Heuristic Rules for Protocol Synthesis

To verify a security protocol, an idealised protocol and initial principal assumptions are given beforehand. The inference rules are used to guide the search from initial assumptions to further beliefs in the order of protocol execution. Belief logics verify whether expected

protocol goals are covered by initial assumptions and these further derived beliefs.

To synthesise (design) a security protocol, only protocol goals and initial principal assumptions are given beforehand. A number of well-ordered protocol steps need to be derived from given protocol goals and given assumptions. Figure 5.5 depicts the protocol synthesis process. Note that this diagram is effectively the 'reverse' of the diagram for protocol verification in section 5.2.

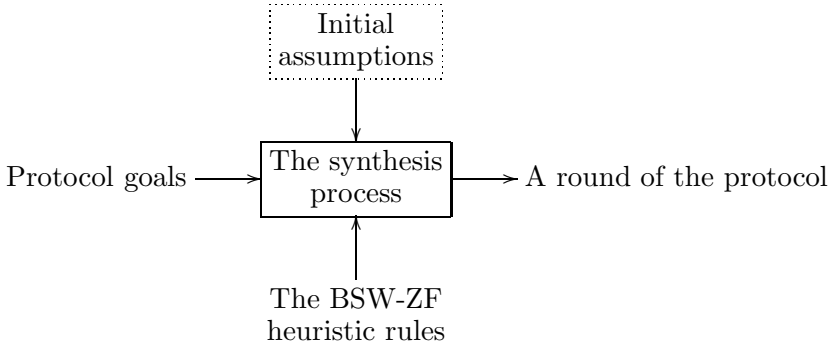


Figure 5.5: The Protocol Synthesis Diagram

While the inference rules of the BSW-ZF logic are used to verify security protocols, the BSW-ZF logic can also be used for synthesising security protocols. Since our protocol synthesis process is a reversed version of the protocol verification process, we effectively "reverse" the inference rules described in Section 5.2 to obtain heuristic rules for protocol synthesis. The validity of the heuristic rules are justified by the corresponding inference rules. To verify a protocol synthesised using these heuristic rules, the satisfied subgoals and protocol steps serve as the premises of the inference rules. It is sufficient to deduce protocol goals from these subgoals and protocol steps using the corresponding inference rules. Therefore, the heuristic rules can be used as synthesis rules to guide the *backward construction* for candidate subprotocols from their goals. In this case, the backward construction means that the protocol steps are constructed by the backward order of protocol execution. The reason for this is that the last required protocol step for a given protocol goal is naturally derived from applying heuristic rules to the protocol goal. The diagram in Figure 5.6 presents the synthesis process of a protocol step.

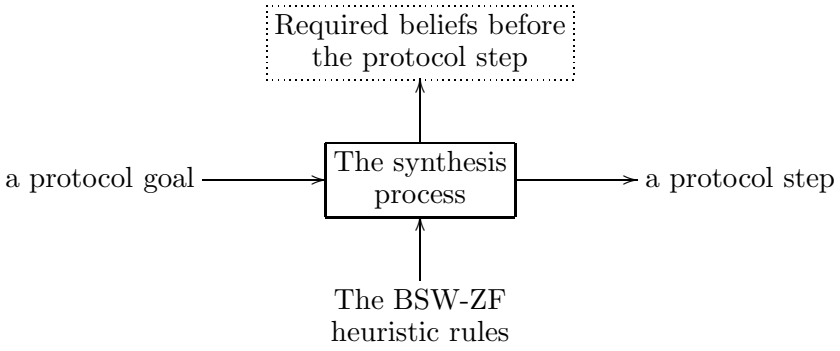
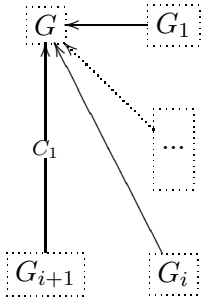


Figure 5.6: A Protocol Synthesis Step

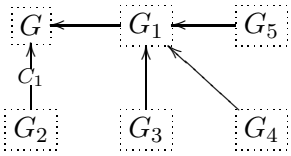
5.5.1 Notation

Our heuristic rules take the general form of a *rooted tree* of temporally ordered goals. A rooted tree is a directed acyclic graph (DAG) with a vertex that is singled out as the root.



This is interpreted to mean that in order to reach the goal G, all subgoals G_1, ..., G_i, have to be established without any conditions or in any particular temporal order. Subgoal G_{i+1} is an optional conditional subgoal under condition C_1, with the interpretation that if condition C_1 is satisfied, then G_{i+1} has to be established before the goal G can be established. Otherwise, G_{i+1} does not need to be considered. Arcs indicate the temporal relationship between a goal and its subgoals.

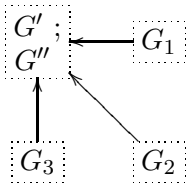
In some scenarios, subgoals have to be established in a particular temporal order. For example, before a principal receives a message via a channel, there must be another principal who first sent the message in that channel. Therefore, we use the following form to illustrate the ordered subgoals.



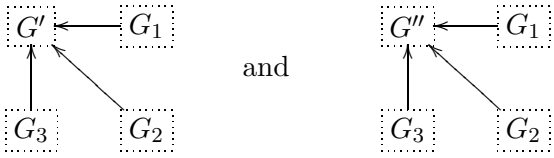
This is interpreted to mean that in order to establish the goal G, all subgoals G_1, ...,

G_5, have to be reached in a particular temporal order. Here, G_3, G_4, and G_5 must be established before G_1. However, G_3, G_4, and G_5 do not need to be reached in any temporal order relative to each other.

The goals are expressed in the heuristic rules using a composed form $G'; G''$, as follows,

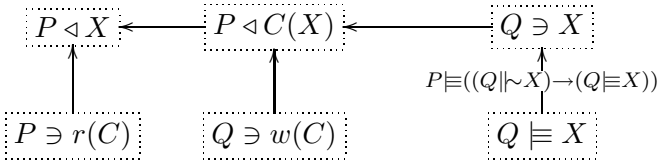


This is an abbreviated form of two heuristic rules and means that these rules have the same subgoals but different goals. They can be rewritten as



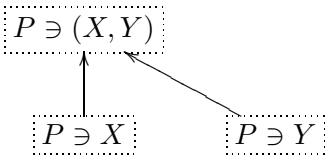
5.5.2 Heuristic Rules

Heur1 (derived from **S1**) To see message X, P must receive X on channel C and be able to read C. Before P receives X from C, some principal Q should hold X and may write in channel C. In addition, if the honesty assumption is presented, then Q is required to believe X before it can hold X.



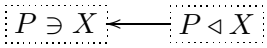
That $P \triangleleft X$ can be synthesised from $P \ni r(C)$ and $P \triangleleft C(X)$ is justified by Axiom **S1**. These subgoals are sufficient to verifying the goal for a given protocol. However, they are not sufficient to synthesise a practical protocol. The reason for this is that when there are no given protocols, the origin of the message $C(X)$ is undecidable. For the fact "P sees a message, only when there is a principal who sends the message" and fact "principals say only messages that they hold", the subgoals have to follow a particular temporal order, that is, to ensure that practical protocols are generated, we require that some principal Q holds X and that Q may write in C before P may receive X from channel C. This is similar in intent to the "required precondition" that is used in [10].

Heur2 (derived from **H1**) To hold a compound message (X, Y), P may hold the message components X and Y separately.

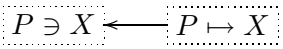


Note that this rule does not require that P must hold the message components X and Y separately. The reason for this is that P may obtain the compound message (X, Y) from the same message that he received. For example, when principal B sees a compound message (A, Na), it is not necessary for B to hold A and Na separately in order to hold the compound message.

Heur3 (derived from **H2**) To hold a message X, P may see X.

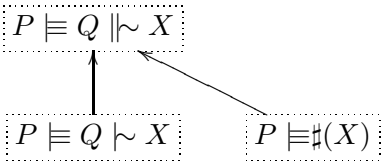


Heur4 (derived from **H3**) To hold a message X, P may generate X.



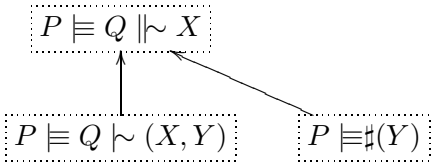
This rule is used to track the origin of all message components in a protocol on the basis that even if a principal holds all components of a message, he may not be willing to send out the message, unless he receives some request to do so or as the protocol initiator.

Heur5 (derived from **F1**) To believe Q says X, P may believe that Q said X and X is fresh.



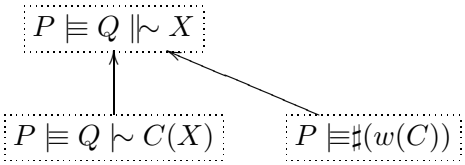
Sometimes a principal may believe another principal says something when it was once said with something fresh. This is reflected by **Heur6**.

Heur6 (derived from **F1** and **F2**) To believe Q says X, P may believe that Q said (X, Y) and Y is fresh.

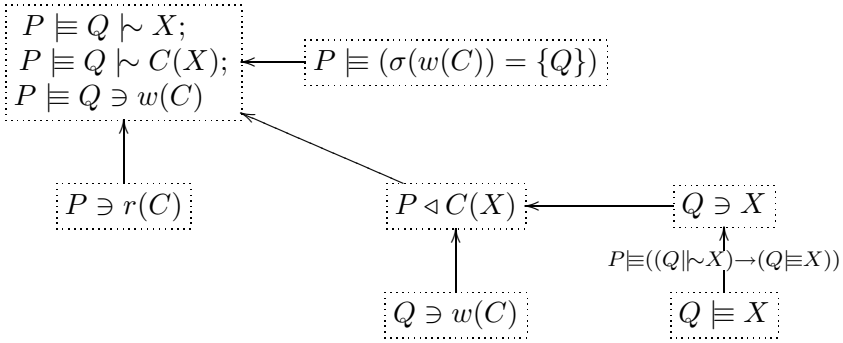


When the subgoal $P \models \sharp(Y)$ is reached, by **F2** $P \models \sharp(X, Y)$ is also reached. Thus, together with $P \models Q \mid\sim (X, Y)$, and by rules **F1** and **F4**, $P \models Q \mid\!\sim (X, Y)$ and $P \models Q \mid\!\sim X$ are reached.

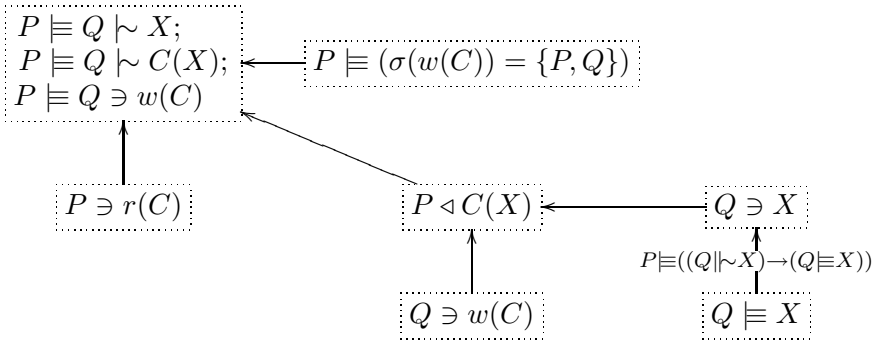
Heur7 (derived from **F1** and **F2**) To believe Q says X, P may believe that Q said X on C and $w(C)$ is fresh.



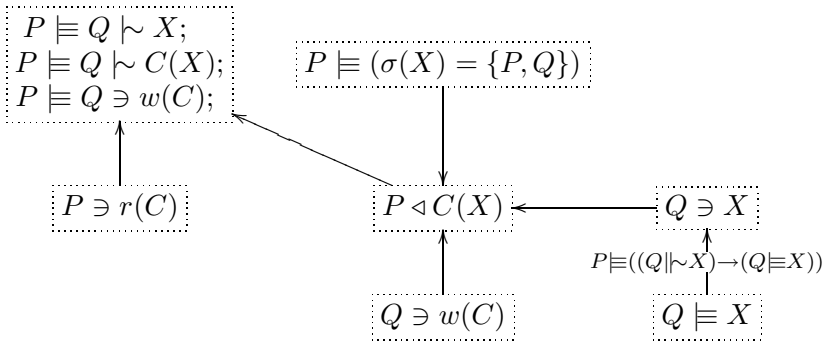
Heur8 (derived from **I2**) To believe that Q said X, Q said X on channel C, and Q holds $w(C)$, P has to receive X via a channel C that he can read and that he believes that it can be written to only by Q. Furthermore, Q must hold X and $w(C)$. If X is a formula and P believes that Q is honest, then Q must also believe X.



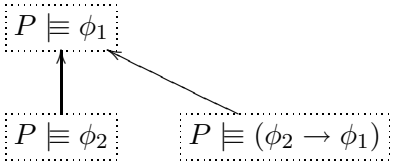
Heur9 (derived from **I1**) To believe that Q said X, Q said X on channel C, and Q holds $w(C)$, P has to receive X via a channel C that he can read and that he believes it can be written to only by P and Q. Furthermore, Q must hold X and $w(C)$. If X is a formula and P believes that Q is honest, then Q must also believe X.



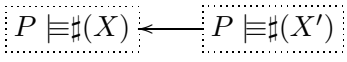
Heur10 (derived from **I3**) To believe that Q said X, P has to receive X via a channel C that he can read and that he believes X is a secret between Q and himself before he sees it. Furthermore, P must believe that Q is able to write to C, and that Q must believe that he sees X.



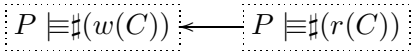
Heur11 (derived from **R1**) To believe ϕ_1, P must believe ϕ_2 and $\phi_2 \rightarrow \phi_1$.



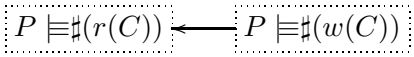
Heur12 (derived from **F3**) To believe a compound message X is fresh, P must believe some part X' of X is fresh.



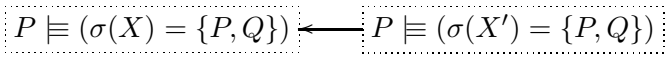
Heur13 (derived from **F4**) To believe $w(C)$ is fresh, P must believe $r(C)$ is fresh.



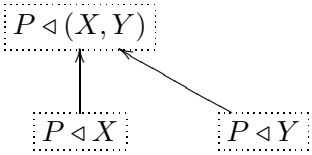
Heur14 (derived from **F4**) To believe $r(C)$ is fresh, P must believe $w(C)$ is fresh.



Heur15 (derived from **M1**) To believe a message X is a shared secret only between P and Q, P must believe some part of the message is a shared secret only between P and Q.



Heur16 (derived from **S2**) To see a compound message (X, Y), P may see the message components X and Y separately.



PROPOSITION **5.5.1 (Provable Synthesis)**

A protocol synthesised using the BSW-ZF heuristic rules can be proven to uphold its protocol goal using the BSW-ZF inference rules.

□

PROOF

The validity of the heuristic rules is justified by their corresponding inference rules, except for **Heur1, 8, 9, 10**. The heuristic rules **Heur1, 8, 9, 10** require additional subgoals to ensure valid temporal ordering of the subgoals of generated protocols, as discussed in **Heur1**.

These heuristic rules require that necessary subgoals are satisfied before a protocol goal is achieved. Any protocol synthesised using these heuristic rules guarantees that all of the subgoals for the protocol goals are satisfied. To verify a protocol, its subgoals serve as the premises of the inference rules. It is sufficient to deduce protocol goals from these subgoals using the corresponding inference rules.

5.5.3 Manual Synthesis Example

To help readers to understand the use of the heuristic rules, a one-way authentication protocol and a mutual authentication protocol are manually synthesised to demonstrate the

G_1 : $A \models (B \mid\!\sim A)$; G_2 : $B \models (A \mid\!\sim B)$.

Figure 5.7: Labeled Protocol goals used in Figure 5.14

A_1 : $A \models \sharp(Na)$; A_2 : $B \models \sharp(Nb)$;
A_3 : $A \mapsto Na$; A_4 : $B \mapsto Nb$;
A_5 : $A \ni A$; A_6 : $A \ni w(Cp)$;
A_7 : $A \ni r(Cas)$; A_8 : $B \ni r(Cp)$;
A_9 : $B \ni w(Cbs)$; A_{10} : $S \ni w(Cas)$;
A_{11} : $S \ni r(Cbs)$; A_{12} : $A \models (\sigma(w(Cas)) = \{A, S\})$;
A_{13} : $S \models (\sigma(w(Cbs)) = \{B, S\})$; A_{14} : $A \models ((S \mid\!\sim \phi_1) \rightarrow (S \models \phi_1))$;
A_{15} : $A \models ((S \models (B \mid\sim \phi_2)) \rightarrow (B \mid\sim \phi_2))$;

Figure 5.8: Labeled Assumptions used in Figure 5.14

IG_1 : $A \models (B \mid\sim (A, Na))$; IG_2 : $A \models (S \models (B \mid\sim (A, Na)))$;
IG_3 : $A \models (S \mid\!\sim (B \mid\sim (A, Na)))$; IG_4 : $A \models \sharp(B \mid\sim (A, Na))$;
IG_5 : $A \models (S \mid\sim (B \mid\sim (A, Na)))$; IG_6 : $B \models (A \mid\sim (B \mid\sim Nb))$;
IG_7 : $B \models (\sigma(B \mid\sim Nb) = \{A, B\})$; IG_8 : $A \ni (B \mid\sim Nb)$;
IG_9 : $A \triangleleft (B \mid\sim Nb)$; IG_{10} : $S \ni (B \mid\sim (A, Na, Nb))$;
IG_{11} : $S \models (B \mid\sim (A, Na, Nb))$; IG_{12} : $B \ni (A, Na, Nb)$;
IG_{13} : $B \ni (A, Na)$; IG_{14} : $B \ni Nb$;
IG_{15} : $B \triangleleft (A, Na)$; IG_{16} : $A \ni (A, Na)$;
IG_{17} : $A \ni Na$; IG_{18} : $B \models (\sigma(Nb) = \{A, B\})$;
IG_{19} : $S \models (B \mid\sim (A, Na, Nb))$; IG_{20} : $S \ni (B \mid\sim (A, Na))$;

Figure 5.9: Labeled Subgoals used in Figure 5.14

M_1 : $B \triangleleft Cp(A, Na)$; M_2 : $S \triangleleft Cbs(A, Na)$;
M_3 : $A \triangleleft Cas(B \mid\sim (A, Na))$; M_4 : $S \triangleleft Cbs(A, Na, Nb)$;
M_5 : $A \triangleleft Cas(B \mid\sim (A, Na, Nb))$; M_6 : $B \triangleleft Cp(B \mid\sim Nb)$.

Figure 5.10: Labeled Protocol Steps used in Figure 5.14

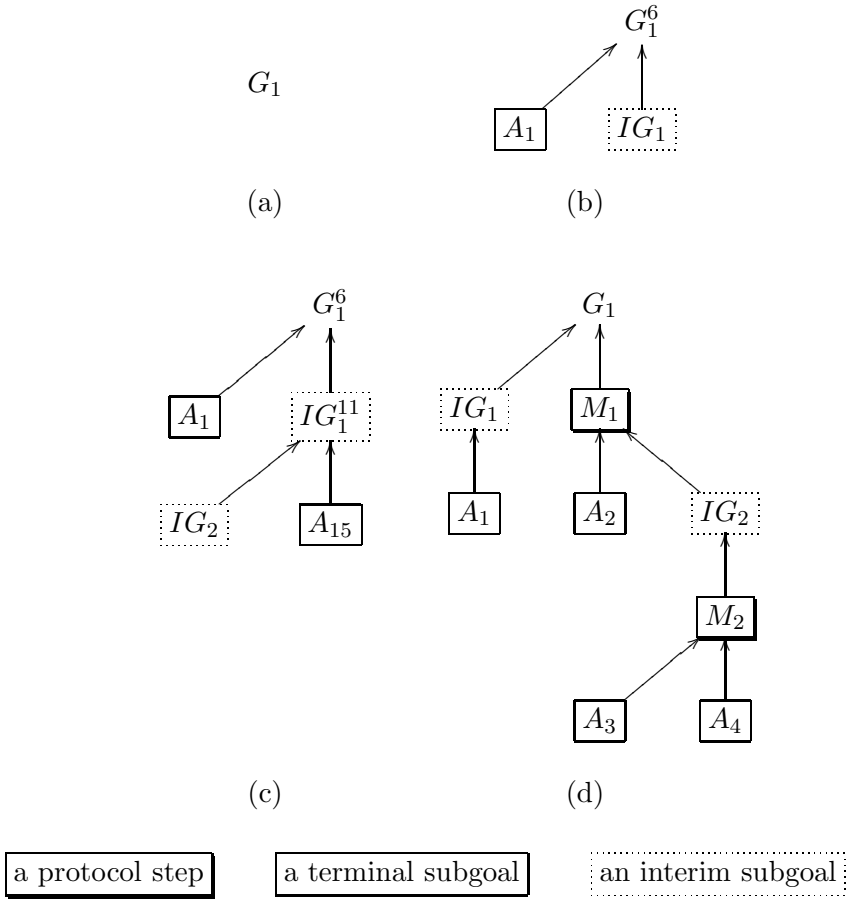


Figure 5.11: Incomplete and Complete Formula Trees

synthesis approach using the BSW-ZF logic. Both protocols are based on Example 8. The protocol goals for Example 8 are restated in Figure 5.7. While the one-way authentication protocol upholds goal G_1, the mutual authentication protocol upholds both G_1 and G_2. Assumptions that are described in Example 8 and used in the current manual synthesis examples are labeled and restated in Figure 5.8; further generated (required) subgoals, and protocol steps are labeled and listed in Figure 5.9, and Figure 5.10.

We first consider the synthesis of a one-way authentication protocol that upholds G_1. Goal G_1 which must be upheld by the synthesised protocol is considered as the single (root) node of a rooted tree. This single-node rooted tree is regarded as a *initial formula tree* for synthesis, such as Figure 5.11(a). Since the heuristic rules of the BSW-ZF logic are also defined in the form of rooted trees, the synthesis of a one-way authentication protocol can be illustrated by *tree grafting*, whereby instantiations of suitable heuristic rules are grafted into *incomplete formula trees* that are extended from the initial formula tree until a

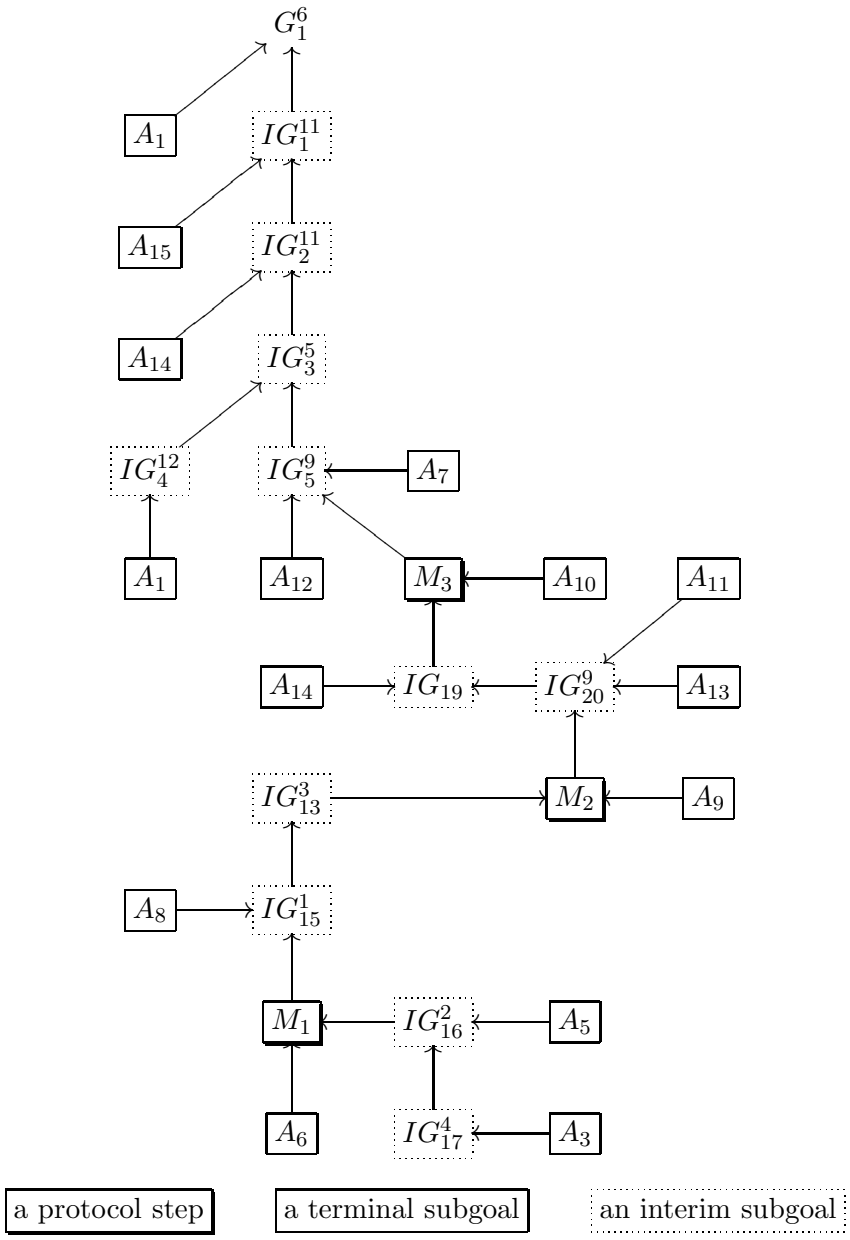


Figure 5.12: One-way authentication Protocol Synthesis for Goal $A \models (B \mid\!\sim A)$

complete formula tree is held. An incomplete formula tree has one or more leaves that are not assumptions from the requirement specification, such as (a), (b) and (c) in Figure 5.11. All the leaves of a complete formula tree correspond to assumptions from the requirement specification, such as Figure 5.11 (d) and Figure 5.12.

If a protocol goal is not an assumption from the requirement specification, then the corresponding initial formula tree is an incomplete formula tree. If a protocol goal is an assumption from the requirement specification, then the corresponding initial formula tree is a complete formula tree. The fact that an initial formula tree can be a complete formula tree, reflects the fact that complete formula trees may contain no protocol steps, since the corresponding protocol goals do not require the cooperation of principals.

By distinguishing complete formula trees from incomplete complete formula trees, the end-point of a synthesis process is determinable. More specifically, the synthesis for the root of the complete formula tree is finished, when a complete formula tree is generated. Figure 5.11(b) and (c) describe two grafting steps from initial formula trees. Figure 5.12 gives an entire example of a complete formula tree generated from the goal G_1.

Initially, the variables (P, Q, X and Y) in the Heuristic rule **Heur6** are substituted by A, B, A and Na in turn. As a result, the goal of **Heur6** is instantiated as the same as goal G_1 that is the root of initial formula tree. By grafting the whole instantiation of **Heur6** into the initial formula tree at the root node G_1, the formula tree in Figure 5.11(b) is obtained. The superscript of goal G_1 indicates the current applied heuristic rule **Heur6**. According to Figure 5.11(b), two new subgoals $A \models \sharp(Na)$ and $A \models (B \mid\sim (A, Na))$ are required to achieve G_1. Formula $A \models \sharp(Na)$ is an assumption A_1 in the protocol requirement specification and thus the search on this branch terminates. $A \models (B \mid\sim (A, Na))$ as an interim subgoal IG_1 requires further synthesis. When **Heur11** is applied to the formula tree in Figure 5.11(b) at node IG_1, the formula tree in Figure 5.11(c) is obtained. The heuristic rules for other subgoals are similarly applied, resulting in the formula tree in Figure 5.12. In order to provide an intuitive deduction process to readers, Figure 5.13 represents the same formula tree with the original formulae (unlike Figure 5.12, we use the formula labels in Figure 5.7 – 5.10).

A one-way authentication protocol is constructed from the 'sees' formulae of the form $P \triangleleft C(X)$ of a complete formula tree as follows.

$$
\begin{aligned}
Step_1(M_1): &\quad B \triangleleft Cp(A, Na);\\
Step_2(M_2): &\quad S \triangleleft Cbs(A, Na);\\
Step_3(M_3): &\quad A \triangleleft Cas(B \mid\sim (A, Na));
\end{aligned}
$$

The one-way authentication protocol, together with the assumptions from the tree, meets

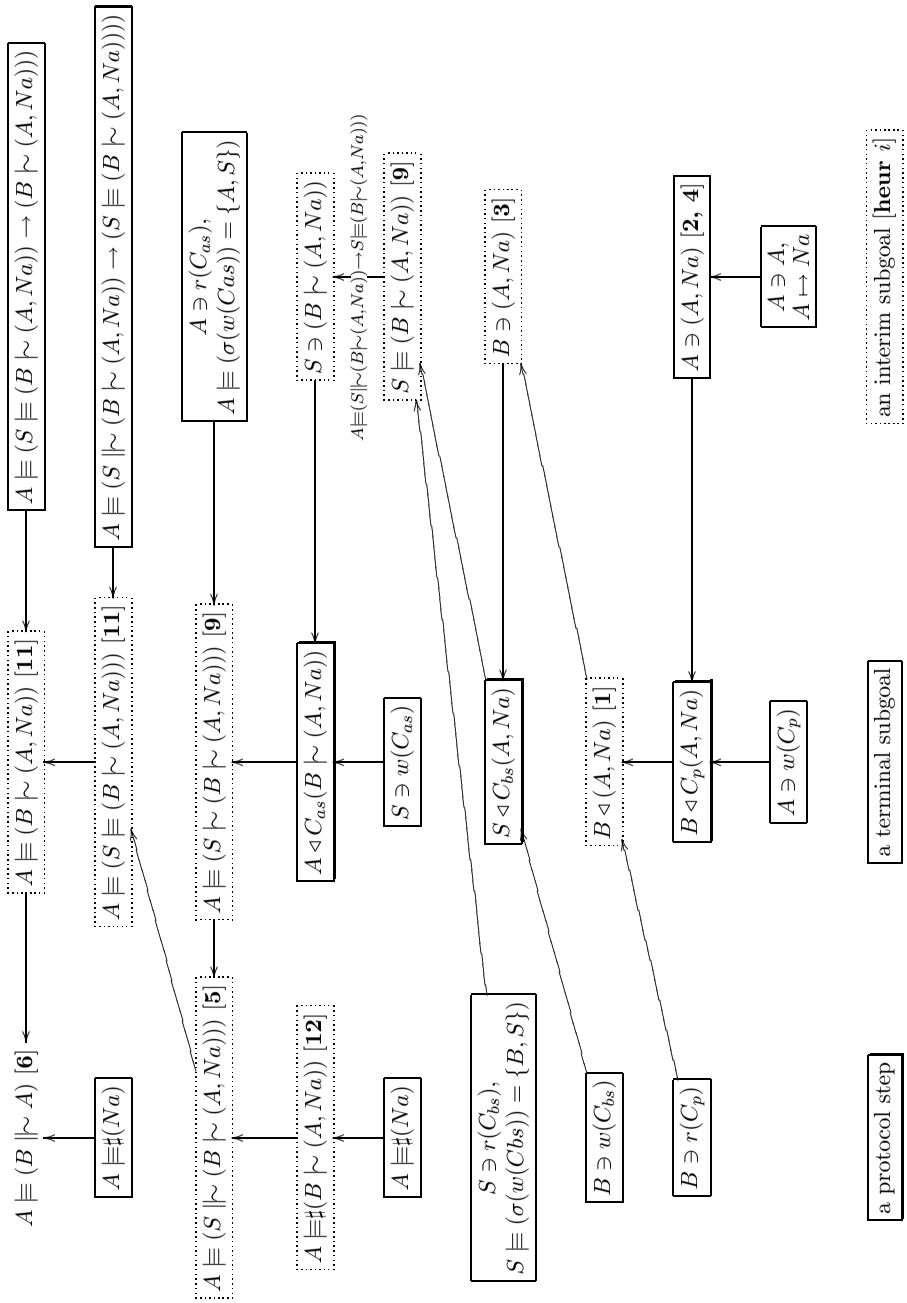


Figure 5.13: A Complete formula tree for Goal $G_1 : A \models (B \mid\!\sim A)$

the goal of the initial synthesis state within the BSW-ZF logic. Note that in using the heuristic rules of the BSW-ZF logic, the order of protocol steps is automatically generated in a backward order of protocol execution sequence. The temporal ordering of subgoals in the heuristics is important. When **Heur9** is applied to the subgoal $S \models (B \mid\sim (A, Na))$ in Figure 5.13, B must hold the message (A, Na) *before* it is seen by S, otherwise B cannot write X to channel C_{bs}. This guarantees the "executability" [63] of generated protocols, whereby, at any stage, each principal possesses enough information to construct the messages that it is supposed to send according to the protocol.

The synthesis for multiple goals can be considered as a composition of a number of formula trees. The roots of these formula trees are the goals from the protocol requirement. Certain subtrees are shared by different trees in the composition. This means that the composition is in the form of a directed acyclic graph (DAG). Therefore, the synthesis for multiple goals can be illustrated in the form of a directed acyclic graph (DAG). A systematic way to compose formula trees will be considered in the next chapter.

Figure 5.14 demonstrates an entire synthesis process for both G_1 and G_2. For example, to achieve goal G_1, assumption A_1 and subgoal IG_1 are required when **Heur6** is applied to G_1. Consequently, heuristic rules **Heur11, 11, 5, 12**, and **Heur9** are applied to corresponding subgoals until subgoal IG_{10} is required to achieve G_1. Then, we consider goal G_2. Heuristic rules **Heur6, 10, 3**, and **Heur1** are applied to corresponding goal and subgoals, then subgoal IG_{10} is also required to achieve goal G_2. Note that subgoal IG_{18} about message secrecy is not synthesised further. It is used to guard the entire further synthesis, that is the related secrecy may not be leaked out before the current protocol step. From Figure 5.14, we can see that only subgoal IG_{10} is required to achieve both G_1 and G_2 now. By repeatedly applying suitable heuristic rules to subgoals, the mutual authentication protocol described in Example 9 is generated. Note that the order of protocol steps M_1, M_4, M_5 and M_6 is automatically generated in a backward order of the execution sequence of the protocol in Example 9.

5.5.4 Discussion

The BSW Logic [28] also provides a synthesis technique that can be used to guide the (manual) systematic calculation of a protocol from its goals. The general form of the BSW

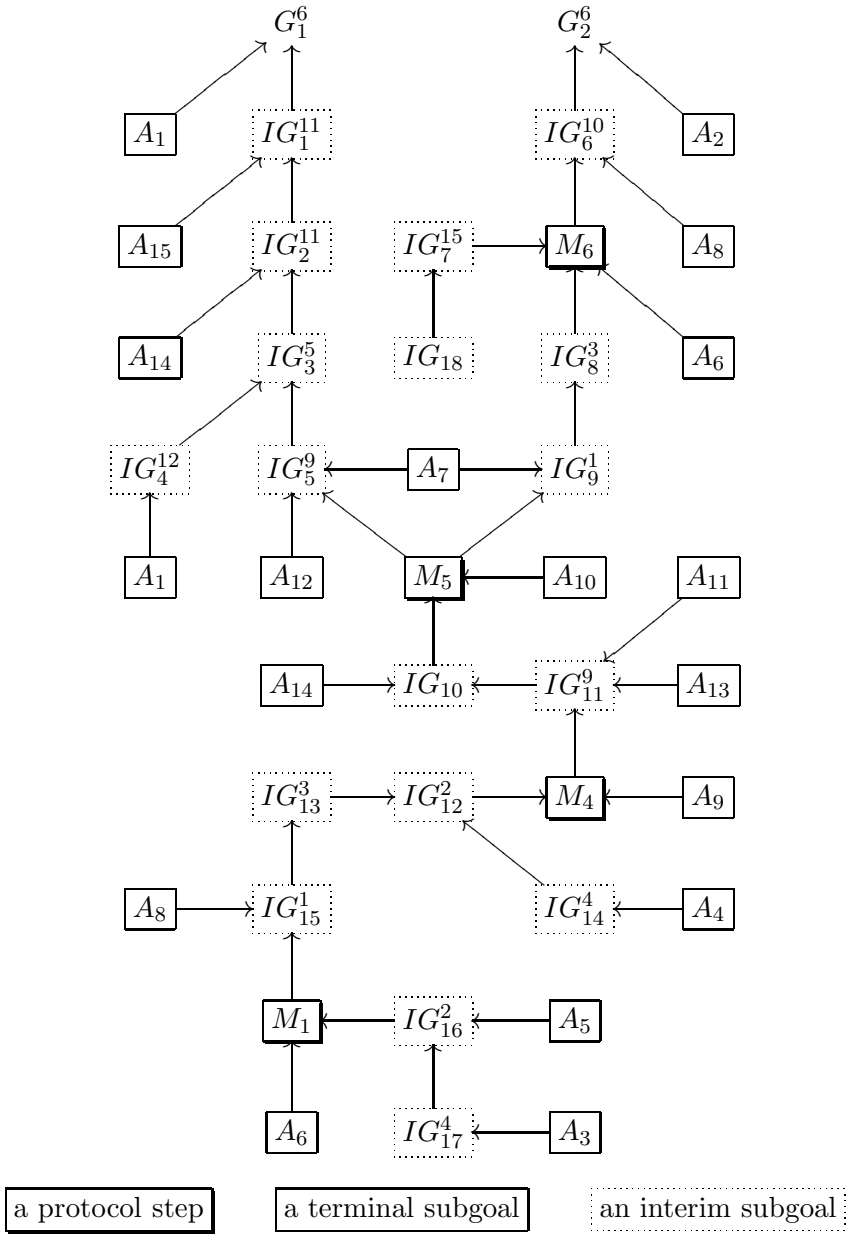
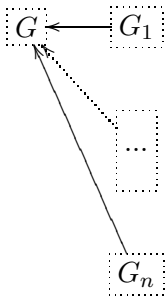


Figure 5.14: Manual Protocol Synthesis for Goals in Example 8

synthesis rules can be re-interpreted in our notation as:



This means that in order to reach the goal G, all subgoals G_1, ..., G_n have to be reached. Nine synthesis rules of this form are proposed in [28] by reversing the inference rules of the original BSW logic. This limitation of the original BSW logic implies that only certain classes of authentication protocols can be synthesised. For example, authentication protocols based on message secrecy and key agreement protocols may not be synthesised within the BSW logic. By extending the BSW logic to the richer BSW-ZF logic, the proposed heuristic rules can synthesis a wider range of authentication and key-exchange protocols.

The BSW synthesis rules do not consider the temporal order of subgoals. When two subgoals in a synthesis rule correspond to messages exchanged between principals, then the proper order of these messages is not specified within the BSW logic. The BSW synthesis rules rely on the user to manually order the synthesised protocol steps to work effectively and are therefore insufficient to automatically generate security protocols that require a number of ordered messages.

The BSW-ZF heuristics rules have temporal order built-in so that the ordering of protocol messages can be done automatically. With the temporal order, the heuristic rules **Heur1–4** and **Heur8–11** guarantee that if certain assumptions do not hold, then protocol messages may not be generated. As noted above, this guarantees the "executability" [63] of generated protocols.

In this chapter, we proposed and discussed the BSW-ZF logic, and also proposed a backward searching approach based on the heuristic rules of the BSW-ZF logic to manually generate protocols. In the next chapter, we will propose a novel automatic backward searching approach based on the heuristic rules of the BSW-ZF logic, as used by the Automatic Security Protocol Builder (ASPB), to automatically generate security protocols. Given a number of protocol goals and assumptions, the heuristic rules of the BSW-ZF logic are adapted to guide an automatic backward search for sub-protocols (message sequences) from each protocol goal, whereby, the desired messages are generated in a reversed order of the protocol execution sequence.

Chapter 6

Automatic Security Protocol Builder

In this chapter, we describe the Automatic Security Protocol Builder (ASPB), and discuss the advantages of our approach by comparing it with other approaches. Figure 6.1 outlines the architecture of the Automatic Security Protocol Builder. A protocol specification is parsed and decomposed into a series of single goal protocol requirements using the Specification Parser (Section 6.1). From these single goal protocol requirements, a collection of subprotocols are synthesised to satisfy the individual goals using the Single Goal Synthesiser (Section 6.2). The Protocol Composer (Section 6.3) merges these subprotocols to form complete candidate protocols. The Protocol Selector (Section 6.4) selects what is considered the most suitable protocols from the candidate protocols. Section 6.5 provides a series of examples and Section 6.6 compares it with the existing approaches.

6.1 The Requirement Specification and the Parser

When designing a protocol, the designer should understand the security context of the principals. It reflects the keys that principals know, the trust relationships between principals, and any other assumptions that they may hold. Our protocol requirement specification defines the terms that are used in the specification, initial known assumptions, and goals for the protocol to be designed. These are represented using the BSW-ZF logic.

Example 10 Figure 6.2 gives a complete requirement specification for Example 8. Note that *unreachable assumptions* may also be included in the requirement specification. These do not affect the logic, but are used to direct the synthesis during pruning, and will be explained in Section 6.2.1. △

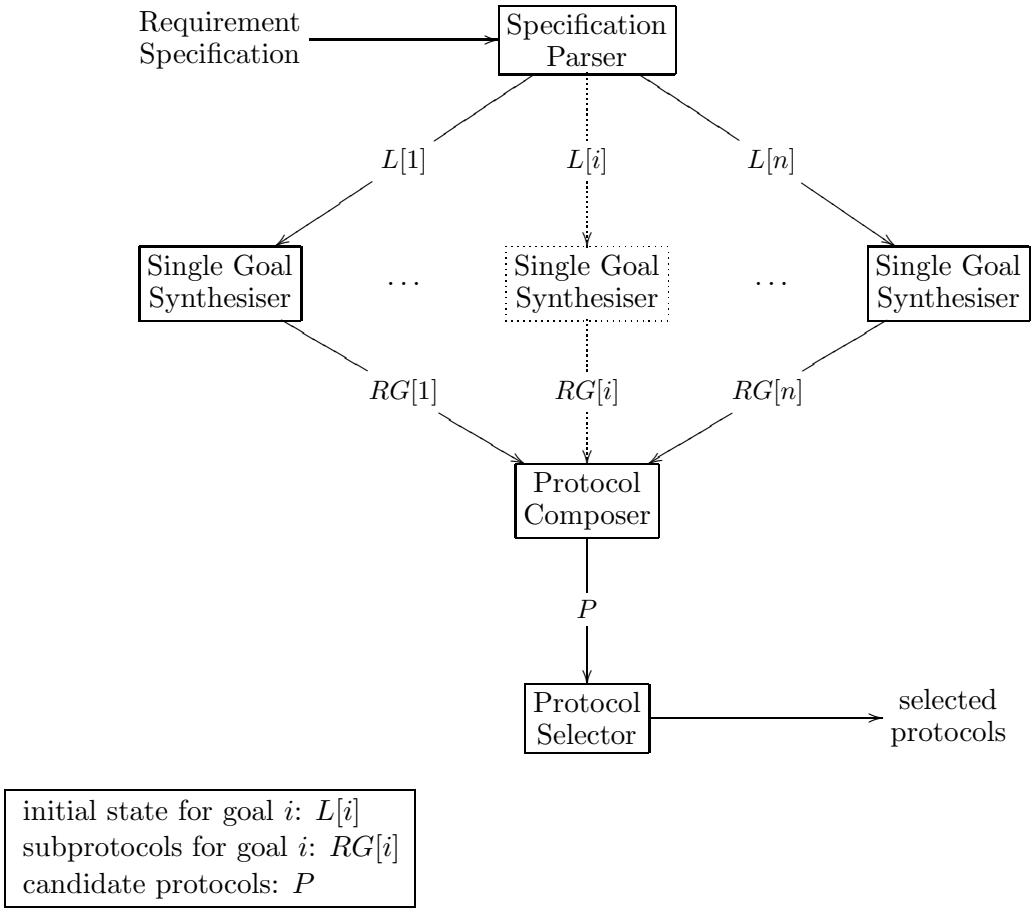


Figure 6.1: Overview of Automatic Security Protocol Builder

The Requirement Specification Parser parses the specification and checks whether all the formulae are valid based on the logic syntax and the types defined in the "declarations" section of the specification. For each goal to be proven, the parser generates a single-node formula tree (of that goal) as an initial synthesis state. The BSW-ZF heuristic rules define how this tree may be expanded, and a snapshot of this tree (rooted by that goal), represents a synthesis state. The single (root) node of an initial formula tree is the goal that any synthesised protocol must uphold. ASPB uses two kinds of formula trees: incomplete formula trees and complete formula trees, that have been defined in Section 5.5.3.

6.2 The Single Goal Synthesiser

The Single Goal Synthesiser accepts an initial synthesis state from the Requirement Specification Parser, automatically synthesises its goal using the heuristic rules of the BSW-ZF logic, and builds complete formula trees from the goal. Algorithm 1 describes this process.

Operation *choose*(L) picks an arbitrary synthesis state s from the set of current states L.

declarations {
Channel Cas, Cbs, Cp;
Principal A, B, S;
Nonce Na, Nb;
Message X;
Formula ϕ;
}
assumptions {
$A \models (\sigma(w(Cas)) = \{A, S\})$;
$S \models (\sigma(w(Cas)) = \{A, S\})$;
$B \models (\sigma(w(Cbs)) = \{B, S\})$;
$S \models (\sigma(w(Cbs)) = \{B, S\})$;
$A \models (\sigma(r(Cas)) = \{A, S\})$;
$S \models (\sigma(r(Cas)) = \{A, S\})$;
$B \models (\sigma(r(Cbs)) = \{B, S\})$;
$S \models (\sigma(r(Cbs)) = \{B, S\})$;
$A \ni r(Cas)$; $A \ni w(Cas)$;
$A \ni r(Cp)$; $A \ni w(Cp)$;
$B \ni r(Cbs)$; $B \ni w(Cbs)$;
$B \ni r(Cp)$; $B \ni w(Cp)$;
$S \ni r(Cbs)$; $S \ni w(Cbs)$;
$S \ni r(Cas)$; $S \ni w(Cas)$;
$S \ni r(Cp)$; $S \ni w(Cp)$;
$A \models \sharp(Na)$; $B \models \sharp(Nb)$;
$A \mapsto Na$; $B \mapsto Nb$;
$A \ni A$; $B \ni B$;
$A \models ((S \mid\sim \phi) \rightarrow (S \models \phi))$;
$B \models ((S \mid\sim \phi) \rightarrow (S \models \phi))$;
$A \models ((S \models (B \mid\sim X)) \rightarrow (B \mid\sim X))$;
$B \models ((S \models (A \mid\sim X)) \rightarrow (A \mid\sim X))$;
}
unreachable assumptions{
$A \ni r(Cbs)$; $A \ni w(Cbs)$; $B \ni r(Cas)$; $B \ni r(Cas)$;
}
goals {
$A \models (B \mid\sim A)$; /* G_1 */
$B \models (A \mid\sim B)$; /* G_2 */
}

Figure 6.2: The complete requirement specification for Example 8

Algorithm 1 StateSet syn(State *initState*)

```
State s;
StateSet S', R = φ;
StateSet L = {initState};
while ¬ empty(L) do
  s = choose(L);
  L = L \ s;
  if hasGoal(s) then
    S' = subsyn(s);
    L = add(L, S');
  else
    R = add(R, s);
  end if
end while
return R;
```

If the given state s is an incomplete formula tree, then operation $subsyn(s)$ selects a leaf node that is not an assumption, and applies the currently applicable heuristic rules to this leaf node. A number of new formula trees are generated by appending the subgoals of the applicable heuristic rules to the formula tree. Operation $subsyn(s)$ returns these new formula trees as a collection of synthesis states. For example, the different formula trees in Figures 5.11(b) and (c) could be generated and returned by $subsyn(s)$ for the formula tree in Figure 5.11(a).

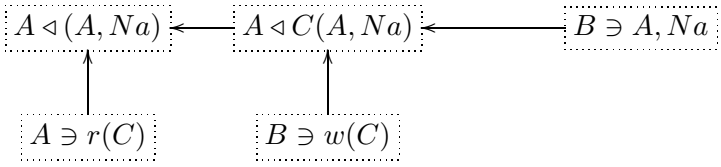
If no heuristic rule can be applied to a leaf node (terminal subgoal) then the Single Goal Synthesiser tests whether this leaf matches an assumption. If so, then it is considered reachable. If all leaves of a tree are reachable, then the tree is a complete formula tree, and represents a candidate protocol for the goal. It is added to the set of complete formula trees R for the given goal.

An automatic verification tool [92] for the original BSW Logic has been implemented using Theory Generation [68]. This tool [92] also supports (manually guided) synthesis of protocols using the synthesis rules described in [28]. The Single Goal Synthesiser described in this section builds on and extends this manual tool in [92] by automatically carrying out the synthesis process for the BSW-ZF logic.

6.2.1 Improving Performance

To improve the performance of the Single Goal Synthesiser, a number of ad-hoc strategies are used as part of the operation $subsyn$(s), including early pruning, variable instantiation, and tree pruning.

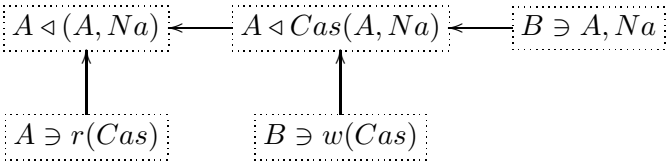
Early Pruning. Having applied a heuristic rule, consider generated formulae that are



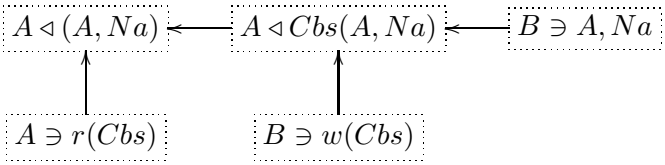
(a) Applied **Heur1** on $A \triangleleft (A, Na)$, and instantiated variable Q by B

declarations{
Channel Cas, Cbs, Cp;
...

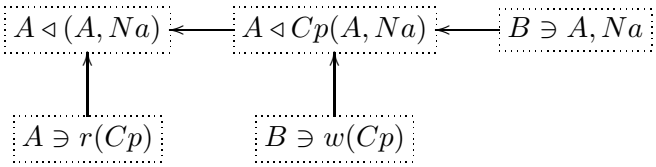
(b) Possible Instantiations for variable C in Figure 6.2



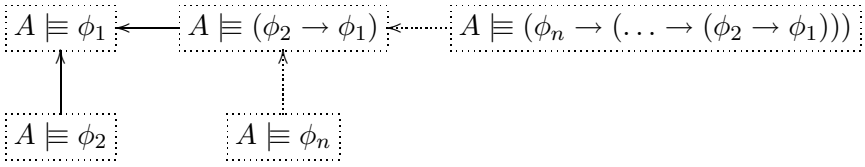
(c) Instantiated variable C by Cas



(d) Instantiated variable C by Cbs



(e) Instantiated variable C by Cp



(f) A Formula Tree for Recursively Applying **Heur11** on $A \models \phi_1$

Figure 6.3: Examples for Early Pruning and Variable Instantiation

of the form of $P \ni r(C)$, $P \ni w(C)$, or $P \models\!\sharp(Y)$. To speed up the judgement process, the operation $subsyn(s)$ checks whether leaves of this form in generated formula trees match the initial assumptions or specify unreachable assumptions of the requirement specification. If a leaf matches an assumption, then the leaf is reachable and the search terminates at this point. For example, $A \ni r(Cp)$ and $B \ni w(Cp)$in Figure 6.3(e) match specified assumptions, then the search terminates at this point. Only $B \ni (A, Na)$ remains for further synthesis. If any leaf matches a specified unreachable assumption, then the corresponding formula tree is also unreachable, and the formula tree is simply discarded. For example, if $B \ni w(Cas)$ in Figure 6.3(c) or $A \ni r(Cbs)$ in Figure 6.3(d) match specified unreachable assumptions, then the corresponding formula trees are simply discarded. Otherwise, the formula tree is kept for further synthesis. The rationale for early pruning is that formulae of this form are frequently generated using the heuristic rules, such as **Heur1, 5 − 10, and 12 − 14**, and it is easy to immediately determine whether they are reachable by this approach.

Variable Instantiation. Some heuristic rules introduce new variables that do not appear in the goal, but appear in the subgoals. For example, **Heur1**, and **Heur6 - 10**. If the type of the variable is known (for example, variable Q is a principal, and variable C is a channel in **Heur1**), then the heuristic rule is repeatedly applied for all possible instantiations of the variable from the "declarations" section of the requirement specification. As a result of this, a number of independent states that have no variables are generated. For example, Figure 6.3(a) introduces channel variable C. By applying all possible instantiations of variable C from Figure 6.3(b), independent states Figure 6.3(c), (d), and (e) that have no variables are generated. Operation $subsyn(s)$ then applies the 'early pruning' strategy to these instantiated states.

Some heuristic rules introduce variables of unknown type. For example, in theory, variable ϕ_2 in **Heur11** could be bound to any logical formula, resulting in an infinite state space. In practice, we consider that in order to reach **Heur11**'s goal, both of its two subgoals must be reached beforehand. In order to determine whether **Heur11** is applicable to $P \models \phi_1$, where ϕ_1 has no unbound variables, a search is made for a pattern $P \models (\phi_2 \rightarrow \phi_1)$ for some arbitrary ϕ_2 in the assumptions of the requirement specification. If matched, the heuristic rule is applied. If no match occurs, then a search is done to test if it is possible to synthesise a formula of the form $P \models (\phi_2 \rightarrow \phi_1)$, from existing assumptions in the requirement specification. In practice, this search is relatively straightforward as only **Heur11** can be applied to $P \models (\phi_2 \rightarrow \phi_1)$, whereupon it recursively searches for formulae of the form $P \models (...(... \rightarrow \phi_1))$ in assumptions, such as the formula tree in Figure 6.3(f).

To further minimise the search space, **Heur11** disregards higher order belief formulae [27] of the form $P \models (Q \mid\!\sim (Q \mid\!\sim X))$ and $P \models (Q \models (Q \models X))$, etc., on the basis that they do not provide any additional information than the first order beliefs $P \models (Q \mid\!\sim X)$ and $P \models (Q \models X)$, etc.. With these strategies, we restrict the state search space, and generate a wide range of useful protocols.

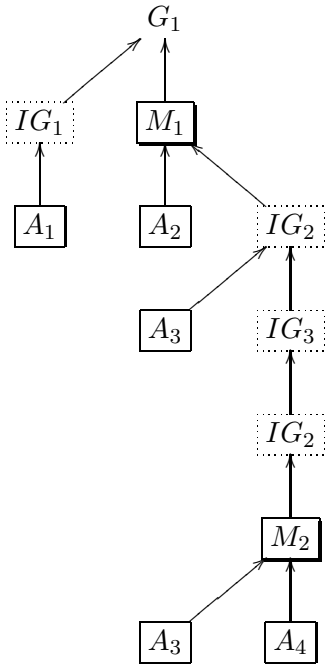
Tree Pruning. The Single Goal Synthesiser does not allow any formula as its own direct or indirect parent in a formula tree. The reason for this is that the Single Goal Synthesiser terminates the extension for any branch of a formula tree as early as possible. If a goal can be achieved, then the simplest subtree does not involve itself as a subgoal. Otherwise, it involves an infinite search for this goal, since any branch for this goal in a formula tree requires itself as a subgoal, such as the formula tree in Figure 6.4(c). For a goal to have itself as its own direct or indirect parent in a formula tree, the high level subtree for this goal can be safely replaced by the low level subtree for the same goal. The replacing formula tree is already in the state set, because the Single Goal Synthesiser extends a formula to all possible formula trees without any formula as its own direct or indirect parent. For example, formula trees in Figure 6.4(a), (b) can be simplified to the formula tree in Figure 6.4(d).

Example 11 Two subprotocols are automatically synthesised from Goal G_1 in the requirement specification of Figure 6.2. Subprotocol 1.1 corresponds to the protocol messages from the search tree in Figure 5.13. Further synthesis generates an additional search tree with corresponding Subprotocol 1.2.

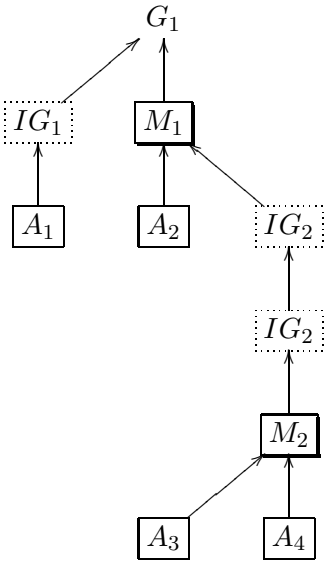
Subprotocol 1.1	**Subprotocol 1.2**
A,	A,
$B \triangleleft C_p(A, Na)$,	$S \triangleleft C_{as}(A, Na)$,
$S \triangleleft C_{bs}(A, Na)$,	$B \triangleleft C_{bs}(A, Na)$,
$A \triangleleft C_{as}(B \mid\!\sim (A, Na))$.	$A \triangleleft C_p(A, Na)$.

Note that the first line of a subprotocol indicates the subprotocol initiating principal (initiator) that generates the first message of the subprotocol. The synthesis of the symmetrically similar goal G_2 generates two similar search trees from which symmetrically similar subprotocols 2.1 and 2.2 are obtained.

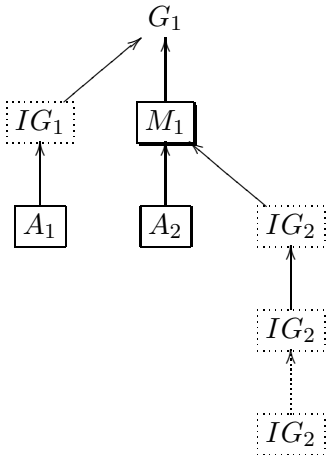
Subprotocol 2.1	**Subprotocol 2.2**
B,	B,
$A \triangleleft C_p(B, Nb)$,	$S \triangleleft C_{bs}(B, Nb)$,
$S \triangleleft C_{as}(B, Nb)$,	$A \triangleleft C_{as}(B, Nb)$,
$B \triangleleft C_{bs}(A \mid\!\sim (B, Nb))$.	$B \triangleleft C_p(B, Nb)$.



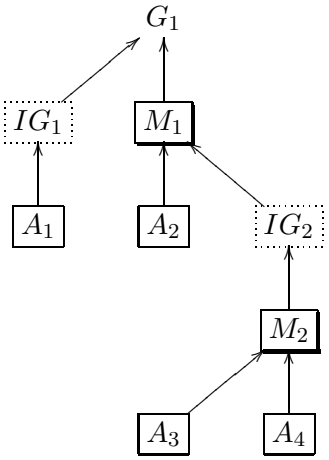
(a) Subgoal IG_2 as Its Own Indirect Parent



(b) Subgoal IG_2 as Its Own Direct Parent



(c) Infinite Search for Subgoal IG_2



(d) No Subgoal as Its Own Parent

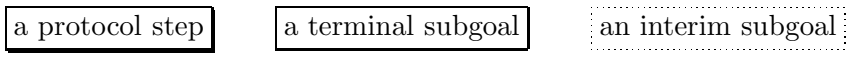


Figure 6.4: Self-Parent Formula Trees and Infinite Search

△

6.3 The Protocol Composer

The Single Goal Synthesiser is used to generate subprotocols from a single goal. While not considered in the original paper [28], it is possible, in theory, to use the original BSW synthesis rules to synthesise multiple goals. In the same way, the heuristic rules of the BSW-ZF logic could be used to synthesis from multiple goals. For example, a mutual authentication protocol is synthesised in Section 5.5.3. However, in practice, building a search tree for multiple goals results in a potential state explosion as each step must consider the application of all possible combinations of heuristic rules that could be applied in the current state. ASPB avoids this problem by first synthesising only single goal protocols and it then uses the Protocol Composer to, in turn, merge these single goal protocols into a single candidate protocol that meets the composition of goals.

A feasible candidate protocol for a requirement specification must achieve all of the goals described in the requirement specification, otherwise the candidate protocol does not satisfy the requirement specification (within the logic). For this reason, the Protocol Composer generates a feasible candidate protocol by composing a number of subprotocols. Each of these subprotocols is selected from the generated subprotocol set for each goal in the requirement specification, and the composition of these subprotocols satisfies all the goals. There can be many possible ways to merge subprotocols. The easiest way to merge two subprotocols is to append one to the end of the other. However, this may lead to lengthy and inefficient protocols. In addition, such protocols are certainly not fail-stop [57], since an attacker may run part of the protocol (achieving one goal) without completing the other subprotocol.

6.3.1 Merging Principal sequences

A security protocol is a sequence of messages exchanged between principals in order to achieve a number of security goals. At an abstract level, these message exchanges can be described just in terms of a *principal sequence*. A principal sequence is a sequence of principal identities based on the order of message exchanges between the principals in a protocol. For example, Subprotocol 1.1 implementing goal G_1 (Example 11) has principal sequence $A \rightarrow B \rightarrow S \rightarrow A$; Subprotocol 2.2 implementing goal G_2 has principal sequence $B \rightarrow S \rightarrow A \rightarrow B$.

Given principal sequences X and Y, then we say that sequence X *covers* sequence Y if Y appears as a fragmented subsequence of X. For example, Figure 6.5(a) illustrates

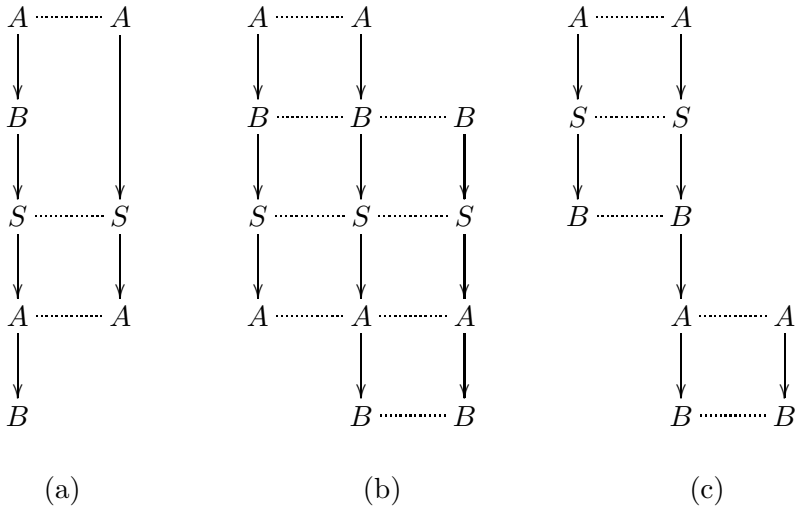


Figure 6.5: Sequence Covering Examples

that $A \rightarrow B \rightarrow S \rightarrow A \rightarrow B$ covers $A \rightarrow S \rightarrow A$. The Protocol Composer uses principal sequences to guide the construction of new protocols when merging subprotocol messages.

Given a collection of subprotocols $\mathrm{P}_1 \ldots \mathrm{P}_n$ that meet goals $G_1 \ldots G_n$ of a protocol specification, respectively, the Protocol Composer tests whether a given principal sequence covers the principal sequences of the subprotocols $\mathrm{P}_1 \ldots \mathrm{P}_n$. This is done using a variation of the shortest subsequence algorithm [69]. If the given principal sequence covers the principal sequences of the subprotocols $\mathrm{P}_1 \ldots \mathrm{P}_n$, then it is a possible principal sequence of a candidate protocol. For example, in Figure 6.5(b), principal sequence $A \rightarrow B \rightarrow S \rightarrow A \rightarrow B$ covers both principal sequence $A \rightarrow B \rightarrow S \rightarrow A$ for Subprotocol 1.1 and principal sequence $B \rightarrow S \rightarrow A \rightarrow B$ for Subprotocol 2.2, then principal sequence $A \rightarrow B \rightarrow S \rightarrow A \rightarrow B$ is a possible principal sequence of a candidate protocol that satisfies both G_1 and G_2. The candidate protocol is generated by merging the messages of the individual subprotocols according to the generated (covering) principal sequence. The behaviour of the Protocol Composer is specified by Algorithm 2.

The Protocol Composer accepts two parameters: the protocol initiator *prin* and an array *RG*. Each element $RG[i]$ of *RG* is the state set for the ith goal of the given requirement specification. The state sets are generated by the Single Goal Synthesiser and correspond to the possible subprotocols for a given goal. The Protocol Composer generates candidate protocols based on the rule of shorter first, so that principal sequences (protocol patterns) are generated from the shortest to the longest.

Operation $sPattern(RG)$ returns the shortest length of all the possible candidate protocols in *RG*. If a candidate protocol is composed of a number of subprotocols, then the principal sequencing of the candidate protocol must cover all the principal sequences of these

Algorithm 2 ProtocolSet compose(Principal *prin*, Array *RG*)

```
Array RS = φ;
ProtocolSet P', P = φ;
PatternSet T; Pattern t;
for i = sPattern(RG) to lPattern(RG) do
  T = genPatterns(prin, i);
  while ¬ empty(T) do
    t = choose(T);
    T = T \ t;
    RS = match(t, RG);
    P' = subCompose(t, RS);
    P = add(P, P');
  end while
end for
return P;
```

subprotocols. Thus, the candidate protocol is not shorter than the longest of these subprotocols. Note that the shortest pattern only guarantees that possible candidate protocols are not shorter than the shortest pattern. It does not means that a candidate protocol as the length of given shortest pattern can be generated. The reason for this is that the principal sequence of a candidate protocol is a common principal sequence that covers the principal sequences of its composing subprotocols. If the principal sequence of any subprotocol may not cover all other subprotocols, then a common principal sequence should be longer than all of its composing subsequences. For example, the length of all subprotocol principal sequences in Example 11 is 4, then the principal sequence length of a candidate protocol may not be shorter than 4. Otherwise, the principal sequence of candidate protocols may not cover any subprotocol. It follows that no protocol can be composed by given principal sequence length and subprotocols. On the other hand, the length of the shortest principal sequence that may cover subprotocols for all goals is 5. Figure 6.5(b) presents a possible covering example. This principal sequence length is longer than the length returned by operation $sPattern(RG)$.

Operation $lPattern(RG)$ returns the longest possible pattern length of all the possible candidate protocols in RG. Given n goals, where l_i is the length of the longest subprotocol for the ith goal, then the longest pattern length is $\sum_{i=0}^{n}(l_i - 1) + 1$. The motivation here is to generate a category of protocols such that the message receiver of a protocol step is the message sender of the next protocol step. Since the first line of a subprotocol only indicates the subprotocol initiator that generates the first message of the subprotocol, when merging subprotocols into a candidate protocol, all of the subprotocol initiators should be a receiver of the previous protocol step, except for one initiator that is the candidate protocol initiator. The length of the longest pattern of a candidate protocol should be the sum

of the number of steps of all the subprotocols plus the candidate protocol initiator. Note that the number of steps in a subprotocol is the length of the associated principal sequence minus one (which is used to indicate the subprotocol initiator). For example, principal sequence of the given candidate protocol in Figure 6.5(c) is $A \rightarrow S \rightarrow B \rightarrow A \rightarrow B$. It covers its subprotocol principal sequences $A \rightarrow S \rightarrow B$ and $A \rightarrow B$. If a candidate protocol is composed as Figure 6.5(c) by the subprotocols corresponding to the given subprotocol principal sequences. No message is sent at the step $B \rightarrow A$ of the generated candidate protocol. Therefore the last step $A \rightarrow B$ of the generated candidate protocol can be executed without executing the previous protocol steps. In this case, an intruder can generate and send a message to B in this protocol step without participating in previous protocol steps of the same round.

Operation $genPatterns(prin, i)$ generates all of the principal sequences, that are initiated by principal $prin$ and with length i, as possible protocol patterns.

Operation $match(t, RG)$ returns an array RS such that each element $RS[i]$ is a subset of the corresponding state set $RG[i]$ that is covered by the principal sequences of all $RS[i]$'s states that are, in turn, covered by the given principal sequence t. This is done by using the longest common subsequence algorithm [69].

Operation $subCompose(t,RS)$ returns all possible candidate protocols that follow principal sequence t, composed from the subprotocols of array RS. A candidate protocol is generated by composing combinations of subprotocols from $R[i]$ ($i \in [0, \ldots, n]$).

If a goal G_1 is a subgoal of another goal G_2 in the same requirement specification then G_1 is considered *relevant* to G_2. In this case, it is not necessary to further synthesise G_1 as doing so will simply generate subprotocols that will also be generated as parts of the subprotocols for G_2.

6.3.2 Merging Subprotocols

The Protocol Composer merges subprotocols according to the following rules.

Merg1 (Early appearing rule) A message from a subprotocol $\mathbb{P}_i$ should appear in the candidate protocol $\mathbb{P}$ as early as possible, constrained only by the principal sequencings.

Merg2 By the inference rule **S2** of the BSW-ZF logic

$$\frac{P \triangleleft (X, Y)}{P \triangleleft X, P \triangleleft Y}$$

we get the following rule

$$\frac{P \triangleleft X, P \triangleleft Y}{P \triangleleft (X, Y)}$$

which means that the messages, that are received by P from the different subprotocols, may be composed into one message in the final protocol.

Merg3 Messages on common channels in the subprotocols should be merged in the candidate protocol, subject to the constraints of the principal sequences. For example, message $C_1(X_1), C_2(X_2, Y_1)$ and message $C_1(X_3), C_2(X_2, Y_2)$ from two subprotocols merge into a resulting message $C_1(X_1, X_3), C_2(X_2, X_2, Y_1, Y_2)$.

Merg4 (Reducing rule) Any redundant message components should be reduced. For example, message $C(X_2, X_2)$ should be reduced to $C(X_2)$.

Example 12 The synthesis of goals G_1 and G_2 generate subprotocols 1.1 and 1.2 and subprotocols 2.1 and 2.2, respectively (Example 11). Thus there are 2×2 possible combinations of the subprotocols to be considered for merging. Furthermore, for each pair of subprotocols, we must find the shortest merge of the two subprotocols. ASPB generates the following 'best' protocol that corresponds to the merge of subprotocols 1.1 and 2.2 from Example 11 (in 3.1 seconds):

$$
\begin{array}{lcl}
A & , & \\
B & \triangleleft & C_p(A, Na), \\
S & \triangleleft & C_{bs}(A, B, Na, Nb), \\
A & \triangleleft & C_{as}(B, Na, Nb, A), \\
B & \triangleleft & C_p(Nb, B).
\end{array}
$$

△

6.3.3 Realising Idealised protocols

The Single Goal Synthesiser generates messages expressed as formulae within the BSW-ZF logic. The final implementation of these protocols is given in terms of conventional protocol message steps. For example, both message $Cas(A \mid\sim Na)$ and $Cas(A \triangleleft Na)$ are expressed by the same notation $Cas(A, Na)$. To minimise the potential for replay attacks where 'similar' messages appear in different parts of a protocol, the messages are modified to make them distinct from one another [12]. For example, $Cas(A, Na, 0)$ and $Cas(A, Na, 1)$ or $Cas(A, Na)$ and $Cas(Na, A)$. In ASPB, we distinguish 'similar' messages by the latter approach, that is, exchanging message component order.

Since the message receiver of a protocol step is the message sender of the next protocol

step, the above protocol in Example 12 can be rewritten in the following format:

$$\begin{aligned} A \rightarrow B: &\quad C_p(A, Na), \\ B \rightarrow S: &\quad C_{bs}(A, B, Na, Nb), \\ S \rightarrow A: &\quad C_{as}(B, Na, Nb, A), \\ A \rightarrow B: &\quad C_p(Nb, B) \end{aligned}$$

Each protocol step means that a principal sends a message and another principal receives this message. For example, the first step of the above protocol $A \rightarrow B : C_p(A, Na)$ means that $A \mid\sim C_p(A, Na)$ and $B \triangleleft C_p(A, Na)$.

6.3.4 Removing Redundant Components

The Protocol Composer may use a redundancy removing strategy to further remove redundant components of protocol messages. This is inspired by Mao's protocol idealisation process [80], that is used to transform protocol messages into BAN-like formulae via a context-sensitive syntactic analysis of the protocol syntax. Our redundancy removing process is used to transform BSW-ZF formulae into protocol messages using a similar analysis of the protocol syntax as follows.

Let the *relevant principal set* $\mathcal{N}_P(N)$ represent a set of principals to which principal P currently believes that the fresh nonce N is uttered as a reference. P should remember this reference wherever P sees N. At the beginning of a protocol, the relevant principal sets for all principals are initialised to empty.

The Protocol Composer uses the following relevancy rules to calculate the relevant principal set for every principal at each protocol step.

RR1 If P believes that only P and Q may write in channel C, and P sees nonce N together with principal R from C, then P believes that N is uttered as a reference to the principals P, Q, and R.

$$\frac{P \models (s(w(C)) = \{P, Q\}), P \ni r(C), P \triangleleft C(N, \ldots, R)}{\mathcal{N}_P(N) = \mathcal{N}_P(N) \cup \{P, Q, R\}}$$

RR2 If P believes that only Q may write in channel C, and P sees nonce N with principal R from C, then P believes that N is uttered as a reference to the principals Q and R.

$$\frac{P \models (s(w(C)) = \{Q\}), P \ni r(C), P \triangleleft C(N, \ldots, R)}{\mathcal{N}_P(N) = \mathcal{N}_P(N) \cup \{Q, R\}}$$

RR3 If P believes that only P and Q may read from channel C, and P writes nonce N with principal R in C, then P believes that N is uttered as a reference to the principals P, Q, and R.

$$\frac{P \models (s(r(C)) = \{P, Q\}), P \ni w(C), P \mid\sim C(N, \ldots, R)}{\mathcal{N}_P(N) = \mathcal{N}_P(N) \cup \{P, Q, R\}}$$

RR4 If P believes that only Q may read from channel C, and P writes nonce N with principal R in C, then P believes that N is uttered as a reference to the principals Q and R.

$$\frac{P \models (s(r(C)) = \{Q\}), P \ni w(C), P \mid\sim C(N, \ldots, R)}{\mathcal{N}_P(N) = \mathcal{N}_P(N) \cup \{Q, R\}}$$

Generally, if $\mathcal{N}_P(N) \cap \mathcal{N}_Q(N) = \psi_1$ before a protocol step $P \rightarrow Q : (N, \psi_2)$, then the protocol step is rewritten by $P \rightarrow Q : C(N, \psi_2/\psi_1)$ This is done in reverse order of protocol execution. If a message may not be distinguished from other messages after applying a relevancy rule, then further principal identifiers from ψ_2 are kept in that message, until the message can be distinguished from other messages. In addition, if $\psi_2/\psi_1 = \{\}$ in the first encrypted message of a protocol, then at least one principal identity of that message is kept to stop principals misusing that message. If we remove all principal identities from the first encrypted message, the protocol may be subject to reflection/oracle attacks. The reason for this is that the message receiver may not judge who generates this message (another principal in the current protocol run, or itself in a previous run). This is because it is not practical for principals to keep messages from previous protocol runs. An example to illustrate this problem will be given in Section 6.5.2.

Example 13 The messages in the generated protocol from Example 12 are reduced and then simplified using above rules to give:

$$\begin{aligned}
&Message\ 1 \quad A \rightarrow B : \quad A, Na, \\
&Message\ 2 \quad B \rightarrow S : \quad \{A, Na, Nb\}_{K_{bs}}, \\
&Message\ 3 \quad S \rightarrow A : \quad \{Na, Nb, B\}_{K_{as}}, \\
&Message\ 4 \quad A \rightarrow B : \quad Nb.
\end{aligned}$$

△

6.4 The Protocol Selector

In general, given subprotocols $\mathbb{P}_1$ and $\mathbb{P}_2$ that uphold goals G_1 and G_2, respectively, then the monotonicity of the BSW-ZF logic ensures that the resulting merged candidate protocol as outlined above also upholds the goals G_1 and G_2 within the logic. Therefore, all the protocols generated by the Protocol Composer are valid within our logic.

However, belief logics do have weaknesses. Regardless of whether we deal with type flaw attacks by assuming that component types can be recognised by principals, the logic is still vulnerable to other classes of attacks. For example, the Protocol Composer generates the following simple mutual authentication protocol.

$$\begin{aligned} Message\ 1 \quad & A \rightarrow B: \quad A, Na, \\ Message\ 2 \quad & B \rightarrow A: \quad B, Nb, \{A, Na\}_{Kab} \\ Message\ 3 \quad & A \rightarrow B: \quad \{B, Nb\}_{Kab} \end{aligned}$$

While secure within the BSW-ZF and many other belief logics, this protocol is subject to a reflection/oracle attack:

$$\begin{aligned} Message\ 1 \quad & A \rightarrow B \quad & : A, Na, \\ Message\ 2 \quad & B \rightarrow I(A) \quad & : B, Nb, \{A, Na\}_{Kab}, \\ Message\ 2' \quad & I(B) \rightarrow A \quad & : B, Nb', \{A, Na\}_{Kab}, \\ Message\ 3 \quad & A \rightarrow B \quad & : \{B, Nb'\}_{Kab}. \end{aligned}$$

Here, Nb' is generated by the intruder. There are many examples of secure protocols, which when composed are vulnerable to attack [118]. Such unsuitable protocols that are easy to identify are discarded following verification by the Protocol Selector. For example, when a subprotocol $\mathbb{P}_1$ does not merge any message with any other subprotocols in the merging process, we consider that the resulting candidate protocol is an unsuitable protocol and discarded by the Protocol Selector. The reason for this is that the subprotocol $\mathbb{P}_1$ could be executed independently. An intruder can participate in the subprotocol without participating in the rest protocol of the same round.

It is useful to also consider verification of additional ad-hoc properties. For example, the non-injective agreement property [78]: *"For certain data items ds, if each time a principal B completes a run of the protocol as responder using ds, apparently with A, then there is a unique run of the protocol with the principal A as initiator using ds, apparently with B."* The generated protocols could be re-analysed using more sophisticated protocol analysis

tools, such as the NRL Analyzer [82], the Interrogator model [86], FDR [76], Murφ [88], Athena [107]. In this case, the Protocol Selector of ASPB would be used to narrow down the set of candidate protocols to be verified. However, we point out that many existing protocol checkers are also limited and also include ad-hoc strategies, and they do not guarantee the correctness of verified protocols.

We also suggest that practical techniques such as [12] may prove useful in making candidate protocols robust against such attacks. For example, by ensuring that the initiator challenge looks different to the respondent challenge.

6.5 Protocol Examples

ASPB generates a large number of protocols for different requirements. For the sake of illustration, only a small number of generated protocols are explored in this section. Note that we do not consider type-flaw attacks [34] in our prototype. However, this could be done by using other protocol checkers, such as the NRL Analyzer [82], the Interrogator model [86], FDR [76], Murφ [88], Athena [107], during protocol selection.

6.5.1 Mutual Authentication with TTP

Figure 6.2 illustrates a protocol requirement specification of a simple mutual authentication protocol that uses a trusted third party. ASPB generates Protocol 1.1 that was described in Example 13.

Protocol 1.1. (Perrig and Song's real optimal protocol [100])

$$
\begin{aligned}
&Message\ 1 \quad A \rightarrow B: \quad A, Na,\\
&Message\ 2 \quad B \rightarrow S: \quad \{A, Na, Nb\}_{K_{bs}},\\
&Message\ 3 \quad S \rightarrow A: \quad \{Na, Nb, B\}_{K_{as}},\\
&Message\ 4 \quad A \rightarrow B: \quad Nb.
\end{aligned}
$$

Protocol 1(a). (Perrig and Song's optimal protocol 1 [100])

$$
\begin{aligned}
&Message\ 1 \quad A \rightarrow B: \quad A, Na,\\
&Message\ 2 \quad B \rightarrow S: \quad \{Na, Nb, A\}_{K_{bs}},\\
&Message\ 3 \quad S \rightarrow A: \quad \{Na, Nb, A, B\}_{K_{as}},\\
&Message\ 4 \quad A \rightarrow B: \quad Nb.
\end{aligned}
$$

Protocol 1(b). (Perrig and Song's optimal protocol 2 [100])

$$\begin{array}{lll} Message\ 1 & A \rightarrow B: & A, Na, \\ Message\ 2 & B \rightarrow S: & \{A, B, Na, Nb\}_{K_{bs}}, \\ Message\ 3 & S \rightarrow A: & \{Na, Nb, B\}_{K_{as}}, \\ Message\ 4 & A \rightarrow B: & Nb. \end{array}$$

Protocol 1(a) and Protocol 1(b) were generated by Perrig and Song's APG [100]. ASPB also generates these two protocols, but considers them as interim protocols with redundant components. For example, the redundant components in Protocol 1(a) is A in $Message$ 3. Before S sends out $Message$ 3, $\mathcal{N}_A(Na) = \mathcal{N}_S(Na) = \{A, S\}$ is obtained by using **RR1** and **RR3**. Therefore, $\mathcal{N}_A(Na) \cap \mathcal{N}_S(Na) = \{A, S\}$. Since $\{A, B\}/\{A, S\} = \{B\}$, A is safely removed from $Message$ 3. The protocol properties do not change after the redundant components are removed. The redundant components in Protocol 1(b) is B in $Message$ 2. Similar redundant removing process can be applied on Protocol 1(b).

6.5.2 Mutual Authentication without TTP

Figure 6.2 illustrates a requirement specification of a simple mutual authentication protocol that uses a trusted third party. If the assumptions are changed to reflect the absence of this third party (all of the assumptions using S, Cas, and Cab are replaced by proper assumptions) then the synthesised protocols include the following.

Using symmetric keys

In this case, principals A and B share a symmetric key K_{ab}. The following assumptions about channel Cab are used in the modified requirement specification that is illustrated in Figure 6.6. For this modified requirement specification, ASPB generates the following six correct protocols, Protocol 2.1–2.6.

Protocol 2.1. (Perrig and Song's Optimal Minimum Cost Protocol [99])

$$\begin{array}{lll} Message\ 1 & A \rightarrow B: & \{A, Na\}_{K_{ab}}, \\ Message\ 2 & B \rightarrow A: & \{Na, Nb\}_{K_{ab}}, \\ Message\ 3 & A \rightarrow B: & Nb. \end{array}$$

declarations {
Channel Cab, Cp;
Principal A, B;
Nonce Na, Nb;
}
assumptions {
$A \models (\sigma(w(Cab)) = \{A, B\})$;
$B \models (\sigma(w(Cab)) = \{A, B\})$;
$A \models (\sigma(r(Cab)) = \{A, B\})$;
$B \models (\sigma(r(Cab)) = \{A, B\})$;
$A \ni r(Cab)$; $A \ni w(Cab)$;
$A \ni r(Cp)$; $A \ni w(Cp)$;
$B \ni r(Cab)$; $B \ni w(Cab)$;
$B \ni r(Cp)$; $B \ni w(Cp)$;
$A \models \sharp(Na)$; $B \models \sharp(Nb)$;
$A \mapsto Na$; $B \mapsto Nb$;
$A \ni A$; $B \ni B$;
}
goals {
$A \models (B \mid\sim A)$; /* G_1 */
$B \models (A \mid\sim B)$; /* G_2 */
}

Figure 6.6: A Requirement Specification for Mutual Authentication without TTP using Symmetric Keys

Protocol 2.2. (Perrig and Song's Minimum Cost Protocol 2 [99])

$$
\begin{aligned}
&Message\ 1 \quad A \rightarrow B: \quad A, Na, \\
&Message\ 2 \quad B \rightarrow A: \quad \{B, Na, Nb\}_{K_{ab}}, \\
&Message\ 3 \quad A \rightarrow B: \quad Nb.
\end{aligned}
$$

Protocol 2.3. (Perrig and Song's Minimum Cost Protocol 1 [99])

$$
\begin{aligned}
&Message\ 1 \quad A \rightarrow B: \quad A, Na, \\
&Message\ 2 \quad B \rightarrow A: \quad \{A, Na, Nb\}_{K_{ab}}, \\
&Message\ 3 \quad A \rightarrow B: \quad Nb.
\end{aligned}
$$

The above three protocols were first generated by APG [99]. Compare with Protocol 2.2 and Protocol 2.3, Protocol 2.1 has an encrypted first message, and a shorter second message. Perrig and Song consider that Protocol 2.1 can be used in environments that perform fast encryption and decryption operations, but using slow links for data transitions. Protocol 2.2

and Protocol 2.3 may be used in regular environments, that the cost of data transitions is lower than encryption and decryption operations.

Note that Protocol 2.1 also satisfies the secrecy requirement that is described as goal $G_{a1} : A \models (\sigma(Na) = \{A, B\})$ and goal $G_{a2} : B \models (\sigma(Na) = \{A, B\})$. By satisfying G_{a1} and G_{a2} after a round of these protocols, it is possible for A and B to use Na as a secret shared only between them for further communication. Protocol 2.2 and Protocol 2.3 do not satisfy goal G_{a1} and G_{a2}.

Before obtaining Protocol 2.2 and Protocol 2.3, ASPB generates the following interim Protocol 2(a) that has a redundant component in the second message.

$$
\begin{aligned}
Message\ 1 \quad & A \rightarrow B : \quad A, Na, \\
Message\ 2 \quad & B \rightarrow A : \quad \{A, B, Na, Nb\}_{K_{ab}}, \\
Message\ 3 \quad & A \rightarrow B : \quad Nb.
\end{aligned}
$$

In the redundancy removing process, either protocol initiator's identifier A or protocol responder's identifier B can be removed from $Message\ 2$. This does not change the properties of Protocol 2(a). After removing one of the message components A and B, Protocol 2.2 and Protocol 2.3 are obtained individually. On the other hand, according to the redundancy removing rules, message components A and B may not be removed at the same time to obtain Protocol 2(b).

Protocol 2(b).

$$
\begin{aligned}
Message\ 1 \quad & A \rightarrow B : \quad A, Na, \\
Message\ 2 \quad & B \rightarrow A : \quad \{Na, Nb\}_{K_{ab}}, \\
Message\ 3 \quad & A \rightarrow B : \quad Nb.
\end{aligned}
$$

The reason for this is that Protocol 2(b) is subject to the following reflection/oracle attack. When E_B intercepts message 1, E_B may start another round of Protocol 2(b) by forwarding all of A's messages to A. After message 3, A believes that A finished two rounds

of the protocol with B, but B does not participate in any round of the protocol.

$$
\begin{array}{lll}
Message\ 1 & A \rightarrow E_B: & A, Na, \\
Message\ 1' & E_B \rightarrow A: & B, Na, \\
Message\ 2' & A \rightarrow E_B: & \{Na, Na'\}_{K_{ab}}, \\
Message\ 2 & E_B \rightarrow A: & \{Na, Na'\}_{K_{ab}}, \\
Message\ 3 & A \rightarrow E_B: & Na'. \\
Message\ 3' & E_B \rightarrow A: & Na'.
\end{array}
$$

Protocol 2.4.

$$
\begin{array}{lll}
Message\ 1 & A \rightarrow B: & \{A, Na\}_{K_{ab}}, \\
Message\ 2 & B \rightarrow A: & \{Na, Nb\}_{K_{ab}}, \\
Message\ 3 & A \rightarrow B: & \{Nb\}_{K_{ab}}.
\end{array}
$$

Protocol 2.5.

$$
\begin{array}{lll}
Message\ 1 & A \rightarrow B: & A, Na, \\
Message\ 2 & B \rightarrow A: & \{B, Na, Nb\}_{K_{ab}}, \\
Message\ 3 & A \rightarrow B: & \{Nb\}_{K_{ab}}.
\end{array}
$$

Protocol 2.6.

$$
\begin{array}{lll}
Message\ 1 & A \rightarrow B: & A, Na, \\
Message\ 2 & B \rightarrow A: & \{A, Na, Nb\}_{K_{ab}}, \\
Message\ 3 & A \rightarrow B: & \{Nb\}_{K_{ab}}.
\end{array}
$$

Protocol 2.4–2.6 are similar to Protocol 2.1–2.3, but each of them has an encrypted third message. Comparing with Protocol 2.1–2.3, we consider that Protocol 2.4–2.6 also satisfy the secrecy requirement, that are described as additional protocol goals, $G_{a3} : B \models (\sigma(Nb) = \{A, B\})$ and $G_{a4} : A \models (\sigma(Nb) = \{A, B\})$. By satisfying G_{a3} and G_{a4} after a round of these protocols, A and B can use Nb as a secret shared only between them for further communication.

ISO/IEC 9798-2 [3] proposes the ISO/IEC Symmetric-Key Three-Pass Mutual Authentication Protocol, as follows:

$$\begin{aligned} Message\ 1 \quad & A \rightarrow B: \quad A, Na, \\ Message\ 2 \quad & B \rightarrow A: \quad \{Na, Nb, A\}_{K_{ab}}, \\ Message\ 3 \quad & A \rightarrow B: \quad \{Na, Nb\}_{K_{ab}}. \end{aligned}$$

Clearly, Protocols 2.1–2.6, generated by ASPB, are simpler than the ISO standard protocol, yet achieve the same mutual authentication goals. Therefore, it is reasonable to consider that component Na in $Message$ 3 of the standard ISO protocol is a redundant component for the purposes of mutual authentication.

Using signature keys

In this case, both principals can verify each other's signature key SK_a and SK_b. Assumptions are adapted to represent the corresponding authenticated channel Ca and Cb. The modified requirement specification is represented in Figure 6.7. For this modified requirement specification, ASPB generated two correct protocols.

Protocol 2.7.

$$\begin{aligned} Message\ 1 \quad & A \rightarrow B: \quad A, Na, \\ Message\ 2 \quad & B \rightarrow A: \quad \{A, Na, Nb\}_{SK_b}, \\ Message\ 3 \quad & A \rightarrow B: \quad \{Nb, B\}_{SK_a}. \end{aligned}$$

Protocol 2.8.

$$\begin{aligned} Message\ 1 \quad & A \rightarrow B: \quad \{A, Na\}_{SK_a}, \\ Message\ 2 \quad & B \rightarrow A: \quad \{A, Na, Nb\}_{SK_b}, \\ Message\ 3 \quad & A \rightarrow B: \quad \{Nb, B\}_{SK_a}. \end{aligned}$$

The only difference between Protocol 2.7 and Protocol 2.8 is whether the first message is signed by A's signature key. ASPB generates these two different protocols because ASPB's Single Goal Synthesiser instantiates typed variables with all possible instantiations that are defined in requirement specification. Since both Protocol 2.7 and Protocol 2.8 satisfy given mutual authentication goals, the above difference does not influence the authentication properties of these protocols. Therefore, it is reasonable to consider that the signature in $Message$ 1 of Protocol 2.8 is a redundant component for the purposes of mutual authentication.

declarations {
Channel Cab, Cp;
Principal A, B;
Nonce Na, Nb;
}
assumptions {
$A \models (\sigma(w(Ca)) = \{A\})$;
$B \models (\sigma(w(Ca)) = \{A\})$;
$A \models (\sigma(w(Cb)) = \{B\})$;
$B \models (\sigma(w(Cb)) = \{B\})$;
$A \ni r(Ca)$; $A \ni w(Ca)$; $A \ni r(Cb)$;
$A \ni r(Cp)$; $A \ni w(Cp)$;
$B \ni r(Cb)$; $B \ni w(Cb)$; $B \ni r(Ca)$;
$B \ni r(Cp)$; $B \ni w(Cp)$;
$A \models \sharp(Na)$; $B \models \sharp(Nb)$;
$A \mapsto Na$; $B \mapsto Nb$;
$A \ni A$; $B \ni B$;
}
goals {
$A \models (B \mid\sim A)$; /* G_1 */
$B \models (A \mid\sim B)$; /* G_2 */
}

Figure 6.7: A requirement specification for Mutual Authentication without TTP using signature keys

ISO/IEC 9798-3 [4] proposes the ISO/IEC Signature-Key Three-Pass Mutual Authentication Protocol, as follows:

$$\begin{aligned} Message\ 1 \quad & A \rightarrow B: \quad A, Na, \\ Message\ 2 \quad & B \rightarrow A: \quad \{A, Na, Nb\}_{SK_b}, \\ Message\ 3 \quad & A \rightarrow B: \quad \{Na, Nb, B\}_{SK_a}. \end{aligned}$$

Clearly, Protocol 2.7, a protocol generated by ASPB, is simpler than the ISO standard protocol, yet achieves the same mutual authentication goals. Therefore, it is reasonable to consider that component Na in $Message$ 3 of the standard ISO protocol is a redundant component for the purposes of mutual authentication.

Using Public Keys

In this case, both principals know each other's public keys. The following assumptions about channel Ca and Cb are used in the modified requirement specification. The modified requirement specification is represented in Figure 6.8. For this modified requirement

specification, the following two protocols were generated by ASPB.

declarations {
Channel Cab, Cp;
Principal A, B;
Nonce Na, Nb;
}
assumptions {
$A \models (\sigma(r(Ca)) = \{A\})$;
$B \models (\sigma(r(Ca)) = \{A\})$;
$A \models (\sigma(r(Cb)) = \{B\})$;
$B \models (\sigma(r(Cb)) = \{B\})$;
$A \ni r(Ca)$; $A \ni w(Ca)$; $A \ni w(Cb)$;
$A \ni r(Cp)$; $A \ni w(Cp)$;
$B \ni r(Cb)$; $B \ni w(Cb)$; $B \ni w(Ca)$;
$B \ni r(Cp)$; $B \ni w(Cp)$;
$A \models \sharp(Na)$; $B \models \sharp(Nb)$;
$A \mapsto Na$; $B \mapsto Nb$;
$A \ni A$; $B \ni B$;
}
goals {
$A \models (B \mid\sim A)$; /* G_1 */
$B \models (A \mid\sim B)$; /* G_2 */
}

Figure 6.8: A requirement specification for Mutual Authentication without TTP using Public Keys

Protocol 2.9. (Corresponds to Needham-Schroeder-Lowe protocol [76])

$$
\begin{aligned}
Message\ 1 \quad & A \rightarrow B: \quad \{A, Na\}_{K_b}, \\
Message\ 2 \quad & B \rightarrow A: \quad \{Na, Nb, B\}_{K_a}, \\
Message\ 3 \quad & A \rightarrow B: \quad \{Nb\}_{K_b}.
\end{aligned}
$$

Protocol 2.10. (Perrig and Song's Minimum Cost Protocol 3 [99])

$$
\begin{aligned}
Message\ 1 \quad & A \rightarrow B: \quad \{A, Na\}_{K_b}, \\
Message\ 2 \quad & B \rightarrow A: \quad \{Na, Nb, B\}_{K_a}, \\
Message\ 3 \quad & A \rightarrow B: \quad Nb.
\end{aligned}
$$

The difference between Protocol 2.9 and Protocol 2.10 is whether the last message is encrypted. Consequently, Protocol 2.9 achieves the secrecy requirement, that is described as additional protocol goals $G_{a3} : B \models (\sigma(Nb) = \{A, B\})$ and $G_{a4} : A \models (\sigma(Nb) = \{A, B\})$,

while Protocol 2.10 does not satisfy these secrecy requirements.

6.5.3 Mutual Authentication and Key Agreement Protocol

Figure 6.9 gives a complete requirement specification for a mutual authentication and key agreement protocol that involves a Trusted Third Party.

Four Message Protocols

ASPB synthesises a number of four-message mutual authentication protocols. For the purpose of illustration, we describe and discuss the following generated protocols, Protocols 3.1–3.7, that satisfy the requirement specification in Figure 6.9.

Protocol 3.1. (Perrig and Song's protocol 1 in Protocol-Set S1 [100])

$$
\begin{array}{lll}
Message\ 1 & A \rightarrow B: & A, Na, \\
Message\ 2 & B \rightarrow S: & B, \{A, Na, Nb\}_{K_{bs}}, \\
Message\ 3 & S \rightarrow A: & \{Nb, K_{ab}\}_{K_{bs}}, \{B, Na, Nb, K_{ab}\}_{K_{as}} \\
Message\ 4 & A \rightarrow B: & \{Nb\}_{K_{ab}}, \{Nb, K_{ab}\}_{K_{bs}}.
\end{array}
$$

Protocol 3.2. (Perrig and Song's protocol in Protocol-Set S3 [100])

$$
\begin{array}{lll}
Message\ 1 & A \rightarrow B: & A, Na, \\
Message\ 2 & B \rightarrow S: & B, \{A, Na, Nb\}_{K_{bs}}, \\
Message\ 3 & S \rightarrow A: & \{Nb, K_{ab}\}_{K_{bs}}, \{B, Na, Nb, K_{ab}\}_{K_{as}} \\
Message\ 4 & A \rightarrow B: & Nb, \{Nb, K_{ab}\}_{K_{bs}}.
\end{array}
$$

In Protocol 3.1 B believes that A receives K_{ab} from S. However, this is not the case in Protocol 3.2 since A does not use the key K_{ab} (to encrypt the nonce). Protocols 3.1 and 3.2 also appear in Protocol-Sets S1 and S3 from [100]. In addition, Protocol 3(a) was generated by APG in Protocol-Set S1 [100], as follows:

Protocol 3(a). (Perrig and Song's protocol 2 in Protocol-Set S1 [100])

$$
\begin{array}{lll}
Message\ 1 & A \rightarrow B: & A, Na, \\
Message\ 2 & B \rightarrow S: & B, \{A, Na, Nb\}_{K_{bs}}, \\
Message\ 3 & S \rightarrow A: & \{Nb, K_{ab}\}_{K_{bs}}, \{B, Na, Nb, K_{ab}\}_{K_{as}} \\
Message\ 4 & A \rightarrow B: & \{\{Nb, K_{ab}\}_{K_{bs}}\}_{K_{ab}}, \{Nb, K_{ab}\}_{K_{bs}}.
\end{array}
$$

ASPB does not generate Protocol 3(a). The reason for this is that ASPB does not allow a

declarations {
Channel Cas, Cbs, Cab, Cp;
Principal A, B, S;
Nonce Na, Nb;
Message X;
Formula ϕ;
}
assumptions {
$A \models (\sigma(w(Cas)) = \{A, S\})$; $A \models (\sigma(r(Cas)) = \{A, S\})$;
$S \models (\sigma(w(Cas)) = \{A, S\})$; $S \models (\sigma(r(Cas)) = \{A, S\})$;
$B \models (\sigma(w(Cbs)) = \{B, S\})$; $B \models (\sigma(r(Cbs)) = \{B, S\})$;
$S \models (\sigma(w(Cbs)) = \{B, S\})$; $S \models (\sigma(r(Cbs)) = \{B, S\})$;
$S \models (\sigma(w(Cab)) = \{A, B\})$; $S \models (\sigma(r(Cab)) = \{A, B\})$;
$A \ni r(Cas)$; $A \ni w(Cas)$; $A \ni r(Cp)$; $A \ni w(Cp)$;
$B \ni r(Cbs)$; $B \ni w(Cbs)$; $B \ni r(Cp)$; $B \ni w(Cp)$;
$S \ni r(Cas)$; $S \ni w(Cas)$; $S \ni r(Cp)$; $S \ni w(Cp)$;
$S \ni r(Cbs)$; $S \ni w(Cbs)$;
$A \models \sharp(Na)$; $B \models \sharp(Nb)$; $S \models \sharp(w(Cab))$;
$A \mapsto Na$; $B \mapsto Nb$; $S \mapsto r(Cab)$; $S \mapsto w(Cab)$;
$A \ni A$; $B \ni B$; $S \ni S$;
$A \models ((S \mid\sim \phi) \rightarrow (S \models \phi))$;
$B \models ((S \mid\sim \phi) \rightarrow (S \models \phi))$;
$A \models ((S \models (B \mid\sim X)) \rightarrow (B \mid\sim X))$;
$B \models ((S \models (A \mid\sim X)) \rightarrow (A \mid\sim X))$;
$A \models ((S \models (\sigma(w(Cab)) = \{A, B\}) \rightarrow (\sigma(w(Cab)) = \{A, B\})))$;
$B \models ((S \models (\sigma(w(Cab)) = \{A, B\}) \rightarrow (\sigma(w(Cab)) = \{A, B\})))$;
$A \models (S \mid\sim w(Cab) \rightarrow \sharp(w(Cab)))$;
$B \models (S \mid\sim w(Cab) \rightarrow \sharp(w(Cab)))$;
}
goals {
$A \models (B \mid\sim A)$; /* G_1 */
$B \models (A \mid\sim B)$; /* G_2 */
$A \models (\sigma(w(Cab)) = \{A, B\})$; /* G_3 */
$B \models (\sigma(w(Cab)) = \{A, B\})$; /* G_4 */
}

Figure 6.9: The requirement specification for the mutual authentication and key agreement protocol using Trusted Third Party.

message to be sent into multiple channels at the same protocol step (as $\{\{Nb, K_{ab}\}_{K_{bs}}\}_{K_{ab}}$ in Message 4). Instead, ASPB generates Protocol 3.3 that is similar to Protocol 3(a). Compare to Protocol 3(a), Protocol 3.3 has a simpler message component $\{B\}_{K_{ab}}$ in Message 4. Since both message components $\{\{Nb, K_{ab}\}_{K_{bs}}\}_{K_{ab}}$ and $\{B\}_{K_{ab}}$ are used to prove that principal A holds session key K_{ab}, and the use of session key K_{ab} may prove the freshness of the current message component, the content encrypted by K_{ab} is only required to be recognisable by B, but is not required to be fresh.

Protocol 3.3.

$$
\begin{array}{lll}
Message\ 1 & A \rightarrow B: & A, Na, \\
Message\ 2 & B \rightarrow S: & B, \{A, Na, Nb\}_{K_{bs}}, \\
Message\ 3 & S \rightarrow A: & \{Nb, K_{ab}\}_{K_{bs}}, \{B, Na, Nb, K_{ab}\}_{K_{as}} \\
Message\ 4 & A \rightarrow B: & \{B\}_{K_{ab}}, \{Nb, K_{ab}\}_{K_{bs}}.
\end{array}
$$

However, for the requirement specification in Figure 6.8, ASPB does not generate protocols in the Protocol-Set S2 for authentication and key agreement from [100] (this includes the original Yahalom protocol). The reason is that the protocols in Protocol-Set S2 have an assumption that B believes that A is honest. If B believes A accepts a session key then B will accept it. Otherwise, A can make B to accept an old session key in the current round of the protocol. For example, Protocol 3(b) is Protocol 1 in Protocol-Set S2 [100].

Protocol 3(b).

$$
\begin{array}{lll}
Message\ 1 & A \rightarrow B: & A, Na, \\
Message\ 2 & B \rightarrow S: & B, \{A, Na, Nb\}_{K_{bs}}, \\
Message\ 3 & S \rightarrow A: & \{K_{ab}\}_{K_{bs}}, \{B, Na, Nb, K_{ab}\}_{K_{as}} \\
Message\ 4 & A \rightarrow B: & \{Nb\}_{K_{ab}}, \{K_{ab}\}_{K_{bs}}.
\end{array}
$$

When assumption, that B believes that A is honest, is not made, A is able to generate and send message

$$Message\ 4' \quad A \rightarrow B: \quad \{Nb\}_{K'_{ab}}, \{K'_{ab}\}_{K_{bs}}$$

where K'_{ab} is an old (expired) session key between A and B.

For the sake of illustration, this assumption was not made in the requirement specification in Figure 6.9 which was used to conduct our experiments.

Protocol 3.4.

$$
\begin{array}{lll}
Message\ 1 & A \rightarrow B: & A, Na, \{B, Na\}_{K_{as}}, \\
Message\ 2 & B \rightarrow S: & B, \{B, Na\}_{K_{as}}, \{A, Na, Nb\}_{K_{bs}}, \\
Message\ 3 & S \rightarrow A: & \{Nb, K_{ab}\}_{K_{bs}}, \{Na, Nb, K_{ab}\}_{K_{as}} \\
Message\ 4 & A \rightarrow B: & \{Nb\}_{K_{ab}}, \{Nb, K_{ab}\}_{K_{bs}}.
\end{array}
$$

In the new Protocol 3.4, when the TTP S receives $Message\ 2$, S may check whether two principals know who the other party is in the current round. If S finds that one principal attempts to cheat the other by generating a different $Message\ 2$, then it can stop the current round as early as possible. While Protocol 3.4 has a higher cost (in terms of message size) than the other protocols, it provides a more powerful TTP. Two new protocols, that are similar to Protocol 3.4, are generated by ASPB as Protocol 3.5 and 3.6.

Protocol 3.5.

$$
\begin{array}{lll}
Message\ 1 & A \rightarrow B: & A, Na, \\
Message\ 2 & B \rightarrow S: & B, Nb, \{A, Na\}_{K_{bs}}, \\
Message\ 3 & S \rightarrow A: & \{A, Nb, K_{ab}\}_{K_{bs}}, \{B, Na, Nb, K_{ab}\}_{K_{as}} \\
Message\ 4 & A \rightarrow B: & \{Nb\}_{K_{ab}}, \{A, Nb, K_{ab}\}_{K_{bs}}.
\end{array}
$$

Protocol 3.6.

$$
\begin{array}{lll}
Message\ 1 & A \rightarrow B: & A, Na, \\
Message\ 2 & B \rightarrow S: & B, \{A, Na, Nb\}_{K_{bs}}, \\
Message\ 3 & S \rightarrow A: & Nb, \{Nb, K_{ab}\}_{K_{bs}}, \{B, Na, K_{ab}\}_{K_{as}} \\
Message\ 4 & A \rightarrow B: & \{Nb\}_{K_{ab}}, \{Nb, K_{ab}\}_{K_{bs}}.
\end{array}
$$

Protocol 3.7.(BAN optimized Yahalom protocol [27])

$$
\begin{array}{lll}
Message\ 1 & A \rightarrow B: & A, Na, \\
Message\ 2 & B \rightarrow S: & B, Nb, \{A, Na\}_{K_{bs}}, \\
Message\ 3 & S \rightarrow A: & Nb, \{A, Nb, K_{ab}\}_{K_{bs}}, \{Na, K_{ab}, B\}_{K_{as}} \\
Message\ 4 & A \rightarrow B: & \{Nb\}_{K_{ab}}, \{A, Nb, K_{ab}\}_{K_{bs}}.
\end{array}
$$

Syverson [111] describes a flaw in Protocol 3.7 (when B is not able to distinguish the

format of different components) as follows.

$$
\begin{array}{lll}
Message\ 1(\alpha) & A \rightarrow B: & A, Na, \\
Message\ 2(\alpha) & B \rightarrow S: & B, Nb, \{A, Na\}_{K_{bs}}, \\
Message\ 1(\beta) & I(A) \rightarrow B: & A, (Na, Nb) \\
Message\ 2(\beta) & B \rightarrow I(S): & B, Nb', \{A, Na, Nb\}_{K_{bs}}, \\
Message\ 3(\alpha) & omitted, & \\
Message\ 4(\alpha) & I(A) \rightarrow B: & \{Nb\}_{K_{ab}}, \{A, Na(=K_{ab}), Nb\}_{K_{bs}}.
\end{array}
$$

However, ASPB uses the following assumptions.

- A principal can recognise his own nonces of the running rounds, and refuse to use them as other principal's nonces.
- If a principal may understand a message context, the principal may distinguish the format of different components, such as principal name, nonce, key, etc.

By these assumptions, Protocol 3.7 is also a correct protocol.

ASPB generates Paulson amended Yahalom Protocol [96], but considers it as an interim protocol. After removing the redundant message components, Protocol 3.7 is obtained.

Protocol 3(c).(Paulson amended Yahalom Protocol [96])

$$
\begin{array}{lll}
Message\ 1 & A \rightarrow B: & A, Na, \\
Message\ 2 & B \rightarrow S: & B, Nb, \{A, Na\}_{K_{bs}}, \\
Message\ 3 & S \rightarrow A: & Nb, \{A, B, Nb, K_{ab}\}_{K_{bs}}, \{B, Na, K_{ab}\}_{K_{as}} \\
Message\ 4 & A \rightarrow B: & \{Nb\}_{K_{ab}}, \{A, B, Nb, K_{ab}\}_{K_{bs}}.
\end{array}
$$

Five Message Protocols

The ISO/IEC Symmetric-Key Five-Pass Mutual Authentication Protocol proposed in ISO/IEC 9798-2 [3].

$$
\begin{array}{lll}
Message\ 1 & A \rightarrow B: & A, Na, \\
Message\ 2 & B \rightarrow S: & A, B, Na, Nb, \\
Message\ 3 & S \rightarrow B: & \{Na, K_{ab}, B\}_{K_{as}}, \{Nb, K_{ab}, A\}_{K_{bs}}, \\
Message\ 4 & B \rightarrow A: & \{Na, K_{ab}, B\}_{K_{as}}, \{Na, Nb'\}_{K_{ab}}, \\
Message\ 5 & A \rightarrow B: & \{Nb', Na\}_{K_{ab}}.
\end{array}
$$

Carlsen also describes a five message protocol, the Secret Key Initiator Protocol [29], as follows:

$$
\begin{array}{lll}
Message\ 1 & A \rightarrow B: & A, Na, \\
Message\ 2 & B \rightarrow S: & A, B, Na, Nb, \\
Message\ 3 & S \rightarrow B: & \{Na, B, K_{ab}\}_{K_{as}}, \{A, Nb, K_{ab}\}_{K_{bs}}, \\
Message\ 4 & B \rightarrow A: & \{Na, B, K_{ab}\}_{K_{as}}, \{Na\}_{K_{ab}}, Nb', \\
Message\ 5 & A \rightarrow B: & \{Nb'\}_{K_{ab}}.
\end{array}
$$

In these protocols, principal B uses two nonces. The protocol requirement specification in Figure 6.9 that formed the basis of our experiments specified that principal B uses one nonce. A consequence of this is that the exact ISO/IEC Symmetric-Key Five-Pass Mutual Authentication Protocol and Carlsen protocol are not generated for our given requirement specification in Figure 6.9. However, ASPB generates the Carlson protocol when the protocol specification is extended to include B's use of two nonces.

The remainder of this section discusses a number of five-message mutual authentication protocols generated by ASPB, that satisfy the requirement specification in Figure 6.9. For the purpose of illustration, we describe and discuss the following generated protocols, Protocol 4.1–4.8.

Protocol 4.1.

$$
\begin{array}{lll}
Message\ 1 & A \rightarrow B: & A, Na, \\
Message\ 2 & B \rightarrow S: & A, B, Na, Nb, \\
Message\ 3 & S \rightarrow B: & \{A, Nb, K_{ab}\}_{K_{bs}}, \{Na, Nb, K_{ab}, B\}_{K_{as}}, \\
Message\ 4 & B \rightarrow A: & \{Na, Nb, K_{ab}, B\}_{K_{as}}, \{Na\}_{K_{ab}}, \\
Message\ 5 & A \rightarrow B: & \{Nb\}_{K_{ab}}.
\end{array}
$$

Protocol 4.1 is similar to Carlsen's Secret Key Initiator Protocol. While B generates only one nonce Nb in Protocol 4.1, it achieves the same result as Carlsen's protocol. Once B receives $Message$ 3, he can check whether S believes that B needs a session key with A, and S believes Nb is B's nonce. When A receives $Message$ 4, she may believe that Nb is generated by B (otherwise, B may not generate $\{Na\}_{K_{ab}}$).

Protocol 4.2.

$$
\begin{array}{lll}
Message\ 1 & A \rightarrow B: & A, Na, \\
Message\ 2 & B \rightarrow S: & B, \{A, Na, Nb\}_{K_{bs}}, \\
Message\ 3 & S \rightarrow A: & \{Nb, K_{ab}\}_{K_{bs}}, \{B, Na, Nb, K_{ab}\}_{K_{as}}, \\
Message\ 4 & A \rightarrow B: & \{Nb\}_{K_{ab}}, \{Nb, K_{ab}\}_{K_{bs}}, \\
Message\ 5 & B \rightarrow A: & \{Na\}_{K_{ab}}.
\end{array}
$$

The first four messages of Protocol 4.2 are the same as Protocol 3.1. From the additional message $Message$ 5, Protocol 4.2 meets an additional goal that A believes B has received the session key Kab.

Protocol 4.3.

$$
\begin{array}{lll}
Message\ 1 & A \rightarrow B: & A, Na, \\
Message\ 2 & B \rightarrow A: & B, Nb, \{A, Na\}_{K_{bs}}, \\
Message\ 3 & A \rightarrow S: & A, \{Nb, B\}_{K_{as}}, \{A, Na\}_{K_{bs}}, \\
Message\ 4 & S \rightarrow A: & \{Na, Nb, K_{ab}\}_{K_{as}}, \{A, Nb, K_{ab}\}_{K_{bs}}, \\
Message\ 5 & A \rightarrow B: & \{A, Nb, K_{ab}\}_{K_{bs}}, \{Nb\}_{K_{ab}}.
\end{array}
$$

Protocol 4.3 is a novel protocol. When S receives $Message$ 3, he may check whether the components have been generated by A and B. If this is the case then S sends $Message$ 4. Further five-message protocols that were generated by ASPB include the following.

Protocol 4.4.

$$
\begin{array}{lll}
Message\ 1 & A \rightarrow B: & A, Na, \{A, Na\}_{K_{as}}, \\
Message\ 2 & B \rightarrow S: & B, \{A, Na, Nb\}_{K_{bs}}, \{A, Na\}_{K_{as}}, \\
Message\ 3 & S \rightarrow A: & \{Na, Nb, K_{ab}\}_{K_{bs}}, \{B, Na, Nb, K_{ab}\}_{K_{as}}, \\
Message\ 4 & A \rightarrow B: & \{Nb\}_{K_{ab}}, \{Na, Nb, K_{ab}\}_{K_{bs}}, \\
Message\ 5 & B \rightarrow A: & \{Na\}_{K_{ab}}.
\end{array}
$$

Protocol 4.5.

$$
\begin{array}{lll}
Message\ 1 & A \rightarrow B: & A, \{B, Na\}_{K_{as}}, \\
Message\ 2 & B \rightarrow S: & B, \{B, Na\}_{K_{as}}, \{Nb, A\}_{K_{bs}}, \\
Message\ 3 & S \rightarrow B: & \{Na, Nb, K_{ab}\}_{K_{as}}, \{Na, Nb, K_{ab}\}_{K_{bs}}, \\
Message\ 4 & B \rightarrow A: & Na, \{Na, Nb, K_{ab}\}_{K_{as}}, \\
Message\ 5 & A \rightarrow B: & Nb.
\end{array}
$$

Protocol 4.6.

$$
\begin{array}{lll}
Message\ 1 & A \rightarrow B: & A, \{B, Na\}_{K_{as}}, \\
Message\ 2 & B \rightarrow S: & B, \{B, Na\}_{K_{as}}, \{Nb, A\}_{K_{bs}}, \\
Message\ 3 & S \rightarrow A: & \{Na, Nb, K_{ab}\}_{K_{as}}, \{Na, Nb, K_{ab}\}_{K_{bs}}, \\
Message\ 4 & A \rightarrow B: & \{Nb\}_{K_{ab}}, \{Na, Nb, K_{ab}\}_{K_{as}}, \\
Message\ 5 & B \rightarrow A: & \{Na\}_{K_{ab}}.
\end{array}
$$

Protocol 4.7.

$$
\begin{array}{lll}
Message\ 1 & A \rightarrow B: & A, Na, \\
Message\ 2 & B \rightarrow S: & A, B, Na, \{A, Nb\}_{K_{bs}}, \\
Message\ 3 & S \rightarrow A: & \{B, Na, Nb, K_{ab}\}_{K_{as}}, \{Nb, K_{ab}\}_{K_{bs}}, \\
Message\ 4 & A \rightarrow B: & \{Nb, K_{ab}\}_{K_{bs}}, \{Nb\}_{K_{ab}}, \\
Message\ 5 & B \rightarrow A: & \{A\}_{K_{ab}}.
\end{array}
$$

Protocol 4.8.

$$
\begin{array}{lll}
Message\ 1 & A \rightarrow B: & A, Na, \\
Message\ 2 & B \rightarrow S: & A, B, Na, Nb, \\
Message\ 3 & S \rightarrow A: & \{A, Na, Nb, Kab\}_{K_{bs}}, \{B, Na, Kab\}_{K_{as}}, \\
Message\ 4 & A \rightarrow B: & \{Nb, Na\}_{K_{ab}}, \{A, Na, Nb, Kab\}_{K_{bs}} \\
Message\ 5 & B \rightarrow A: & \{Nb\}_{K_{ab}}.
\end{array}
$$

Six Message Protocols

ASPB synthesises a number of six-message mutual authentication protocols. For the purpose of illustration, we only describe the following protocol, Protocol 5.1, that satisfies the

requirement specification in Figure 6.9.

Protocol 5.1.

$$
\begin{array}{lll}
Message\ 1 & A \rightarrow B: & A, Na, \\
Message\ 2 & B \rightarrow S: & A, B, Na, Nb, \\
Message\ 3 & S \rightarrow B: & \{A, Nb, K_{ab}\}_{K_{bs}}, \{Na, Nb, K_{ab}, B\}_{K_{as}}, \\
Message\ 4 & B \rightarrow A: & \{Na, Nb, K_{ab}, B\}_{K_{as}}, \\
Message\ 5 & A \rightarrow B: & \{A, Na, Nb\}_{K_{ab}}, \\
Message\ 6 & B \rightarrow A: & \{B, Na\}_{K_{ab}}.
\end{array}
$$

ASPB does not generate the six-message protocols in [32] that are listed as Protocol 5(a) and 5(b). The reason for this is that ASPB generates protocols according to corresponding principal sequences, whereby the receiver of a message is the sender of the next message. Protocol 5(a) and 5(b) can not be described in terms of principal sequences. For example, in Protocol 5(a), after A receives message 2, B (not A) sends Message 3. We observe that without the participation of previous protocol steps, B is unable to determine when Message 3 should be sent. Therefore, we argue that if a protocol is not described in terms of a principal sequence, then it is difficult to properly execute the protocol in practice.

Protocol 5(a).

$$
\begin{array}{lll}
Message\ 1 & A \rightarrow S: & A, B, Na, \\
Message\ 2 & S \rightarrow A: & \{B, Na, Kab\}_{K_{as}}, \\
Message\ 3 & B \rightarrow S: & B, A, Nb, \\
Message\ 4 & S \rightarrow B: & \{A, Na, Nb, Kab\}_{K_{bs}}, \\
Message\ 5 & A \rightarrow B: & \{B, Nb, Na\}_{K_{ab}}, \\
Message\ 6 & B \rightarrow A: & \{Nb, A\}_{K_{ab}}.
\end{array}
$$

Protocol 5(b).

$$
\begin{array}{lll}
Message\ 1 & A \rightarrow B: & A, Na, \\
Message\ 2 & B \rightarrow S: & A, B, Na, Nb, \\
Message\ 3 & S \rightarrow B: & \{A, Nb, K_{ab}\}_{K_{bs}}, \\
Message\ 4 & S \rightarrow A: & \{Na, Nb, K_{ab}, B\}_{K_{as}}, \\
Message\ 5 & A \rightarrow B: & \{A, Na, Nb\}_{K_{ab}}, \\
Message\ 6 & B \rightarrow A: & \{B, Na\}_{K_{ab}}.
\end{array}
$$

Protocol 5(a) can be refined as Protocol 4.8 by reordering the message to satisfy the ASPB principal sequence and by refining messages by using the ASPB redundancy removing rules. Clearly, our 5-message protocols can be considered to be more compact and efficient than the 6-message versions in [32]. Similarly, Protocol 5(b) can be refined as Protocol 5.1, that is generated by ASPB.

6.6 Discussion and Evaluation

In this section, we evaluate ASPB and compare it with existing approaches, including, the Automatic Protocol Generator (APG) [100], and the Evolutionary approach [32].

Table 6.1: The time performance comparison between ASPB and APG

protocol purpose		ASPB		APG[a]
		Stage 1[b]	Stage 2[c]	
mutual authentication without TTP	signature keys	1.2 sec.	1.3 sec.	N/A
	public keys	1.2 sec.	1.3 sec.	23 sec.
	symmetric keys	1.2 sec.	1.3 sec.	10 sec.
mutual authentication with TTP	symmetric keys	3 sec.	4 sec.	10 min.
mutual authentication and key agreement	(4 messages)	15 sec.	20 sec.	2 hr.
	(5 messages)	25 sec.	80 sec.	N/A

[a] APG timing is based on generating the best protocol result running on a 400MHz Intel Pentium III [100].

[b] Time to synthesise, compose, and generate all candidate protocols running on a 1.8GHz Intel Pentium IV.

[c] Estimated time that the Athena [107] checker would take to further validate the ASPB generated candidate protocols.

Table 6.1 provides a time performance comparison between ASPB and APG [100]. The first three rows give the performance results for generating the mutual authentication protocol without TTP described in Sec 6.5 (and specified in Figures 6.6, 6.7, and 6.8). The fourth row gives the performance results for generating the mutual authentication protocol with TTP described in Figure 6.2. The other experiment was for mutual authentication and key agreement with TTP described in Figure 6.9. The fifth row gives the result for four message protocols. The last row gives the result for five message protocols. The results of similar experiments were reported in [100]. While ASPB was tested on a faster computer than APG, given the marked difference in speed, it is nevertheless reasonable to conclude that ASPB runs significantly faster than APG.

ASPB generates approximately 500 valid five-message candidate protocols in 25 seconds. However, on manual inspection, many of these protocols are similar, containing minor

textual and redundant variations. On manual inspection, we estimate that in this set there are 24 reasonably distinct four-message candidate protocols and 76 reasonably distinct five-message protocols. We selected several protocols from each category to demonstrate our results in Section 6.5.

APG has not been tested for five message protocols; in this case we conjecture that direct application of the forward search approach of APG would result in a very large and potentially infeasible search space. Perrig and Song's estimate [100] is based on the average number of messages that a principal can generate in a given round of the protocol and is explained as follows. In a given principal sequence four-message three-party authentication and key agreement protocol (as the requirement described in Figure 6.9 and, for example, the given principal sequence is $A \rightarrow B \rightarrow S \rightarrow B \rightarrow A$), A generates over 100 different messages to B, then B in turn generates about 500 different messages and sends them to S. S can generate 30,000 messages and sends to B. B can generate around 500 messages in the final round. The combination of this number of messages is on the order of 10^{12}. If these four-message protocols extend to five-message versions by one message that is sent from A to B, there are around 500 different messages generated by A when using Perrig and Song's estimation approach in [100]. We conjecture that the search space for a given principal sequence five-message three-party authentication and key agreement protocol is over 500×10^{12} The search space for all possible principal sequences is $500 \times 10^{12} \times 2^5$ (based on the limitation that one does not send messages to itself, and therefore there are 2 possible message receivers at each protocol step).

APG [100] generates the "best" protocol for each protocol requirement in the entire protocol search space with respect to its metric functions. The metric function of APG [100] gives each message operation a cost value. For example, the cost value of a message decryption/encryption is 3; the cost value of generating a nonce is 1. The cost value of a protocol is the total cost of all the protocol operations. The minimal cost protocol that meets all of the protocol goals is APG's "best" protocol. Compared with APG, the evolutionary approach [32] only searches a part of the entire protocol space. Its goal is to find a "good" solution for each protocol requirement in a given protocol search space with respect to its fitness function. Unlike APG, the evolutionary approach does not guarantee that the "best" protocol in the entire search space can be found in the selected part of the search space. On the other hand, because the evolutionary approach searches a smaller protocol search space than APG, it may find a "good" protocol in a shorter time than APG. For example, APG generates the "best" protocol for mutual authentication and key exchange protocol in two hours; Chen, Clark, and Jacob generate a "good" protocol for the same set of protocol goals only in 3 minutes using an evolutionary approach [32] based on the SVO logic.

Since variable instantiation in ASPB's single goal synthesiser disregards higher order

belief formulae, and it is possible to use these higher order belief formulae to generate some protocols that are different to protocols generated using the first order beliefs, then ASPB does not search the entire protocol space to generate all possible protocols that satisfy requirement specifications. When compared to the above approaches, ASPB generates a number of "good" protocols, including the "best" protocol, in a shorter period. For example, ASPB generates a number of four message mutual authentication and key exchange protocols for the same set of protocol goals and assumptions within 20 seconds. The reason for this is that the Single Goal Synthesiser uses the heuristics to direct its backwards search for valid protocols from a protocol goal. Unlike [35] and [100] which compose random messages, our strategy ensures that all candidate protocols obtained from the search tree are valid in that they uphold the goal within the logic. This contrasts with the forward searching approaches that may process many invalid candidate protocols before encountering a valid protocol.

Athena [107] is used to guarantee that APG generates correct protocols. However, fine grained distinctions between honesty and dishonesty cannot be made, since the underlying analysis model of APG, the strand space model, does not consider the concepts of honesty and dishonesty. Sometimes, an improper trust relationship may lead to some principals being cheated by others. With the BSW-ZF Logic, it is possible to distinguish honest (and competent) from dishonest (and incompetent) principals. This fine-grain distinction is useful for a protocol designer.

Since the Single Goal Synthesiser is completely independent of the Protocol Composer, our approach does not depend on the BSW-ZF logic. In future research, we could extend the logic (or change the basic logic to another) to suit more complex requirements.

Furthermore, before using Single Goal Synthesiser, the BSW-ZF inference rules could be applied on known assumptions to enrich the set of known assumptions. This strategy is a common strategy in prolog searching engines. It may reduce the searching space for generating formula trees.

A LCF-style foreward check can be applied on generated candidate protocols. It checks whether the protocol goals can be generated from a candidate protocol and all related assumptions. This may disgard protocols that may not meet all protocol goals after sub-protocol merging.

Finally, increased performance could be achieved by parallelising the single goal synthesis and composition steps across separate processors.

Chapter 7

Authorisation Subterfuge

Many commercial access control systems are closed and rely on centralised authorisation policy/servers. An access control decision corresponds to determining whether some authenticated user has been authorised for the requested operation. This strategy of first determining who the user is and then whether that user is authorised has its critics, citing, for instance, single point of failure, scalability issues and excessive administrative overhead. An, perhaps overlooked, advantage of this approach is that administrators exercise tight control when granting access. The administrators are familiar with all of the resources that are available and they make sure that the user gets the appropriate permissions – no more and no less. The opportunity to subvert the intentions of a good administrator is usually small.

Cryptographic authorisation certificates bind authorisations to public keys and facilitate a decentralised approach to access control in open systems. Trust Management [62, 36, 43, 73, 19, 38] is an approach to constructing and interpreting the trust relationships among public-keys that are used to mediate access control. Authorisation certificates are used to specify delegation of authorisation among public keys. Determining authorisation in these systems typically involves determining whether the available certificates can prove that the key that signed a request is authorised for the requested action.

However, these approaches do not consider how the authorisation was obtained. They do not consider whether a principal can somehow bypass the intent of a complex series of authorisation delegations via some unexpected circuitous but authorised route. In an open system no individual has a complete picture of all the resources and services that are available. Unlike the administrator of the closed system, the principals of an open system are often ordinary users and are open to confusion and subterfuge when interacting with resources and services. These users may inadvertently delegate un-intended authorisation to recipients.

In this chapter, we explore the problem of *authorisation subterfuge*, whereby, in a poorly designed authorisation system, delegation chains that are used by principals to prove authorisation may not actually reflect the original intention of all of the participants in the chain. For example, the intermediate principals of a delegation chain may inadvertently issue incorrect certificates, when the intended resource owner is unclear to intermediate participants in the chain. We argue that subterfuge is a realistic problem that should be addressed in a certificate scheme for payment systems. For example, the Trust Management payment systems [22, 23, 52] are also vulnerable to authorisation subterfuge (leading to a breakdown in authorisation accountability) if care is not taken to properly identify the 'permissions' indicating the payment authorisations when multiple banks and/or provisioning agents are possible. The micro-billing scheme [23] uses KeyNote to help determine whether a micro-check (a KeyNote credential, signed by a customer) should be trusted and accepted as payment by a merchant. The originator of the chain is the provisioning agent, who is effectively responsible for ensuring that the transaction is paid for. In [52], delegation credentials are used to manage the transfer of micropayment contracts between public keys; delegation chains provide evidence of contract transfer and ensure accountability for double-spending.

The chapter is organised as follows. In Section 7.1 we describe a series of subterfuge attacks that can be carried out on certificate chains. Section 7.2 discusses the underlying problem of authorisation subterfuge. Section 7.3 illustrates how subterfuge can also arise in local naming. Section 7.4 explores similarities between these attacks on certificates and replay attacks on authentication protocols. Analysing a collection of certificates for potential subterfuge is not unlike checking whether it is possible for an 'intruder' to interfere with a certificate chain. Section 7.5 proposes the Subterfuge logic which can be used to determine whether performing a delegation operation might leave the delegator open to subterfuge. Examples from Section 7.1 are analysed in Section 7.6. Finally, we conclude in Section 7.7.

7.1 Authorisation Subterfuge in SPKI/SDSI

SPKI/SDSI [43] relies on the cryptographic argument that a public key provides a globally unique identifier that can be used to refer to its owner in some way. However, public keys are not particularly meaningful to users and, therefore, SPKI/SDSI provides local names which provide a consistent scheme for naming keys relative to one another. For example, the local name that *Alice* uses for *Bob* is (Alice's Verisign's Bob), which refers to *Bob*'s public key as certified by the Versign that *Alice* knows. By binding local names to public keys with name certificates, principals may delegate their authorisation to others beyond their locality through a chain of local relationships.

A SPKI/SDSI name certificate is denoted as (K, A, S), where: K specifies the certificate issuer's signature key, and identifier A is defined as the local name for the subject S. For example, $(K_B, Alice, K_A)$ indicates that K_B refers to K_A using the local name *Alice*. A SPKI/SDSI authorisation certificate is denoted as (K, S, d, T), where: K specifies the certificate issuer's signature key; tag T is the authorisation delegated to subject S (by K) and d is the delegation bit (0/1). If the delegation bit is 1, the subject of this certificate is allowed to re-delegate tag T to others. If it is 0, the subject can not re-delegate T to others. For example, K_B delegates authorisation T to *Alice* by signing $(K_B, Alice, 0, T)$, where 0 indicates no further delegation. Note that for the sake of simplicity we do not consider the SPKI/SDSI validity period. We assume that all credentials are within their validity periods.

Authorisation tags are specified as s-expressions and Example 2.6 in [44] specifies tag T1= tag (purchase(*range le <amount>),(*set <<items>>)) to mean that it

> *"[...] might indicate permission to issue a purchase order. The amount of the purchase order is limited by the second element of the (purchase) S-expression and, optionally, a list of purchasable items is given as the third element. The company whose purchase orders are permitted to be signed here will appear in the certificate permission chain leading to the final purchase order. Specifically, that company's key will be the issuer at the head of the (purchase). [...]" [44]*

Given two authorisation certificates $(K_1, S_1, 1, T_1)$ and (S_1, S_2, d, T_2), if the subject of the first certificate is the issuer of the second certificate, and the delegation bit of the first certificate is 1, then SPKI/SDSI reduction rules conclude $(K_1, S_2, d, (T_1 \cap T_2))$, whereby $T_1 \cap T_2$ means that if $T_1 \subseteq T_2$ (or $T_2 \subseteq T_1$), then T_1 (or T_2) is the result of this operation.

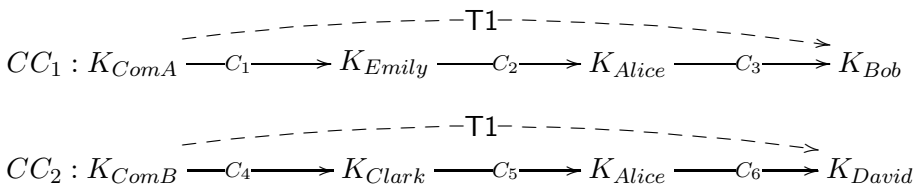
7.1.1 Authorisation Examples

A company *ComA* allows its manager *Emily* to issue purchase orders, and *Emily* may also delegate this right to others. After *Emily* receives the certificate from *ComA*, *Emily* delegates this right (issuing a purchase order) to an employee *Bob* via *Alice*. We have the following certificates:

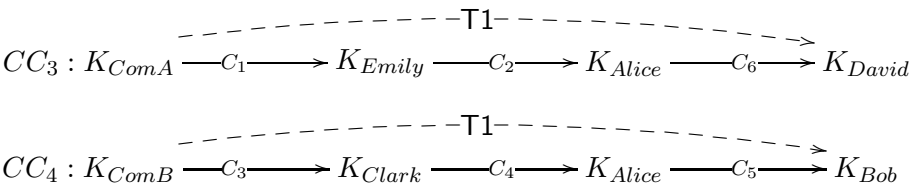
C_1=(K_{ComA}, K_{Emily}, 1, T1);
C_2=(K_{Emily}, K_{Alice}, 1, T1); and
C_3=(K_{Alice}, K_{Bob}, 0, T1)

(*Alice* delegates this right to employee *Bob*, But *Bob* may not delegate this right to others).

Suppose that there is another company *ComB* which also uses the tag T1 to issue purchase orders. Suppose that *Alice* also works for *ComB*. *Clark*, a senior manager in



(a) Expected Certificate Chain CC_1 and CC_2



(b) Unexpected Certificate Chain CC_3 and CC_4

$K_1 \dashrightarrow K_2$ means that K_1 delegates T to K_2, and
$K_1 \xrightarrow{C} K_2$ means that K_1 sends certificate C to K_2.

Figure 7.1: Certificate Chains in SPKI/SDSI Example

ComB, holds the right to issue purchase orders, and delegates it to *Alice*. *ComB* employee *David* accepts authority from *Alice* to issue purchase orders. We have certificates:

C_4=(K_{ComB}, K_{Clark}, 1, T1);
C_5=(K_{Clark}, K_{Alice}, 1, T1); and
C_6=(K_{Alice}, K_{David}, 0, T1).

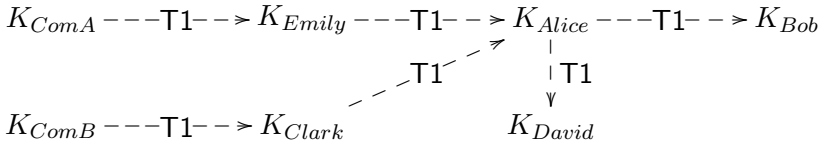
Figure 7.1(a) gives the certificate chain CC_1 and CC_2 that *Bob* and *David* respectively use to prove authorisation to issue purchase orders.

7.1.2 Authorisation Subterfuge Examples

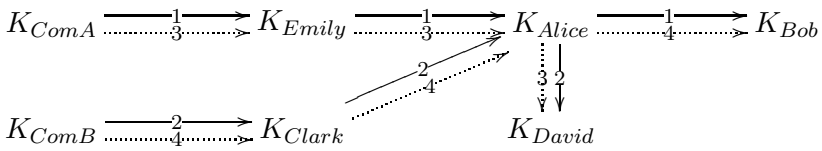
The examples above are effective when separate chains CC_1 and CC_2 are used to prove authorisation. However, their combination, depicted in Figure 7.2(a), result in further delegation chains CC_3 and CC_4 depicted in Figure 7.1(b) and these lead to some surprising interpretations of how authorisation is acquired [122].

Subterfuge 1: passive attack. In Figure 7.3, *Alice*'s intention, when she signed C_6, was that *David* should use chain CC_2 as proof of authorisation when making purchases. However, unknown to *Alice*, dishonest *David* collects all other certificates and uses the chain CC_3 as his proof of authorisation.

This confusion may introduce problems if the certificate chains that are used to prove



(a) Delegation Graph for T1



(b) Delegation Paths for T1

Figure 7.2: Delegations in SPKI/SDSI Example

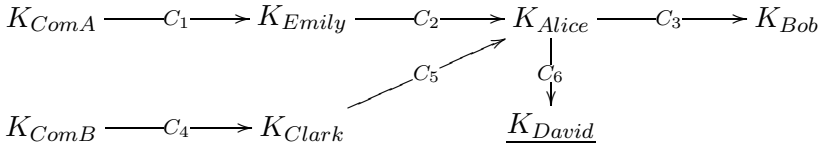


Figure 7.3: Attack Graphs: Passive Attack

authorisation are also used to provide evidence of who should be billed for the transaction. In delegating, *Alice* believes that chain CC_2 (from $ComB$) provides the appropriate accountability for $Clark$'s authorisation.

Subterfuge 2: outer-active attack. The above passive attack can be transformed into a more active attack. In Figure 7.4, *David* sets up a shelf company $ComB$ with fictitious employee $Clark$. Using attractive benefits, *David* masquerading as $Clark$, lures *Alice* to join $ComB$. $Clark$ delegates authorisations (T1) to *Alice* that correspond to authorisation already held by *Alice*. However, *Alice* does not realise this and, in the confusion, further delegates the authorisation to *David*; an authorisation from $ComA$ that normally he would not be expected to hold.

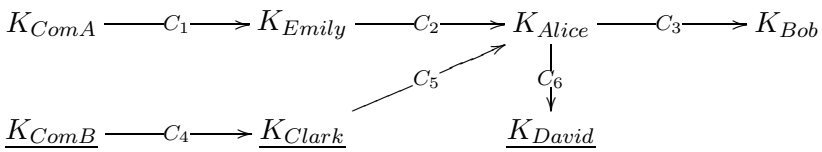


Figure 7.4: Attack Graphs: Outer-Active Attack

In both of these cases we think of *Alice* as more *confused* in her delegation actions rather than incompetent; the permission naming scheme influences her local beliefs and it was the inadequacy of this scheme that led to her confusion. Perhaps *Alice* has too many certificates to manage and in the confusion loses track of which permissions should be associated with which keys.

$ComA$ may attack $ComB$ in the same way to get the money back by CC_4. However, if $ComB$ updates its certificate, then *Alice* does not hold the right for $ComB$. $ComA$ cannot get its money back.

Subterfuge 3: inner-active attack. In Figure 7.5, $Clark$ is a manager in $ComA$ and $ComB$ and colludes with *David* ($ComB$ employee). $Clark$ delegates authorisation T1 legitimately obtained from $ComB$ to *Alice*. However, suppose that unknown to *Alice*, $Clark$ is coincidentally authorised to do T1 by $ComA$ (via C_7) and $Clark$ intercepts the issuing of credential C_1 and conceals it. *Alice* delegates what she believes to be T1 from $ComB$ to *David* via C_6. However, *David* can present chain $[C_7; C_5; C_6]$ as proof that his authorisation originated from $ComA$.

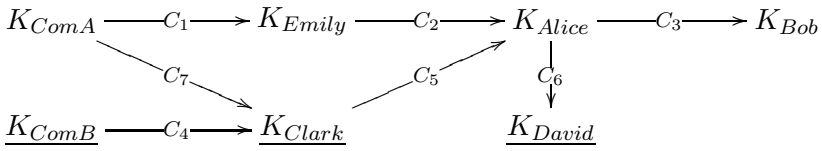


Figure 7.5: Attack Graphs: Inner-Active attack

The above authorisation subterfuge may be avoided if *Alice* is very careful about how she delegates. However the following attacks are a bit more difficult for *Alice* to avoid.

Subterfuge 4: (outer-intercept attack) In Figure 7.6, $Clark$ intercepts certificate C_2 and conceals it. When delegating authorisation to *David*, *Alice* believes that the chain is $[C_4; C_5; C_6]$ from $ComB$, however *David* knowingly or unknowingly uses a different chain $[C_1; C_2; C_6]$.

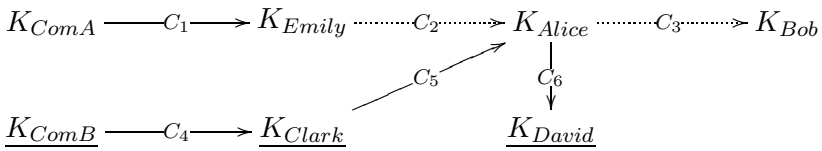


Figure 7.6: Attack Graphs: Outer-Intercept attack

Subterfuge 5: (inner-outer active attack). In Figure 7.7, *Alice* has a legitimate expectation that so long as she delegates competently then she should not be liable for

any confusion that is a result of poor system/permission design. *Alice* can use this view to act dishonestly. In signing a certificate she can always deny knowledge of the existence of other certificates and the inadequacy of permission naming in order to avoid accountability. While *Alice* secretly owns company $ComB$, she claims that he cannot be held accountable for the 'confusion' when *Bob* (an employee of $ComA$) uses the delegation chain $[C_4; C_5; C_3]$ to place an order for *Alice*.

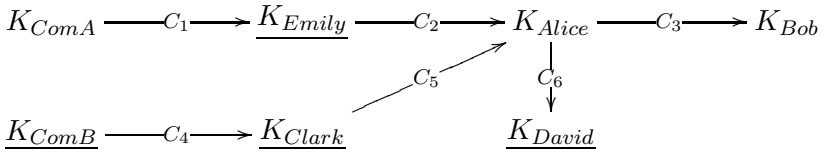


Figure 7.7: Attack Graphs: Inner-Outer Active Attack

The above subterfuge examples demonstrate that it is impossible to distinguish malicious principals from regular principals in some scenarios. For example, $ComA$ may consider *Alice* as a malicious principal, since *Alice* improperly delegates T1 to an outsider *David*. However, *Alice* is an innocent principal in all of the above subterfuge examples. $ComB$ and $Clark$ are malicious principals in Subterfuge 2 – 5, but they may claim that they are innocent principals as in Subterfuge 1. *Emily* can be a malicious principal as in Subterfuge 5, but she may claim that she is innocent as in Subterfuge 1 – 4. One may consider that *David*, the principal who improperly uses certificates from $ComA$, is implicitly a malicious principal.

7.2 Avoiding Subterfuge: Accountability for Authorisation

The underlying problem with the examples in the previous section is that the permission T1 is not sufficiently precise to permit *Alice* to distinguish the authorisations that are issued by different principals. This is not a problem in closed systems. The reason for this is that the security administrator of a closed system defines every permission within the system and delegates them based on a complete view of the system. Table 7.1 gives the comparison between closed and open systems.

An ad-hoc strategy to avoid this problem would be to ensure that each permission is sufficiently detailed to avoid any ambiguity in the sense that it is clear from whom the authorisation originated. This provides a form of accountability for the authorisation. For example, including a company name as part of the permission may help avoid the vulnerabilities in the particular example above.

	Closed System	Open System
User origin	same domain	different domains
Does a name have a unique and useful meaning?	Yes	No
Someone knows all the names?	Security Administrator	No one
Delegation decisions based on	complete information within the domain	incomplete view of the world

Table 7.1: A Comparison between Open System and Closed System

However, at what point can a principal be absolutely sure that an ad-hoc reference to a permission is sufficiently complete? Achieving this requires an ability to be able to fix a permission within a global context, that is, to have some form of global identifier and/or reference for the permission.

A possible source of suitable identifiers is a global X500-style naming service (if it could be built) that would tie global identities to real world entities, which would in turn be used within permissions. However, X500-style naming approaches suffer from a variety of practical problems [45] when used to keep track of the identities of principals. On the other hand, in the context of subterfuge, a principal might easily be confused between the (non-unique) common name and the global distinguished name contained within a permission that used such identifiers. For example, permission T2 [44] might use some form of global reference, which may result in a subterfuge problem, as outlined in the above scenarios. Example 2.5 in [44] specifies tag T2 = tag (spend ⟨bank⟩⟨account⟩ (* range le ⟨amount⟩)) to indicate that

> *"[...] the subject has authority to authorise spending up to ⟨amount⟩ per electronic check from ⟨account⟩ at ⟨bank⟩. [...]" [44]*

Permission T2 gives only a bank name and a sequence number as the original authority's account number. Suppose that T2 specifies $ComA$'s bank account in the scenario in the previous section. When *Alice* receives permission T2, she may not be able to recognise whether $ComB$ is a legitimate authority to delegate the T2. If *Alice* makes further delegation decisions based on its visible permission delegator $ComB$ and delegates T2 to principals that are members of $ComB$, *Alice* is misled and unintentionally delegates T2 out of $ComA$.

When a subject cannot associate given web addresses with their authorities, the following permissions T3 – T5 have the same problem as T2. Tag Example 2.1 in [44] specifies tag T3 = tag (ftp cybercash.com cme) to indicate that

> *"[...] This tag indicates that the subject has permission to do FTP into host cybercash.com as user cme. [...]" [44]*

Example 2.2 in [44] uses tag T4 = tag (http http://acme.com/company-private/personnel) to indicate that

> *"[...]This tag gives the Subject permission to access the web page at the given URI. To give permission for an entire tree under a given URI, one might use:"*
> T5 = tag (http (* prefix http://acme.com/company-private/personnel/)) *[44]*

From above discussion, we can see that schemes with unregulated names cannot guarantee global uniqueness of a permission, and schemes that regulate global names (e.g., X509) cannot guarantee proper use of a globally unique permission.

Authorisation subterfuge is possible when one cannot precisely account for how an authorisation is held. In signing a certificate, we assume that the signer is in some way willing to account for the authorisation that they are delegating. The authorisation provided by a certificate chain that is not vulnerable to subterfuge can be accounted for by each signer in the chain. A principal who is concerned about subterfuge will want to check that the permission that is about to be delegated can also be accounted for by others earlier in the chain: the accountability 'buck' should preferably stop at the head of the chain!

SPKI [43] characterises the checking of authorisation as *"is principal X authorised to do Y?"*. However, the examples above illustrate that this is not sufficient; checking *"is principal X authorised to do Y by Y's owner Z?"* would be more appropriate.

Public keys provide globally unique identifiers that are tied to the owner of the key. These can also be used to avoid permission ambiguity within delegation chains. For example, given SPKI authorisation certificate $(\mathsf{K}_{ComA}, \mathsf{K}_E, 1, [\mathsf{T1}.\mathsf{K}_{ComA}])$, there can be no possibility of subterfuge when *Emily* delegates to *Alice* with $(\mathsf{K}_E, \mathsf{K}_A, 1, [\mathsf{T1}.\mathsf{K}_{ComA}])$. In this case the authorisation $[\mathsf{T1}.\mathsf{K}_{ComA}]$ is globally unique and the certificate makes the intention of the delegation and where it came from (authorisation accountability) very clear.

Needless to say that this strategy does assume a high degree of competence on *Alice*'s part to be able to properly distinguish between permissions $[\mathsf{T1}.\mathsf{K}_{ComA}]$ and $[\mathsf{T1}.\mathsf{K}_{ComB}]$, where, for example, each public key could be 342 characters long (using a common ASCII encoding for a 2048 bit RSA key). One might be tempted to use SDSI-like local names to make this task more manageable for *Alice*. However, in order to prevent subterfuge, permissions require a name that is unique across all name spaces where it will be used, not just the local name space of *Alice*. In *Alice*'s local name space the permission [T1.*(Emily's ComA)*] may refer to a different *ComA* to the *ComA* that *Alice* knows. Subterfuge in name certificates will be discussed in next section.

Existing Trust Management approaches such as [73] avoid subterfuge by assuming that all certificates are correctly in place, well understood by principals, and may not be improperly used. However, we argue that subterfuge is a realistic problem that should be addressed in a certificate scheme. For example, the Trust Management payment systems [22, 23, 52] are also vulnerable to authorisation subterfuge (leading to a breakdown in authorisation accountability) if care is not taken to properly identify the 'permissions' indicating the payment authorisations when multiple banks and/or provisioning agents are possible. The micro-billing scheme [23] uses KeyNote to help determine whether a micro-check (a KeyNote credential, signed by a customer) should be trusted and accepted as payment by a merchant. The originator of the chain is the provisioning agent, who is effectively responsible for ensuring that the transaction is paid for. In [52], delegation credentials are used to manage the transfer of micropayment contracts between public keys; delegation chains provide evidence of contract transfer and ensure accountability for double-spending.

7.3 Subterfuge in Name Certificates

Authorisation subterfuge is also possible when using SPKI/SDSI local name certificates. For example, Ellison and Dohrmann [42] describe a model based on SPKI/SDSI name certificates for access control in mobile computing platforms. A group leader controls all rights of a group. A group leader may delegate the right of "admitting members" to other principals. For example, K_G is a group leader; K_G admits K_A as its group member by certificate C_1. K_G defines a large random number n, which will be used as K_A's local name for K_G's membership. Then, K_G issues certificate C_2 to K_A which means that if K_A accepts a principal as (K_A's n), then the principal also becomes K_G's group G's member. K_A admits K_B as K_A's n by C_3. Together with C_2, K_B also becomes a member of K_G's G as presented in C_4. The certificates are as follows.

$$C_1 = (K_G, G, K_A);$$
$$C_2 = (K_G, G, (K_A\text{'s } n)); \text{ and}$$
$$C_3 = (K_A, n, K_B)$$

From these we can deduce (K_G, G, K_B), that is, K_B is now a member of group G.

The scheme works in a decentralised manner and thus no single member will hold the entire membership list. This means that there is no easy way to prove non-membership. The strategy described in the paper is sufficiently robust as it relies on face-to-face verification of certificate C_2 when a member joins.

However, the nonce is large and there may be potential for confusion during the face-to-face verification and this can lead to subterfuge. Consider the following certificates.

$$C'_1 = (K_I,\, G_I,\, K_A);$$
$$C'_2 = (K_I,\, G_I,\, (K_A\text{'s } n));$$
$$C'_3 = (K_A,\, n,\, K_I)$$

Suppose that the intruder K_I wants to join K_G's group G. K_I intercepts C_2 and issues C'_2 by using the number in C_2. In the confusion, K_A issues C'_3 which corresponds to admitting K_C (which the intruder controls) as a member of K_I's G_I for K_A. In this case, K_C may use C_2 and C'_3 to prove its membership in K_G's group G.

7.4 Subterfuge in Satan's Computer

We are interested in determining whether, given a collection of known certificates, it is safe for a principal to delegate some held authorisation to another principal. By safe we mean that subterfuge is not possible. In simple terms, this requires determining if it is possible for a malicious outsider to interfere with a certificate chain with a view to influencing the authorisation accountability. In order to help understand this we draw comparisons between subterfuge attacks and attacks on authentication protocols. Our hypothesis is that techniques for analysing one can be used to analyse the other (as we shall see in the next section when we use a BAN-like logic to analyse subterfuge in delegation chains).

A certificate is a signed message that is exchanged between principals; an authentication protocol step can be an encrypted message that is exchanged between principals. A certificate chain is an ordering of certificates exchanged between principals. An authentication protocol is an ordering of encrypted messages exchanged between principals. For example, the chain CC_1 could be represented by the following protocol.

$$\begin{array}{lcl} msg1 & ComA \rightarrow E & : \{K_{ComA}, K_E, 1, T1\}_{K_{ComA}} \\ msg2 & E \rightarrow A & : \{K_E, K_A, 1, T1\}_{K_E} \\ msg3 & A \rightarrow B & : \{K_A, K_B, 0, T1\}_{K_A} \end{array}$$

There are differences between authentication protocols and certificate chains. A round of a typical authentication protocol has a fixed and small number of pre-defined messages, while the number of participants and messages in a certificate chain are unlimited and, sometimes, it may not be predetermined.

An attack from Section 7.1.2 is represented as follows.

$$\begin{array}{lcl} msg2'. & I(CA) \rightarrow A & : \{K_I, K_A, 1, T1\}_{K_I} \\ msg3'. & A \rightarrow D & : \{K_A, K_D, 0, T1\}_{K_A} \end{array}$$

Subterfuge attacks involve a malicious user (the intruder I) removing/hiding and replaying certificates between different certificate chains. These actions are comparable to a combination of the replay attacks [34]:

Freshness attack "When a message (or message component) from a previous run of a protocol is recorded by an intruder and replayed as a message component in the current run of the protocol."

Parallel session attack "When two or more protocol runs are executed concurrently and messages from one are used to form messages in another."

The analysis of an authentication protocol typically centres around an analysis of nonce properties: *if one correctly responds to the nonce challenge in a round of an authentication protocol, it is the regular responder.*

Freshness A nonce is a number used once in a message. Message freshness fixes a message as unique and ties it to a particular protocol run.

Relevancy to originator A nonce is related to its originator. The nonce verifier is also the nonce provider (originator). The nonce originator generates the nonce and this means that it can recognise and understand its relationship with the nonce.

Relevance of message In a two-party mutual authentication protocol, each principal generates its own nonce. A principal uses its own nonce and the other principal's nonce to relate its own message to the other's message.

There are some similarities between these nonce properties and the permission properties that rely on unique permissions.

Uniqueness is required in a permission string to account for its originator within a particular certificate chain.

Relevancy to originator A permission should be related to its originator and it should be possible for others along the chain to recognise this relationship.

Relevance of certificates Certificates can be used to delegate combinations of permissions that originated from different sources. These new certificates should account for the authorisation of the originators.

From the above, we may see that a certificate chain is similar to a security protocol using signature keys. An authorisation certificate chain is a sequence of ordered certificates that are exchanged between principals for delegating rights.

Similar to the definition of the correctness of authentication [78]: *A protocol guarantees agreement with a participant B (say, as the responder) for certain data items x if: each time a principal B completes a run of the protocol as responder using x, which to B appears to be a run with A, then there is a unique run of the protocol with the principal A as initiator using x, which to A appears to be a run with B.*

We define the accountability of authorisation as follows.

DEFINITION **7.4.1** *A certificate chain guarantees agreement to a participant B (say, as the delegatee) for certain right R if: each time a principal B is delegated a right R, which to B appears to be a certificate chain with A, then there is a unique certificate chain with the principal A as initial delegator authorising R.*

We use a BAN-style logic to reason about this notion of accountability of authorisation.

7.5 A Logic for Analysing Certificate Chains

The previous section demonstrated similarities between (freshness) vulnerabilities in authentication protocols and (subterfuge) vulnerabilities in delegation chains. In this section we develop the Subterfuge Logic (SL) which draws on some of the techniques from BAN-like logics to analyse subterfuge in certificate chains.

7.5.1 The language

The logic uses the following basic formulae. P, Q,R and S range over principals; X represents a message, which can be data or formulae or both; ϕ will be used to denote a formula. The basic formulae are the following:

- $\sharp(X)$: Formula X is a globally unique identifier. For example, this is typically taken as true for X.500 distinguished names and for public keys.
- $X \mid P$: represents the message X, as guaranteed/accounted for by principal P; this means that P is willing to be held accountable for the consequences of action X. For example, it is in Alice's interest to delegate $\mathsf{T1} \mid K_{ComA}$ to Bob, as opposed to just $\mathsf{T1}$.
- $X \rightsquigarrow P$: Principal P is an originator of formula X. In the examples above, we write $\mathsf{T1}|K_{ComA}$ to mean that permission $\mathsf{T1}$ was first uttered by K_{ComA} in some chain. Note that we assume that the same global unique formula (permission) cannot originate from two different principals, that is, if $X \rightsquigarrow P$, $X \rightsquigarrow Q$ and $\sharp(X)$ then $P = Q$.

- $P \ni X$: P is authorised for the action X.
- $P \succ X$: P is authorised to delegate X to others.
- $P \,|\!\approx X$: P directly says X. This represents a credential that is directly exchanged between principals.
- $P \,|\!\sim X$: P says X. P directly says X or others say X (who have been delegated to speak on X by P).

Further formulae can be derived by using propositional logic. If ϕ_1 and ϕ_2 are formulae, then $\phi_1 \wedge \phi_2$ (ϕ_1 and ϕ_2), $\phi_1 \vee \phi_2$ (ϕ_1 or ϕ_2), and $\phi_1 \rightarrow \phi_2$ are formulae.

SPKI/SDSI credentials can be encoded within the logic as follows. An authorisation credential $(\mathsf{K}, \mathsf{S}, 0, \mathsf{T})$ is represented as $\mathsf{K} \,|\!\approx (\mathsf{S} \ni \mathsf{T})$, and credential $(\mathsf{K}, \mathsf{S}, 1, \mathsf{T})$ represented as $\mathsf{K} \,|\!\approx (\mathsf{S} \ni \mathsf{T} \wedge \mathsf{S} \succ \mathsf{T})$. The purpose of the logic is to permit a principal decide whether it would be safe for it to delegate an authorisation based on the collection of credentials that it currently holds. For the examples above, Alice would like to be able to test whether it is safe for her to write a credential corresponding to $K_{Alice} \,|\!\approx (K_{David} \ni \mathsf{T1})$. That is, she wishes that someone further back on the chain will accept accountability for the action, that is, $K_{Alice} \succ \mathsf{T1} | K_{ComA}$ can be deduced (which is not possible for the examples in Section 7.1.1). Note that in signing the credential, Alice is also accepting accountability for the authorisation.

7.5.2 Inference rules

Gaining Rules

G1 If P holds authorisation for X, for which Q can be held accountable, and Q may delegate X then P is also authorised for X.

$$\frac{P \ni X \mid Q, Q \succ X}{P \ni X}$$

G2 We have a similar rule for authorisation to delegate.

$$\frac{P \succ X \mid Q, Q \succ X}{P \succ X}$$

Direct delegation

D1 Direct delegation of authority assumes that the delegator accepts accountability for the action.

$$\frac{P \,|\!\approx (Q \ni X)}{P \,|\!\sim (Q \ni X \mid P), Q \ni X \mid P}$$

D2 We have a similar rule for authorisation to delegate.

$$\frac{P \mid\!\approx (Q \succ X)}{P \mid\!\sim (Q \succ X \mid P), Q \succ X \mid P}$$

D3 The usual conjunction rules apply.

$$\frac{P \mid\!\approx (\phi_1 \wedge \phi_2)}{P \mid\!\approx \phi_1, P \mid\!\approx \phi_2}$$

Indirect delegation

I1 If principal P says that Q is authorised to perform an action X (with R accountable), and P is authorised to delegate X (with R accountable), then Q is authorised to perform X (with R accountable).

$$\frac{P \mid\!\sim (Q \ni X \mid R), P \succ X \mid R}{Q \ni X \mid R}$$

I2 We have a similar rule for authorisation to delegate.

$$\frac{P \mid\!\sim (Q \succ X \mid R), P \succ X \mid R}{Q \succ X \mid R}$$

I3 If principal P says that Q is authorised to perform action X by P, then P says that Q is authorised to perform X.

$$\frac{P \mid\!\sim (Q \ni X \mid P)}{P \mid\!\sim (Q \ni X)}$$

I4 Accountability can be stripped from an authorisation. Note, however, that stripping accountability does not refute the existence of the accountability.

$$\frac{P \mid\!\sim (Q \succ X \mid P)}{P \mid\!\sim (Q \succ X)}$$

I5 Accountability is transitive along certificate chains.

$$\frac{R \mid\!\sim (P \succ X \mid S), P \mid\!\sim (Q \succ X \mid R)}{R \mid\!\sim (Q \succ X \mid S)}$$

I6 We have a similar rule for authorisation.

$$\frac{P \mid\!\sim (Q \ni X \mid R), R \mid\!\sim (P \succ X \mid S)}{R \mid\!\sim (Q \ni X \mid S)}$$

Unique Origin Rules

U1 If Q is authorised for unique X that originated from P then P can be held accountable for X.

$$\frac{\sharp(X), X \rightsquigarrow P, Q \ni X}{Q \ni X \mid P}$$

U2 We have a similar rule for authorisation to delegate.

$$\frac{\sharp(X), X \rightsquigarrow P, Q \succ X}{Q \succ X \mid P}$$

7.6 Analysing Authorisation Subterfuge

The examples from Section 7.1 and Section 7.3 are analysed using the Subterfuge Logic as follows.

Example 14 Certificates C_1 and C_2 in the example from Section 7.1 are encoded by the following formulae. Note that principal names are abbreviated to their first initial if no ambiguity can arise.

$$\begin{aligned} K_{ComA} &\mid\!\approx ((K_E \ni \mathsf{T1}) \wedge (K_E \succ \mathsf{T1})) \\ K_E &\mid\!\approx ((K_A \ni \mathsf{T1}) \wedge (K_A \succ \mathsf{T1})) \end{aligned}$$

Assumptions regarding uniqueness include the following.

$$\sharp(K_{ComA}), \sharp(K_{ComB}), \sharp(K_A), \sharp(K_B), \sharp(K_C), \sharp(K_E)$$

Principal $ComA$ is assumed authorised to delegate and accept accountability for the authorisation $\mathsf{T1}$ that it originates.

$$K_{ComA} \succ (\mathsf{T1} \mid K_{ComA})$$

Before delegating authority for $\mathsf{T1}$ to *Bob*, *Alice* wishes to test whether it is safe to do so. *Alice* tests whether $ComA$ accepts accountability for this action, that is she attempts to deduce $K_A \succ \mathsf{T1} \mid K_{ComA}$ using the above assumptions within the logic. This is not possible since no assumption is made regarding uniqueness of $\mathsf{T1}$, and, therefore, we cannot deduce $K_E \mid\!\sim (K_A \succ T_1 \mid K_{ComA})$; thus *Alice* refrains from the delegation. △

In Trust Management public keys provide globally unique identifiers that are tied to the owner of the key. These can also be used to avoid authorisation ambiguity within delegation chains. For example, given SPKI certificate $(\mathsf{K}_{ComA}, \mathsf{K}_E, 1, [\mathsf{T1}.\mathsf{K}_{ComA}])$, there

can be no possibility of subterfuge when Emily delegates to Alice by signing the certificate $(\mathsf{K}_E, \mathsf{K}_A, 1, [\mathsf{T1}.\mathsf{K}_{ComA}])$. In this case the authorisation $[\mathsf{T1}.\mathsf{K}_{ComA}]$ is globally unique, that is $\sharp(\mathsf{T1}|\mathsf{K}_{ComA})$ and the certificate makes the intention of the delegation and accountability very clear.

The revised certificates are represented in the logic as follows.

$$\begin{aligned} K_{ComA} &\mid\!\approx ((K_E \ni T_1 \mid K_{ComA}) \wedge (K_E \succ T_1 \mid K_{ComA})) \\ K_E &\mid\!\approx ((K_A \ni T_1 \mid K_{ComA}) \wedge (K_A \succ T_1 \mid K_{ComA})) \end{aligned}$$

Given these certificates then Alice can deduce

$$K_A \succ \mathsf{T1} \mid K_{ComA}$$

and can safely delegate to Bob as

$$K_A \mid\!\approx (K_B \ni T_1 \mid K_{ComA})$$

and we can deduce that $K_B \ni T_1 \mid K_{ComA}$. Considering other certificates, including

$$\begin{aligned} K_{ComB} &\mid\!\approx ((K_C \ni T_1 \mid K_{ComB}) \wedge (K_C \succ T_1 \mid K_{ComB})) \\ K_C &\mid\!\approx ((K_A \ni T_1 \mid K_{ComB}) \wedge (K_A \succ T_1 \mid K_{ComB})) \\ K_A &\mid\!\approx (K_D \ni T_1 \mid K_{ComB}) \end{aligned}$$

we can deduce $K_D \ni T_1 \mid K_{ComB}$, the expected authorisation.

Suppose that $ComB$ issues confusing certificates to Clark, who in turn delegates the incorrect authorisation to Alice.

$$\begin{aligned} K_{ComB} &\mid\!\approx ((K_C \ni T_1 \mid K_{ComA}) \wedge (K_C \succ T_1 \mid K_{ComA})) \\ K_C &\mid\!\approx ((K_A \ni T_1 \mid K_{ComA}) \wedge (K_A \succ T_1 \mid K_{ComA})) \end{aligned}$$

In this case we can deduce $K_{ComB} \mid\!\sim (K_A \succ T_1 \mid K_{ComA})$ and thus and $K_A \succ T_1 \mid K_{ComB}$. However, before A delegates this right for K_{ComA}, she needs (but cannot hold) the following formulae $K_{ComB} \succ T_1 \mid K_{ComA}$, or $K_C \succ T_1 \mid K_{ComA}$. Thus, she should not delegate and therefore resists the subterfuge attack.

The conventional SPKI/SDSI authorisation certificate reduction rule can be described as

$$P \mid\!\approx (Q \succ X) \wedge Q \mid\!\sim (R \succ X) \implies P \mid\!\sim (R \succ X)$$

in the SL logic (with a similar relationship for delegation of authorisation). Such a relationship does not facilitate the tracking of accountability during certificate reduction.

Example 15 Consider the SPKI/SDSI name certificate example in Section 7.3. The group leader K_G issues certificate C_2:(K_G, G, (K_A's n)) that allows K_A to admit group members. If we consider that a local name is a permission without parameter, certificate C_2 can be encoded by the following formulae.

$$\begin{aligned} K_G &\ |\!\approx\ (K_A \succ \mathsf{G} \mid K_G) \\ K_G &\ |\!\approx\ (K_A \succ \mathsf{n} \mid K_A) \end{aligned}$$

Ellison and Dohrmann assume that n is a large random number that is generated by K_G. When n is large enough, it can be assumed to be unique within K_A's local name space. As a result, K_A's n is a global unique permission (name). On the other hand, K_G is the originator of $\mathsf{n} \mid K_A$, because n is generated by K_G. When K_A's n is encoded by the SL formula $\mathsf{n} \mid K_A$, the following assumptions are assumed by Ellison and Dohrmann.

$$\sharp(\mathsf{n} \mid K_A), \mathsf{n} \mid K_A \rightsquigarrow K_G$$

However, $\mathsf{n} \mid K_A \rightsquigarrow K_G$ is not held. The reason for this is that n is only a random number that may not provide any meaningful content that K_G is relevant. On the other hand, it is safe to assume that K_A (not K_G) is explicitly accountable for $\mathsf{n} \mid K_A$, since $\mathsf{n} \mid K_A$ is a local permission of principal K_A. In order to effectively delegate $\mathsf{n} \mid K_A$ to any other principal, such as K_A, K_G must be able to prove that he is accountable for $\mathsf{n} \mid K_A$. It requires that K_G is authorised to delegate $\mathsf{n} \mid K_A$. This can be encoded by $K_G \succ \mathsf{n} \mid K_A$. However, $K_G \succ \mathsf{G} \mid K_A$ cannot be deduced using the above assumptions within the logic. It means that K_G cannot prove that it is accountable for $\mathsf{n} \mid K_A$. Consequently, K_G is not tracked when K_A delegates $\mathsf{n} \mid K_A$ to other principals. K_G cannot effectively delegate $\mathsf{n} \mid K_A$ to other principals.

This problem can be solved by issuing certificate C_4:(K_A, n, (K_G's G) that is generated by K_A. The certificate makes the delegation and accountability from K_A to K_G very clear. Certificate C_4 is represented in the logic as $K_A\,|\!\approx\!(K_G \succ \mathsf{n} \mid K_A)$. Based on this formula, $K_G \succ \mathsf{n} \mid K_A$ is held. Then, K_G can effectively delegate $\mathsf{n} \mid K_A$ to other principals (including K_A). △

7.7 Conclusions

In this chapter we described how poorly characterised permissions within cryptographic credentials can lead to authorisation subterfuge during delegation operations. This subterfuge results in a vulnerability concerning the accountability of the authorisation provided by a delegation chain: does the delegation operations in the chain reflect the true intent of the participants?

The challenge here is to ensure that permissions can be referred to in a manner that properly reflects their context. Since permissions are intended to be shared across local name spaces then their references must be global. In the chapter we discuss some ad-hoc strategies to ensure globalisation of permissions. In particular, we consider the use of global name services and public keys as the sources of global identifiers.

The Subterfuge Logic provides a systematic way of determining whether a particular delegation scheme using particular ad-hoc permissions is sufficiently robust to be able to withstand attempts at subterfuge. This logic provides a new characterisation of certificate reduction that, we argue, is more appropriate to open systems. We believe that it will be straightforward to extend the Subterfuge Logic to consider subterfuge in SDSI-like local names (as considered in Section 7.3).

The Subterfuge Logic is used to determine the source of an authority that constrains how the certificate may be used/delegated by others. The interim principals of a delegation chain may freely make its delegation decisions based on the knowledge of the permission source. It provides a flexible way to the interim principals to make delegation decisions. An alternative strategy to avoiding subterfuge is Constrained Delegation [15]. In Constrained Delegation, the source of an authority uses constraint structures to control the shape of a delegation chain. Only members of the last group of a constraint structure may be allowed to access a given resource. Members of other groups of the constraint structure may only be allowed to delegate permissions. In this way, the source of an authority may separate resource management from resource access, and decide only principals in a specified group may access its resource.

Trust Management, like many other protection techniques, provide operations that are used to control access. As with any protection mechanism the challenge is to make sure that the mechanisms are configured in such a way that they ensure some useful and consistent notion of security. Subterfuge logic helps to provide assurance that a principal cannot bypass security via some unexpected but authorised route. This general goal of analysing unexpected but authorised access is not limited to just certificate schemes. Formal techniques that analyse whether a particular configuration of access controls is effective is considered in [48, 49]; strategies such as well formed transactions, separation of duties and protection

domains help to ensure that a system is sufficiently robust to withstand a malicious principle. We are currently exploring how the subterfuge logic can be extended to include such robustness building strategies.

Chapter 8

DAL: Distributed Authorisation Language

This chapter describes a simple yet expressive logic-based language, Distributed Authorisation Language (DAL). In designing DAL, our motivation has been to provide a safe language that is specialised for open delegation in coalitions. DAL is subterfuge-safe in the sense that properly encoded credentials are not vulnerable to subterfuge. While other languages such as [75, 43, 38, 74] offer comparable levels of expressiveness to DAL, credentials written in these languages require formal analysis [124] and/or pre-agreed global naming services to ensure subterfuge-safe delegation. Like type-safe languages, DAL is intended to provide flexibility while preventing classes of unsafe formulae from being encoded. DAL supports coalition regulation and contract in large scale distributed systems without requiring pre-agreed global naming services. Authorisation decisions are based on DAL logic axioms and on statements made by principals.

This chapter is organised as follows. Section 8.1 introduces the notation of the DAL language. Section 8.2 gives a number of examples that demonstrate how DAL is used to express common security policies. The DAL proof system is given in Section 8.3. Section 8.4 discusses how authorisation subterfuge is avoid in DAL. Section 8.5 discusses DAL features for open delegation, and support different kinds of delegation. How DAL is used to support coalition frameworks is described in Chapter 9.

8.1 Notation

This chapter introduces the notations that are used in DAL, such as principals and statements.

8.1.1 Principals

In DAL, only principals can obtain, use, and issue permissions. A *DAL principal* is an entity that can be identified and verified via Authentication. A DAL principal can be an identifier, a role, or their combination.

Identifiers

In the 'real' world, various entities, such as computers, people, and organisations, should be identified as globally unique. *Identifiers* represent these global unique entities and are denoted by a triple (K, N, $\mathcal{T}$), where K specifies the entity's signature key; text string N is the entity's user-friendly self-description; and $\mathcal{T}$ is a type flag for the entity (value $\mathcal{I}$ or $\mathcal{C}$). We assume that no two identifiers share the same public key and thus are globally unique and verifiable by others.

DAL defines two types of identifiers: individual (denoted as $\mathcal{I}$) and coalition (denoted as $\mathcal{C}$). An *individual* is a single entity who may make decisions by itself and use its own signature key to sign credentials. Examples include computers, people, and so forth. A *coalition* is a virtual space which may not make any decisions by itself: all of its decisions are made by its participants. Examples include departments, organisations, and so forth. If no ambiguity can arise, we refer to a globally unique entity as an identifier.

For the sake of simplicity, the shorthand $ID^{\mathcal{T}}$ is used to replace the triple (K, N, $\mathcal{T}$), where ID is the global unique identifier replacing K and N.

Roles

Roles are principals and are useful for managing the delegation of different permissions to groups of principals. A role is denoted as $ID^{\mathcal{T}}.\mathsf{n}$, where $ID^{\mathcal{T}}$ is an identifier, and n is an auxiliary local name defined by the identifier $ID^{\mathcal{T}}$. For example, $UnivA^{\mathcal{C}}.\mathsf{student}$ represents the role $\mathsf{student}$ of coalition $UnivA^{\mathcal{C}}$. An identifier controls, and may discretionarily define, all of its own roles, but may only define roles of other identifiers when explicitly authorised. Firozabadi has a similar role definition in his PhD dissertation [46].

Threshold Principals

Some situations may require cooperation of several entities to provide a high degree of confidence. These compound entities are called *threshold principals*.

Threshold principals are used in DAL to provide threshold-based delegation. For example, professor $A^{\mathcal{I}}$ has three teaching assistants $B^{\mathcal{I}}$, $C^{\mathcal{I}}$, and $D^{\mathcal{I}}$ for the course $\mathsf{CS101}$. To give any student an experimental score in $\mathsf{CS101}$, two of the three teaching assistants are required to agree with the score.

Like the SPKI/SDSI 'k-of-n' threshold [43], we use a *static threshold structure* to express this kind of compound principal, denoted as $\mathsf{threshold}(k,\ [p_1,\ p_2, \ldots,\ p_\mathsf{n}])$, which means that at least k principals from the list of n principals are required. The threshold of the above example is represented as $\mathsf{threshold}(2,\ [B^{\mathcal{I}}, C^{\mathcal{I}}, D^{\mathcal{I}}])$.

As a further example, in order to combat fraud, a company (coalition) $ComA^{\mathcal{C}}$ has a policy that two managers are required to sign purchase orders for goods with a value over $5,000. We may not a-priori know the number of managers in the company (this number may change with time), and thus, a *dynamic threshold structure*[1] is required to express such compound principals. This is denoted as $\mathsf{threshold}(k,\ r)$ which means that at least k principals in the role r are required. The threshold of the above example is represented as $\mathsf{threshold}(2,\ \mathsf{ComA}^{\mathcal{C}}.\mathsf{manager})$.

Threshold principals also represent a shorthand and their encoding in terms of the basic logic operators is described as follows. The static threshold principal $\mathsf{threshold}(k,\ [p_1, \ldots, p_\mathsf{n}])$ can be simply encoded in the usual way in terms of the logic operators $\wedge$ and $\neg$. For example, given principals p_1 and p_2, then $\mathsf{threshold}(2,\ [p_1, p_2])$ and $\mathsf{threshold}(1,\ [p_1, p_2])$ may be represented by $p_1 \wedge p_2$ and $p_1 \vee p_2$, respectively.

Principals

Let G and R represent the set of all identifiers and roles respectively. The symbols g, g_0, g_1, ..., range over G, and the symbols r, r_0, r_1, ..., range over R. The set of *principals* P is the set of all identifiers, roles, and their combinations using the logic operators $\wedge$ (and) and $\neg$ (negation). The symbols p, p_0, p_1, ..., range over P.

8.1.2 Statements

DAL statements represent facts that are held by principals (identifiers, roles, and threshold principals). DAL statements are made using basic logic operators, functions and the says and directly says operators. We assume that statements may contain temporal information to indicate their validity periods. For the sake of simplicity, we do not consider the time related issues in this thesis. Therefore, all statements are understood with respect to an arbitrary but fixed current time.

Says and Directly-says Statement

We write $g \,|\!|\!\approx s$ to indicate a 'directly-says' statement; this means that a statement s is directly signed by the signing key of identifier g. The 'says' statement, denoted $p \,|\!|\!\sim s$, is

[1] This contrasts with SDSI which requires the members of a role in a threshold scheme to be a-priori defined.

used to reflect deductive results from a chain of credentials. This means that statement s is directly signed by principal p, or that principal p indirectly makes statement s via a chain of related statements.

Since both identifiers and roles are principals in DAL, the statement $C^{\mathcal{C}}.\mathsf{manager} \mid\!\sim s$ is valid. However, $C^{\mathcal{C}}.\mathsf{manager} \mid\!\approx s$ is not a valid statement as 'directly-says' statements reflect credentials directly signed by a key.

We write $(p_1 \wedge p_2) \mid\!\sim s$ as a shorthand of $p_1 \mid\!\sim s$ and $p_2 \mid\!\sim s$ in the remainder of the dissertation, and similarly for 'directly-says' statements.

A dynamic threshold principal $\mathsf{threshold}(n,\ g.\mathsf{r})$ means that at least n principals in the role $g.\mathsf{r}$ are required. A threshold may appear only as a principal in a statement, that is, in a statement of the form $\mathsf{threshold}(n,\ g.\mathsf{r}) \mid\!\sim s$ for some arbitrary statement s. A statement $\mathsf{threshold}(n,\ g.\mathsf{r}) \mid\!\sim s$ is encoded by the logic statement

$$(?X_1 \mid\!\sim g.\mathsf{r} \mid\!\sim s) \wedge \ldots \wedge (?X_n \mid\!\sim g.\mathsf{r} \mid\!\sim s)$$

where each of the $?X_i$ $(1 \leq i \leq n)$ are distinct principals. A function that is similar to $neq(s_1, s_2)$ defined in Section 8.1 can be defined to distinguish principals and this function can be used to encode $neq(?X_i, ?X_j)$ for the above formula with $0 \leq i, j \leq n$ and $i \neq j$.

Functions

We use *functions* to idealise atomic statements in the language, such as facts, regulations, and permissions. A function name may have different meanings for different principals, and, therefore, function names are meaningful only when bound to a specific identifier. A function $n(\mathsf{a_1}, \ldots, \mathsf{a_n})@g$, has name n, arguments $\mathsf{a_1}, \ldots, \mathsf{a_n}$, and g is the identifier that defines this function and controls the operation described by the function. A function argument may be a constant or a variable. To distinguish a variable, its name is decorated by a "?". For example, function invocation $Read(\mathsf{fileB},\ ?\mathsf{X})@A^{\mathcal{I}}$ specifies that any (authorised) principal may read fileB. Evaluating statement $Read(\mathsf{fileB},\ \mathsf{C}^{\mathcal{I}})@A^{\mathcal{I}}$ to true means that individual $C^{\mathcal{I}}$ may read fileB by $A^{\mathcal{I}}$'s local function Read. This statement is true only when the permission provider $A^{\mathcal{I}}$ makes the statement as $A^{\mathcal{I}} \mid\!\sim Read(\mathsf{fileB},\ \mathsf{C}^{\mathcal{I}})@A^{\mathcal{I}}$, and $C^{\mathcal{I}}$ makes the request $C^{\mathcal{I}} \mid\!\sim Read(\mathsf{fileB},\ \mathsf{C}^{\mathcal{I}})@A^{\mathcal{I}}$.

A small number of widely used *global functions* are defined by the provider identified as $DAL^{\mathcal{I}}$. These are similar in intent to the standard classes for programming languages, such as Java. When no ambiguity can rise, the identifier $DAL^{\mathcal{I}}$ is omitted from 'global' functions, and a global function may be denoted as $n(a_1, \ldots, a_n)$. The reader should note that identifier $DAL^{\mathcal{I}}$ provides only schema explanations for global functions: actual role relationships are constructed by participants' statements in the logic and can, therefore, be

fully decentralised.

- Given two valid statements s_1 and s_2, then global function $neq(s_1, s_2)$ represents a fact within the logic as to whether s_1 is not equal to s_2[2].
- Global function $actAs(p_0,\ [p_1, \ldots, p_j])$ relates principals (identifiers and roles): each principal $p_i,\ (i \in [1, \ldots, j])$ acts as principal p_0 (a role or an identifier). If $j = 1$, the bracket is omitted in the expression. The function expression $actAs(p_0,\ [p_1, \ldots,\ p_i])$ is a shorthand for the conjunction $actAs(p_0,\ p_1) \wedge \ldots \wedge actAs(p_0,\ p_j)$.

Example 16 Coalitions $UK^{\mathcal{C}}$, $IE^{\mathcal{C}}$, $FR^{\mathcal{C}}$, and $IT^{\mathcal{C}}$ represent virtual countries, each supporting its necessary operational structure and regulations for running a virtual government. These coalitions form a further coalition $EU^{\mathcal{C}}$, of which each is a member, that is, $actAs(EU^{\mathcal{C}}.\mathsf{member},\ [IE^{\mathcal{C}},\ UK^{\mathcal{C}},\ FR^{\mathcal{C}},\ IT^{\mathcal{C}}])$. △

The global uniqueness of signature keys ensure that identifiers and roles are distinguishable from each other. Roles are distinguishable from each other in statements because DAL does not have a SPKI-like rule of the form $actAs(p_2,\ p_1) \supset (p_1 \mid\!\sim s \supset p_2 \mid\!\sim s)$. Instead, DAL uses axiom **R8** in Figure 8.1 to force a principal to provide an unambiguous role name in the statements clarifying the current role that it is acting as. For any two different roles $g_1.\mathsf{r}_1$ and $g_2.\mathsf{r}_2$, the statement $g_1.\mathsf{r}_1 \mid\!\sim s$ may not be deduced from $g_2.\mathsf{r}_2 \mid\!\sim s$. For example, given $actAs(B^{\mathcal{C}}.\mathsf{student}, A^{\mathcal{I}})$, then statements $A^{\mathcal{I}} \mid\!\sim s$ and $A^{\mathcal{I}} \mid\!\sim B^{\mathcal{C}}.\mathsf{student} \mid\!\sim s$ are different, since from the latter we may deduce $B^{\mathcal{C}}.\mathsf{student} \mid\!\sim s$, which cannot be deduced from the former $A^{\mathcal{I}} \mid\!\sim s$ in DAL.

Statements

Following [5], given principal p, identifier g, statements s, s_1 and s_2, then DAL statements are defined inductively as follows.

- a function is a statement;
- $\neg s_1$ and $s_1 \wedge s_2$ are statements; we write the implication $s_1 \supset s_2$ as an abbreviation for $\neg(s_1 \wedge \neg s_2)$; we also write $s_2 \Longleftarrow s_1$ as an equivalence for $s_1 \supset s_2$, and $s_1 \vee s_2$ as an equivalence for $\neg(\neg s_1 \wedge \neg s_2)$;
- $g \mid\!\approx s$ and $p \mid\!\sim s$ are statements.

Operators $\mid\!\approx$ (directly says) and $\mid\!\sim$ (says) have the same precedence in a statement and have higher precedence than $\wedge$. Operator $\mid\!\sim$ is right-associative. Statements involving

[2]In the absence of a formal sematics for DAL, the equivalence of two statements is based on syntactic comparision.

principal conjunction, such as $(p_1 \wedge p_2) \mid\!\sim s$, are treated as a shorthand for $p_1 \mid\!\sim s$ and $p_2 \mid\!\sim s$.

8.2 DAL Examples

This section demonstrates how DAL is used to express common security policies.

- Decentralised authority. Authorisation is delegated by the issuing of certificates. For example, Alice signs a certificate stating that Bob is a student at University A:

$$Alice^{\mathcal{I}} \mid\!\sim actAs(UnivA^{\mathcal{C}}.\mathsf{student},\ Bob^{\mathcal{I}})$$

- Role-based Authority. A principal may take on different roles within the same or different coalitions. These roles can have different authority. In order to prevent accidental misuse of authority, a principal should indicate which role it is acting as when issuing a credential. For example, $Alice^{\mathcal{I}}$ (a $UnivA^{\mathcal{C}}$ staff member) asserts that $Bob^{\mathcal{I}}$ is a student at $UnivA^{\mathcal{C}}$. Notes that operator $\mid\!\sim$ is right-associative.

$$Alice^{\mathcal{I}} \mid\!\sim UnivA^{\mathcal{C}}.\mathsf{staff} \mid\!\sim actAs(UnivA^{\mathcal{C}}.\mathsf{student},\ Bob^{\mathcal{I}})$$

- Identity-based delegation. A principal can delegate rights to other specific principals. For example, $UnivB^{\mathcal{C}}$ trusts $UnivA^{\mathcal{C}}$ to identify students.

$$\begin{aligned} UnivB^{\mathcal{C}} \mid\!\sim (&actAs(UnivB^{\mathcal{C}}.\mathsf{student},\ ?X) \\ &\Longleftarrow UnivA^{\mathcal{C}} \mid\!\sim actAs(UnivB^{\mathcal{C}}.\mathsf{student},\ ?X)) \end{aligned}$$

- Role-based delegation. A principal can delegate rights to certain roles. This kind of delegation can be done across coalitions. For example, $UnivB^{\mathcal{C}}$ trusts $UnivA^{\mathcal{C}}.\mathsf{staff}$ to identify students.

$$\begin{aligned} UnivB^{\mathcal{C}} \mid\!\sim (&actAs(UnivB^{\mathcal{C}}.\mathsf{student},\ ?\mathsf{X}) \\ &\Longleftarrow UnivA^{\mathcal{C}}.\mathsf{staff} \mid\!\sim actAs(UnivB^{\mathcal{C}}.\mathsf{student},\ ?\mathsf{X})) \end{aligned}$$

  Another example, $UnivA^{\mathcal{C}}$ trusts any coalition that is a university to identify a student.

$$\begin{aligned} UnivA^{\mathcal{C}} \mid\!\sim (&actAs(UnivA^{\mathcal{C}}.\mathsf{student},\ ?\mathsf{X}) \\ &\Longleftarrow actAs(UnivA^{\mathcal{C}}.\mathsf{univ}, ?\mathsf{Y}) \wedge\ ?\mathsf{Y} \mid\!\sim actAs(UnivA^{\mathcal{C}}.\mathsf{student}, ?\mathsf{X})) \end{aligned}$$

- Conjunction inference. A principal may require another principal to satisfy several separate conditions to be authorised for some action. For example, $UnivA^{\mathcal{C}}$ allows anyone who is both a lecturer and a manager to access a document.

$$UnivA^{\mathcal{C}} \mid\sim (access(\mathsf{fileB},\ ?\mathsf{X})@UnivA^{\mathcal{C}} \Longleftarrow actAs(UnivA^{\mathcal{C}}.\mathsf{lecturer},\ ?\mathsf{X}) \wedge actAs(UnivA^{\mathcal{C}}.\mathsf{manager},\ ?\mathsf{X}))$$

- Assert Inference. A principal authorises anyone who satisfies some condition to do a particular action. For example, $Alice^{\mathcal{I}}$ allows anyone who is a student to access a file fileA.

$$Alice^{\mathcal{I}} \mid\sim (Access(\mathsf{fileA},\ ?\mathsf{X})@Alice^{\mathcal{I}} \Longleftarrow actAs(\mathsf{student},\ ?\mathsf{X}))$$

- Static threshold structures. For example, professor $Harry^{\mathcal{I}}$ requires two out of three students $Bob^{\mathcal{I}}$, $Carl^{\mathcal{I}}$, and $David^{\mathcal{I}}$ to cooperate in order to to give student $Alice^{\mathcal{I}}$ a score for course CS101 of University $UnivA$.

$$Harry^{\mathcal{I}} \mid\sim (Score(\mathsf{Alice}^{\mathcal{I}}, \mathsf{CS101}, ?\mathsf{score})@UnivA^{\mathcal{C}} \Longleftarrow \mathsf{threshold}(2,\ [Carl^{\mathcal{I}}, David^{\mathcal{I}}, Bob^{\mathcal{I}}]) \mid\sim Score(\mathsf{Alice}^{\mathcal{I}}, \mathsf{CS101}, ?\mathsf{score})@UnivA^{\mathcal{C}})$$

- Dynamic threshold structures. Professor $Harry^{\mathcal{I}}$ requires two students from $sec^{\mathcal{C}}$ cooperate to approve $Alice^{\mathcal{I}}$'s score for course CS101 of University $UnivA$.

$$Harry^{\mathcal{I}} \mid\sim (Score(\mathsf{Alice}^{\mathcal{I}}, \mathsf{CS101}, ?\mathsf{score})@UnivA^{\mathcal{C}} \Longleftarrow \mathsf{threshold}(2,\ sec^{\mathcal{C}}.\mathsf{student}) \mid\sim Score(\mathsf{Alice}^{\mathcal{I}}, \mathsf{CS101}, ?\mathsf{score})@UnivA^{\mathcal{C}})$$

- Limited role-based delegation. A principal may require a role from a specified coalition to do a job. See the example for dynamic threshold structures.

8.3 Proof System

The proof system has three kinds of axioms:

1. The standard axioms and rules of propositional logic, such as the axiom $(s_1 \wedge s_2) \supset s_1$, the rules $(s_2 \Longleftarrow s_1) \equiv (s_1 \supset s_2)$, and

$$\frac{s_1, s_1 \supset s_2}{s_2}$$

R1 : $(g \mid\!\approx s) \supset (g \mid\!\sim s)$
R2 : $(g_1 \mid\!\sim actAs(g_1,\ g_2)) \supset ((g_2 \mid\!\sim actAs(g_1,\ g_2)) \supset (actAs(g_1,\ g_2)))$
R3 : $(g_1 \mid\!\sim actAs(g_1.\mathsf{n}_1,\ g_2)) \supset ((g_2 \mid\!\sim actAs(g_1.\mathsf{n}_1,\ g_2)) \supset (actAs(g_1.\mathsf{n}_1,\ g_2)))$
R4 : $(g_1 \mid\!\sim actAs(g_1,\ g_2.\mathsf{n}_2)) \supset ((g_2 \mid\!\sim actAs(g_1,\ g_2.\mathsf{n}_2)) \supset (actAs(g_1,\ g_2.\mathsf{n}_2)))$
R5 : $(g_1 \mid\!\sim actAs(g_1.\mathsf{n}_1,\ g_2.\mathsf{n}_2)) \supset ((g_2 \mid\!\sim actAs(g_1.\mathsf{n}_1,\ g_2.\mathsf{n}_2)) \supset (actAs(g_1.\mathsf{n}_1,\ g_2.\mathsf{n}_2)))$
R6 : $(actAs(p_2,\ p_1)) \supset (actAs(p_3,\ p_2) \supset actAs(p_3,\ p_1))$
R7 : $actAs(p_1,\ p_1)$
R8 : $(actAs(p_2,\ p_1)) \supset ((p_1 \mid\!\sim p_2 \mid\!\sim s) \supset (p_2 \mid\!\sim s))$

Figure 8.1: New DAL axioms

2. The standard axioms and rules of modal logic:

$$(p \mid\!\approx (s_1 \supset s_2)) \supset ((p \mid\!\approx s_1) \supset (p \mid\!\approx s_2))$$
$$(p \mid\!\sim (s_1 \supset s_2)) \supset ((p \mid\!\sim s_1) \supset (p \mid\!\sim s_2))$$

$$\frac{s}{p \mid\!\sim s}$$

These are useful for manipulating 'directly-says' and 'says' statements. The axioms define that the statements that a principal says are closed under consequence; the rule expresses that every principal says all provable statements.

3. A small number of additional intuitive axioms, **R1 - R8**, are listed in Figure 1.

R1 reflects the fact that a 'says' statement may be deduced from a single credential. **R2** – **R5** define that only when both the role appointor g_1 and role acceptor g_2 explicitly confirm the role binding that g_2 (or g_2's role $g_2.\mathsf{n}_2$) is bound to g_1 (or g_1's role $g_1.\mathsf{n}_1$), then the role binding becomes a fact. **R6** and **R7** define that the role binding relation is transitive and reflexive. **R8** defines that, if p_1 is bound to p_2, then a statement by p_1 is taken as a statement by p_2 only when p_1 is explicitly representing p_2 in the statement. We informally consider the soundness of this proof system.

PROPOSITION **8.3.1 (Statement Validity)**

If statement s is provable (that is, deducible by the axioms) in the logic, then s is valid (in the sense that the authorisation request is granted).

□

Proof sketch (following the approach in [5]). The propositional axioms and rules are valid, since the propositional connectives are interpreted in the usual manner. So too are the

standard axioms and rules of modal logic. **R1**, **R6** and **R7** are straightforwardly valid. We argue that the remaining axioms are valid using the following comparison with SPKI/SDSI [43], a well-known sound [5] naming schema to link roles(names) between principals.

SPKI/SDSI principal expression $p_1's \ \ldots \ 's \ p_n$ means that p_n is a local name in p_{n-1}'s ... in p_1's local name space, where each of p_i is an identifier or an auxiliary local name. A DAL principal may be regarded as SPKI/SDSI principal with the restrictions: p_1 must be an identifier; p_2 must be an auxiliary local name; and the linking name length $n \leq 2$. Intuitively, when $n = 1$, the principal is an identifier; when $n = 2$, the principal is a role. Thus the set of DAL principal expression is a subset of SPKI/SDSI principal expression.

Given this restriction and the fact that an identifier can be recognised by all principals, the SPKI/SDSI name linking rule may be represented in DAL as

$$(g_1 \ \|\sim \ actAs(g_1.\mathsf{n_1}, g_2)) \supset actAs(g_1.\mathsf{n_1}, g_2)$$
$$(g_1 \ \|\sim \ actAs(g_1.\mathsf{n_1}, g_2.\mathsf{n_2})) \supset actAs(g_1.\mathsf{n_2}, g_2.\mathsf{n_2})$$

R2 – **R5** are constructed from these SPKI/SDSI axioms together with the restriction that not only is the role appointor required to make the name linking statement, but also the role acceptor. Similarly, **R8** and **R9** are constructed from SPKI/SDSI's 'speaks-for' axiom $actAs(p_2,\ p_1) \supset ((p_1 \ \|\sim \ s) \supset (p_2 \ \|\sim \ s))$ with the restriction that, to represent another principal, a principal must explicitly represent that principal in the statement. Therefore, DAL's provable statements form a subset of SPKI/SDSI provable statements. The soundness of SPKI/SDSI— *that all provable statements are valid*— has been proven [5]. Therefore, all provable statements in DAL are also valid.

Soundness implies consistency: *if s is provable in the logic, then $\neg s$ is not provable.*

8.4 Discussion: DAL and Authorisation Subterfuge

The previous chapter explored the problem of authorisation subterfuge, whereby a principal receiving a permission in one domain, can somehow misuse the permission in another domain via some unexpected circuitous but apparently authorised route. Existing Trust Management languages, such as Keynote, SPKI/SDSI may be subject to this problem if care is not taken. In this section we consider the subterfuge problem in the context of DAL.

PROPOSITION **8.4.1 (Safe Delegation)**

Permission delegation in DAL is subterfuge-free.

□

Proof Sketch. A general DAL function can be represented in the following form.

$$act(\mathsf{obj}, ?\mathsf{X})@g_0$$

It means that principal ?X may take principal g_0's local action act on object obj. If we consider action act on obj as permission T, the above DAL function can be rewritten as Subterfuge Logic statement $?\mathsf{X} \ni T \mid g_0$.

The following DAL statement represents a fact that a principal g_n allows another principal g_{n+1} to delegate permission $act(\mathsf{obj}, ?\mathsf{X})@g_0$.

$$g_n \,|\!\approx\!(act(\mathsf{obj}, ?\mathsf{X})@g_0 \impliedby g_{n+1} \,|\!\sim act(\mathsf{obj}, ?\mathsf{X})@g_0)$$

This DAL statement can be safely translated into the following Subterfuge Logic statement.

$$s_n : g_n \,|\!\approx\!(g_{n+1} \succ T \mid g_0)$$

The above statement explicitly indicates that permission T is a local permission of principal g_0. When principal g_{n+1} receives the above statement $g_n \,|\!\sim (g_{n+1} \succ t \mid g_0)$, he considers the originator of this delegation is g_0. Therefore, before delegating $T \mid g_0$ to other principals, g_{n+1} wishes to test whether it is safe to do so. g_{n+1} tests whether g_0 accepts accountability for this action, that is, she attempts to deduce $g_{n+1} \succ T \mid g_0$ using the above assumptions within the Delegation Logic [75]. It requires a delegation chain that delegates permission t from principal g_0 to g_n.

Since $T \mid g_0$ specifies a local permission T of principal g_0, it is assumed that statement $s_{a0} : g_0 \succ T \mid g_0$ is held.

We consider the following two possible scenarios.

- A delegation chain for T from g_0 to g_n exists. It means that g_n is a legitimate principal for the delegation.

  The delegation chain can be constituted by a sequence of statements $s_0, \ldots, s_{n-1}$ as follows.

$$\begin{aligned} s_0 &: g_0 \,|\!\approx\!(g_1 \succ T \mid g_0); \\ s_1 &: g_1 \,|\!\approx\!(g_2 \succ T \mid g_0); \\ &\ldots \\ s_{n-1} &: g_{n-1} \,|\!\approx\!(g_n \succ T \mid g_0); \end{aligned}$$

Using SL rules **D2** and **I3**, $s_{d0} : g_0 \mid\!\sim g_1 \succ T \mid g_0$ is deducible from s_0. Then, using SL rule **I2**, $s_{d1} : g_1 \succ T \mid g_0$ is deducible from statements s_{a0} and s_{d0}. Apply similar SL deduction, $s_{dn-1} : g_n \succ T \mid g_0$ can be deduced consequently. Therefore, g_{n+1} believes that g_n is authorised to delegate permission $T \mid g_0$, and g_0 accepts accountability for this action.

Using **D2**, **I3**, and **I2**, statement $g_{n+1} \succ T \mid g_0$ is deduced from s_n and s_{dn-1}. Then, g_{n+1} believes that he is authorised to delegate permission $T \mid g_0$, and g_0 accepts accountability for this action.

- A delegation chain for T from g_0 to g_n does not exist. It means that g_n is a malicious principal for delegating permission $T \mid g_0$.

  Since no delegation chain delegates $T \mid g_0$ to g_n, statement $g_n \succ T \mid g_0$ is not held. It means that g_n cannot prove that g_n is authorised to delegate permission $T \mid g_0$. Therefore, g_{n+1} will not accept the delegation for $T \mid g_0$ from g_n.

Since principal g_{n+1} may distinguish legitimate principals and malicious principals in delegation chains, she should not delegate permissions from malicious principals and therefore she resists the subterfuge attack. It means that authorisation subterfuge may not occur in DAL permission delegation.

On the other hand, if a certificate is generated without obeying the rules for generating DAL statements, the certificate can not be translated into DAL. Therefore, such a certificate can not be used in a DAL deduction. Consequently, it dose not effect DAL's subterfuge-free property.

8.5 DAL Features

DAL provides a framework for specifying and reasoning about delegation and authorisation in a manner that avoids subterfuge and has a number of novel features that make it particularly applicable to open systems. This is explored in this section.

8.5.1 Global Unique Permissions

Feature 1: Atomic DAL permissions are globally unique.

Existing Trust Management languages, such as SPKI/SDSI and Keynote are designed to express arbitrary permissions. Writing a subterfuge-safe permission is dependent on the user's experience. Even though designed by experts, the KeyNote based payment systems [22, 23, 52] are vulnerable to authorisation subterfuge if care is not taken to properly identify the 'permissions' indicating the payment authorisations when multiple banks and/or provisioning agents are possible.

Atomic DAL permissions are expressed by *globally unique functions*, whereby the identifier is the permission's originator (or the permission authority). The originator has the full authority to explain and manage its own permissions. Together with its own defined local function name, the identifier ensures that the permission is globally unique and the relationship between the permission and its originator is globally verifiable. For example, in statement $access(\mathsf{fileB}, \mathsf{B}^{\mathcal{I}})@A^{\mathcal{I}}$, $A^{\mathcal{I}}$ is the permission authority, and when this permission is delegated to other principals, all delegatees can verify that the permission is from $A^{\mathcal{I}}$.

8.5.2 Global Naming Services

Feature 2: DAL does not require global name services.

A number of existing coalition frameworks rely on some form of global name provider to ensure that different parties get the right name for resources, and so forth. For example, Keynote [19] relies on IANA; RT [74] relies on ADSD; and X509 relies on the X500 name service. However, global name providers are not coalition security administrators; they only provide each name with a unique meaning and have no control over how names are used. Entities from different coalitions may still use arbitrary names to represent their own and the resources of others.

While SPKI/SDSI does not rely on any global name provider, its name schema is subject to authorisation subterfuge. This can be explained as follows. When identifier g_1 binds its local name n_1 to g_2's local name n_2, the SPKI/SDSI 'linking' axiom does not require g_2 to be notified. Therefore, g_2 might not know that its n_2 is bound to g_1's local name, and may accidentally use n_2 for other purposes.

DAL axioms **R2** – **R5** are used to relate principals. With these axioms, g_2 in the above example is required to accept a given name(role) binding. Therefore, g_2 may not use n_2 for other purposes. Thus, in DAL, binding roles(names) between different name spaces does not require global name providers, and can avoid authorisation subterfuge. For example,

$$A^{\mathcal{I}} \mid\!\sim (Read(\mathsf{fileB}, ?\mathsf{X})@A^{\mathcal{I}} \Longleftarrow {C_1}^{\mathcal{C}} \mid\!\sim actAs({C_1}^{\mathcal{C}}.\mathsf{manager}, ?\mathsf{X}) \wedge {C_2}^{\mathcal{C}} \mid\!\sim actAs({C_2}^{\mathcal{C}}.\mathsf{manager}, ?\mathsf{X}))$$

does not require that C_1 and C_2 agree on the meaning of the role manager. The meanings of "manager" in these separate coalitions C_1 and C_2 are decided within their own coalitions. $A^{\mathcal{I}}$ only needs to know the respective meanings of "manager" in these two separate coalitions C_1 and C_2.

8.5.3 Role in Statements

Feature 3: DAL requires a principal to explicitly indicate its current role within

the statement.

Existing authorisation languages, such as SPKI/SDSI and Keynote use the 'speak-for' axiom, whereby if p_2 is bound to p_1, then any statement by p_1 is taken as a statement by p_2. However, a principal may represent different principals. For example, consider the following statements,

$$
\begin{aligned}
&s_1 : actAs(C^{\mathcal{C}}.\mathsf{manager}, A^{\mathcal{I}}),\\
&s_2 : actAs(B^{\mathcal{I}}, A^{\mathcal{I}}),\\
&s_3 : C^{\mathcal{C}} \mid\!\sim (Read(\mathsf{fileB}, ?\mathsf{X})@C^{\mathcal{C}} \Longleftarrow C^{\mathcal{C}}.\mathsf{manager} \mid\!\sim Read(\mathsf{fileB}, ?\mathsf{X})@C^{\mathcal{C}}),\\
&s_4 : A^{\mathcal{I}} \mid\!\sim Read(\mathsf{fileB},\ \mathsf{B}^{\mathcal{I}})@C^{\mathcal{C}}
\end{aligned}
$$

Using the SPKI/SDSI 'speak-for' axiom only, we can deduce: $C^{\mathcal{C}} \mid\!\sim Read(\mathsf{fileB},\ \mathsf{B}^{\mathcal{I}})@C^{\mathcal{C}}$, and $B^{\mathcal{I}} \mid\!\sim Read(\mathsf{fileB},\ \mathsf{B}^{\mathcal{I}})@C^{\mathcal{C}}$. The former statement specifies the policy that $C^{\mathcal{C}}$ authorises $B^{\mathcal{I}}$ to read fileB by $C^{\mathcal{C}}$'s local function. The latter statement defines the request made by $B^{\mathcal{I}}$ to read fileB by $C^{\mathcal{C}}$'s local function. In this scenario, it is not clear what $A^{\mathcal{I}}$ intends to do.

In order to provide more precise permissions, we use the axiom **R8** instead of the SPKI/SDSI 'speak-for' axiom. When a permission is delegated to a certain role, in order to validate the permission, DAL requires that principals must explicitly indicate that role in the statement. Otherwise, even though the permission is signed by the principal who obtains that role, the certificate is useless. It also means that an entity may only speak in roles it knows. This requirement prevents principals from issuing ambiguous permissions.

For example, in a coalition all permissions for use only in the coalition should be delegated to coalition roles. If a principal wants to write a valid 'inner' coalition permission statement, it should know which role it is representing. Since this 'inner' coalition statement is invalid in other coalitions, the principal obtains other roles from other coalitions, and this statement is useless for use in other coalitions. Therefore, cross-coalition subterfuge may be prevented in DAL.

Applying the DAL axioms to this example, it is not possible to derive statements $C^{\mathcal{C}} \mid\!\sim Read(\mathsf{fileB},\ \mathsf{B}^{\mathcal{I}})@C^{\mathcal{C}}$ and $B^{\mathcal{I}} \mid\!\sim Read(\mathsf{fileB},\ \mathsf{B}^{\mathcal{I}})@C^{\mathcal{C}}$. Presenting $A^{\mathcal{I}}$'s extended statement $A^{\mathcal{I}} \mid\!\approx (C^{\mathcal{C}}.\mathsf{manager} \mid\!\sim Read(\mathsf{fileB},\ \mathsf{B}^{\mathcal{I}})@C^{\mathcal{C}})$, results in a policy that coalition $C^{\mathcal{C}}$ authorises $B^{\mathcal{I}}$ to read fileB by $C^{\mathcal{C}}$'s local function. Presenting $A^{\mathcal{I}} \mid\!\approx (B^{\mathcal{I}} \mid\!\sim Read(\mathsf{fileB},\ \mathsf{B}^{\mathcal{I}})@C^{\mathcal{C}})$ results in a request by $B^{\mathcal{I}}$ to read fileB by $C^{\mathcal{C}}$'s local function.

8.5.4 Coalition Delegation

Feature 4: DAL supports coalition delegation in addition to individual delegation.

We give several statements that describe different categories of inner- and outer- coalition delegation.

Closed delegation

Closed delegation means that only coalition participants may work within the coalition. Entities are required to obtain local roles in a closed coalition before they may use that coalition's resources. All 'inner' coalition permissions are delegated only to local roles of that coalition. For example, a closed delegation of coalition $C^{\mathcal{C}}$ may be defined by the statement

$$C^{\mathcal{C}} \mid\sim (read(\mathsf{f_B}, ?\mathsf{X})@C^{\mathcal{C}} \Longleftarrow actAs(C^{\mathcal{C}}.\mathsf{member}, ?\mathsf{X}) \wedge ?\mathsf{X} \mid\approx C^{\mathcal{C}}.\mathsf{member} \mid\sim read(\mathsf{f_B}, ?\mathsf{X})@C^{\mathcal{C}})$$

This means that if a principal wants to operate file $\mathsf{f_B}$ by coalition $C^{\mathcal{C}}$'s local function *read*, it must be in the role member of $C^{\mathcal{C}}$, and must explicitly indicate the role that it is representing in its request (a 'directly-says' statement). Otherwise, the principal may not use that coalition's resource.

Semi-open Delegation

Semi-open delegation means that the resource of coalition ${C_1}^{\mathcal{C}}$ may be used by a group of principals from another coalition ${C_2}^{\mathcal{C}}$. These principals are not required to obtain any local role of coalition ${C_1}^{\mathcal{C}}$. However, they must obtain a specified role from coalition ${C_2}^{\mathcal{C}}$. In addition, these principals may not delegate their authority to others. In this case, all delegation decisions are still made by the resource owner coalition ${C_1}^{\mathcal{C}}$. The coalition ${C_2}^{\mathcal{C}}$ decides who may obtain its specified local role to grant the permission.

An example of a semi-open delegation is given by the following statement.

$${C_1}^{\mathcal{C}} \mid\sim (read(\mathsf{f}, ?\mathsf{X})@{C_1}^{\mathcal{C}} \Longleftarrow actAs({C_2}^{\mathcal{C}}.\mathsf{member}, ?\mathsf{X}) \wedge ?\mathsf{X} \mid\approx {C_2}^{\mathcal{C}}.\mathsf{member} \mid\sim read(\mathsf{f}, ?\mathsf{X})@{C_1}^{\mathcal{C}})$$

This means that ${C_2}^{\mathcal{C}}$'s members are authorised to use ${C_1}^{\mathcal{C}}$'s local resource when representing its membership of ${C_2}^{\mathcal{C}}$, but may not delegate it to others. Note that the directly-says substatement is made by ?X. This restricts ?X to identifiers because only identifiers (with their associated public key), may sign permissions.

Cross Coalition Delegation

Cross coalition delegation means that permissions are delegated from coalition ${C_1}^{\mathcal{C}}$ to a group of principals from another coalition ${C_2}^{\mathcal{C}}$. Unlike semi-open delegation, these principals may delegate their authority to others. For example, statement

$$\begin{aligned}{C_1}^{\mathcal{C}} \mid\!\sim\ & (read(\mathsf{f}, ?\mathsf{X})@{C_1}^{\mathcal{C}} \\ & \Longleftarrow\ actAs({C_2}^{\mathcal{C}}.\mathsf{member}, ?\mathsf{X}) \wedge ?\mathsf{X} \mid\!\sim {C_2}^{\mathcal{C}}.\mathsf{member} \mid\!\sim read(\mathsf{f}, ?\mathsf{X})@{C_1}^{\mathcal{C}})\end{aligned}$$

means that ${C_1}^{\mathcal{C}}$ allows ${C_2}^{\mathcal{C}}$'s members to access file f by ${C_1}^{\mathcal{C}}$'s local function *read*, and ${C_2}^{\mathcal{C}}$'s members may also delegate the permission to others. For example, assuming that $actAs({C_2}^{\mathcal{C}}.\mathsf{member}, A^{\mathcal{I}})$ is true, then issuing the statement

$$A^{\mathcal{I}} \mid\!\sim ({C_2}^{\mathcal{C}}.\mathsf{member} \mid\!\sim read(\mathsf{f}, ?\mathsf{X})@{C_1}^{\mathcal{C}} \ \Longleftarrow\ B^{\mathcal{I}} \mid\!\approx read(\mathsf{f}, ?\mathsf{X})@{C_1}^{\mathcal{C}})$$

means that ${C_2}^{\mathcal{C}}.\mathsf{member}\ A^{\mathcal{I}}$ delegates the above statement to another principal $B^{\mathcal{I}}$. However, $A^{\mathcal{I}}$ still holds accountability for $B^{\mathcal{I}}$'s behavior regarding this delegated permission. This is because, given the request made by $B^{\mathcal{I}}$, we may deduce that $A^{\mathcal{I}}$ makes the request in the role ${C_2}^{\mathcal{C}}.\mathsf{member}$. Note that ${C_1}^{\mathcal{C}}$'s policy statement has a 'says' sub-statement, which is unlike the 'directly-says' used in semi-open delegation.

Open Delegation

Open delegation means that permissions may be delegated from one coalition ${C_1}^{\mathcal{C}}$ to another coalition ${C_2}^{\mathcal{C}}$ which, in turn, may independently decide who may use the permission. For example,

$${C_1}^{\mathcal{C}} \mid\!\sim (read(\mathsf{f}, ?\mathsf{X})@{C_1}^{\mathcal{C}} \ \Longleftarrow\ {C_2}^{\mathcal{C}} \mid\!\sim read(\mathsf{f}, ?\mathsf{X})@{C_1}^{\mathcal{C}})$$

Constrained Delegation

Constrained delegation means that permissions may be delegated from one coalition ${C_1}^{\mathcal{C}}$ to another coalition ${C_2}^{\mathcal{C}}$, however, only principals from coalition ${C_1}^{\mathcal{C}}$ may use the permission. The original concept of constrained delegation is defined in [15] that is similar to our definition. This can be implemented in DAL using a combinition of closed delegation and cross coalition delegation. For example,

$$\begin{aligned}{C_1}^{\mathcal{C}} \mid\!\sim\ & (read(\mathsf{f}, ?\mathsf{X})@{C_1}^{\mathcal{C}} \\ & \Longleftarrow\ ({C_2}^{\mathcal{C}} \mid\!\sim ?\mathsf{X} \mid\!\approx {C_1}^{\mathcal{C}}.\mathsf{member} \mid\!\sim read(\mathsf{f}, ?\mathsf{X})@{C_1}^{\mathcal{C}} \\ & \quad \wedge ?\mathsf{X} \mid\!\approx {C_1}^{\mathcal{C}}.\mathsf{member} \mid\!\sim read(\mathsf{f}, ?\mathsf{X})@{C_1}^{\mathcal{C}})\end{aligned}$$

8.5.5 Complex Principal Expressions

Feature 5: DAL does not support complex principal expressions, such as SPKI/SDSI compound names.

The compound name `UnivA's professor's student` represents the local name `student` in all the local name spaces of `UnivA`'s `professor`. This kind of local name requires that all `UnivA's professor`s must have the same meaning for their local role `student` based on `UnivA`'s understanding.

We argue that this is unreasonable in many scenarios. For example, before joining `UnivA` as a `professor`, Alice may already define her local name `student` with its own meaning. According to the conflicting local name meanings, Alice may not join `UnivA`, or may have to adapt the meaning of `student` according to `UnivA`'s local name space and withdraw all certificates related to the meaning of her own `student`. On the other hand, when Alice joins `UnivA`, she may not be aware of the name conflict, which may lead to further unexpected authorisations/delegations subterfuge. If `UnivA` wishes to delegate its permission to `UnivA's professor's student`, it should ensure that the name `student` has the same meaning for all related principals' local name space. In practice, these are unreasonable solutions.

DAL, in avoiding the use of complex role principals, permits a local name to resolve only to an identifier.

Chapter 9

Secure Coalition Framework

Many coalition-supporting frameworks provide not only operational mechanisms to ensure correct operation, but also provide security mechanisms to ensure that only authorised entities may participate in a coalition. However, with regard to authorisation, we argue that existing coalition security mechanisms are limited in many respects.

In existing frameworks [13, 53, 55, 87, 98], coalitions may or may not have an associated signing key. If a coalition does not have its own signature key, then the coalition may not be uniquely identified over the network. In this case an outsider cannot verify whether a statement is from the coalition and cross-coalition operations may not be possible. When a coalition has a signature key, its "super" security administrator controls that key. Other coalition participants cannot prevent the administrator from arbitrarily signing any statement using that key. Furthermore, if the coalition signing key is compromised then the coalition has to be reformed, whereby, the coalition revoke all certificates signed by the compromised key, all coalition members should accept the new coalition key.

Existing frameworks, such as [13, 53, 55, 87, 98], rely on a "super" administrator who has unlimited authority within the coalition. However, in many cases such flexibility is not desirable. For example, when several mutually suspicious entities establish a coalition to share resources only within the coalition, the concern may be that such an administrator can arbitrarily authorise entities outside of the coalition. In this case no one is willing to authorise any other entity as the coalition security administrator. Another example is that such an omnipotent administrator controls all of the company resources over the network, while those company resources should be controlled by the CEO (or other senior executives) of the company. It is preferable that coalition security mechanisms do not rely on the notion of a "super" administrator.

In addition, existing coalition frameworks rely on the appointment of the "super" security administrator outside of the mechanics of the coalition framework, and he must be

accepted by all coalition participants before a coalition can be established. Regulations concerning the resources under the control of this administrator should be carefully issued by the administrator, and well understood by all participants in advance. Different coalitions may require different establishment and regulations, and thus a high degree of expertise is required for an administrator to properly form and manage a coalition. We believe that coalition establishment should not be done in this ad-hoc manner, rather, it should be formalised as an integral part of the coalition framework.

This chapter is organised as follows. Section 9.1 introduces the desirable characteristics in any coalition supporting frameworks. How DAL is used to support coalition frameworks is described in Section 9.2. Section 9.3 gives core coalition regulations in any valid DAL coalition. A coalition establishment process is proposed in Section 9.4. Section 9.5 provides a informal security analysis of coalition establishment. Section 9.6 gives a number of examples of coalition establishment, coalition merging, and coalition spawning. Section 9.7 discusses the features of our coalition forming framework.

9.1 Coalition Characteristics

A coalition may be formed by one or more principals (individuals and/or further coalitions). We argue that the following characteristics are desirable in any framework that is to support coalitions.

- *Unique identity*. Every coalition has a permanent and unique coalition identity to identify itself over network environments.
- *Regulation of authorisation*. Principal authorisation is regulated according to the controls of the coalitions that it participates in.
- *Decentralised authorisation*. The regulation of authorisation can be decentralised. For example, a coalition supporting framework should not have to rely on a "super" security administrator or a centralised authorisation server.
- *Autonomy*. The establishment and operation of a coalition can be achieved entirely by its participants.
- *Dynamic establishment*. Coalitions may spawn further coalitions. Coalitions may also come together to dynamically form further coalitions.
- *Self-determination*. A resource owner may regulate resource authorisation in different (possibly conflicting) ways according to the coalitions set up to share it.

- *Maintainability.* Under agreed circumstances it must be possible to change the regulations that govern the participants of a coalition.

We are not aware of any existing research that provides an infrastructure that has all of the above characteristics. Using DAL, a framework for establishing secure coalitions is proposed whereby coalitions can be dynamically formed in a fully distributed manner without relying on a "super" security administrator or any particular threshold cryptography algorithms, and the framework can be used to merge and spawn coalitions.

9.2 DAL Support for Coalitions

DAL is used to make statements regarding relationships between principals including coalition membership. In this chapter we consider the process by which new coalitions are negotiated and formed.

To establish a purpose-independent coalition, intended participants should understand and obey a small number of basic coalition regulations. To interact with such coalitions, principals should know how the coalition works in order to correctly negotiate with the coalition. The regulations that are used to characterise the nature of a coalition and its normal operation are specified in terms of DAL.

In designing DAL, our motivation has been to provide a safe language that has been tailored for open delegation in coalitions. While other languages such as [75, 43, 38, 74] offer comparable levels of expressiveness to DAL, credentials written in these languages require formal analysis [124] and/or pre-agreed global naming services to ensure subterfuge-safe delegation. Like type-safe languages, DAL is intended to provide flexibility while preventing classes of unsafe formulae from being encoded.

In our coalition framework, a coalition has a unique identifier that includes its signature key as defined in DAL. The purpose of the coalition key is to sign the initial coalition regulations during the establishment of the coalition and is not intended for any other purpose. The coalition key is generated and initially held by a trusted principal (the constructor) of the proposed coalition. The constructor is selected by the coalition founders and may be a trusted external third party or intended member of the coalition.

Once the initial coalition regulations that identify the coalition constructor, founders and oversights have been signed and the coalition established, the coalition key should not be used for further signing. Further specified coalition regulations will be signed by coalition founders. In establishing a coalition, the constructor signs a penalty contract accepting responsibility for the proper use of the signing key. If the key is misused then the constructor becomes liable under the terms of the contract. The coalition regulations

are such that it is not possible to establish a coalition without signing this contract. In practice, it is expected that having established the coalition, the constructor will destroy the (ephemeral private) coalition key in order to avoid accidental compromise.

This coalition establishing framework does not depend on any particular threshold cryptography scheme and, therefore, the users of this framework are free to use the cryptographic algorithms of their choice.

9.3 Core Coalition Regulations

The reader is reminded that it is only the function schemata that are global; their (decentralised) definitions are provided by DAL statements made by participating principals. The coalition establishing process relies on this small number of global functions to bootstrap subterfuge-safe coalitions. DAL ensures that all other names and permissions that are used within the coalitions are managed in a subterfuge-safe way and do not require a global name service such as X.500. The following predefined global functions are used in establishing coalitions.

- Function $Pay(amount,\ unit,\ payer,\ payee)$ means that the principal payer is willing to pay a certain amount of money to another principal payee. For example, $Pay(500, USD, A^{\mathcal{I}}, B^{\mathcal{I}})$ means that $A^{\mathcal{I}}$ is willing to pay \$500 to $B^{\mathcal{I}}$. When the payee is a coalition role then the mechanism does not specify how the payment should be divided among coalition partners.
- Given two valid statements s_1 and s_2, then function $neq(s_1, s_2)$ represents a fact within the logic as to whether s_1 is not equal to s_2.

Every valid coalition must include two core statements which act as the basic regulation for the coalition. These core statements have the following Coalition Formation (**CF**) patterns.

CF$_1$ A coalition ${id_1}^{\mathcal{C}}$ is formed by an individual ${id_2}^{\mathcal{I}}$ in the role ${\mathsf{id_1}}^{\mathcal{C}}.\mathsf{constructor}$ with the agreement of all founding principals. The responsibility of ${id_2}^{\mathcal{I}}$ extends to the formation of the coalition, but no further.

$$\begin{aligned} {id_1}^{\mathcal{C}} \mid\!\!\approx (&actAs({id_1}^{\mathcal{C}}.\mathsf{constructor}, {id_2}^{\mathcal{I}}) \\ &\wedge\ actAs({id_1}^{\mathcal{C}}.\mathsf{r_1}, [p_1, \ldots, p_n]) \\ &\wedge\ (?\mathsf{X} \Longleftarrow \mathsf{threshold}(n,\ {id_1}^{\mathcal{C}}.\mathsf{r_1}) \mid\!\!\sim ?\mathsf{X})) \end{aligned}$$

This statement from ${id_1}^{\mathcal{C}}$, is interpreted as follows,

1. $id_1{}^{\mathcal{C}} \mid\!\sim actAs(id_1{}^{\mathcal{C}}.\mathsf{constructor},\ id_2{}^{\mathcal{I}})$. This statement indicates that the coalition $id_1{}^{\mathcal{C}}$ claims the individual $id_2{}^{\mathcal{I}}$ is its constructor and holds its coalition key. After obtaining this information, the principal who wants to join $id_1{}^{\mathcal{C}}$, or collaborate with the coalition, may require further insurance information ($\mathbf{CF_2}$) from $id_2{}^{\mathcal{I}}$.
2. $id_1{}^{\mathcal{C}} \mid\!\sim actAs(id_1{}^{\mathcal{C}}.\mathsf{r_1}, [p_1, \ldots, p_n])$. Here, $id_1{}^{\mathcal{C}}$ defines the role $id_1{}^{\mathcal{C}}.\mathsf{r_1}$ in $id_1{}^{\mathcal{C}}$ when $id_1{}^{\mathcal{C}}$ established, and $p_i, i \in [1, n]$ are the only initial principals for this role. The role $id_1{}^{\mathcal{C}}.\mathsf{r_1}$ represents the most important role in the coalition, for example, a role to which founders belong.
3. $id_1{}^{\mathcal{C}} \mid\!\sim (?\mathsf{X} \Longleftarrow \mathsf{threshold}(n,\ id_1{}^{\mathcal{C}}.\mathsf{r_1}) \mid\!\sim ?\mathsf{X})$. When a principal accepts the role $id_1{}^{\mathcal{C}}.\mathsf{r_1}$, it needs to be sure that it does not delegate any unintended rights, that no one can force it to accept conditions it does not want to accept, such as sharing its own resources without its permission and so forth, as a consequence of joining a coalition.

   Only when n initial principals of this coalition agree on a decision, may the coalition make this decision. In other-words, if any one playing the role $id_1{}^{\mathcal{C}}.\mathsf{r_1}$ does not agree on a decision, then the coalition may not be established. If the coalition key does not sign any other credentials, then this statement guarantees that the $id_1{}^{\mathcal{C}}$ authority is distributed to all initial principals who are playing the role $id_1{}^{\mathcal{C}}.\mathsf{r_1}$.

$\mathbf{CF_2}$ Coalition constructor $id_2{}^{\mathcal{I}}$ agrees to a penalty contract regarding proper use of the coalition key.

$$id_2{}^{\mathcal{I}} \mid\!\sim (Pay(AMT, U, id_2{}^{\mathcal{I}}, id_1{}^{\mathcal{C}}.\mathsf{r_2}) \Longleftarrow neq(id_1{}^{\mathcal{C}} \mid\!\approx ?\mathsf{Y},\ \mathbf{CF_1}))$$

This is a necessary part of a properly formed coalition and provides evidence that, if $id_1{}^{\mathcal{C}}$ signs any statement other than founding regulation $\mathbf{CF_1}$ above, then $id_2{}^{\mathcal{I}}$ is willing to pay a penalty amount AMT in the currency U to the principals in the oversight role $id_1{}^{\mathcal{C}}.\mathsf{r_2}$.

Note that the oversight role $id_1{}^{\mathcal{C}}.\mathsf{r_2}$ may be different to the founding role $id_1{}^{\mathcal{C}}.\mathsf{r_1}$. The presence of some penalty regulation clause is sufficient in a valid coalition; whether it is acceptable is part of the coalition establishment process with the founders and is considered in Section 9.4.

The signature keys for $id_1{}^{\mathcal{C}}$ and $id_2{}^{\mathcal{I}}$ are known only by $id_2{}^{\mathcal{I}}$. Therefore, $id_2{}^{\mathcal{I}}$ is the only principal who can generate the necessary regulation credentials above. The signature

key for $id_1{}^{\mathcal{C}}$ should only be used for establishing the coalition; to use it for any other action results in a penalty on $id_2{}^{\mathcal{I}}$ according to $\mathbf{CF_2}$.

Example 17 $A^{\mathcal{I}}$, $B^{\mathcal{I}}$, $C^{\mathcal{I}}$ wish to form a coalition $M^{\mathcal{C}}$. They will be the $M^{\mathcal{C}}$'s founders. They all trust a third party $TTP^{\mathcal{I}}$ to generate the basic coalition regulations so long as $TTP^{\mathcal{I}}$ is willing to promise that if $TTP^{\mathcal{I}}$ misuses the coalition key, then it will pay \$50 collective penalty to principals in the role $\mathsf{M}^{\mathcal{C}}.\mathsf{oversight}$. The regulations generated are as follows.

$$\begin{aligned}\mathbf{CF_1} \hat{=}\ & M^{\mathcal{C}} \mid\!\approx (actAs(M^{\mathcal{C}}.\mathsf{constructor}, TTP^{\mathcal{I}}) \\ & \wedge actAs(M^{\mathcal{C}}.\mathsf{founder},\ [A^{\mathcal{I}}, B^{\mathcal{I}}, C^{\mathcal{I}}])) \\ & \wedge (?\mathsf{X} \Longleftarrow \mathsf{threshold}(3,\ M^{\mathcal{C}}.\mathsf{founder}) \mid\!\sim ?\mathsf{X}))\end{aligned}$$

$$\begin{aligned}\mathbf{CF_2} \hat{=}\ & TTP^{\mathcal{I}} \mid\!\sim (Pay(50, USD, TTP^{\mathcal{I}}, M^{\mathcal{C}}.\mathsf{oversight}) \\ & \Longleftarrow neq(M^{\mathcal{C}} \mid\!\approx ?\mathsf{Y}, \mathbf{CF_1}))\end{aligned}$$

△

9.4 Coalition Establishment Process

Establishing a new coalition requires cooperation and agreement between the coalition constructor and all of the coalition founders. This Coalition Establishment (**CE**) process involves three steps. Given $\mathbf{i}, \mathbf{j} \in [1, n]$:

CE$_1$ $id_2{}^{\mathcal{I}} \rightarrow p_{\mathsf{i}} : \mathbf{CF_1} \wedge \mathbf{CF_2}$;

Each principal p_{i} invited to join the coalition's founding role $id_1{}^{\mathcal{C}}.\mathsf{r_1}$ receives and checks the two regulations generated from $id_2{}^{\mathcal{I}}$. After signing $\mathbf{CF_1}$ it will not be in the interest of $id_1{}^{\mathcal{C}}$ to sign further credentials, due to the penalty signed by $id_2{}^{\mathcal{I}}$ (in $\mathbf{CF_2}$).

CE$_2$ $p_{\mathsf{i}} \rightarrow id_2{}^{\mathcal{I}}$: $p_{\mathsf{i}} \mid\!\sim (actAs(id_1{}^{\mathcal{C}}.\mathsf{r_1}, p_{\mathsf{i}}) \wedge \mathbf{CF_1} \wedge \mathbf{CF_2})$;

Each founder p_{i} that accepts the coalition regulations signs an agreement on the regulations and its role membership. Before engaging this step, coalition founders may communicate to consider and informally agree the regulations.

CE$_3$ $p_{\mathsf{i}} \rightarrow p_{\mathsf{j}}$: $p_{\mathsf{i}} \mid\!\sim id_1{}^{\mathcal{C}}.\mathsf{r_1} \mid\!\sim actAs(id_1{}^{\mathcal{C}}.\mathsf{r_2}, id_1{}^{\mathcal{C}}.\mathsf{r_1})$.

The final step in the process requires the founders to agree the enforcement role, that is, the oversight role to which payment will be made if the coalition constructor breaks the regulations of the coalition.

After establishing a coalition, anyone can trust ${id_1}^{\mathcal{C}}$ or ask its constructor ${id_2}^{\mathcal{I}}$ to provide all above credentials as evidence for the coalition establishment.

9.5 Security Analysis

The coalition establishment process is secure, in the sense that no principal can cheat or mis-represent the coalition. We provide a security analysis of coalition establishment; the analysis is informal.

This coalition establishment process relies on three roles: constructor, founder, and oversight. Role constructor is a predefined role name for all coalitions, making it clear who creates the coalition. The founders and oversight are user-definable role names. We analyse the accountability of each role as follows.

Regulation $\mathbf{CF_1}$ is a prearranged agreement by all founders regarding coalition structure and initial participants. $\mathbf{CF_2}$ is a prearranged penalty contract agreed by the coalition constructor and all of the coalition founders. All founders knows the constructor's public key and agree that the constructor generates the basic coalition regulations.

Only the constructor knows the coalition and constructor signature (private) keys. He is therefore the only principal that can generate and accept accountability for $\mathbf{CF_1}$ and $\mathbf{CF_2}$. If a valid constructor generates just one version of $\mathbf{CF_1} \wedge \mathbf{CF_2}$ then it is not possible to deduce $neq({id_1}^{\mathcal{C}} \mid\!\approx ?X, \mathbf{CF_1})$ and consequently it is not possible to deduce ${id_2}^{\mathcal{I}} \mid\!\sim pay(AMT, U, {id_2}^{\mathcal{I}}, {id_1}^{\mathcal{C}}.\mathsf{r_2})$, that is, a valid constructor cannot be penalised using $\mathbf{CF_2}$.

Founders are willing to accept some pre-agreed $\mathbf{CF_1}$ and $\mathbf{CF_2}$ when establishing the coalition. If a constructor (or any principal masquerading as constructor) generates regulations $\mathbf{CF_1}$ and $\mathbf{CF_2}$ that are not acceptable to the founders, then the coalition establishment process stops at Step $\mathbf{CE}_2$.

If the coalition constructor attempts to mislead principals by generating different versions of $\mathbf{CF_1}$ for founders and other participants, then upon detection the penalty regulation can be applied. Similarly, once a coalition has been established, if the constructor attempts to speak for the coalition (using the coalition key), then upon detection the penalty regulation may be applied. Thus, a constructor cannot misrepresent a coalition without the application of the penalty regulation. On the other hand, a valid constructor cannot be penalised.

Having completed Step $\mathbf{CE}_2$, the founders can collectively speak for the coalition. However, each founder is not willing to participate until it has received acceptable declaration concerning the oversight role (Step $\mathbf{CE}_3$). This agreement is also necessary before the penalty regulation can be properly applied and effectively protects the constructor if a

coalition is not properly established.

Before accepting the role founder, coalition founders may informally negotiate with each other to determine whether they received the same message. Any founder may stop the establishment process, if it received an unexpected $\mathbf{CF_1}$ or $\mathbf{CF_2}$.

The threshold structure in $\mathbf{CF_1}$ ensures that agreement is required between all founders before Step $\mathbf{CE}_2$ can be successfully completed: it is not possible to establish a coalition without the agreement of all founders. Once established all the founders share authority of the coalition.

The oversight role does not have any inherent authority other than authority over the penalty. However, when a principal accepts this role, it may protect itself by keeping necessary evidence to carry out the penalty contract.

9.6 Examples

The following examples provide examples of coalition establishment, coalition merging, and coalition spawning.

Example 18 (Security Group) This is a centralised coalition, in which the constructor $Alice^{\mathcal{I}}$ is also the founder and controls all authority of the coalition $sec^{\mathcal{C}}$. $Alice^{\mathcal{I}}$ follows the three protocol steps defined above.

First, $Alice^{\mathcal{I}}$ acting as coalition constructor sends the founding regulations to herself (in role $sec^{\mathcal{C}}.\mathsf{head}$).

$$\begin{aligned}\mathbf{CF_{sec1}} \hat{=} sec^{\mathcal{C}} \mid\!\approx (&actAs(sec^{\mathcal{C}}.\mathsf{constructor},\ Alice^{\mathcal{I}})\\ &\wedge actAs(sec^{\mathcal{C}}.\mathsf{head},\ Alice^{\mathcal{I}})\\ &\wedge (?\mathsf{X} \Longleftarrow \mathsf{threshold}(1,\ sec^{\mathcal{C}}.\mathsf{head}) \mid\!\sim ?\mathsf{X}))\end{aligned}$$

$$\begin{aligned}\mathbf{CF_{sec2}} \hat{=} Alice^{\mathcal{I}} \mid\!\approx (&Pay(500,\ USD,\ Alice^{\mathcal{I}},\ sec^{\mathcal{C}}.\mathsf{oversight})\\ &\Longleftarrow neq(sec^{\mathcal{C}} \mid\!\approx ?\mathsf{Y},\ \mathbf{CF_{sec1}}))(2)\end{aligned}$$

Using DAL axiom **R1**, statement 9.1 is derived from $\mathbf{CF_{sec1}}$.

$$\begin{aligned}sec^{\mathcal{C}} \mid\!\sim\ \ &(actAs(sec^{\mathcal{C}}.\mathsf{constructor},\ Alice^{\mathcal{I}})\\ &\wedge actAs(sec^{\mathcal{C}}.\mathsf{head},\ Alice^{\mathcal{I}})\\ &\wedge (?\mathsf{X} \Longleftarrow \mathsf{threshold}(1,\ sec^{\mathcal{C}}.\mathsf{head}) \mid\!\sim ?\mathsf{X}))\end{aligned} \tag{9.1}$$

Consequently, statement 9.2 is obtained from statement 9.1.

$$sec^{\mathcal{C}} \mid\!\sim actAs(sec^{\mathcal{C}}.\mathsf{head}, Alice^{\mathcal{I}}) \tag{9.2}$$

Given her financial commitment, it is in $Alice^{\mathcal{I}}$'s own interest to avoid accidental compromise and and improper use of the coalition key $\mathsf{K}_{\mathsf{sec}c}$. Therefore, principal $Alice^{\mathcal{I}}$ destroys the (ephemeral private) coalition key.

Then, $Alice^{\mathcal{I}}$ accepts founding role $sec^{\mathcal{C}}.\mathsf{head}$ defined by herself using statement 9.3.

$$Alice^{\mathcal{I}} \mid\!\approx\!(actAs(sec^{\mathcal{C}}.\mathsf{head},\ Alice^{\mathcal{I}}) \wedge \mathbf{CF_{sec1}} \wedge \mathbf{CF_{sec2}}) \tag{9.3}$$

Using DAL axiom **R1**, statement 9.4 is derived from statement 9.3.

$$Alice^{\mathcal{I}} \mid\!\sim actAs(sec^{\mathcal{C}}.\mathsf{head},\ Alice^{\mathcal{I}}) \tag{9.4}$$

Using DAL axiom **R3**, statement 9.5 is derived from statement 9.2 and 9.4.

$$actAs(sec^{\mathcal{C}}.\mathsf{head},\ Alice^{\mathcal{I}}) \tag{9.5}$$

That is, $Alice^{\mathcal{I}}$ is the founder of the coalition. Similar deductions can be made on the coalition regulations to deduce $actAs(sec^{\mathcal{C}}.\mathsf{constructor},\ Alice^{\mathcal{I}})$ and $actAs(sec^{\mathcal{C}}.\mathsf{oversight},\ Alice^{\mathcal{I}})$ ($Alice^{\mathcal{I}}$ is the coalition constructor and has oversight).

$Alice^{\mathcal{I}}$ specifies that role $sec^{\mathcal{C}}.\mathsf{oversight}$ is the oversight role:

$$Alice^{\mathcal{I}} \mid\!\approx sec^{\mathcal{C}}.\mathsf{head} \mid\!\sim actAs(sec^{\mathcal{C}}.\mathsf{oversight}, sec^{\mathcal{C}}.\mathsf{head}) \tag{9.6}$$

At this point $Alice^{\mathcal{I}}$ is the constructor, founder and sole member of the coalition $sec^{\mathcal{C}}$. She introduces a new role $sec^{\mathcal{C}}.\mathsf{member}$ for members and assigns $Bob^{\mathcal{I}}$ to that role:

$$Alice^{\mathcal{I}} \mid\!\sim sec^{\mathcal{C}}.\mathsf{head} \mid\!\sim actAs(sec^{\mathcal{C}}.\mathsf{member},\ Bob^{\mathcal{I}}) \tag{9.7}$$

$Alice^{\mathcal{I}}$ assigns all group members to the oversight role:

$$Alice^{\mathcal{I}} \mid\!\sim sec^{\mathcal{C}}.\mathsf{head} \mid\!\sim actAs(sec^{\mathcal{C}}.\mathsf{oversight}, sec^{\mathcal{C}}.\mathsf{member}) \tag{9.8}$$

If $Bob^{\mathcal{I}}$ is satisfied with the regulation on oversight then he is willing to participate within the coalition and signs

$$Bob^{\mathcal{I}} \mid\!\sim actAs(sec^{\mathcal{C}}.\mathsf{member},\ Bob^{\mathcal{I}}) \tag{9.9}$$

and $Bob^{\mathcal{I}}$ can now speak as a member of the coalition.

△

Example 19 (GRID Group) This is a decentralised coalition, in which the constructor $John^{\mathcal{I}}$, together with other two individuals $Ellen^{\mathcal{I}}$ and $Alice^{\mathcal{I}}$, are the founders of the coalition $grid^{\mathcal{C}}$. These three individuals control all $grid^{\mathcal{C}}$'s authority. We have:
$John^{\mathcal{I}} \rightarrow Ellen^{\mathcal{I}}, Alice^{\mathcal{I}} : \mathbf{CF_{grid1}} \wedge \mathbf{CF_{grid2}}$, where,

$$\begin{aligned}\mathbf{CF_{grid1}} \hat{=}\, & grid^{\mathcal{C}} \mid\!\approx (actAs(grid^{\mathcal{C}}.\mathsf{constructor},\ John^{\mathcal{I}}) \\ & \wedge actAs(grid^{\mathcal{C}}.\mathsf{committee}, [Ellen^{\mathcal{I}}, John^{\mathcal{I}}, Alice^{\mathcal{I}}]) \\ & \wedge (?\mathsf{X} \Longleftarrow \mathsf{threshold}(3,\ grid^{\mathcal{C}}.\mathsf{committee}) \mid\!\sim ?\mathsf{X}))\end{aligned}$$

$$\begin{aligned}\mathbf{CF_{grid2}} \hat{=}\, & John^{\mathcal{I}} \mid\!\approx (Pay(500, USD, John^{\mathcal{I}}, grid^{\mathcal{C}}.\mathsf{oversight}) \\ & \Longleftarrow neq(grid^{\mathcal{C}} \mid\!\approx ?\mathsf{Y}, \mathbf{CF_{grid1}}))\end{aligned}$$

$Alice^{\mathcal{I}}$, $Ellen^{\mathcal{I}}$, and $John^{\mathcal{I}}$ accept the founding role $grid^{\mathcal{C}}.\mathsf{committee}$ of coalition $grid^{\mathcal{C}}$:

$$\begin{aligned}Ellen^{\mathcal{I}} \rightarrow John^{\mathcal{I}} :\quad & Ellen^{\mathcal{I}} \mid\!\approx (actAs(grid^{\mathcal{C}}.\mathsf{committee},\ Ellen^{\mathcal{I}}) \\ & \wedge \mathbf{CF_{grid1}} \wedge \mathbf{CF_{grid2}})\end{aligned}$$

$$\begin{aligned}John^{\mathcal{I}} \rightarrow John^{\mathcal{I}} :\quad & John^{\mathcal{I}} \mid\!\approx (actAs(grid^{\mathcal{C}}.\mathsf{committee},\ John^{\mathcal{I}}) \\ & \wedge \mathbf{CF_{grid1}} \wedge \mathbf{CF_{grid2}}) \\ Alice^{\mathcal{I}} \rightarrow John^{\mathcal{I}} :\quad & Alice^{\mathcal{I}} \mid\!\approx (actAs(grid^{\mathcal{C}}.\mathsf{committee},\ Alice^{\mathcal{I}}) \\ & \wedge \mathbf{CF_{grid1}} \wedge \mathbf{CF_{grid2}})\end{aligned}$$

$Alice^{\mathcal{I}}$, $Ellen^{\mathcal{I}}$, and $John^{\mathcal{I}}$ specify that role $grid^{\mathcal{C}}.\mathsf{oversight}$ is the oversight role:

$$\begin{aligned}Ellen^{\mathcal{I}} \rightarrow John^{\mathcal{I}}, Alice^{\mathcal{I}} :\quad & Ellen^{\mathcal{I}} \mid\!\approx grid^{\mathcal{C}}.\mathsf{committee} \mid\!\sim \\ & actAs(grid^{\mathcal{C}}.\mathsf{oversight}, grid^{\mathcal{C}}.\mathsf{committee}) \\ John^{\mathcal{I}} \rightarrow Alice^{\mathcal{I}}, Ellen^{\mathcal{I}} :\quad & John^{\mathcal{I}} \mid\!\approx \mathsf{grid}^{\mathcal{C}}, \mathsf{committee} \mid\!\sim \\ & actAs(grid^{\mathcal{C}}.\mathsf{oversight}, grid^{\mathcal{C}}.\mathsf{committee}) \\ Alice^{\mathcal{I}} \rightarrow Ellen^{\mathcal{I}}, John^{\mathcal{I}} :\quad & Alice^{\mathcal{I}} \mid\!\approx grid^{\mathcal{C}}.\mathsf{committee} \mid\!\sim \\ & actAs(grid^{\mathcal{C}}.\mathsf{oversight}, grid^{\mathcal{C}}.\mathsf{committee})\end{aligned}$$

Everyone can verify that $grid^{\mathcal{C}}$ has been established properly. The committee member role $grid^{\mathcal{C}}.\mathsf{committee}$ share all $grid^{\mathcal{C}}$'s authority. All new regulations must be certificated by all

of them. $grid^{\mathcal{C}}$ needs a director to deal with routine works.

After the coalition $grid^{\mathcal{C}}$ is established, coalition committees may build their own coalition structure. The following statement shows that a director role $grid^{\mathcal{C}}.\mathsf{director}$ is defined and authorised to accept new members for $grid^{\mathcal{C}}$. From

$$Ellen^{\mathcal{I}} \mid\!\sim grid^{\mathcal{C}}.\mathsf{committee} \mid\!\sim (IsRole(\mathsf{grid}^{\mathcal{C}}, \mathsf{member}, ?\mathsf{X}) \Longleftarrow grid^{\mathcal{C}}.\mathsf{director} \mid\!\sim IsRole(grid^{\mathcal{C}}.\mathsf{member}, ?\mathsf{X}))$$

$$Alice^{\mathcal{I}} \mid\!\sim grid^{\mathcal{C}}.\mathsf{committee} \mid\!\sim (IsRole(grid^{\mathcal{C}}.\mathsf{member}, ?\mathsf{X}) \Longleftarrow grid^{\mathcal{C}}.\mathsf{director} \mid\!\sim IsRole(grid^{\mathcal{C}}.\mathsf{member}, ?\mathsf{X}))$$

$$John^{\mathcal{I}} \mid\!\sim grid^{\mathcal{C}}.\mathsf{committee} \mid\!\sim (IsRole(grid^{\mathcal{C}}.\mathsf{member}, ?\mathsf{X}) \Longleftarrow \mathsf{grid}^{\mathcal{C}}, \mathsf{director} \mid\!\sim IsRole(grid^{\mathcal{C}}.\mathsf{member}, ?\mathsf{X}))$$

We get the following statement

$$grid^{\mathcal{C}} \mid\!\sim (IsRole(grid^{\mathcal{C}}.\mathsf{member}, ?\mathsf{X}) \Longleftarrow grid^{\mathcal{C}}.\mathsf{director} \mid\!\sim IsRole(grid^{\mathcal{C}}.\mathsf{member}, ?\mathsf{X}))$$

Then, $grid^{\mathcal{C}}$ appoints John as the director of $grid^{\mathcal{C}}$ by $grid^{\mathcal{C}} \mid\!\sim IsRole(grid^{\mathcal{C}}.\mathsf{director},\ John^{\mathcal{I}})$, and $John^{\mathcal{I}} \mid\!\sim IsRole(grid^{\mathcal{C}}.\mathsf{director},\ John^{\mathcal{I}})$.

After accepting the director role in $grid^{\mathcal{C}}$, $John^{\mathcal{I}}$ is able to admit $Philip^{\mathcal{I}}$ as a member of $grid^{\mathcal{C}}$.

$$John^{\mathcal{I}} \mid\!\sim grid^{\mathcal{C}}.\mathsf{director} \mid\!\sim IsRole(grid^{\mathcal{C}}.\mathsf{member}, Philip^{\mathcal{I}})$$

△

Example 20 (Spawning a New Web-Grid Group) $grid^{\mathcal{C}}$ wants to spawn a further group $web^{\mathcal{C}}$ to manage the Web GRID project. $grid^{\mathcal{C}}$ director holds all authority of $web^{\mathcal{C}}$.
Firstly, $John^{\mathcal{I}}$ acting as the constructor sends the founding regulations to himself (in role $web^{\mathcal{C}}.\mathsf{founder}$):

$$\begin{aligned}\mathbf{CF_{web1}} \hat{=} web^{\mathcal{C}} \mid\!\approx (&actAs(web^{\mathcal{C}}.\mathsf{constructor},\ John^{\mathcal{I}}) \\ &\wedge actAs(web^{\mathcal{C}}.\mathsf{founder},\ grid^{\mathcal{C}}.\mathsf{director}) \\ &\wedge (?\mathsf{X} \Longleftarrow \mathsf{threshold}(1,\ web^{\mathcal{C}}.\mathsf{founder}) \mid\!\sim ?\mathsf{X}))\end{aligned}$$

$$\begin{aligned}\mathbf{CF_{web2}} \hat{=} John^{\mathcal{I}} \mid\!\approx (&Pay(500,\ USD,\ John^{\mathcal{I}},\ web^{\mathcal{C}}.\mathsf{oversight}) \\ &\Longleftarrow neq(web^{\mathcal{C}} \mid\!\approx ?\mathsf{Y},\ \mathbf{CF_{web1}}))\end{aligned}$$

Then, $John^{\mathcal{I}}$ accepts the founding role $web^{\mathcal{C}}.\mathsf{founder}$ defined by himself:

$$John^{\mathcal{I}} \mid\!\approx grid^{\mathcal{C}}.\mathsf{director} \mid\!\sim (actAs(web^{\mathcal{C}}.\mathsf{founder},\ grid^{\mathcal{C}}.\mathsf{director}) \wedge \mathbf{CF_{web1}} \wedge \mathbf{CF_{web2}})$$

Finally, $John^{\mathcal{I}}$ specifies that role $web^{\mathcal{C}}.\mathsf{oversight}$ is the oversight role:

$$John^{\mathcal{I}} \mid\!\approx grid^{\mathcal{C}}.\mathsf{director} \mid\!\sim web^{\mathcal{C}}.\mathsf{founder} \mid\!\sim actAs(web^{\mathcal{C}}.\mathsf{oversight}, web^{\mathcal{C}}.\mathsf{founder})$$

△

Example 21 (Web Security Group) Coalitions $sec^{\mathcal{C}}$ and $web^{\mathcal{C}}$ merge to a new coalition, which is called $websec^{\mathcal{C}}$. Coalitions merging to a further coalition is similar to coalitions formed by individuals. We have:
$Alice^{\mathcal{I}} \rightarrow Bob^{\mathcal{I}}, Philip^{\mathcal{I}} : \mathbf{CF_{websec1}} \wedge \mathbf{CF_{websec2}}$;

$$\begin{aligned}\mathbf{CF_{websec1}} \hat{=} websec^{\mathcal{C}} \mid\!\approx (&actAs(websec^{\mathcal{C}}.\mathsf{constructor},\ Alice^{\mathcal{I}})\\ &\wedge actAs(websec^{\mathcal{C}}.\mathsf{cmtte},\ [sec^{\mathcal{C}}, grid^{\mathcal{C}}])\\ &\wedge (?\mathsf{X} \Longleftarrow \mathsf{threshold}(2,\ [websec^{\mathcal{C}}.\mathsf{cmtte}]) \mid\!\sim ?\mathsf{X}))\end{aligned}$$

$$\begin{aligned}\mathbf{CF_{websec2}} \hat{=} Alice^{\mathcal{I}} \mid\!\approx (&Pay(500,\ USD,\ Alice^{\mathcal{I}},\ websec^{\mathcal{C}}.\mathsf{oversight})\\ &\Longleftarrow neq(websec^{\mathcal{C}} \mid\!\approx ?\mathsf{Y},\ \mathbf{CF_{websec1}}))\end{aligned}$$

Before the next step, following statements are required.

$$\begin{aligned}&sec^{\mathcal{C}} \mid\!\sim actAs(\mathsf{sec}^{\mathcal{C}}, \mathsf{ws.committee}, Bob^{\mathcal{I}})\\ &sec^{\mathcal{C}} \mid\!\sim (websec^{\mathcal{C}}.\mathsf{committee} \mid\!\sim ?\mathsf{X} \Longleftarrow sec^{\mathcal{C}}.\mathsf{ws.committee} \mid\!\sim ?\mathsf{X})\\ &web^{\mathcal{C}} \mid\!\sim actAs(web^{\mathcal{C}}.\mathsf{ws.committee}, Philip^{\mathcal{I}})\\ &web^{\mathcal{C}} \mid\!\sim (websec^{\mathcal{C}}.\mathsf{committee} \mid\!\sim ?\mathsf{X} \Longleftarrow web^{\mathcal{C}}.\mathsf{ws.committee} \mid\!\sim ?\mathsf{X})\end{aligned}$$

Then, $Bob^{\mathcal{I}}$ and $Philip^{\mathcal{I}}$ accept the founding role $websec^{\mathcal{C}}.\mathsf{cmtte}$ defined by Alice:

$$\begin{aligned}Bob^{\mathcal{I}} \rightarrow Alice^{\mathcal{I}} \quad :\quad & Bob^{\mathcal{I}} \mid\!\approx sec^{\mathcal{C}}.\mathsf{wscmtte} \mid\!\sim\\ &(GFactAswebsec^{\mathcal{C}}.\mathsf{cmtte},\ sec^{\mathcal{C}} \wedge\ \mathbf{CF_{websec1}}\ \wedge\ \mathbf{CF_{websec2}})\\ Philip^{\mathcal{I}} \rightarrow Alice^{\mathcal{I}}: \quad & Philip^{\mathcal{I}} \mid\!\approx web^{\mathcal{C}}.\mathsf{wscmtte} \mid\!\sim\\ &(actAs(websec^{\mathcal{C}}.\mathsf{cmtte},\ web^{\mathcal{C}}) \wedge\ \mathbf{CF_{websec1}}\ \wedge\ \mathbf{CF_{websec2}})\end{aligned}$$

Finally, $Bob^{\mathcal{I}}$ (speaking for $sec^{\mathcal{C}}$) and $Philip^{\mathcal{I}}$ (speaking for $web^{\mathcal{C}}$) specify that $websec^{\mathcal{C}}.\mathsf{oversight}$

is the oversight role.

$$Philip^{\mathcal{I}} \rightarrow Bob^{\mathcal{I}}: \quad Philip^{\mathcal{I}} \mid\!\approx web^{\mathcal{C}}.\mathsf{wscmtte} \mid\!\sim websec^{\mathcal{C}}.\mathsf{cmtte} \mid\!\sim actAs(websec^{\mathcal{C}}.\mathsf{oversight}, websec^{\mathcal{C}}.\mathsf{cmtte})$$

$$Bob^{\mathcal{I}} \rightarrow Philip^{\mathcal{I}}: \quad Bob^{\mathcal{I}} \mid\!\approx sec^{\mathcal{C}}.\mathsf{wscmtte} \mid\!\sim websec^{\mathcal{C}}.\mathsf{cmtte} \mid\!\sim actAs(websec^{\mathcal{C}}.\mathsf{oversight}, websec^{\mathcal{C}}.\mathsf{cmtte})$$

△

9.7 Discussion

Using DAL, a formal framework for regulating the establishment of dynamic coalitions is proposed in this chapter. Coalitions are formed with the involvement of founders, constructors and oversight and do not rely on the traditional notion of a "super" administrator. Constructors are responsible for properly creating a coalition; this service can be provided by a third party. If the service is improperly provided then the constructor is subject to a penalty, which may be collected by another third party providing oversight. With this framework, a coalition can be dynamically formed in a fully distributed manner without relying on a "super" security administrator or any particular threshold cryptography algorithms. The coalition forming framework has a number of characteristics.

- *Authentication.* Each individual and coalition has a unique signature key. Every coalition and individual can be uniquely authenticated.
- *Delegation.* DAL can express a wide range of delegation actions, including, identity-based, role-based, conjunction, cross-coalition and static and dynamic threshold-based delegation.
- *Decentralisation.* Our framework does not need a centralized or 'super' coalition administrator. Once a coalition is established, then all the authority of the coalition lies with the founders who can create and regulate their own coalition structure. If some of the coalition founders' signature keys are compromised, then the other coalition founders may choose to no longer issue further regulations in order to protect the coalition. However, revocation of compromised keys and revocation of regulations that have been issued with the participation of compromised keys remain an issue. The focus in this paper is on providing a logic and framework for establishing coalitions and we do not consider revocation in this paper.
- *Accountability.* Principals wishing to negotiate with a coalition check the coalition's two regulations. Following delegation of coalition authority, principals determine

whether they are engaging with the right role/principal in that coalition. If incorrect regulations are received then they can be kept as evidence. The constructor has accountability for incorrect regulations. This provides autonomy and self-determination.

- *Maintainability.* Once the coalition is created the coalition (signing) key should be destroyed by the constructor; the coalition key provides a unique identity and is used to validate the basic regulations. Founders can introduce further regulations (speaking for the coalition) that control existing and new participants. Principals, including the founders, may also leave a coalition under regulations that were agreed upon by the other founders.

- *Fairness.* No principal has advantages over any other during or after the coalition establishment process. This fair coalition establishing process does not require any trusted third party who has total power over issuance of certificates.

- *Non-repudiation.* We consider only the simple meaning of non-repudiation, that is, once a statement is signed by a principal then it may not subsequently deny that statement. We do not consider non-repudiation of recipient.

- *Dynamic establishment.* Since coalitions are principals then the establishment process can be used by coalitions to form further coalitions.

- *Revocation.* We do not consider key revocation or certificate revocation. The reason for this is that the analysis of subterfuge problem in our research is based on a delegation chain that is composed of a number of current effective certificates. Revocation provides a separate scenario. When a certificate is revoked, it is not a current effective certificate.

Part IV

Conclusions and Future Work

Chapter 10

Conclusions and Future Work

In this chapter, we draw conclusions, summarise our contributions made in this dissertation, and indicate some future research directions.

10.1 Overview

Designing well behaved security mechanisms is a challenging task, since security mechanisms often fail to fulfil their goals for various reasons. One possible explanation for this is that many existing security mechanisms are designed in an ad-hoc manner. Their design follows best practice based on the experiences of their designers, and prevent only classes of known malicious behaviour. However, effective subterfuge is often based on other unexpected behaviour. Another possible explanation for this is that many existing security mechanisms are designed to work properly only for coalitions that satisfy a number of specified assumptions, but they are used for coalitions that do not satisfy all of these specified assumptions.

Understanding subterfuge provides a new approach to the analysis and development of protection mechanisms for coalitions. In this dissertation, we considered the design of three types of security mechanisms: authentication protocols, authorisation languages, and coalition frameworks. Both analysis and synthesis approaches are proposed and used for evaluating the design of these types of security mechanisms.

Security mechanism analysis approaches attempt to analyse existing security mechanisms under specified assumptions. Such an analysis approach specifies a number of possible subterfuge vulnerabilities, and verifies whether a given security mechanism may prevent those subterfuge scenarios under specified assumptions.

Security mechanism synthesis approaches attempt to generate well behaved security mechanisms in a systematic manner. These well behaved security mechanisms should be able to defeat subterfuge. Thus, these well behaved security mechanisms should be

subterfuge-free security mechanisms. When a synthesis approach is provided, anyone may generate security mechanisms by following the steps that are specified in such an approach. A security mechanism generated in a systematic manner is sufficient to prevent a number of known subterfuge automatically. It does not rely on the experiences of its designers. This is unlike security mechanisms that are designed by experience.

10.2 Subterfuge Revisited

The statement at the heart of our thesis is

> The design of a security mechanism can be evaluated by answering the question "Is the security mechanism sufficient to prevent subterfuge from malicious principals?"

Understanding the question provides the groundwork for answering the question. A security mechanism is composed of a sequence of legitimate actions that are performed by a number of specified (well-behaved) principals. The intended controls of a security mechanism are based on the deduction of these legitimate actions. They provide a way to ensure that only these specified (well-behaved) principals may achieve their purposes.

When designing a security mechanism we would like assurance that a malicious principal cannot bypass security in some unexpected, but permitted (according to the design), manner. This malicious principal corresponds to: the Trojan Horse exploiting a covert channel; the Spy engaging in a replay attack on a security protocol; the principal engaging in authorisation subterfuge in a trust management scheme, etc.

In this dissertation, the unexpected, but permitted (according to the design) deceptive actions that are performed by malicious principals with the goal of evading the intended controls of a security mechanism are interpreted in terms of *subterfuge*.

- **Authentication subterfuge** may occur when a principal obtaining a message from a round of a security protocol, can somehow misuse the message in another round of a security protocol via some unexpected circuitous but apparently permitted route. For example, the flawed security protocol in Section 2.2.1 that is used by a bank for customer authentication is subject to authentication subterfuge, and, in particular, freshness attack, whereby, a malicious principal uses messages from a previous protocol round to subvert the bank in the current protocol round.

- **Authorisation subterfuge** may occur when a principal receiving a permission in one domain, can somehow misuse the permission in another domain via some unexpected circuitous but apparently authorised route. For example, the delegation

chain in Section 7.1 that is used by a company for purchase order permission delegation is subject to authorisation subterfuge, whereby, a malicious principal obtains the company's permission by issuing a specious certificate for delegating a permission for another company.

Authentication subterfuge has been investigated and attacks analysed in terms of protocol attacks in the security protocol analysis literature. However, comparable malicious behavior—authorization subterfuge— has not attracted attention in the literature of authorisation mechanism analysis. The notion of subterfuge that is introduced and explored in this dissertation provides a new and unifying perspective for understanding security mechanisms for authentication and authorisation, and, in particular, in coalition frameworks.

This dissertation investigated the design of authentication protocols and decentralised authorisation mechanisms based on the analysis of authentication subterfuge and authorisation subterfuge. Based on this investigation, a decentralised framework has been developed to support the establishment and the management of secure coalitions.

10.3 Summary of Contributions

The contributions contained within this dissertation are as follows.

A BAN-like logic, the BSW-ZF logic, is proposed in Chapter 5 that extends the BSW logic to support reasoning about message secrecy, fresh channels, and 'holding' statements. Given a number of protocol goals and assumptions, the heuristic rules of the BSW-ZF logic are adapted to guide a backward search for sub-protocols (message sequences) from each protocol goal, whereby, the desired messages are generated in a reversed order of the protocol execution sequence.

Chapter 6 describes an automatic security protocol generator that uses the heuristic rules of the BSW-ZF logic to guide it in a backward search for suitable protocols from protocol goals. The approach taken is unlike existing automatic protocol generators which typically carry out a forward search for candidate protocols from the protocol assumptions. A prototype generator ASPB has been built that performs well in the automatic generation of authentication and key exchange protocols. ASPB is faster and more efficient than other existing protocol generators.

In Chapter 7, we described how poorly characterised permissions within cryptographic credentials can lead to the problem of authorisation subterfuge during delegation operations. This subterfuge results in a vulnerability concerning the accountability of the authorisation provided by a delegation chain: does the delegation operations in the chain reflect the true intent of the participants? The challenge here is to ensure that permissions can be referred to in a manner that properly reflects their context.

The Subterfuge Logic proposed in this chapter provides a systematic way of determining whether a particular delegation scheme using particular ad-hoc permissions is sufficiently robust to be able to withstand attempts at authorisation subterfuge. This logic is the first approach for analyzing authorisation subterfuge. It provides a new characterisation of certificate reduction. We argue that this logic is more appropriate to analyse delegation mechanisms in open systems than existing analysis approaches.

In Chapter 8, a simple yet expressive logic-based language Distributed Authorization Language (DAL) is proposed that supports open delegation in large scale distributed systems. From the outset, DAL has been designed with open systems in mind; flexible cross-domain delegation can be achieved without subterfuge and without having to rely on the proper use of a global name service.

Using DAL, a formal framework for regulating the establishment of dynamic coalitions is proposed in Chapter 9. Coalitions are formed with the involvement of founders, constructors and oversight and do not rely on the traditional notion of a super administrator. Constructors are responsible for properly creating a coalition; this service can be provided by a third party. If the service is improperly provided then the constructor is subject to a penalty, which may be collected by another third party providing oversight. With this framework, a coalition can be dynamically formed in a fully distributed manner without relying on a "super" security administrator or any particular threshold cryptography algorithms. The framework can be used to merge and spawn collaborations.

10.4 Future Work

While significant progress has been made, there is much work to be done. In this section, we discuss some future work related to security protocol generation, authorisation subterfuge, and coalition frameworks.

Protocol Generation

Automated security protocol synthesis techniques can be explored in many interesting research directions.

As a topic of ongoing research, we are investigating the use of protocol synthesis techniques to allow principals negotiate and on-the-fly generate security protocols [123]. When principals wish to interact then, rather than offering each other a fixed menu of 'known' protocols, the protocol negotiation process generates a new "session" protocol that is tailored specifically to their current security environment and requirements. A change in the security environment of a principal may result in the re-negotiation of a new security protocol. This provides a basis for survivable security protocols that have the potential to, in

effect, self-heal and adapt to recover from changes in the security environment.

Another research direction is to explore how automated protocol synthesis techniques can be used to manage more sophisticated protocols such as nonrepudiation and delegation. A further research direction can explore how techniques such as [84] can be used to translate (generated) protocol specifications into executable code. It will be a significant foundation for future self-adaptive security mechanisms.

Authorisation Subterfuge

Trust Management, like many other protection techniques, provides operations that are used to control access. As with any protection mechanism the challenge is to make sure that the mechanisms are configured in such a way that they ensure some useful and consistent notion of security.

Subterfuge logic helps to provide assurance that a principal cannot bypass security via some unexpected but authorised route. This general goal of analysing unexpected but authorised access is not limited to just certificate schemes. Formal techniques that analyse whether a particular configuration of access controls is effective is considered in [48, 49]; strategies such as well formed transactions, separation of duties and protection domains help to ensure that a system is sufficiently robust to a malicious principle.

We argue that non-interference provides the basis for a formal understanding of this attack. A further research topic can explore how the subterfuge logic can be extended to include such robustness building strategies.

Coalition Framework

As with any protection framework the challenge is to make sure that it provides some useful and consistent notion of security. Assurance is required that a principal cannot bypass security via some unexpected but authorised route.

In the case of DAL, we seek formal proof that it is a subterfuge-safe language. It is argued in [48] that verifying whether a particular configuration of access controls is effective can be achieved by analysing its consistency, that is, whether it is possible for a malicious principle to interfere with the the normal operation of the system. This type of analysis [48, 49] is not unlike the analysis carried out on authentication protocols. In the case of mechanisms based on trust management schemes, such as DAL, it is a question of ensuring consistency between potential delegation chains. Further research on this topic could explore how a non-interference analysis such as [48, 49] might be done on DAL in order to prove subterfuge-safety.

Finally, current security mechanisms in GRID and web services are generally centralised

mechanisms. This kind of practical mechanism conflicts with their purpose of supporting collaboration among principals. It would be interesting to explore how DAL might be used to provide subterfuge-safe authorization and coalition establishment within Virtual Organisations in GRIDs and web services.

Bibliography

[1] *Data Encryption Standard (DES), Federal Information Processing Standard 46 – the Data Encryption Standard*. National Institute of Standards and Technology, Washington, D.C., 1976.

[2] *Advanced Encryption Standard (AES), Federal Information Processing Standard 197 – the Data Encryption Standard*. National Institute of Standards and Technology, Washington, D.C., November 2001.

[3] ISO/IEC 9798-2. Information technology - security techniques – entity authentication part 2: Mechanisms using symmetric encipherment algorithems (second edition), 1999.

[4] ISO/IEC 9798-3. Information technology - security techniques – entity authentication part 3: Mechanisms entity authentication using digital signature techniques (second edition), 1998.

[5] M. Abadi. On SDSI's linked local name spaces. In *Proceedings of the 10th Computer Security Foundations Workshop (CSFW '97)*, pages 98–108, Washington, DC, USA, 1997. IEEE Computer Society.

[6] M. Abadi. Logic in access control. In *Proceedings of the Eighteenth Annual IEEE Symposium on Logic in Computer Science*, pages 228–233, Ottawa, Canada, June 2003. IEEE Computer Society Press.

[7] M. Abadi and A. D. Gordon. A calculus for cryptographic protocols: The spi calculus. In *Proceedings of Fourth ACM Conference on Computer and Communications Security*, pages 36–47, Zurich, 1997. ACM Press.

[8] M. Abadi and R. Needham. Prudent engineering practice for cryptographic protocols. In *Proceedings of 1994 IEEE Computer Society Symposium on Research in Security and Privacy*, pages 122–136, Los Alamitos, CA, 1994. IEEE Computer Society Press.

[9] R. Alfieri, R. Cecchini, V. Ciaschini, L. dellÁgnello, A. Frohner, A. Gianoli, K. Lörentey, and F. Spataro. VOMS: an authorization system for virtual organizations. In *the proceedings of the 1st European Across Grids Conference*, pages 33–40, Santiago de Compostela, February 2003.

[10] J. Alves-Foss and T. Soule. A weakest precondition calculus for analysis of cryptographic protocols. In *Proceedings of the DIMACS Workshop on Design and Formal Verification of Security Protocols*, 1997.

[11] N. Asokan and P. Ginzboorg. Key-agreement in ad-hoc networks. *Computer Communications*, 23(17):1627–1637, 2000.

[12] T. Aura. Strategies against replay attacks. In *Proceedings of the 10th IEEE Computer Security Foundations Workshop (CSFW'97)*, pages 59–68, Washington, DC, USA, 1997. IEEE Computer Society Press.

[13] T. Aura and S. Mäki. Towards a survivable security architecture for ad-hoc networks. In *the 9th International Workshop on Security Protocols*, volume 2467 of *LNCS*, pages 63–79, Cambridge, UK, April 2001. Springer-Verlag.

[14] D. Balfanz, D. Smetters, P. Stewart, and H. Wong. Talking to strangers: Authentication in adhoc wireless networks. In *Proceedings of the 9th Annual Network and Distributed Systems Security Symposium (NDSS'02)*, San Diego, California, February 2002.

[15] O. Bandmann, M. Dam, and B. Firozabadi. Constrained delegations. In *Proceedings of the 23th Annual IEEE Symposium on Security and Privacy*, pages 131–140, Oakland, CA, May 2002. IEEE Computer Society Press.

[16] D. E. Bell and L. J. La Padula. Secure computer system: unified exposition and multics interpretation. Technical Report ESD-TR-75-306, The MITRE Corporation, Bedford, MA, March 1976.

[17] E. Bertino, B. Catania, E. Ferrari, and P. Perlasca. A logical framework for reasoning about access control models. *ACM Transactions on Information and System Security (TISSEC)*, 6(1):71–127, 2003.

[18] K. J. Biba. Integrity considerations for secure computer systems. Technical Report MTR-3153 Rev 1 (ESD-TR-76-372), The MITRE Corporation, Bedford, MA, April 1977.

[19] M. Blaze, J. Feigenbaum, J. Ioannidis, and A. D. Keromytis. The keynote trust-management system, version 2, IETF RFC 2704, September 1999.

[20] M. Blaze, J. Feigenbaum, J. Ioannidis, and A. D. Keromytis. The role of trust management in distributed systems security. *Secure Internet Programming: Issues in Distributed and Mobile Object Systems*, 1603:185–210, 1999.

[21] M. Blaze, J. Feigenbaum, and J. Lacy. Decentralized trust management. In *Proceedings of the 19th Annual IEEE Symposium on Research in Security and Privacy*, pages 164–173, Oakland, CA, 1996. IEEE Computer Society Press.

[22] M. Blaze, J. Ioannidis, S. Ioannidis, A. Keromytis, P. Nikander, and V. Prevelakis. TAPI: Transactions for accessing public infrastructure. In *Proceedings of the 8th IFIP Personal Wireless Communications (PWC) Conference*, pages 90–100, 2003.

[23] M. Blaze, J. Ioannidis, and A. D. Keromytis. Offline micropayments without trusted hardware. In *Proceedings of the 5th International Conference on Financial Cryptography (FC'01)*, pages 21–40, London, UK, 2002. Springer-Verlag.

[24] C. Boyd and W. Mao. On a limitation of BAN logic. In *Proceedings of Workshop on the theory and application of cryptographic techniques on Advances in cryptology (EUROCRYPT'93)*, pages 240–247, Secaucus, NJ, USA, 1994. Springer-Verlag New York, Inc.

[25] D. F. C. Brewer and M. J. Nash. The chinese wall security policy. In *Proceedings of the 1989 IEEE Symposium on Security and Privacy*, pages 206–214. IEEE Computer Society Press, May 1989.

[26] M. Burnside, D. Clarke, T. Mills, A. Maywah, S. Devadas, and R. Rivest. Proxy-based security protocols in networked mobile devices. In *Proceedings of the 2002 ACM symposium on Applied computing (SAC'02)*, pages 265–272, New York, NY, USA, 2002. ACM Press.

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

[28] L. Buttyán, S. Staamann, and U. Wilhelm. A simple logic for authentication protocol design. In *Proceedings of the 11th IEEE Computer Security Foundations Workshop*, pages 153–162. IEEE Computer Society Press, 1998.

[29] U. Carlsen. Optimal privacy and authentication on a portable communications system. *SIGOPS Operating Systems Review*, 28(3):16–23, 1994.

[30] D. W. Chadwick and A. Otenko. The PERMIS X.509 role based privilege management infrastructure. *Future Generation Computer Systems*, 19(2):277–289, 2003.

[31] A. Chander, D. Dean, and J.C. Mitchell. Reconstructing trust management. *Journal of Computer Security*, 12(1):131–164, 2004.

[32] H. Chen, J. A. Clark, and J. L. Jacob. Synthesising efficient and effective security protocols. *Electronic Notes in Theoretical Computer Science*, 125(1):25–41, 2005.

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

[34] J. A Clark and J. L Jacob. A survey of authentication protocol literature, version 1.0. In *http://www.cs.york.ac.uk/jac/*, 1997.

[35] J. A Clark and J. L Jacob. Searching for a solution: Engineering tradeoffs and the evolution of provable secure protocols. In *Proceedings of the 2000 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, May 2000.

[36] D. Clarke, J. Elien, C. Ellison, M. Fredette, A. Morcos, and R. L. Rivest. Certificate chain discovery in SPKI/SDSI. *Journal of Computer Security*, 9(4):285–322, 2001.

[37] S. Creese, M. H. Goldsmith, B. Roscoe, and I. Zakiuddin. The attacker in ubiquitous computing environments: Formalising the threat model. In T. Dimitrakos and F. Martinelli, editors, *Workshop on Formal Aspects in Security and Trust*, Pisa, Italy, September 2003.

[38] J. DeTreville. Binder, a logic-based security language. In *Proceedings of the 2002 IEEE Symposium on Research in Security and Privacy*, pages 105–113. IEEE Computer Society Press, 2002.

[39] W. Diffie and M. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, November 1976.

[40] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.

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

[42] C. Ellison and S. Dohrmann. Public-key support for group collaboration. *ACM Transactions on Information and System Security (TISSEC)*, 6(4):547–565, 2003.

[43] C. Ellison, B. Frantz, B. Lampson, R. L. Rivest, B. Thomas, and T. Ylonen. SPKI certificate theory, IETF RFC 2693, September 1999.

[44] C. M. Ellison, B. Frantz, B. Lampson, R. Rivest, B. M. Thomas, and T. Ylonen. SPKI examples, Draft-ietf-spki-cert-examples-01, September 1998.

[45] C.M. Ellison. The nature of a usable PKI. *Computer Networks*, 31:823–830, 1999.

[46] B. S. Firozabadi. *Decentralized privilege management for access control*. PhD thesis, Imperial University, London, UK, September 2005.

[47] B. S. Firozabadi, M. J. Sergot, and O. Bandemann. Using authority certificates to create management structures. In *Security Protocols. 9th International Workshop*, volume 2467 of *LNCS*, pages 134–145, Cambridge, April 2001.

[48] S. N. Foley. A non-functional approach to system integrity. *IEEE Journal on Selected Areas in Communications*, 21(1):36–43, January 2003.

[49] S. N. Foley. Believing in the integrity of a system. In *IJCAR Workshop on Automated Reasoning for Security Protocol Analysis*. Springer Verlag Electronic Notes in Computer Science, 2004.

[50] S. N. Foley, B. Mulcahy, and T. B. Quillinan. Dynamic administrative coalitions with WebCom DAC. In *Proceedings of Web 2004: Third Workshop on e-Business*, Washington DC, December 2004.

[51] S. N. Foley and H. Zhou. Authorisation subterfuge by delegation in decentralised networks. In *the 13th International Security Protocols Workshop*, LNCS, pages 127–141, Cambridge, UK, April 2005. Springer-Verlag.

[52] S.N. Foley. Using trust management to support transferable hash-based micropayments. In *Proceedings of the 7th International Financial Cryptography Conference*, pages 1–14, Gosier, Guadeloupe, FWI, January 2003.

[53] I. Foster, C. Kesselman, and G. Tsudik. A security architecture for computational grids. In *Proceedings of ACM Conference on Computers And Security*, pages 83–91. ACM Press, October 1998.

[54] B. Gleeson, A. Lin, J. Heinanen, G. Armitage, and A. Malis. A framework for IP based virtual private networks, IETF RFC 2764, February 2000.

[55] L. Gong. Enclaves: Enabling secure collaboration over the internet. *IEEE Journal on Selected Areas in Communications*, 15(3):567–575, 1997.

[56] L. Gong, R. Needham, and R. Yahalom. Reasoning about belief in cryptographic protocols. In *Proceedings of the IEEE 1990 Symposium on Security and Privacy*, pages 234–248, Oakland, California, May 1990. IEEE Computer Society Press.

[57] L. Gong and P. Syverson. Fail-stop protocols: An approach to designing secure protocols. In *Proceedings of the 5th International Working Conference on Dependable Computing for Critical Applications (DCCA-5)*, pages 44–55, 1995.

[58] The Object Management Group. Common object request broker architecture (CORBA/IIOP), http://www.omg.org/technology/documents/formal/corba iiop.htm, December 2002.

[59] J. D. Guttman. Security protocol design via authentication tests. In *Proceedings of 15th IEEE Computer Security Foundations Workshop*, pages 92–103. IEEE Computer Society Press, April 2002.

[60] J. D. Guttman and F. J. Thayer. Authentication tests. In *Proceedings of the 2000 IEEE Symposium on Security and Privacy*, pages 96–109. IEEE Computer Society Press, 2000.

[61] J. Heather, G. Lowe, and S. Schneider. How to prevent type flaw attacks on security protocols. *Journal of Computer Security*, 11(2):217–244, 2003.

[62] R. Housley, W. Polk, W. Ford, and D. Solo. Internet X.509 public key infrastructure certificate and certificate revocation list (CRL) profile, IETF RFC 3280, April 2002.

[63] F. Jacquemard, M. Rusinowitch, and L. Vigneron. Compiling and verifying security protocols. In *Proceedings of Logic for Programming and Automated Reasoning*, pages 131–160, Reunion Island, November 2000.

[64] S. L. Keoh, E. Lupu, and M. Sloman. PEACE : A policy-based establishment of ad-hoc communities. In *Proceedings of the 20th Annual Computer Security Applications Conference (ACSAC'04)*, pages 386–395, Tucson, Arizona, December 2004. IEEE Computer Society.

[65] A. Kerckhoffs. *La Cryptographie militare*. 1883.

[66] A. Keromytis and J. Smith. Creating efficient fail-stop cryptographic protocols. Technical Report MS-CIS-96-32, University of Pennsylvania, December 1996.

[67] H. Khurana, V. Gligor, and J. Linn. Reasoning about joint administration of access policies for coalition resources. In *Proceedings of IEEE International Conference On Distributed Computing (ICDCS)*, Vienna, Austria, 2002.

[68] D. Kindred. *Theory Generation for Security Protocols*. PhD thesis, Carnegie Mellon University, 1999. Also available as Carnegie Mellon University, Computer Science Report No. CMU-CS-99-130.

[69] D. E. Knuth. *The art of computer programming*, volume 3. Addison-Wesley, Reading, 2nd edition, 1998. sorting and searching.

[70] S. A. Kripke. A completeness theorem in modal logic. *The Journal of Symbolic Logic*, 31(2):276–277, June 1966.

[71] B. Lampson. Protection. In *Proceedings of 5th Princeton Conference on Information Sciences and Systems*, Princeton, 1971.

[72] B. Lampson, M. Abadi, M. Burrows, and E. Wobber. Authentication in distributed systems: Theory and pratice. *ACM Transactions on Computer Systems*, 10(4):265–310, 1992.

[73] N. Li, , W. Winsborough, and J. Mitchell. Beyond proof-of-compliance: Safety and availability analysis in trust management. In *Proceedings of the 2003 IEEE Symposium on Security and Privacy*, pages 123–139, Berkeley, California, May 2003. IEEE Computer Society.

[74] N. Li and J. C. Mitchell. RT: A role-based trust-management framework. In *The Third DARPA Information Survivability Conference and Exposition (DISCEX III)*, pages 201–212, Washington, D.C., April 2003. IEEE Computer Society Press, Los Alamitos, California.

[75] N. Li, W. H. Winsborough, and J. C. Mitchell. Delegation logic: A logic-based approach to distributed authorization. *ACM Transactions on Information and System Security (TISSEC)*, 6(1):128–171, February 2003.

[76] G. Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using FDR. In *Proceedings of the Second International Workshop on Tools and Algorithms for Construction and Analysis of Systems (TACAS)*, volume 1055 of *Lecture Notes In Computer Science*, pages 147–166. Springer-Verlag, Berlin Germany, 1996.

[77] G. Lowe. Some new attacks upon security protocols. In *Proceedings of The 9th Computer Security Foundations Workshop (CSFW'96)*, pages 162–169, Washington, DC, USA, 1996. IEEE Computer Society Press.

[78] G. Lowe. A hierarchy of authentication specifications. In *Proceedings of the 10th Computer Security Foundations Workshop (CSFW'97)*, pages 31–43, Los Alamitos, 1997. IEEE Computer Society Press.

[79] S. Mäki, T. Aura, and M. Hietalahti. Robust membership management for ad-hoc groups. In *Proceedings of the 5th Nordic Workshop on Secure IT Systems (NORDSEC 2000)*, Reykjavik, Iceland, October 2000.

[80] W. Mao. An augmentation of BAN-like logics. In *Proceedings of the Eighth IEEE Computer Security Foundations Workshop (CSFW'95)*, pages 44–57. IEEE Computer Society Press, 1995.

[81] C. Meadows. Formal verification of cryptographic protocols: A survey. In *Proceedings of the 4th International Conference on the Theory and Applications of Cryptology (ASIACRYPT'94)*, volume 917 of *Lecture Notes In Computer Science*, pages 135–150, London, UK, 1994. Springer-Verlag.

[82] C. Meadows. The NRL protocol analyzer: An overview. *Journal of Logic Programming*, 26(2):113–131, 1996.

[83] Sun Microsystems. Enterprise JavaBeans(EJB tm) specification, version 2.1. http://java.sun.com/products/ejb/docs.html, June 2003.

[84] J. Millen and F. Muller. Cryptographic protocol generation from CAPSL. Technical Report SRI-CSL-01-07, SRI International, December 2001.

[85] J. K. Millen. The interrogator: A tool for cryptographic protocol security. In *Proceedings of the 1984 IEEE Symposium on Security and Privacy*, pages 134–141. IEEE Computer Society, 1984.

[86] J. K. Millen. The interrogator model. In *Proceedings of the 1995 IEEE Symposium on Security and Privacy (SP'95)*, pages 251–260, Washington, DC, USA, 1995. IEEE Computer Society.

[87] N. H. Minsky and V. Ungureanu. Law-governed interaction: a coordination and control mechanism for heterogeneous distributed systems. *ACM Transactions on Software Engineering and Methodology*, 9(3):273–305, 2000.

[88] J. C. Mitchell, M. Mitchell, and U. Stern. Automated analysis of cryptographic protocols using murϕ. In *Proceedings of the 1997 IEEE Symposium on Security and Privacy*, pages 141–151. IEEE Computer Society Press, May 1997.

[89] R. Needham and M. Schroeder. Using encryption for authentication in large networks of computers. *Communications of the ACM*, 21(12), December 1978.

[90] R. Needham and M. Schroeder. Authentication revisited. *Operating Systems Review*, 21(7):7–7, January 1987.

[91] D. M. Nessett. A critique of the Burrows, Abadi and Needham logic. *ACM SIGOPS Operating Systems Review*, 24(2):35–38, 1990.

[92] D. O'Crualaoich. Theory generation for the simple logic, Bachelor Thesis, University College Cork, 2002.

[93] D. Otway and O. Rees. Efficient and timely mutual authentication. *Operating Systems Review*, 21(1):8–10, January 1987.

[94] S. Pancho. Paradigm shifts in protocol analysis. In *Proceedings of the 1999 Workshop on New Security Paradigms*, pages 70–79, Caledon Hills, Ontario, Canada, 1999.

[95] L. Paulson. Designing a theorem prover. In Abramsky, Gabbay, and Maibaum, editors, *Handbook of Logic in Computer Science, Volumes 1 (Background: Mathematical Structures) and 2 (Background: Computational Structures)*, volume 2. Clarendon, 1992.

[96] L. Paulson. Relations between secrets: Two formal analyses of the Yahalom protocol. *Journal of Computer Security*, 9(3):197–216, 2001.

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

[98] L. Pearlman, C. Kesselman, V. Welch, I. Foster, and S. Tuecke. The Community Authorization Service: Status and future. In *Proceedings of the Conference for Computing in High Energy Physics 03 (CHEP03)*, La Jolla, California, March 2003.

[99] A. Perrig and D.X. Song. A first step towards the automatic generation of security protocols. In *Proceedings of Network and Distributed System Security Symposium*, February 2000.

[100] A. Perrig and D.X. Song. Looking for diamonds in the desert: Extending automatic protocol generation to three-party authentication and key agreement protocols. In *Proceedings of 13th IEEE Computer Security Foundations Workshop*, pages 64–76. IEEE Computer Society Press, July 2000.

[101] T.B. Quillinan. *Secure Naming for Distributing Computing using the Condensed Graph Model*. PhD thesis, University College Cork, Cork, Ireland, July 2006.

[102] R. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public key cryptosystems. *Communications of the ACM*, 21(2):120–126, February 1978.

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

[104] H. Saidi. Towards automatic synthesis of security protocols. In *Proceedings of the Logic-Based Program Synthesis Workshop, AAAI 2002 Spring Symposium*, Stanford University, California, March 2002.

[105] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman. Role-based access control models. *IEEE Computer*, 29(2):38–47, 1996.

[106] A. Shamir. Identity-based cryptosystems and signature schemes. In *Proceedings of CRYPTO 84 on Advances in cryptology*, pages 47–53, New York, NY, USA, 1985. Springer-Verlag New York, Inc.

[107] D. X. Song. Athena: a new efficient automated checker for security protocol analysis. In *Proceedings of the 12th IEEE Computer Security Foundations Workshop*, pages 192–202. IEEE Computer Society Press, June 1999.

[108] F. Stajano. The resurrecting duckling – what next? In *the 8th International Security Protocols Workshop*, volume 2133 of *Lecture Notes in Computer Science*, pages 204–214. Springer-Verlag, 2000.

[109] F. Stajano and R. Anderson. The resurrecting duckling: Security issues for ad-hoc wireless networks. In *the 7th International Security Protocols Workshop*, Lecture Notes in Computer Science, pages 172–194. Springer-Verlag, 1999.

[110] P. Syverson. On key distribution for repeated authentication. *Operating Systems Review*, pages 24–30, 1994.

[111] P. Syverson. A taxonomy of replay attacks. In *Proceedings of the IEEE Computer Security Foundations Workshop*, pages 187–191. IEEE Press, 1994.

[112] P. Syverson. Limitations on design principles for public key protocols. In *Proceedings of the IEEE Symposium on Security and Privacy 1996*, pages 62–72. IEEE Computer Society Press, April 1996.

[113] P. Syverson. Towards a strand semantics for authentication logics. *Electronic Notes in Theoretical Computer Science*, 20, 1999. 31, 1999.

[114] P. Syverson and I. Cervesato. The logic of authentication protocols. In *the IFIP WG 1.7 International School on Foundations of Security Analysis and Design on Foundations of Security Analysis and Design (FOSAD'00)*, volume 2171 of *Lecture Notes in Computer Science*, pages 63–136, London, UK, 2001. Springer-Verlag.

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

[116] F. J. Thayer, J. C. Herzog, and J. D. Guttman. Strand spaces: Why is a security protocol correct? In *Proceedings of the 20th IEEE Symposium on Security and Privacy*, pages 160–171. IEEE Computer Society Press, 1998.

[117] M. Thompson, W. Johnston, S. Mudumbai, G. Hoo, K. Jackson, and A. Essiari. Certificate-based access control for widely distributed resources. In *Proceedings of the Eighth USENIX Security Symposium (Security '99)*, pages 215–228, Washington, D.C., August 1999.

[118] W. Tzeng and C. Hu. Inter-protocol interleaving attacks on some authentication and key distribution protocols. *Information Processing Letters*, 69(6):297–302, 1999.

[119] T. Y. C. Woo and S. S. Lam. A lesson on authentication protocol design. *Operating System Review*, pages 24–37, 1994.

[120] R. Wright, A. Getchell, T. Howes, S. Sataluri, P. Yee, and W. Yeong. Recommendations for an X.500 production directory service, Request for Comments (RFC) 1803, Internet Engineering Task Force, June 1995.

[121] H. Zhou. Email communication with N. Li on bypassing delegation depth. 2005.

[122] H. Zhou. Email communication with R. Rivest on subterfuge in example 2.6 of SPKI examples. 2005.

[123] H. Zhou and S. N. Foley. A collaborative approach to autonomic security protocols. In *Proceedings of the ACSA New Security Paradigms Workshop*, pages 13–21, Nova Scotia, Canada, September 2004.

[124] H. Zhou and S. N. Foley. A logic for analysing subterfuge in delegation chains. In *Proceedings of the Workshop on Formal Aspects in Security and Trust (FAST2005)*, LNCS, pages 127–141, Newcastle upon Tyne, UK, July 2005. Springer-Verlag.

Index