Technical Report

UCAM-CL-TR-657
ISSN 1476-2986

Number 657

UNIVERSITY OF CAMBRIDGE
Computer Laboratory

Security protocol design by composition

Hyun-Jin Choi

January 2006

15 JJ Thomson Avenue
Cambridge CB3 0FD
United Kingdom
phone +44 1223 763500

http://www.cl.cam.ac.uk/

This technical report is based on a dissertation submitted December 2004 by the author for the degree of Doctor of Philosophy to the University of Cambridge, Churchill College.

Technical reports published by the University of Cambridge Computer Laboratory are freely available via the Internet:

http://www.cl.cam.ac.uk/TechReports/

ISSN 1476-2986

Summary

The aim of this research is to present a new methodology for the systematic design of compound protocols from their parts. Some security properties can be made accumulative, i.e. can be put together without interfering with one another, by carefully selecting the mechanisms which implement them. Among them are authentication, secrecy and non-repudiation. Based on this observation, a set of accumulative protocol mechanisms called *protocol primitives* are proposed and their correctness is verified. These protocol primitives are obtained from common mechanisms found in many security protocols such as challenge and response. They have been carefully designed not to interfere with each other. This feature makes them flexible building blocks in the proposed methodology. Equipped with these protocol primitives, a scheme for the systematic construction of a complicated protocol from simple protocol primitives is presented, namely, design by composition. This design scheme allows the combination of several simple protocol parts into a complicated protocol without destroying the security properties established by each independent part. In other words, the composition framework permits the specification of a complex protocol to be decomposed into the specifications of simpler components, and thus makes the design and verification of the protocol easier to handle. Benefits of this approach are similar to those gained when using a modular approach to software development.

The applicability and practicality of the proposed methodology are validated through many design examples of protocols found in many different environments and with various initial assumptions. The method is not aimed to cover all existent design issues, but a reasonable range of protocols is addressed.

Acknowledgements

First and foremost, I would like to thank my supervisor Prof. Lawrence Paulson for taking me as his research student and guiding me throughout the course of my research. His insight, enthusiasm and encouragement have helped me to make the research a fulfilling experience. I greatly appreciate him for reviewing this dissertation and many valuable suggestions.

Secondly, I would like to thank my former supervisor Prof. Ross Anderson for introducing various interesting topics in the field and helping me to find out the topic of my research. I should also thank him for many useful discussions about my research.

Thirdly, I should thank all my friends and colleagues at Cambridge, for their help, support and friendship. I wish to express my gratitude especially to Andrew Faszer, Islam Mohammad, Keri Faszer, Maria Fernanda Gonzalez, Rasmus Bertelsen, Samir El-Bachir, Sigbjørn Hervik, Thierry Balzarq, Tomas Flaragan, Younes Mokrab, and Zdenek Burger for their special friendship and interesting discussions concerning many different subjects.

Finally, grateful thanks go to my mother and sister for suggesting me to start PhD as well as encouraging me to finish it, and financial support from Korean government is gratefully acknowledged.

Contents

List of Figures

List of Tables

1

Introduction

1.1 Overview

Due to the rapid growth of the Internet and the increasing need for E-commerce and M-commerce, computer security has recently become a very hot topic. As many more people go online and as more services such as banking, shopping, and trading are offered online, it becomes critical to guarantee that these online services are safe and secure to use. These guarantees are generally provided by means of security protocols.

Security protocols are communication protocols that use cryptographic transformations and whose primary purpose is to achieve security related goals. These goals typically include the authentication of the communicating parties and the establishment of a secret session key. Despite their seeming simplicity, experience has shown that security protocols are extremely hard to be correctly designed on the first attempt. Quite surprisingly, many proposed security protocols have later been found to be flawed. The Needham-Schroeder symmetric key protocol [39] was shown to be flawed three years after its publication by Denning and Sacco [15]. CCITT X.509 protocols were discovered to be flawed by Burrows, Abadi, and Needham [7]. Gavin Lowe found the Needham Schroeder public key protocol to be flawed in 1995, almost twenty years after its invention [31]. Many other examples exist. This pattern of protocol design, implementation and flaw discovery has been repeated an alarming number of times in the years that have followed Denning and Sacco's discovery. This has been the main problem with security protocols: too many of them have had flaws that were discovered too late, and this has caused many people to become sceptical about security protocols in general. Especially when flawed security protocols become widely deployed, there can be grave consequences. Security protocols are already being used for electronic funds transfers, and voting protocols have been proposed and may be used for elections in the near future. Many more security protocols will be introduced with time. If a dishonest person is the first to discover a flaw in one of these protocols, he could exploit it and steal a large amount of money or influence the results of an election. Since the reward for breaking some of these protocols is so large, serious and sophisticated attacks on these protocols might increase as more services adopt them. Withstanding these attacks will require a variety of analysis techniques to reduce the number of flaws in protocols and to try to ensure that either no flaws can exist or any flaws that may remain will not be easy to find. Many researchers have worked on applying formal techniques to the analysis and verification of security protocols. BAN Logic, NRL Analyzer, FDR, Spi calculus, Inductive method and Strand space theory [1,7,32,36,41,52] are some of them. Protocol correctness can be checked using these tools. However, these tools were primarily developed for protocol verification rather than for design purposes.

Another problem in the design and verification of security protocols is the difficulty in specifying correctness criteria or protocol goals. Typically, many protocols are poorly designed because their designers are unclear about the protocol goals they should achieve. Specifying the requirements for security protocols formally clarifies the problem that designers should solve and bridges the abstraction gap between informal correctness criteria and formal protocol description. Understanding protocol goals also helps designers to choose proper and unambiguous mechanisms in implementing protocols. Without exact knowledge of protocol goals, it is a difficult task for designers to select or implement correct mechanisms. However, no precise definitions for security properties seem to be currently available, and even a precise notion of authentication is still a topic of research [20,33,47]. One common mistake found in many formal specifications of protocols is that protocols are usually specified at a high level, where cryptographic services are expressed ambiguously by one uniform notation, encryption. Many people use encryption mechanisms to provide both confidentiality and integrity. However, using a uniform notation for any and all cryptographic services results in an incomplete specification. Such a specification does not support a sensible expression of the different kinds of cryptographic services needed in different contexts. Therefore, the formal specification of a protocol should differentiate different cryptographic services using different notations. In other words, the formal specification needs to specify what the mechanism is being used for, not just what mechanism it is.

The protocol design work is typically a repeated process of design and verification. Due to many years of research on protocol verification, the analysis and verification of protocols have become much easier. Nonetheless, the design process still heavily depends on rules of thumb and designers' experience. No reasonable systematic design methodology is yet available, in spite of a relatively long history of the studies on these issues. General principles, which advise things to follow and things to avoid in protocol design, can be helpful, but their applicability is very limited [2, 4]. Common mechanisms for guaranteeing freshness, achieving one-way authentication, avoiding replay attacks, committing to an action without completing the action are some tools that designers can use to construct a new protocol. Many security protocols are in a way some combinations of these mechanisms. For example, a two-way mutual authentication protocol can be seen as a combination of two one-way authentication protocols. However, it is yet unknown how to combine these mechanisms to build more complicated protocols and how to derive more complicated protocols from simple ones without violating security guarantees that simple protocols satisfy. This lack of understanding on protocol composition makes the design task harder, more time consuming, and it also complicates the verification process.

1.2 Scope and Aims

The general aim of this research is to investigate modular construction of security protocols and to present a new methodology for systematic design of compound protocols from their parts. A set of basic components called protocol primitives, which serve as building blocks in the proposed methodology, are identified through the recognition that many security protocols are built using common mechanisms such as challenge and response [19]. A scheme for systematic constructions and derivations of a complicated protocol from simple primitives is presented, namely, design by composition, which allows to combine several simple protocol parts (protocol primitives) into a complicated protocol without destroying security properties already established by each independent part.

Many security properties such as authentication and secrecy are generally not preserved under protocol composition. Protocol composition is complicated because one mechanism of a protocol may reveal information that should remain secret or may degrade security properties achieved by another mechanism of the protocol. For example, when two independent protocols of one-way authentication are combined together to build a compound protocol, it should be guaranteed that neither one of them breaks the authentication achieved by the other. To preserve each security property of a single protocol component in a combined authentication protocol, which is obtained by a composition of single protocol components, the proposed composition scheme should be accumulative. That is, the scheme should only allow to combine protocol components in a way that accumulates security properties. Some security properties can be made accumulative by carefully selecting the mechanisms which implement them. Among them are such security properties as authentication, secrecy and non-repudiation. Based on this observation, accumulative protocol mechanisms are proposed and their correctness is verified.

Equipped with these mechanisms, the applicability and practicality of the proposed methodology are validated through many design examples of protocols found in many different environments and with various initial assumptions. The method is not aimed to cover all existent design approaches, but a reasonable range of protocols is addressed.

1.3 Outline of Dissertation

The layout of this dissertation is as follows. Chapter 2 reviews protocol design methodologies and concepts currently available, which include the design principles serving as informal guidelines, the fail-stop protocol methodology, a simple design logic and several automatic protocol synthesis schemes.

Chapter 3 deals with the practical and theoretical background material upon which the proposed methodology is built. Strand space theory is explained and key concepts such as ideal, coideal, etc are described. Regularity and discreetness properties of protocols are introduced and later used to prove secrecy goals of protocols.

Chapter 4 proposes a protocol composition methodology, which makes it possible to design or derive complicated security protocols starting from protocol primitives. Protocol primitives are presented and their properties are fully explained. Protocol primitives are carefully engineered from mechanisms used in many existing security protocols.

Chapter 5 shows how the proposed design methodology is used in the design of two-party authentication protocols. Several design examples of authentication protocols are presented and a protocol similar to TLS is derived from a simple mutual authentication protocol.

Chapter 6 demonstrates the applicability of the proposed methodology by showing many design examples of authentication protocols where an authentication server is involved. The behaviour of a faithful authentication server is described and later used in the composition of the protocols requiring such a server.

Chapter 7 extends the scope of the proposed methodology to the protocols required in mobile network and secure payment systems. First, common structures of these protocols are identified and the idea of mutual equations is proposed in order to utilise these structures. Second, the design issues of authentication protocols for mobile network environments are discussed. Finally, the properties necessary for secure payment protocols are defined and then a protocol which satisfies these requirements is shown to be composable step by step from simple primitives.

Chapter 8 concludes the dissertation by discussing the main achievements of this research and discusses further work to be done in this field.

2

Literature Review

This chapter reviews previous work relevant to the research presented in this dissertation. Section 2.1 gives a brief outline of design principles intended to act as "rules of thumb" for protocol designers. Section 2.2 explains a design methodology based on the notion of fail-stop protocols. Section 2.3 reviews a design logic for authentication protocol design. Section 2.4 describes a strand space approach based on authentication tests. Section 2.5 addresses various automatic protocol generation schemes. Finally, Section 2.6 outlines a derivation system for security protocols.

2.1 Prudent Design Principles

Abadi and Needham [2] and Anderson and Needham [4] have proposed a set of principles which were derived from the observation of the most common errors that have been found in published protocols. These approaches of structured design rules are complementary in many ways to formal proof approaches: On one hand, by following these principles, designers are less likely to make common mistakes and fall victim to confusion often found in a number of published protocols, and are more likely to produce protocols whose security is easy to evaluate by formal proof tools. On the other hand, protocol flaws discovered through formal analysis may lead researchers to new insights into the nature of robustness [4]. These principles are paraphrased as follows.

Abadi and Needham's Principles for Design of Cryptographic Protocols

Principle 1: Every message should say what it means — the interpretation of the message should depend only on its content.

Principle 2: The conditions for a message to be acted upon should be clearly set out so that someone reviewing a design can see whether they are acceptable or not.

Principle 3: If the identity of a principal is essential to the meaning of a message, it is prudent to mention the principal's name explicitly in the message.

Principle 4: Be clear about why encryption is being done.

Principle 5: When a principal signs material that has already been encrypted, it should not be inferred that the principal knows the content of the message. On the other hand, it is proper to infer that the principal that signs a message and then encrypts it for privacy knows the content of the message.

Principle 6: Be clear what properties you are assuming about nonces. What may do for ensuring temporal succession may not do for ensuring association — and perhaps association is best established by other means.

Principle 7: The use of a predictable quantity can serve in guaranteeing newness, through a challenge and response exchange. But if a predictable quantity is to be effective, it should be protected so that an intruder cannot simulate a challenge and later reply a response.

Principle 8: If timestamps are used as freshness guarantees by reference to absolute time, then the difference between local clocks at various machines must be much less than the allowable age of a message deemed to be valid. Furthermore, the time maintenance mechanism everywhere becomes part of the trusted computing base.

Principle 9: A key may have been used recently, for example to encrypt a nonce, yet be quite old, and possibly compromised. Recent use does not make the key look any better than it would otherwise.

Principle 10: If an encoding is used to present the meaning of a message, then it should be possible to tell which encoding is being used. In the common case where the encoding is protocol dependent, it should be possible to deduce that the message belongs to this protocol, and in fact to a particular run of the protocol, and to know its number in the protocol.

Principle 11: The protocol designer should know which trust relation his protocol depends on, and why the dependence is necessary. The reasons for particular trust relations being acceptable should be explicit thought they will be founded on judgement and policy rather than on logic.

Anderson and Needham's Robustness Principles for Public Key Protocols

Principle 1: Sign before encrypting.

Principle 2: Be careful how entities are distinguished. If possible, avoid using the same key for two different purposes, and be sure to distinguish different runs of the same protocol from each other.

Principle 3: Be careful when signing or decrypting data that you never let yourself be used as an oracle by your opponent.

Principle 4: Account for all the bits — how many provide equivocation, redundancy, computational complexity and so on. Make sure that the redundancy you need is based on mechanisms which are robust in the application context, and that any extra bits cannot be used against you in some way.

Principle 5: Do not assume the secrecy of anybody else's secrets (except possibly those of a certification authority).

Principle 6: Do not assume that a message you receive has a particular form.

Principle 7: Be explicit about the security parameters of crypto primitives.

Principle 8: Robust security is about explicitness; one must be explicit about any properties which can be used to attack a public key primitive, such as multiplicative homomorphism, as well as the usual security properties such as naming, typing, freshness, the starting assumption and what one is trying to achieve.

These principles only act as "rules of thumb" for protocol designers, so the principles are not necessary for correctness, nor are they sufficient. There are many examples of protocols which ignore one or more of the principles and yet are believed to be secure, and there are also many examples of protocols which keep all the principles but are still vulnerable. It is also recognised that principles may conflict with each other: following one principle may result in the violation of another [50]. However, these principles have been found to be very useful as a checklist for protocol designers. Design principles can help to identify weaknesses in a protocol and they can be used as a checklist to guard against certain attacks. In order to use the design principles effectively, protocol designers are in need of tools to help them to design protocols and automatically check that the design principles are not violated.

2.2 Fail-stop Protocols

The secrecy assumption that a secret remains secret during an execution of a protocol is paradoxical because whether a secret can remain secret may depend on whether the protocol is secure. Thus, the assumption cannot be used to derive the security of the protocol unless a separate mechanism can justify this assumption. To tackle this problem, Gong and Syverson have proposed a new protocol design approach based on the idea of fail-stop protocols [23]. In a fail-stop protocol, if a received message is inconsistent with the protocol specification, then all those messages that are causally after the altered message in term of Lamport's definition of causality [30], will not be sent. That is, a protocol execution automatically halts immediately after there is any deviation from the designed protocol execution path. This restricts the effects of active attacks on the protocol because active attacks only cause early termination of a protocol execution. Thus, active attacks do not cause more or different messages to be sent; so an attacker using active attacks cannot obtain more secrets than one using passive eavesdropping. Therefore, only passive attacks need to be considered in the verification process of the protocol. This makes the protocol verification process easier, because such passive attacks and protection against them are much better understood than active attacks.

Informally, a protocol is fail-stop if any attack interfering with a message sent in one step will cause all causally after messages in the next step or later not to be sent. The fail-stop design methodology is composed of three different steps:

Step 1: Verify that the given protocol is fail-stop.

Step 2: Validate the secrecy assumption of the protocol.

Step 3: Apply BAN-like logics to prove the secrecy of the protocol.

Step 1 and **Step 2** can be repeated several times until the secrecy assumption becomes fully validated. To verify that a given protocol is fail-stop, it should be checked whether the protocol confirms to one of the known specifications of fail-stop protocols. The proposed specifications of fail-stop protocols by the authors are summarised as follows.

1. The content of each message has a header containing the identity of its sender[1], the identity of its intended recipient, the protocol identifier, its version number, a message sequence number, and a freshness identifer.
2. In symmetric cryptosystems, each message is encrypted under the key shared between its sender and its intended recipient. In asymmetric cryptosystems, each message is signed by the sender's private key. The message can then be optionally encrypted under the public key of the recipient.
3. An honest process follows the protocol and ignores all unexpected messages (FA: a faithfulness assumption).
4. A process halts any protocol run in which an expected message does not arrive within a specified timeout period (CR: a causality requirement).

If a message header uniquely identifies the position of the message within a protocol execution by means of a message sequence number included in the header, it is impossible to use this message elsewhere without any modification of the message. In a protocol which satisfies the above specification, every message is either encrypted

[1]When one can assume that encryption is sufficient proof of the sender's identity, this requirement can be omitted

under the key shared between its sender and recipient, or signed under the private signing key of its sender, so no one else can make undetectable modifications without obtaining these keys first, and any detectable modification will make the protocol run stop immediately. Therefore, any protocol which satisfies the specifications above is fail-stop.

To validate the security assumption that certain information is known only to a set of participants, it needs to be shown that no other participants can obtain the information through attacks. There are two types of secrets. The first type of secret includes those used as keys to encrypt messages but which are not sent as message content during a protocol execution. Clearly, these keys cannot be obtained by an attacker. The other type includes secrets sent as message contents. In a fail-stop protocol, it is a better strategy for an attacker to wait for the protocol to complete and to gather as many messages as possible, that is, passive eavesdropping is better for him than active manipulation. The information that an attacker has gathered by recording the execution of a protocol is easily deduced through the possession rules similar to the possession rules of the GNY logic [22]. Showing that the attacker cannot possess the secret information is enough to validate the secrecy assumption.

The last step is to apply BAN-like logics to verify the correctness of the protocol. Since the secrecy assumption is already validated, the secrecy assumption can be used to derive the security of the protocol. This makes protocol verification simple and more credible.

The authors have also presented extensible fail-stop protocols that enjoy sequential and parallel composition. Informally, a protocol is extensible fail-stop if adding any last message to the protocol results in a fail-stop protocol. If all individual protocols or building blocks are extensible fail-stop, then the analysis of an overall complex protocol can be built on the analysis of the individual protocols. While many fail-stop protocols are not extensible fail-stop, extensible fail-stop protocols are not substantially harder to design than fail-stop protocols[2].

This approach to secure protocol design does not help the designer in the actual construction of the fail-stop protocol, it is more concerned with verification of the protocol. The designer has to construct a protocol, without a formal design framework,

[2]To be an extensible fail-stop protocol, the protocol needs to satisfy a causal consistency criterion (CCC), which is expressed in terms of a faithfulness assumption (FA) and a causality requirement (CR).

which conforms to the definition of the fail-stop protocol and then the protocol is verified using an existing verification technique. As Gong and Syverson state, some protocols may have other requirements which conflict with those of fail-stop protocols.

2.3 Design Logic

Buttyan et al. have proposed a design logic similar to the BAN logic for the synthesis of security protocols [8], essentially reversing the inference rules of their BAN-like logic. The approach uses a relatively abstract model to construct and verify a protocol. It is based on the concept of the channel. Channels are abstract views of various types of secure communication links between principals. The abstraction of a link as a channel enables designers to design and analyse protocols without addressing the complexity of the actual implementation issues. Indeed, instead of encryption or decryption operations, channels with various access restrictions are used in the synthesis process.

A channel $\mathcal{C}$ is characterised by its set of readers $r(\mathcal{C})$, those who can read a message from the channel, and its set of writers $w(\mathcal{C})$, those who can write a message to the channel. To use a channel, a principal needs a capability to access the channel. This capability is denoted as $\mathcal{C}^r$ for reading from the channel, $\mathcal{C}^w$ for writing to the channel, respectively. If a principal P possesses $\mathcal{C}^r$, then P is a reader of the channel $\mathcal{C}$ and denoted as $P \in r(\mathcal{C})$. If a principal P possesses $\mathcal{C}^w$, then P is a writer of the channel $\mathcal{C}$ and denoted as $P \in w(\mathcal{C})$. It is assumed that a principal can always detect a message arrival on any channel that it can read. There are several types of channels with different characteristics:

Public Channel: A channel which anybody in the system can write and read, i.e. $r(\mathcal{C}) = w(\mathcal{C}) = \mathcal{A}$, where $\mathcal{A}$ is the set of all principals.

Authentic Channel: A channel which anybody can read, but only one principal P can write, i.e. $r(\mathcal{C}) = \mathcal{A}$ and $w(\mathcal{C}) = \{P\}$. Authentic channels can be implemented by signatures.

Confidential Channel: A channel which anybody can write, but only one principal P can read, i.e. $r(\mathcal{C}) = \{P\}$ and $w(\mathcal{C}) = \mathcal{A}$. Confidential channels can be established by encryption with the public key of P.

Dedicated Channel: A channel which one principal P can read and one principal Q can write, i.e. $r(\mathcal{C}) = \{P\}$ and $w(\mathcal{C}) = \{Q\}$. Dedicated channels can be constructed by combining the properties of the authentic channel with the properties of the confidential channel.

Closed Group Channel: A channel which a set of principals can write and the same set of principals can read, i.e. $r(\mathcal{C}) = w(\mathcal{C}) = \mathcal{A}$, where $\mathcal{A}$ is a set of principals. Closed group channels can be implemented by using symmetric key encryption and distributing the key to a set of principals.

Conventional Secret Channel: A channel which two principals P and Q can read and write, i.e. $r(\mathcal{C}) = w(\mathcal{C}) = \{P, Q\}$.

The logic itself is similar to the BAN logic except some minor modifications on the see ($\triangleleft$) formula. That is, a principal sees information through a channel, written as either $P \triangleleft \mathcal{C}(X)$ which means P sees $\mathcal{C}(X)$, or $P \triangleleft X|\mathcal{C}$ which means P sees X via $\mathcal{C}$.

The synthetic rules are obtained by reversing the inference rules of the logic and have the following general forms:

$$\begin{array}{ll} \mathcal{G} & \\ \hookrightarrow & \mathcal{G}_1 \\ \hookrightarrow & \mathcal{G}_2 \\ \hookrightarrow & \cdots \\ \hookrightarrow & \mathcal{G}_n \end{array}$$

which means that in order to reach the goal $\mathcal{G}$ all new goals $\mathcal{G}_1$, $\mathcal{G}_2$, $\cdots$, $\mathcal{G}_n$ have to be reached. A goal $\mathcal{G}$ can have the form $\mathcal{G}'/\mathcal{G}''$, which means that either $\mathcal{G}'$ or $\mathcal{G}''$ has to be reached.

(Synthesis Rule 1) To recognise that a message X arrived via a channel $\mathcal{C}$, a principal P has to receive $\mathcal{C}(X)$ and it has to be able to read $\mathcal{C}$.

$$\begin{array}{lll} P \mid\equiv (P \triangleleft X|\,\mathcal{C}) & & \\ & \hookrightarrow & P \triangleleft \mathcal{C}(X) \\ & \hookrightarrow & P \in r(\mathcal{C}) \end{array}$$

(Synthesis Rule 2) To see a message X, a principal P has to see a message (X, Y) that contains X or it has to receive X via a channel $\mathcal{C}$.

$$\begin{aligned} &P \lhd X \\ &\qquad \hookrightarrow \quad P \lhd (X, Y) / P| \!\equiv (P \lhd X | \, \mathcal{C}) \end{aligned}$$

(Synthesis Rule 3) To believe that a principal Q said X, a principal P has to believe that Q said a message (X, Y) that contains X.

$$\begin{aligned} &P| \!\equiv (Q| \!\sim X) \\ &\qquad \hookrightarrow \quad P| \!\equiv (Q| \!\sim (X, Y)) \end{aligned}$$

(Synthesis Rule 4) To believe that a principal Q said X, a principal P has to receive X via a channel $\mathcal{C}$ that it can read and that it believes can be written only by Q, or P and Q. Furthermore, Q has to see X.

$$\begin{aligned} &P| \!\equiv (Q| \!\sim X) \\ &\qquad \hookrightarrow \quad P \lhd \mathcal{C}(X) \\ &\qquad \hookrightarrow \quad P \in r(\mathcal{C}) \\ &\qquad \hookrightarrow \quad P| \!\equiv (w(\mathcal{C}) = Q) / P| \!\equiv (w(\mathcal{C}) = \{P, Q\}) \\ &\qquad \hookrightarrow \quad Q \lhd X \end{aligned}$$

(Synthesis Rule 5) To believe that a principal Q has recently said X, the following is required: if X is a formula and P believes that Q is honest, then P has to receive X via a channel $\mathcal{C}$ that it can read and that it believes can be written by Q, or P and Q. Furthermore, P has to believe that X is fresh and Q has to believe X.

$$\begin{aligned} &P| \!\equiv (Q \| \!\sim X) \\ &\qquad \hookrightarrow \quad P \lhd \mathcal{C}(X) \\ &\qquad \hookrightarrow \quad P \in r(\mathcal{C}) \\ &\qquad \hookrightarrow \quad P| \!\equiv (w(\mathcal{C}) = Q) / P| \!\equiv (w(\mathcal{C}) = \{P, Q\}) \\ &\qquad \hookrightarrow \quad P| \!\equiv \sharp(X) \\ &\qquad \hookrightarrow \quad Q| \!\equiv X \end{aligned}$$

Alternatively, P has to believe that Q said X and X is fresh.

$$
\begin{array}{ll}
P| \equiv (Q \Vert\!\sim X) & \\
\hookrightarrow & P| \equiv (Q |\!\sim X) \\
\hookrightarrow & P| \equiv \sharp(X)
\end{array}
$$

(Synthesis Rule 6) To believe that a message X is fresh, a principal P has to believe that some part X' of X is fresh.

$$
\begin{array}{ll}
P| \equiv \sharp(X) & \\
\hookrightarrow & P| \equiv \sharp(X')
\end{array}
$$

(Synthesis Rule 7) To believe that a principal Q believes a formula ϕ, a principal P has to believe that Q has recently said ϕ and that Q is honest.

$$
\begin{array}{ll}
P| \equiv (Q| \equiv \phi) & \\
\hookrightarrow & P| \equiv (Q \Vert\!\sim \phi) \\
\hookrightarrow & P| \equiv ((Q \Vert\!\sim \phi) \rightarrow (Q| \equiv \phi))
\end{array}
$$

(Synthesis Rule 8) To believe a formula ϕ, a principal P has to believe that Q has recently said ϕ, and that a principal Q is honest and competent.

$$
\begin{array}{ll}
P| \equiv \phi & \\
\hookrightarrow & P| \equiv (Q \Vert\!\sim \phi) \\
\hookrightarrow & P| \equiv ((Q \Vert\!\sim \phi) \rightarrow \phi)
\end{array}
$$

(Synthesis Rule 9) To believe a formula ϕ, a principal P has to believe a formula ϕ' and the implication $\phi' \rightarrow \phi$.

$$
\begin{array}{ll}
P| \equiv \phi & \\
\hookrightarrow & P| \equiv \phi' \\
\hookrightarrow & P| \equiv (\phi' \rightarrow \phi)
\end{array}
$$

Equipped with these rules, first, the designer identifies protocol goals and describes them with the language of the logic. Then, the designer can generate the whole protocol and the required assumptions in a systematic way by using the synthesis rules. Deriving a protocol from the goals involves repeated application of rules which decompose the goals into simple goals. Synthesis rules are applied until all goals obtained belong to the initial assumptions. The result of the process is a formal description of a protocol in a BAN-like logic form. The clarity of the approach makes it an easy tool to use, but the approach only provides a way of designing protocols at a highest abstraction level. The use of logics to design protocols suffers from the same problem as the verification of protocols using logics: the conversion of the protocol from the formal statements in the BAN-like logic to the implementation, expressed in the ambiguous protocol notation, is an informal process which could lead to errors.

2.4 Strand Space Approach

Guttman has shown how authentication tests can be an effective tool in designing complex protocols [24]. Authentication tests are common tools used to implement authentication mechanisms in many protocols. A successful authentication test guarantees the existence and participation of the intended principal in a protocol run, thus serves as an authentication proof. Generally speaking, two-party protocols are much better understood and much easier to design than protocols with more than two participants. Thus, when a complex protocol design problem is given, the problem is divided into two-party subproblems. This is natural because most authentication and non-repudiation goals are pairwise. Each subproblem can be easily solved using authentication tests. After solving all these subproblems, a protocol which satisfies all these subgoals can be composed from the subprotocols, if there is some common information which can uniquely identify and combine all subprotocols into one. For example, a unique session identifier can be inserted into the messages of each subprotocol in order to tell which session each subprotocol belongs to. Alternatively, each participant can generate a message component using a shared secret together with some session identifiable information which is cryptographically verifiable by the other participants, and send the component to the other participants.

The design process of the approach has the following steps:

1. Formulate a number of precise goals that the protocol is intended to meet. Goals that concern a subset of the principals may be achieved using subprotocols in-

volving only those principals.

2. For each goal, select an authentication test pattern and design an appropriate authentication test mechanism that will satisfy the authentication goal exclusively. Verify that the subprotocol achieves the goal.

3. Piece the subprotocols together to construct a single protocol and justify it using strand space theory.

This approach is interesting because the method shows how to construct a complicated protocol from several two-party protocols. However, the assumption that a common shared secret exists among participants limits its applicability. The design methodology proposed in this thesis is built upon this strand space idea and goes further to extend the idea.

2.5 Automatic Protocol Generation

Several APG (Automatic Protocol Generation) schemes have recently been proposed [12, 18, 43, 44]. The aim of these approaches is to provide a practical means of searching reasonable but not necessarily optimal protocols which satisfy the given goals[3]. Let design space be the set of all possible or feasible designs, then the protocol design problem is to find feasible protocols from the design space within a reasonable time limit. Intuitively, the protocol space of feasible protocols is infinite. Hence, APG systems need a way to limit the number of candidate protocols generated, while not omitting any potential protocols.

[3]In most cases, finding an optimal protocol is impossible due to the large size of the design space.

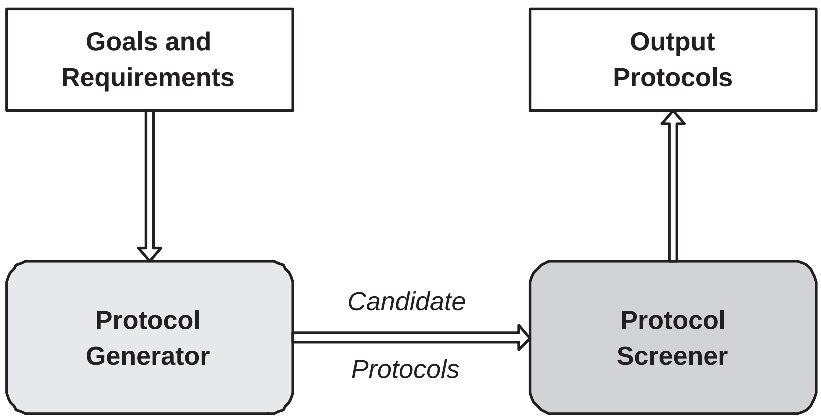


Figure 2.1: **APG Overview**

The first step of these approaches is to input the specification of the desired security properties and the system requirements, then a protocol generator generates many feasible protocols either randomly or with the help of some sophisticated techniques such as genetic algorithms, simulated annealing, etc. A protocol screener analyses these candidate protocols to find out what they actually achieve. Given a candidate protocol, the protocol screener should be able to examine the protocol and tell whether it is correct or not, and the protocol screener needs to be highly efficient to find a good protocol in a reasonable amount of time. If a generated protocol does not satisfy the goals specified, the protocol and any protocols derivable from it can be discarded using reduction techniques. There are tradeoffs in reduction. If a protocol is removed early in the search step, then it may reduce the size of the design space to search but it may also lose some optimal protocols. On the contrary, if a protocol is removed later in the search step, then it may suffer the state-space explosion but it may increase the chance of producing optimal protocols.

The efficiency of these searching frameworks for a reasonably good protocol depends on several factors such as characteristics of the design space, evaluation (fitness or metric) functions and search strategies. The characteristics of the design space determine the way that required protocols are represented. Some approaches restrict the design space to the protocols where principals act honestly [12, 18], whereas some approaches do not [43, 44].

Evaluation functions provide means to evaluate how feasible a protocol is and to characterise precisely how the goodness of each protocol is. Providing a good evaluation function is very difficult because there are too many factors to be considered, such as the strength of assumptions made, the number of interactions with key servers, the length of messages, the number of signatures required, and the amount of encryption needed. To be a good evaluation function, it should prefer a protocol that achieves all of its goals to one that achieves only half of them. Similarly, a good evaluation function should prefer a protocol with less initial assumptions to a protocol with more assumptions. The number of messages in a protocol can be a good metric of efficiency, thus it should also be considered. Formulating an accurate evaluation function seems to be almost impossible and may be unnecessary. However, it is important to find a reasonably good one because an evaluation function forms a crucial component of the search framework, so it may significantly affect the success and the performance of the technique.

Good search strategies offer schemes for searching the design space that examine only a small fraction of possible protocols but still locate good ones. Various heuristic techniques can be used, such as iterative deepening, simulated annealing, tabu searches and genetic algorithms. The efficiency of search strategies determines how good is the solution search strategies outcome in a short time.

APG approaches have several advantages over the current protocol design process, especially compared with a manual design process. First, the approaches are fully automatic. Second, as can be seen from some examples of two-party mutual authentication and key distribution protocols with or without a trusted third party [43, 44], the protocols generated by APG methods offer higher confidence in their correctness because they are already verified by powerful protocol analysers. The approach of automatic protocol generation sounds attractive, but still it is unclear whether it is feasible to generate meaningful and correct protocols automatically. Especially, it is dubious whether the approaches will scale up to more complicated protocols other than two-party mutual authentication ones. With a small increase in principals and messages, the design space which needs to be considered explodes exponentially, and so significantly more aggressive reduction techniques are required. These reduction techniques are difficult to develop due to the increased risk of optimal solutions being discarded by the technique, as more and more potential solutions are discarded.

2.6 Protocol Derivation System

Mechanisms such as Diffie-Hellman key exchange, nonces or timestamps to provide freshness, and signatures for non-repudiation, are common in many authentication and key exchange protocols. Thus, understanding how these mechanisms work and how properties of a compound protocol can be obtained from properties of its parts, especially how various security properties can be accumulated, is crucial for the systematical derivation of complicated security protocols from simple ones. Datta et al. [13] have presented a protocol derivation system for deriving security protocols from basic components using a set of operations such as protocol composition, refinement and transformation.

The system consists of two base protocol components, three transformations, and seven refinement rules. Two protocol components are a Diffie-Hellman key exchange, and a two-message signature challenge and response authentication protocol, which act as basic building blocks for constructing larger protocols. A composition operation combines two protocols together either sequentially or through parameter substitution. Suppose that p_i and p_j are two protocol components. If the postcondition of one protocol p_i matches the precondition of the other protocol p_j, then these two protocols can be sequentially composed as a new protocol $[p_i; p_j]$, where $[p_i; p_j]$ means every step of p_j comes sequentially after every step of p_i. Moreover, if p_i satisfies security property φ with parameter ω, that is, after $p_i(\omega)$, $\varphi(\omega)$ is true, then the substitution of the parameter ω into μ will make the security property $\varphi(\mu)$ true. This parameter substitution together with sequential composition provides a way of composing a protocol. A refinement rule replaces a single message in a protocol with another. A refinement rule does not change the number of messages or the basic structure of the protocol, that is, every refinement rule is nondestructive in the sense that applying the rule does not destroy any security properties already gained. The following are the refinement rules proposed by the authors of [13].

R1: $[m]_A \Rightarrow \{\!|[m]_A|\!\}_k$, where $[\cdot]_A$ is a signature operation with A's signature key and $\{\!| \cdot |\!\}_k$ is an encryption with a shared key k. This refinement provides identity protection against passive attackers.

R2: $[m]_A \Rightarrow [\langle m, ID_A \rangle_k]_A$, where $\langle \cdot \rangle_k$ is a hashing operation with a shared key k and ID_A denotes the public key certificate of A.

R3: $[m]_A \Rightarrow [m]_A, \langle m, ID_A\rangle_k$.

R4: $[m]_A \Rightarrow [m, ID_B]_A$. This refinement provides protection against man in the middle attacks.

R5: $g^x \Rightarrow g^x, n_A$, where n_A is a fresh value. This refinement enables the reuse of exponentials across multiple sessions.

R6: $[m]_A \Rightarrow [m]_A, ID_A$. This refinement discharges the initial assumption that the principals possess each other's public key certificates.

R7: $\{\!|m|\!\}_k \Rightarrow \{\!|m|\!\}_k, \langle role, \{\!|m|\!\}_k\rangle_{k'}$

More refinement rules can be added later through the careful analysis of many published protocols. Finally, a transformation operation acts on a single protocol, which either moves data from a later message to an earlier one through the message component move transformation, or reorders messages using a DoS (Denial of Service) prevention cookie transformation technique. The protocol derivation system can systematically derive many protocols incrementally, starting from simple components, and extending them by features and functions. The approach is elegant in many ways. However, it is unclear whether or not the composition of the secrecy property proposed in the approach still works when the secret is not of the form g^x but of the form x. Moreover, no formal proofs of the correctness of the refinement and transformation rules are presented in the paper, even though the validity of some of the rules is evident.

3

Mathematical Preliminaries

This chapter reviews the practical and theoretical background concepts which are necessary for understanding the proposed design methodology. Section 3.1 shows how message terms are modelled in this thesis. Section 3.2 outlines the fundamental operations on sets of terms and their properties. Some useful properties on fake messages of the intruder are shown in the same section. Definitions of basic concepts such as ideals and coideals, events and strands, and bundles and traces, which are borrowed from strand space theory [51, 52], are given in Section 3.3, Section 3.4, and Section 3.5, respectively. The notion of binding groups is also proposed in Section 3.4 to convey the idea that a variable introduced by an agent during a protocol run is always linked to a specific group of agents. Finally, the regularity and discreetness properties of a protocol, as proposed by Millen and Ruess [38], are discussed in Section 3.6.

3.1 Message Terms

The formalisation task begins by defining the primitive data types that may occur as message terms. A is the set of messages that can be sent between agents. In another context we might use "principal" instead of "agent". We call elements of A *terms*. The set of terms A is assumed to be freely generated from two sets T, K:

- The set $\mathsf{T} \subseteq \mathsf{A}$ is a union of a set of Constant terms C and a set of Variable terms V, i.e. $\mathsf{T} = \mathsf{C} \cup \mathsf{V}$. The set C consists of terms which are session-invariant such as protocol numbers or agent names. There are two special agents: a trusted server Srv and an intruder Spy. The set of all agents is denoted as $\mathcal{A}$. The set V consists of terms which are session-variant such as nonces or timestamps. V and C are assumed to be disjoint.

- The set $\mathsf{K} \subseteq \mathsf{A}$ consists of cryptographic keys disjoint from T. Each key can be private or public and short-term or long-term depending on the properties it satisfies. A public key of agent A is denoted as $\mathsf{pub}(A)$, its corresponding private key $\mathsf{prv}(A)$, and a symmetric key $\mathsf{shr}(A)$. The set of long-term keys of agent A is denoted as $\mathsf{K}_L(A)$ and the set of short-term keys as $\mathsf{K}_S(A)$. Short-term keys are assumed symmetric. A symmetric key is denoted as k_{AB} or k_{BA}, where k_{AB} means a shared key used by A to communicate with B. A pair of signature and verification keys of A are denoted as (sk_A, vk_A), and a pair of encryption and decryption keys of A as (ek_A, dk_A). The variable k is used to refer one of these keys.

There are two unary and four binary operators:

- $\mathsf{inv} : \mathsf{K} \rightarrow \mathsf{K}$
  inv maps each member of a public and private key pair to its mate and maps each symmetric key to itself. We write $\mathsf{inv}(k)$ as k^{-1}.

- $\mathsf{key} : \mathsf{A} \rightarrow \mathsf{K}$
  key is a key generation function which cryptographically transforms a term or terms into a key. This function does not necessarily need to be irreversible, but it is assumed irreversible for simplicity.

- $\mathsf{conf} : \mathsf{K} \times \mathsf{A} \rightarrow \mathsf{A}$
  conf is a reversible cryptographic transformation of a message with a key, which

provides confidentiality to the message. Encryption is generally used for providing this property. We write $\mathsf{conf}(k, m)$ as $\{\!|m|\!\}_k$.

- $\mathsf{intg} : \mathsf{K} \times \mathsf{A} \rightarrow \mathsf{A}$
  intg is a reversible cryptographic transformation of a message with a key, which provides integrity to the message. We denote $\mathsf{intg}(k, m)$ as $[m]_k$.

- $\mathsf{hash} : \mathsf{K} \times \mathsf{A} \rightarrow \mathsf{A}$
  hash is an irreversible (one-way) cryptographic transformation of a message with or without a key. We use notation $\langle m \rangle_k$ for $\mathsf{hash}(k, m)$ and $\langle m \rangle$ for $\mathsf{hash}(\emptyset, m)$.

- $\mathsf{join} : \mathsf{A} \times \mathsf{A} \rightarrow \mathsf{A}$
  join is a concatenation operation on messages. We denote $\mathsf{join}(a, b)$ as $a \cdot b$.

We call conf, intg, and hash operations collectively as *cryptographic operations* or *cryptographic transformations* and use $\mathsf{F}_k(\cdot)$ to denote them when the operator itself is unimportant for the discussion. Sometimes, we use $\mathsf{G}_k(\cdot)$ only to refer to either the operation conf or intg. All cryptographic operations are based on the *free encryption assumption* where

$$\mathsf{F}_k(m) = \mathsf{F}_{k'}(m') \Leftrightarrow m = m' \wedge k = k'$$

Definiton 1. (Simple, Composite and Basic Terms) *A term is* simple *if it is not of the form $x \cdot y$. Otherwise, it is* composite*. All concatenated terms are composite. A composite term can be divided into simple terms, and these simple terms are called* components *of the composite term.* Basic *terms are the simple terms in the set $\mathsf{V} \cup \mathsf{K}$ excluding those constructed by encryption.*

Basic terms are the kinds of primitive data types that may be designed to be secrets. Composite terms and Constant terms are never designed as secrets, though some composite terms may have to be protected to maintain the secrecy of some of their components or some Constant terms need to be protected for the protection of privacy.

3.2 Inductive Relations

The fundamental operations on sets $S \subseteq \mathsf{A}$ of terms as introduced by Paulson [41] are $\mathsf{Parts}(S)$, $\mathsf{Analz}(S)$ and $\mathsf{Synth}(S)$. All three sets are defined inductively, as the least set closed under specified extensions. The set $\mathsf{Parts}(S)$ is the set of all subterms of terms in the set S. The set $\mathsf{Parts}(S)$ is obtained from S by repeatedly adding the components of message terms and the bodies of cryptographically transformed message terms generated by $\mathsf{G}_k(\cdot)$ operations. The set $\mathsf{Analz}(S)$ is the subset of $\mathsf{Parts}(S)$ consisting of only those subterms that are accessible to Spy. The set $\mathsf{Analz}(S)$ is obtained from S by repeatedly adding the components of message terms and the bodies of cryptographically transformed message terms generated by operations $\mathsf{G}_k(\cdot)$ if k^{-1} is in $\mathsf{Analz}(S)$. The set $\mathsf{Synth}(S)$ is the set of message terms constructible by Spy given the elements of S. The set $\mathsf{Synth}(S)$ is obtained by adding all message terms constructible by concatenation and cryptographic operations using terms and keys in S. These sets are formally defined as follows.

Definiton 2. (Parts, Analz, and Synth) *Let S be a set of terms. The set $\mathsf{Parts}(S)$ is the smallest extension of S obtained by recursively including each component of concatenations and the bodies of $\mathsf{G}_k(\cdot)$ transformed message terms.*

- $S \subseteq \mathsf{Parts}(S)$
- *if* $x \cdot y \in \mathsf{Parts}(S)$, *then* $x \in \mathsf{Parts}(S)$ *and* $y \in \mathsf{Parts}(S)$
- *if* $\mathsf{G}_k(x) \in \mathsf{Parts}(S)$, *then* $x \in \mathsf{Parts}(S)$

Given a term x, a term $y \in \mathsf{Parts}(x)$ is called a subterm of x and denoted as $y \sqsubseteq x$.

The set $\mathsf{Analz}(S)$ is the smallest extension of S closed under projection and inverse operations by keys in $\mathsf{Analz}(S)$.

- $S \subseteq \mathsf{Analz}(S)$
- *if* $x \cdot y \in \mathsf{Analz}(S)$, *then* $x \in \mathsf{Analz}(S)$ *and* $y \in \mathsf{Analz}(S)$
- *if* $\mathsf{G}_k(x) \in \mathsf{Analz}(S)$ *and* $k^{-1} \in \mathsf{Analz}(S)$, *then* $x \in \mathsf{Analz}(S)$

Finally, the set $\mathsf{Synth}(S)$ is the smallest extension of S closed under concatenation and $\mathsf{F}_k(\cdot)$ transformation.

- $S \subseteq \mathsf{Synth}(S)$
- *if* $x \in \mathsf{Synth}(S)$ *and* $y \in \mathsf{Synth}(S)$, *then* $x \cdot y \in \mathsf{Synth}(S)$

- *if* $x \in \mathsf{Synth}(S)$ *and* $k \in \mathsf{Synth}(S)$, *then* $\mathsf{F}_k(x) \in \mathsf{Synth}(S)$

The following properties are stated, for similarly defined sets in [41]. They are all proved by straight forward inductions.

Proposition 1. *The set operations* $\mathsf{Parts}(S)$, $\mathsf{Analz}(S)$, *and* $\mathsf{Synth}(S)$ *are closure operators, i.e. they are extensive*[1]*, monotonic, and idempotent. Furthermore:*

$$\begin{array}{rcl} \mathsf{Parts}(\mathsf{Analz}(S)) & = & \mathsf{Parts}(S) \\ \mathsf{Analz}(\mathsf{Parts}(S)) & = & \mathsf{Parts}(S) \\ \mathsf{Parts}(\mathsf{Synth}(S)) & = & \mathsf{Parts}(S) \cup \mathsf{Synth}(S) \\ \mathsf{Analz}(\mathsf{Synth}(S)) & = & \mathsf{Analz}(S) \cup \mathsf{Synth}(S) \end{array}$$

Spy can generate fake messages from analysable parts of a set of available terms. This gives the definition of $\mathsf{Fake}(S)$.

Definiton 3. (Fake) $\mathsf{Fake}(S) = \mathsf{Synth}(\mathsf{Analz}(S))$.

Some useful properties on $\mathsf{Parts}(\mathsf{Fake}(S))$ can be easily obtained from the definition of Fake.

Lemma 1. (Fake-Parts)

1. $\mathsf{Parts}(\mathsf{Fake}(S)) = \mathsf{Parts}(S) \cup \mathsf{Fake}(S)$

2. $\mathsf{Parts}(\mathsf{Fake}(S)) \subseteq \mathsf{Synth}(\mathsf{Parts}(S))$

Proof. Using the properties in Proposition 1.

$$\begin{array}{rcl} \mathsf{Parts}(\mathsf{Fake}(S)) & = & \mathsf{Parts}(\mathsf{Synth}(\mathsf{Analz}(S))) \\ & = & \mathsf{Parts}(\mathsf{Analz}(S)) \cup \mathsf{Synth}(\mathsf{Analz}(S)) \\ & = & \mathsf{Parts}(S) \cup \mathsf{Synth}(\mathsf{Analz}(S)) \\ & = & \mathsf{Parts}(S) \cup \mathsf{Fake}(S) \end{array}$$

Moreover, $\mathsf{Analz}(S) \subseteq \mathsf{Parts}(S) \subseteq \mathsf{Synth}(\mathsf{Parts}(S))$, therefore,

$$\begin{array}{rcl} \mathsf{Parts}(\mathsf{Fake}(S)) & = & \mathsf{Parts}(S) \cup \mathsf{Synth}(\mathsf{Analz}(S)) \\ & \subseteq & \mathsf{Synth}(\mathsf{Parts}(S)) \end{array}$$

□

[1] For example, $S \subseteq \mathsf{Parts}(S)$.

3.3 Ideals and Coideals

If S is a set of basic terms, then there exists a smallest set which should be protected from Spy in order not to reveal any information on the basic terms in S [38, 51]. This set is called the *ideal* of S and denoted as $\mathcal{I}[S]$. The ideal is the smallest set of terms that includes S and which is closed under concatenation with any terms and under encryption with keys whose inverses are not in S. The k-ideal of S is denoted as $\mathcal{I}_{\mathsf{k}}[S]$, where k is the set of keys belonging to S. To protect a set of secrets S, all members of $\mathcal{I}_{\mathsf{k}}[S]$ should be protected from Spy. The ideal is formally defined as follows.

Definiton 4. (Ideals) *Let k be a set of keys belonging to S. The k-ideal of S, $\mathcal{I}_{\mathsf{k}}[S]$ is the smallest set such that*

1. $S \subseteq \mathcal{I}_{\mathsf{k}}[S]$
2. *if $x \in \mathcal{I}_{\mathsf{k}}[S]$ or $y \in \mathcal{I}_{\mathsf{k}}[S]$, then $x \cdot y$, $y \cdot x \in \mathcal{I}_{\mathsf{k}}[S]$*
3. *If $x \in \mathcal{I}_{\mathsf{k}}[S]$ and $k^{-1} \notin S$, then $\mathsf{G}_k(x) \in \mathcal{I}_{\mathsf{k}}[S]$*

The smallest k-ideal containing x is denoted as $\mathcal{I}_{\mathsf{k}}[x]$. If $\mathsf{k} = \mathsf{K}$, then the ideal of S is the closure of S under concatenation. We use $\mathcal{I}[\cdot]$ instead of $\mathcal{I}_{\mathsf{k}}[\cdot]$ if k is clear from the context.

Under the assumption that any term not in the ideal may be already compromised to Spy, it is necessary to protect this whole ideal because compromising any element of the ideal effectively compromises some element of S.

It will be useful to note that any term in the ideal must have a subterm in the generating set S especially if S is constructed from a set of basic terms. This can be easily proved by induction, but it is evident from the definition of the ideal.

Lemma 2. *If $x \in \mathcal{I}[S]$, then there exists $y \sqsubseteq x$ such that $y \in S$ and y is basic.*

The complement of an ideal, which is called a *coideal* [38,51], is denoted by $\mathcal{C}[S]$. The coideal $\mathcal{C}[S]$ defines the set of terms that are public with respect to the set of secrets S, i.e. terms whose release would not compromise any secrets in S.

Definiton 5. (Coideals) *Given a set S and its ideal $\mathcal{I}[S]$, the coideal of S is the complement of the ideal of S and denoted as $\mathcal{C}[S]$.*

Coideals are interesting because they are closed under Spy analysis, thereby implying that protection of the ideal is sufficient.

Lemma 3. (Analz Closure) *For a set S of terms:*

$$\mathsf{Analz}(\mathcal{C}[S]) = \mathcal{C}[S]$$

Proof. From the extensive property of $\mathsf{Analz}(\cdot)$

$$\mathcal{C}[S] \subseteq \mathsf{Analz}(\mathcal{C}[S])$$

We now have to show

$$\mathsf{Analz}(\mathcal{C}[S]) \subseteq \mathcal{C}[S]$$

This can be done by showing that $\mathcal{C}[S]$ is closed under the rules that expand $\mathsf{Analz}(\cdot)$. First, suppose $x \cdot y \in \mathcal{C}[S]$, that is, $x \cdot y \notin \mathcal{I}[S]$. We have to show $x, y \in \mathcal{C}[S]$. From the definition of the ideal, if $x \in \mathcal{I}[S]$, then $x \cdot y \in \mathcal{I}[S]$. This contradicts the assumption that $x \cdot y \in \mathcal{C}[S]$. Therefore, neither of x nor y is in $\mathcal{I}[S]$, i.e. $x, y \in \mathcal{C}[S]$. Second, suppose $\mathsf{G}_k(x) \in \mathcal{C}[S]$ and $k^{-1} \in \mathcal{C}[S]$, then we have to show $x \in \mathcal{C}[S]$. From the definition of the ideal, $\mathsf{G}_k(x) \in \mathcal{C}[S]$ implies either $x \notin \mathcal{I}[S]$, or $x \in \mathcal{I}[S]$ and $k^{-1} \in \mathcal{I}[S]$. The first subcase means $x \in \mathcal{C}[S]$ and the second subcase contradicts the assumption that $k^{-1} \in \mathcal{C}[S]$. Therefore, $x \in \mathcal{C}[S]$. □

A similar result can be proved for $\mathsf{Synth}(\cdot)$ when the elements generating the coideal are basic.

Lemma 4. (Synth Closure) *For a set S of basic terms:*

$$\mathsf{Synth}(\mathcal{C}[S]) = \mathcal{C}[S]$$

Proof. From the extensive property of $\mathsf{Synth}(\cdot)$

$$\mathcal{C}[S] \subseteq \mathsf{Synth}(\mathcal{C}[S])$$

Now we have to show

$$\mathsf{Synth}(\mathcal{C}[S]) \subseteq \mathcal{C}[S]$$

This can be done by showing that $\mathcal{C}[S]$ is closed under the rules that expand $\mathsf{Synth}(\cdot)$. First, suppose $x, y \in \mathcal{C}[S]$. We have to show $x \cdot y \in \mathcal{C}[S]$. In contradiction, assume

$x \cdot y \in \mathcal{I}[S]$. From the definition of the ideal, $x \cdot y \in \mathcal{I}[S]$ either because $x \cdot y \in S$ or $x, y \in \mathcal{I}[S]$. The first subcase is impossible because S is composed of basic terms, and the second subcase contradicts the assumption that $x, y \in \mathcal{C}[S]$. Hence, $x \cdot y \in \mathcal{C}[S]$. We now have to show that $\mathsf{F}_k(x) \in \mathcal{C}[S]$ under the assumption $x, k \in \mathcal{C}[S]$. $\mathsf{F}_k(x)$ implies $\mathsf{G}_k(x)$ or $\langle x \rangle_k$. Assume that $\mathsf{G}_k(x) \in \mathcal{I}[S]$, either because $\mathsf{G}_k(x) \in S$, or $x \in \mathcal{I}[S]$ and $k^{-1} \notin \mathcal{I}[S]$. The first subcase is impossible because S is a set of basic terms, and the second subcase contradicts the hypothesis that $x \in \mathcal{C}[S]$. The same argument can be applied for $\langle x \rangle_k$. Therefore, $\mathsf{F}_k(x) \in \mathcal{C}[S]$. □

From the definition of the ideal, we can inductively define two different types of sets of keys: *penetrable keys* and *safe keys* [52]. The set of penetrable keys is the set of keys that may become known to Spy, and the set of safe keys is the set of keys which are safe from Spy analysis.

Definiton 6. (Penetrable Keys) *Let* P_0 *be the set of keys which are initially known to* Spy *before any protocol activity. The set of penetrable keys* $\mathsf{K_P}$ *is recursively defined as follows.*

1. $\mathsf{K}_{\mathsf{P}_0} = \mathsf{P}_0$
2. $\mathsf{K}_{\mathsf{P}_{i+1}} = \mathsf{K}_{\mathsf{P}_i} \cup \mathsf{X}$, *where* $k \in \mathsf{X}$ *iff there is a term* x *originating from an agent and* $x \in \mathcal{I}_{\mathsf{K}_{\mathsf{P}_i}^{-1}}[k]$
3. $\mathsf{K_P} = \bigcup_i \mathsf{K}_{\mathsf{P}_i}$

Thus, a key becomes a penetrable key either because the key is already penetrated before any protocol activity or because some regular agent, during the protocol runs, emits the key in a message that could be penetrable by Spy.

Definiton 7. (Safe Keys) *The set of safe keys is defined as* $\mathsf{K_S} = \mathsf{K} - \mathsf{K_P}$.

3.4 Events and Strands

Agents are non-deterministic machines with some internal state, communicating over a public channel by sending and receiving messages. The internal state of an agent determines what action an agent will take next. The behaviour of some agents is determined by a protocol. Some agents such as Spy will not adhere to the protocol. She can behave arbitrarily. Other agents may be partially faithful, for example, clients in an electronic commerce protocol may try to cheat. We view a protocol as a constraint on the possible actions of agents. More specifically, each agent is associated with a *next action* defined by the protocol, which maps its current state into the set of possible next actions. Spy is generally considered unconstrained by the protocol, while faithful agents such as Srv are fully constrained.

As a first step towards formalising such a view of agents, we define an *event*, which is an action of agent. Events take one of the following forms: $e^+(x)$ indicating that term x is sent by an agent; $e^-(x)$ indicating that term x is received by an agent; (νx) indicating that a basic term x is generated by an agent; and $e^-(x/p(x))$ matching a term x against a pattern $p(x)$. We use e to refer to one of actions such as $e^+(x)$, $e^-(x)$, (νx), $(x/p(x))$. An event can be subscripted by an agent identifer like e_A to indicate where the event happens.

Definiton 8. (Events) *An event e is an action which an agent can perform to interact with other agents, i.e. $e ::= e^+(x)|e^-(x)|(\nu x)|(x/p(x))$. The events $e^+(x)$ and $e^-(x)$ are called external events and others are called internal events. The contents of a message of an external event e is denoted by $m(e)$.*

We divide events into two categories: *internal* and *external* events. By internal events, we mean events which happen within an agent. The contents of an internal event cannot be seen by other agents unless there is an external event which carries the contents. Hence, we only define the contents of external events. Actions such as decryption and verification of terms are basically the same, so we use pattern matching actions for both purposes. Variable generation actions need special attention. Whenever a new variable term is introduced in a protocol run, i.e. when an action (νx) has been done by some agent A, there is a group of agents to whom x is supposed to mean something. For example, if A creates x as a secret, then there will be a group of agents who are permitted to share the secret with A. If A generates x as an authentication challenge, there will be a set of agents whom A wants to authenticate using x. This group of agents is called the *binding group* of x.

Definiton 9. (Binding Group) *When a new variable term x is introduced by an agent A during a protocol run, there is a group of agents who are linked to x by A. This group of agents is called the binding group of x and denoted as $\mathcal{B}_x$. When x is used as a secret, the binding group of x is called the secret group of x. Sometimes, we use notation $(\nu x : \mathcal{B}_x)$ instead of (νx) in order to specify the binding group of a variable x.*

Each agent is assumed to have a local notion of "before" and "after" such as Lamport's clocks [30] that gives a total order on each agent's events. These local clocks of agents define an order between events. If event e_i happens before event e_j, then it is denoted as $e_i < e_j$ and we say that e_i precedes e_j.

Axiom 1. (Order Restriction)

1. $e^+(x) < e^-(x)$

2. $(\nu x) < e^+(x)$

This axiom simply says that all sending events of a term x precede all receiving events of the term x, and a new variable term cannot be sent before it is generated, that is, no events on the right side can happen without their preceding events. Therefore, the existence of the events on the right side always means the existence of the events on the left side. Two events are independent if no order restriction exists between them.

Definiton 10. (Independent Events) *Two events e_i and e_j are independent if $e_i \not< e_j$ and $e_j \not< e_i$.*

If there is an order restriction between events, then the order relation between the events can be obtained through the application of Axiom 1. In case of independent events, the protocol specification has to provide the order relations among the independent events to avoid ambiguity. This way, an agent can always predict which action it should take next, and can define a total order among events it has done or seen so far.

When an agent interacts with other agents, not all actions are visible to the agent. An agent can only see either actions done by itself or their corresponding external actions done by other agents. A sequence of actions which an agent has done is called a *strand*. A parametric strand shows an agent's local view of a protocol run defined by a parameter known to the agent.

Definiton 11. (Strand) *A parametric strand $\xi(\omega)$ is a sequence of actions which have been done by one particular agent with a parameter "ω". We subscript it by the agent identifier like $\xi_A(\omega)$ to display to whom the strand belongs, or we use $|e_1, e_2, ..., e_m|_A$ to mention the sequence itself. If a particular event e is a part of the sequence $\xi_A(\omega)$, then we say e is in the strand and denote it as $e \in \xi_A(\omega)$. The number of events in a strand is called the height of the strand and denoted $h(\xi(\omega))$.*

Parametric strands are called *strands* for short with the notation ξ instead of $\xi(\omega)$. Every strand has a total order among its events defined by the agent's local clock and the order restrictions imposed on its events.

3.5 Bundles and Traces

A *strand space* $\Sigma = \{\xi_{A_i} : A_i \in \mathcal{A}\}$ is a set of strands [25, 52]. A set of strands $\psi \subseteq \Sigma$ can interact with each other if message exchanges exist between them, that is, there are at least two distinct strands ξ_{A_i}, ξ_{A_j}, where for some term x, $e^+(x) \in \xi_{A_i}$ and $e^-(x) \in \xi_{A_j}$. Such a collection of strands which interact with each other is called a *bundle*. For convenience, we write $e \in \psi$ if $e \in \xi_{A_i}$ where $\xi_{A_i} \in \psi$.

Definiton 12. (Bundle) *Let ψ be a set of strands, i.e. $\psi = \{\xi_{A_i} : A_i \in \mathcal{A}\}$. ψ is a bundle if:*

1. *ψ is finite, i.e. each ξ_{A_i} has a finite height.*
2. *If $e^-(t) \in \xi_{A_i}$, then there is a unique strand $\xi_{A_j} \in \psi$, where $e^+(t) \in \xi_{A_j}$. This relationship is written as $e^+(t) \rightarrow e^-(t)$.*
3. *ψ is prefix-closed, i.e. if $e_i \in \psi$ and e_j precedes e_i, then $e_j \in \psi$. When the preceding event is in the same strand, i.e. $e_j, e_i \in \xi_{A_i}$, this relationship is written as $e_j \Downarrow e_i$.*
4. *ψ is acyclic, i.e. no two events e_i and e_j satisfy both $e_i < e_j$ and $e_j < e_i$.*

We do not assume that the communication medium is secure, and so we do not expect that a message is received only by its intended receiver. Rather, we treat message send events as broadcasts to the world; any agent can receive any message that is sent. Further, we make no assumptions about the correct behaviour of the network — messages may be completely lost, received by only some agents, or even duplicated due to network failures and errors. This view is reflected in the definition of the bundle. In a bundle, when a strand receives a message term x, there is a unique strand sending x from which the message was immediately received. By contrast, when a strand sends a message term x, many strands or none may immediately receive x.

Definiton 13. (Bundle Equivalence) *Two bundles ψ_i and ψ_j are equivalent iff for all $e_A \in \psi_i \Leftrightarrow e_A \in \psi_j$, where A is not* Spy. *A set ϕ of bundles is invariant under bundle equivalences if $\psi_i \in \phi$ and ψ_j is equivalent to ψ_j implies $\psi_j \in \phi$.*

The concept of a bundle is important: all possible runs of a protocol can be represented as bundles, and almost all correctness properties can be stated as properties of bundles. The agreement and non-injective agreement properties of Lowe [33] are invariant under bundle equivalences. For instance, a non-injective agreement property asserts that whenever a bundle contains a protocol strand (for instance, a responder strand) of a certain height, then it also contains a matching strand (for instance, an initiator strand using the same parameter) of a suitable height. A bundle is an entangled structure of a set of strands; a linearised version of a bundle is called a *trace*.

Definiton 14. (Traces) *A trace of ψ, $\mathcal{TR}(\psi)$ is a totally ordered set of events which can occur when events in the strands of ψ happen without violating local order relations of each strand and order restrictions among events. The contents of a trace is denoted by $m(\mathcal{TR})$, which is the union of all contents of the events in the trace.*

The definition above does not admit traces in which two events happen simultaneously. Such a possibility could be accommodated by replacing the notion of a total order with the more general notion of a total pre-order. However, the extra structure afforded by simultaneous events is inconsequential for the security properties.

A trace is a record of actions done so far by agents. Therefore any prefix of a trace is also a trace.

Lemma 5. (Prefix Closure) *If $\mathcal{TR}$ is a trace, then any prefix subsequence of $\mathcal{TR}$ is a trace.*

The definition of a trace captures a notion of agent interaction in which agents are completely free to send and receive messages. However, this notion does not take into account the constraints on interaction imposed by protocol specifications or environments. Therefore, there is no limit to the number of intermediate events that can occur between any two events. This is not true in real situations, so it is useful to restrict the number of events between any two events to be finite. This restriction is imposed by the assumption of Definition 12 that every bundle has a finite height, so traces in this thesis are bounded.

Unused terms are those terms that do not appear in the trace, and which can be used for nonces or secrets.

Definiton 15. (Unused Terms) *A variable t is unused in $\mathcal{TR}$ if t is basic, $t \not\sqsubseteq (\nu x) \in \mathcal{TR}$, and $t \notin \mathsf{Parts}(m(\mathcal{TR}))$. The set of unused variables in $\mathcal{TR}$ is denoted by $\mathsf{Unused}(\mathcal{TR})$*

3.6 Protocols and Secrecy

A protocol is defined as a set of rules which allow principals to generate traces by adding actions, either performed by honest principals faithfully following the protocol specification or introduced by Spy activity.

Definiton 16. (Protocol and Run) *A protocol $\mathcal{P}$ is a set of rules which enable agents to add an event e to a trace $\mathcal{TR}$. Given a trace $\mathcal{TR}$, if the next event defined by the protocol is e, then this rule is denoted as $(\mathcal{TR}, e) \in \mathcal{P}$. The runs of a protocol, $\mathsf{Runs}(\mathcal{P})$, is a set of traces that could be generated by a protocol $\mathcal{P}$ in an environment with Spy activity. Let I be a set of terms known to Spy, then $\mathsf{Runs}(\mathcal{P})$ is defined inductively:*

Rule 1 *: $\epsilon \in \mathsf{Runs}(\mathcal{P})$*

Rule 2 *: If $\mathcal{TR} \in \mathsf{Runs}(\mathcal{P})$ and $(\mathcal{TR}, e) \in \mathcal{P}$, then $\mathcal{TR} \cdot e \in \mathsf{Runs}(\mathcal{P})$, where $\mathcal{TR} \cdot e$ represents a trace obtained by adding e at the end of $\mathcal{TR}$.*

Rule 3 *: If $\mathcal{TR} \in \mathsf{Runs}(\mathcal{P})$ and e is an action done by Spy with $m(e) \in \mathsf{Fake}(m(\mathcal{TR}) \cup \mathsf{I})$, then $\mathcal{TR} \cdot e \in \mathsf{Runs}(\mathcal{P})$.*

Rule 2 corresponds to the way of expanding a trace by an honest agent action, while **Rule 3** corresponds to the introduction of terms by Spy based on the Dolev-Yao intruder model [17].

In each protocol run, terms such as long-term keys of principals or short-term secrets of the run have to be protected from Spy analysis. This set of secrets which should be protected from Spy is defined as the *protective domain* of a protocol run.

Definiton 17. (Protective Domains) *The x-protective domain S_x of a secret x is defined as follows:*

$$\mathsf{S}_x = \{x\} \cup \{\mathsf{prv}(A) | A \in \mathcal{B}_x\} \cup \{\mathsf{shr}(A) | A \in \mathcal{B}_x\}$$

In order to protect x from Spy, not only x but also private keys and shared keys of the members of $\mathcal{B}_x$ should be protected because compromise of these keys may reveal x to Spy. Therefore, all of these keys should be elements of the x-protective domain. Notice that only variable terms are allowed to be secrets.

A x-protective domain S_x is compatible with an initial knowledge set I if I does not contain any of the secrets in S_x as its subterms.

Definiton 18. (I-Compatibility) S_x *is compatible with* I *if* $\mathsf{S}_x \cap \mathsf{Parts}(\mathsf{I}) = \emptyset$, *and this kind of protective domain is called* I*-compatible.*

The following two lemmas are derivable from the definition of I-compatibility.

Lemma 6. *If* S_x *is* I*-compatible, then* $\mathcal{I}[\mathsf{S}_x] \cap \mathsf{Parts}(\mathsf{I}) = \emptyset$.

Lemma 7. (Compatible Coideal) *If* S_x *is* I*-compatible, then* $\mathsf{I} \subseteq \mathcal{C}[\mathsf{S}_x]$.

According to Paulson's regularity results [41], Spy never gets hold of any agent's long-term keys if they are never introduced by an honest agent as parts of messages. A protocol is called *regular* if it protects long-term secrets this way.

Definiton 19. (Regularity) *Let* K_L *be a set of long-term keys to be protected. A protocol* $\mathcal{P}$ *is regular if:*

$\forall k \in K_L$, *if* $(\mathcal{TR}, e) \in \mathcal{P}$ *and* $k \notin \mathsf{Parts}(m(\mathcal{TR}))$, *then* $k \notin \mathsf{Parts}(m(e))$

Checking whether a protocol is regular is easy because we only need to see whether long-term keys are used as parts of messages. Protocols used for long-term key distribution purposes cannot be regular because long-term keys have to be introduced and delivered as parts of messages. However, session keys can be distributed without affecting the regularity property. From a design point of view, regularity simply means that no long-term keys should be used as parts of messages in the process of a protocol construction.

From the definition of the regularity, the regularity lemma follows.

Lemma 8. (Regularity) *Let* K_L *be a set of long-term keys to be protected. Suppose that*

1. $\mathcal{P}$ *is regular*
2. $k \in K_L$
3. $k \notin \mathsf{Parts}(\mathsf{I})$
4. $\mathcal{TR} \in \mathsf{Runs}(\mathcal{P})$

then $k \notin \mathsf{Parts}(m(\mathcal{TR}))$.

Suppose S_x is a set of secret terms. The smallest set we should protect from Spy in order not to reveal any element in S_x is $\mathcal{I}[\mathsf{S}_x]$. If S_x is I-compatible, no terms in $\mathcal{I}[\mathsf{S}_x]$ are obtainable by Spy from I (Lemma 6), as long as a trace generated by a protocol $\mathcal{P}$ does not give any extra information to Spy. A trace which does not expose any terms in $\mathcal{I}[\mathsf{S}_x]$ is called *discreet* [38].

Definiton 20. (Discreet Trace) *A trace* $\mathcal{TR}$ *is* I*-discreet, if for all* x*-protective domains* S_x *that are compatible with* I*,*

$$m(\mathcal{TR}) \subseteq \mathcal{C}[\mathsf{S}_x]$$

A protocol is secure with respect to its secrecy policy and the Spy's initial knowledge if every trace of possible runs of the protocol is discreet. The secrecy proof of a protocol has a protocol independent part and a protocol dependent part. The protocol dependent part is expressed by the discreet property defined below. It says that if the prior trace is discreet, the next message event generated by the protocol does not compromise a secret. This has to be proved individually for each protocol.

Definiton 21. (Discreet Protocol) *A protocol* $\mathcal{P}$ *is discreet if for all* $\mathsf{Runs}(\mathcal{P})$*,* I*, and a secret* x *satisfying the conditions*

1. $(\nu x) \in \mathcal{TR} \in \mathsf{Runs}(\mathcal{P})$ *such that* $\mathcal{TR}$ *is* I*-discreet*

2. $(\mathcal{TR} \cdot e) \in \mathsf{Runs}(\mathcal{P})$

3. S_x *is compatible with* I

it is the case that $m(e) \subseteq \mathcal{C}[\mathsf{S}_x]$.

The protocol independent part of a secrecy proof is the Secrecy theorem [38].

Theorem 1. (Secrecy) *If* $\mathcal{P}$ *is regular and discreet, then every trace in* $\mathsf{Runs}(\mathcal{P})$ *is* I*-discreet.*

Proof. This can be proved by induction on the trace $\mathcal{TR}$. □

4

Composition Theory

This chapter proposes the concepts of agreement tests and discreetness, which enable authentication and secrecy to be composable. Protocol primitives are implemented based on the two proposed concepts and will later be used as basic building blocks in composing complex protocols. Section 4.1 reviews the idea of authentication tests and extends it using the agreement property in order to make authentication composable. The section also shows how to implement protocol primitives which guarantee a matching record of a run. Section 4.2 discusses the non-destructive property of authentication and shows how to make protocol primitives non-destructive for composition of authentication. The section proves regularity and discreetness of the proposed primitives and their composition, which is important for the secrecy of sensitive data within the primitives. Finally, Section 4.3 discusses how to integrate each independent primitive into a single protocol so that all independent goals accomplished by each primitive happen in a single session.

4.1 Authentication with Agreement

Challenge and response mechanisms, or authentication tests [26] are common methods for establishing authentication results. Many authentication protocols depend on these schemes to achieve their authentication goals. It is easy to implement a protocol accomplishing a single independent one-way authentication goal using an authentication test. A multi-way authentication protocol can be seen as a set of one-way authentication protocols. This opens the possibility of implementing a two-way authentication protocol by combining two applications of a secure one-way authentication protocol. Unfortunately, this method generally does not work because two-way authentication is not simply twice one-way [6]. This does not necessarily mean that authentication itself is not composable or that it cannot be composable. If authentication is composable, then the design of complex protocols will be easier. Therefore, it is important to understand whether or not authentication is composable. The main focus of this section is to understand whether a new protocol can be built from several independent protocols, preserving the goals already achieved by each independent protocol. This is what composition means in this thesis.

The section starts by reviewing the definition of authentication tests, which provide mechanical ways of implementing one-way authentication, and then shows why authentication tests are not composable.

4.1.1 Authentication Tests

The concept of authentication tests [26] is a generalisation of the challenge and response mechanism. The basic idea behind the authentication test is simple. There are some operations that Spy cannot do, so Spy cannot apply any non-trivial actions to the messages containing the outputs of those operations. Those messages may be discarded by Spy but if they are delivered to an honest participant, they will be delivered unaltered. If one of those transformations of the message appears some time after a message was sent, then only honest participants are responsible for the generation of the transformed message. Therefore, this pair of messages may be regarded as an authentication test. There are two main types of authentication tests: *outgoing* and *incoming*, depending on the forms of challenge and response messages. An unsolicited test is a weaker version of an authentication test which does not provide any guarantee of the recency of the received message. The original definition of authentication tests is as follows [26]:

Definiton 22. (Authentication Tests & Unsolicited Tests) *Assume that $x \in \mathsf{V}$, $k \in \mathsf{K_S}$ and $k' = \mathsf{inv}(k)$, then incoming and outgoing authentication tests are defined as follows:*

Outgoing Tests *: If a term x has been transmitted in encrypted form $\{\!|\cdots x \cdots|\!\}_{k'}$ and is later received as a component of a different composite term, then a participant who possesses the key k must have been responsible for the emission of x.*

Incoming Tests *: If a term x has been transmitted as a component of a composite term and is later received in an encrypted term $y = \{\!|\cdots x \cdots|\!\}_{k}$, then a participant who possesses the key k must have been responsible for the generation of the message y.*

Unsolicited Tests *: If an encrypted term $y = \{\!|\cdots x \cdots|\!\}_{k}$ is received, then a participant who possesses the key k must have been responsible for the generation of the message y. It does not provide the recency of the received message x.*

Authentication tests are basic mechanisms for implementing entity authentication, especially one-way authentication. More specifically, a single authentication test guarantees the aliveness of a specific correspondent who has a specific key. In principle, any pair of messages (x, y) can be an authentication test if the latter can only be generated by the application of cryptographic operations on the former with the help of safe keys. However, the definition above is too general to serve as a definition of authentication in protocol design. The following protocol is an example of an authentication test. The notations used for specifying the protocol are based on the strand space formalism. Each arrow represents a send event and its corresponding receive event(s) as seen by the named principal. Message 1 is ambiguous in the sense that there is no way for B to believe that the message is destined for him before he decrypts it. It should be $B, \{\!|N_a, A|\!\}_{K_B}$. The same goes for Message 2. In the example, these destination identities are omitted because they do not affect secrecy.

$$\begin{array}{lll} \text{Message 1} & A \rightarrow * : & \{\!|N_a, A|\!\}_{K_B} \\ \text{Message 2} & A \leftarrow * : & N_a \end{array}$$

When A finished the protocol above, what can she conclude? Can she say that B replied to her when she received Message 2? Unfortunately, there is nothing she can conclude from the example protocol except that B has recently performed some cryptographic transformations using his safe key in response to someone's request containing N_a. If B is a bad guy, then Message 2 may not come directly from B, but it may come from some other participant C ($\neq B$) who believes that he is talking with A:

$$\begin{array}{lll} \text{Message 1} & A \rightarrow * : & \{\!|N_a, A|\!\}_{K_B} \\ \text{Message 1}' & B \rightarrow * : & \{\!|N_a, A|\!\}_{K_C} \\ \text{Message 2} & A \leftarrow * : & N_a \end{array}$$

This might be acceptable in some situations but can cause serious problems in others. As pointed out by many authors [20, 33, 47, 48], authentication comes in a number of flavours. For example, Gollmann has identified four different varieties, which raises the question of which kind of authentication a given protocol is designed for, and which kind it actually provides. Therefore, without knowing exactly what authentication means, it seems to be almost impossible to design protocols correctly. Technically speaking, the definition of authentication tests covers the so called "recent aliveness" of a correspondent among many possible definitions of authentication [20, 33]. For most authentication protocols, especially for protocols which aim for mutual authentication, the definition of authentication tests is too abstract to provide a guideline or a tool for implementing correct authentication mechanisms. Particularly for the composition of authentication, a stronger definition of authentication is required. Two most important properties necessary for authentication to be composable are *non-destructiveness* and *non-constructiveness*. By the non-destructiveness of a property, we mean that a property should remain true starting from the point it becomes true. By the non-constructiveness of a property, we mean that a property should not be true without the proper event which makes it true. In the previous example, the authentication property is in a sense constructive. Although, there is no event originating from B which makes the authentication true between A and B, the authentication unexpectedly holds true. This happens because of ambiguities among messages. The message generated by B in response to A, and the message generated by C in response to B are identical in the example. There is no way for A to differentiate between them. Therefore, either of the two can be used to achieve the authentication. This makes the authentication property implemented in the example constructive, and thus too weak.

It is generally required that a participant B should not finish running the protocol believing that he has been running it with a participant A, unless A also believes that he has been running the protocol with B. In other words, there should be a matching record of a run for each participant who is running an instance of the protocol. Gavin Lowe [33] calls this property *agreement*.

Definiton 23. (Agreement) *A protocol guarantees to an initiator A agreement with a responder B on a set of data items x, if whenever A (acting as initiator) completes a run of the protocol, apparently with responder B, then B has previously been running the protocol, apparently with A, and B was acting as responder in his run, and the two participants agreed on the data values corresponding to all the variables in x, and each such run of A corresponds to a unique run of B.*

This is the strongest authentication that a protocol can accomplish. As already seen in the previous example, not all authentication tests satisfy this property. Some authentication goals such as “aliveness” and “recent aliveness” might be achievable without using authentication tests satisfying the agreement property, but using stronger mechanisms than necessary does not cause any harm. The reason that we adopt the agreement property as the meaning of authentication is that for most cases of protocol design and verification, we need to check not only whether some cryptographic operations have recently been done by a particular agent but also for whom those operations have been done. The former corresponds to authentication tests and the latter to the agreement property. Many flaws in published authentication protocols are due to the lack of proper inspections of the agreement property.

Authentication tests and the agreement property are easily formalised using strand space theory. Suppose ψ be a bundle which contains all events of an instance of an authentication test. Let $\xi^{+}_{AB}(\omega)$ be a strand[1] of A playing an initiator role supposedly with a responder B and $\xi^{-}_{BA}(\omega)$ a strand of B playing a responder role supposedly with an initiator A, where ω is the parameter of the run. For an authentication test satisfying the agreement property, whenever a participant finishes its part of a run of the test, there is always its corresponding run in its partner's strand.

Lemma 9. *If ψ is a bundle built up through an authentication test, then for any authentication tests satisfying the agreement property:*

$$\forall \psi \bullet \xi^{+}_{AB}(\omega) \in \psi \Rightarrow \xi^{-}_{BA}(\omega) \in \psi$$

[1]This notation can be understood as a generalisation of $\xi_A(\omega)$. $+/-$ represents a role of a participant during a single authentication test, it does not necessarily mean a role of the participant in a protocol run.

Let $[\xi^-_{BA}(\omega)]$ represent a strand of B which contains all B's actions prior to the last action of $\xi^+_{AB}(\omega)$, and let $[\xi^+_{AB}(\omega)]$ represent a strand of A which contains all A's actions prior to the last action of $\xi^-_{BA}(\omega)$, then the following are true for any authentication tests satisfying the agreement property.

$$\forall \psi \bullet \xi^+_{AB}(\omega) \in \psi \Rightarrow [\xi^-_{BA}(\omega)] \in \psi$$

$$\forall \psi \bullet \xi^-_{BA}(\omega) \in \psi \Rightarrow [\xi^+_{AB}(\omega)] \in \psi$$

Proof. All properties are obvious from the definitions of authentication tests and the agreement property. □

Authentication tests which satisfy the agreement property are called *agreement tests*.

Definiton 24. (Agreement Tests) *An authentication test is an agreement test if it satisfies the agreement property.*

An agreement test is an enhanced version of an authentication test which guarantees authentication with the agreement property. This test is one of the strongest authentication mechanisms. It can be used for any kind of authentication. In order to implement mechanisms for agreement tests, the concept of term bindings is introduced next.

4.1.2 Term Bindings and Protocol Primitives

When a participant A generates a variable such as a nonce or a secret during a protocol run, there is a group of participants to whom she links the variable. This group is called the *binding group* of the variable. For example, if A generates a nonce x to authenticate B, then she will link x to A as initiator and B as responder. The linking between a variable and its binding group is called a *term binding*.

Definiton 25. (Term Bindings) *Given a variable x and its binding group $\mathcal{B}_x$, we say that x is term-bound to $\mathcal{B}_x$ and denoted as $(x; \mathcal{B}_x)$.*

A variable can be bound to another variable, but at the moment we will not discuss this issue. Here, all term bindings are assumed to mean a linking between a variable and a set of agents, and $\mathcal{B}_x$ is considered to be an ordered tuple, that is, (A, B) is not equal to (B, A). The meaning of each element in $\mathcal{B}_x$ is assumed to be defined by a protocol specification, so more specifically, it should be written as $(A^{i(1)}, B^{i(2)})$, where superscript $i(j)$ means the interpretation of the j-th element in the set. As an example, authentication tests between A as an initiator and B as a responder using x are written as $(x; (A^{i(1)}, B^{i(2)}))$, where $i(1)$ is an initiator and $i(2)$ is a responder. However, except for the case when it is necessary to specify the interpretation of each element, it is assumed that $i(1)$ is an initiator and $i(2)$ a responder. It is generally assumed that the term-binding between x and $\mathcal{B}_x$ is unique, i.e. if an honest agent binds a variable x to $\mathcal{B}_x$, then it does not bind the same variable to a group of other agents.

The concept of term bindings makes authentication easier to understand and to define. Now, we can strengthen an authentication test and make it an agreement test. An initiator sends a message containing a nonce and its binding group, and receives a message confirming the term binding from the responder. In other words, every challenge message should specify who are the sender and the receiver, and whenever a responder creates a reply, he should clearly say in his reply what he meant by the message, i.e. to whom he is replying. In protocols where all challenge and response messages are implemented in such ways, when an initiator receives a reply, she can check the binding group in the reply and if it is not what she expected, then she can stop the protocol run. A protocol implemented this way is similar to a *fail-stop protocol* [23] in that it stops as soon as it receives an inappropriate message.

There are several ways to implement agreement tests but not all agreement tests are useful and only a small number of these tests are practical in implementations. We propose a set of protocols or protocol steps which can be used as agreement tests. We call the set of protocols *protocol primitives* or *primitives*. Protocol primitives are designed for efficiency of communication in mind and will later be used as building blocks for composing more complex protocols. Various types of protocol primitives are possible, depending on the keys used to generate reply messages and the forms of reply messages. The following are the protocol primitives mainly utilised in this thesis. All protocol primitives proposed are fail-stop in a sense that a primitive execution stops immediately when there is any deviation from the designed primitive execution path. For notational convenience, a binding group $\mathcal{B}_x$ is treated as a term, so instead of writing x, A, B, it is written as $x, \mathcal{B}_x$ if there is no confusion.

Symmetric Key Cryptosystem. Suppose that two different keys k_{AB} and k_{BA} are shared between A and B (A uses k_{AB} and B uses k_{BA} to communicate with the other, respectively), and suppose $\mathcal{B}_x = (A, B)$. Encryption is not assumed to provide integrity.

$$\begin{array}{llc} \textbf{Type 1}: & A \to B: & B, \{\!|x, \mathcal{B}_x|\!\}_{k_{AB}}, \langle x, \mathcal{B}_x \rangle_{k_{AB}} \\ & B \to A: & A, \langle x, \mathcal{B}_x \rangle \end{array}$$

$$\begin{array}{llc} \textbf{Type 2}: & A \to B: & B, \{\!|x, \mathcal{B}_x|\!\}_{k_{AB}}, \langle x, \mathcal{B}_x \rangle_{k_{AB}} \\ & B \to A: & A, \langle x, \mathcal{B}_x \rangle_{k_{BA}} \end{array}$$

$$\begin{array}{llc} \textbf{Type 3}: & A \to B: & x, \mathcal{B}_x, \langle x, \mathcal{B}_x \rangle_{k_{AB}} \\ & B \to A: & A, \langle x, \mathcal{B}_x \rangle_{k_{BA}} \end{array}$$

As mentioned earlier, notations such as $\{\!|x, A|\!\}_{K_B}$ found in the Needham-Schroeder public key protocol are not acceptable because when B sees the message, he has no way to know its destination. Therefore, every message should clearly say who should receive the message. In case of **Type 3**, remember that $\mathcal{B}_x$ contains the term B as a responder. Special attention should be paid to the term $\langle x, \mathcal{B}_x \rangle_{k_{AB}}$. Some message forms in primitives are similar, so **Type 2** and **Type 3** may suffer replay attacks. There are many ways to avoid this problem, such as introducing more redundancy to these messages or using different hash functions in generating these messages. Several suggested solutions can be used to tackle this problem. A more detailed explanation will be given later. Assuming that each message has a different form, $\langle x, \mathcal{B}_x \rangle_{k_{AB}}$ is denoted as $auth_{k_{AB}}(x, \mathcal{B}_x)$. Technically, $auth_k(*)$ is used to represent a message originating from someone who possesses the key k. It does not need to be an output of a private key transformation because if there is a link proven to originate from someone with the key k, then $auth_k(*)$ can be delivered using this link. The reason to use this notation instead of a specific message is not to stick to some specific message format. For example, if there is another message, say M, which is verified as originating from an agent with k, then the overhead of an extra transformation, necessary for generating $auth_k(x, \mathcal{B}_x)$, can be reduced by including $(x, \mathcal{B}_x)$ in M. We see $auth_k(*)$ as an evidence which shows it originates from an agent with the key k. If encryption is assumed to provide integrity, $auth_k(*)$ can be removed from **Type 1** and **Type 2** primitives.

$$\begin{aligned}
\textbf{Type 1}: \quad & A \rightarrow B: \quad B, \{\!|x, \mathcal{B}_x|\!\}_{k_{AB}}, auth_{k_{AB}}(x, \mathcal{B}_x) \\
& B \rightarrow A: \quad A, \langle x, \mathcal{B}_x \rangle
\end{aligned}$$

$$\begin{aligned}
\textbf{Type 2}: \quad & A \rightarrow B: \quad B, \{\!|x, \mathcal{B}_x|\!\}_{k_{AB}}, auth_{k_{AB}}(x, \mathcal{B}_x) \\
& B \rightarrow A: \quad A, \langle x, \mathcal{B}_x \rangle_{k_{BA}}
\end{aligned}$$

$$\begin{aligned}
\textbf{Type 3}: \quad & A \rightarrow B: \quad x, \mathcal{B}_x, auth_{k_{AB}}(x, \mathcal{B}_x) \\
& B \rightarrow A: \quad A, \langle x, \mathcal{B}_x \rangle_{k_{BA}}
\end{aligned}$$

Asymmetric Key Cryptosystem. Let (sk_*, vk_*) be a pair of signature and verification keys and (ek_*, dk_*) be a pair of encryption and decryption keys.

$$\begin{aligned}
\textbf{Type 1}: \quad & A \rightarrow B: \quad B, \{\!|x, \mathcal{B}_x|\!\}_{ek_B}, auth_{sk_A}(x, \mathcal{B}_x) \\
& B \rightarrow A: \quad A, \langle x, \mathcal{B}_x \rangle
\end{aligned}$$

$$\begin{aligned}
\textbf{Type 2}: \quad & A \rightarrow B: \quad B, \{\!|x, \mathcal{B}_x|\!\}_{ek_B}, auth_{sk_A}(x, \mathcal{B}_x) \\
& B \rightarrow A: \quad A, \langle x, \mathcal{B}_x \rangle_{sk_B}
\end{aligned}$$

$$\begin{aligned}
\textbf{Type 3}: \quad & A \rightarrow B: \quad x, \mathcal{B}_x, auth_{sk_A}(x, \mathcal{B}_x) \\
& B \rightarrow A: \quad A, \langle x, \mathcal{B}_x \rangle_{sk_B}
\end{aligned}$$

All primitives above can be easily derived from symmetric key ones by changing the keys used.

In order to specify authentication goals, using something like "this protocol should achieve one-way authentication" is too vague. For example, the following protocol gives a one-way authentication guarantee to A.

$$\begin{aligned}
\text{Message 1} \quad & A \rightarrow B: \quad x, \mathcal{B}_x \\
\text{Message 2} \quad & A \leftarrow B: \quad A, \langle x, \mathcal{B}_x \rangle_k
\end{aligned}$$

However, there is a clear difference between the protocol above and the suggested primitives in terms of B's assurance. The suggested primitives guarantee to B that the first message comes from A but the protocol above does not. This minor difference is subtle but important especially in a design process. Hence, we need to know what each participant sees at the end of a run, i.e. we need to know specifically which goals each participant achieves when they finish a run of a primitive.

In order to prove that the primitives proposed above are agreement tests, it has to be shown that A gets the guarantee that B is freshly authenticated and B is talking with her, whereas B gets the guarantee that the challenge originates from A. Lemma 10 describes B's guarantee as a responder and Lemma 11 describes A's as an initiator.

Lemma 10. *Whenever B finishes a run of a primitive as a responder supposedly with A, i.e. if $\xi^-_{BA}(\omega) \in \psi$ for a bundle ψ, then $[\xi^+_{AB}(\omega)] \in \psi$.*

$$\forall \psi \bullet \xi^-_{BA}(\omega) \in \psi \Rightarrow [\xi^+_{AB}(\omega)] \in \psi$$

Proof. Let $\xi^-_{BA}(\omega) = |e^-(m_1), e^+(m_2)|_B$, where $m_1 = (B, \{\!|x, \mathcal{B}_x|\!\}_{k_1}, auth_{k_2}(x, \mathcal{B}_x))$ or $(x, \mathcal{B}_x,\ auth_{k_2}(x, \mathcal{B}_x))$, and $m_2 = (A, \langle x, \mathcal{B}_x \rangle_{k_3})$ with $\mathcal{B}_x = (A, B)$. The values of k_1, k_2 and k_3 are determined by the type of the primitive used in the run.

The strand $[\xi^+_{AB}(\omega)]$ is composed of an event $e^+(m_1)$, so we have to show that $e^+(m_1) \in [\xi^+_{AB}(\omega)] \in \psi$. From the definition of the bundle, $e^+(m_1) \in \psi$. Each term in m_1 can be delivered separately, so let $m_1 = (y,\ auth_{k_2}(x, \mathcal{B}_x))$, where y is either $(B, \{\!|x, \mathcal{B}_x|\!\}_{k_1})$ or $(x, \mathcal{B}_x)$. The term $auth_{k_2}(x, \mathcal{B}_x)$ can only originate from A playing an initiator role because $k_2 = sk_A$ or k_{AB}. Therefore,

$$e^+(auth_{k_2}(x, \mathcal{B}_x)) \in \xi^+_{A*}(\omega)$$

If A is honest, then $(\nu x) \in \xi^+_{A*}(\omega)$. Otherwise, it is possible that $(\nu x) \notin \xi^+_{A*}(\omega)$. B will send his reply m_2 only when he receives m_1. Therefore, $e^+(y) \in \psi$. The binding group in $auth_{k_2}(x, \mathcal{B}_x)$ should match with the binding group in y. If A is honest, then $e^+(y) \in \xi^+_{A*}(\omega)$. Therefore, $e^+(m_1) \in \xi^+_{AB}(\omega)$ because the binding groups in y and $auth_{k_2}(x, \mathcal{B}_x)$ should match. If A is dishonest, she can forward a challenge message y' containing x to B, i.e. y' originates from some participant $C \neq A$. However, to make B believe that he is talking with A, she has to put her identity in $\mathcal{B}_x$, which is not the same as the one in y'. Therefore, $y' \neq y$ and $e^+(y) \in \xi^+_{AB}(\omega)$.

Hence, from $e^+(auth_{k_2}(x, \mathcal{B}_x)) \in \xi^+_{AB}(\omega)$ and $e^+(y) \in \xi^+_{AB}(\omega)$,

$$e^+(m_1) \in \xi^+_{AB}(\omega) \qquad \text{and} \qquad [\xi^+_{AB}(\omega)] \in \psi$$

□

A's guarantee can be verified in a similar way.

Lemma 11. *Whenever A finishes a run of a primitive as an initiator supposedly with B, i.e. if $\xi^{+}_{AB}(\omega) \in \psi$ for a bundle ψ, then $[\xi^{-}_{BA}(\omega)] \in \psi$.*

$$\forall \psi \bullet \xi^{+}_{AB}(\omega) \in \psi \Rightarrow [\xi^{-}_{BA}(\omega)] \in \psi$$

Notice that B's goal is achieved ahead of A's. It is possible to achieve A's goal before B's by moving $auth_k(*)$ to the last position as follows.

Transformed Primitives:

Symmetric Key Cryptosystem.

$$\begin{array}{lrc} \textbf{Type 1}': & A \rightarrow B: & B, \{\!|x, \mathcal{B}_x|\!\}_{k_{AB}} \\ & B \rightarrow A: & A, \langle x, \mathcal{B}_x \rangle \\ & A \rightarrow B: & B, auth_{k_{AB}}(x, \mathcal{B}_x) \end{array}$$

$$\begin{array}{lrc} \textbf{Type 2}': & A \rightarrow B: & B, \{\!|x, \mathcal{B}_x|\!\}_{k_{AB}} \\ & B \rightarrow A: & A, \langle x, \mathcal{B}_x \rangle_{k_{BA}} \\ & A \rightarrow B: & B, auth_{k_{AB}}(x, \mathcal{B}_x) \end{array}$$

$$\begin{array}{lrc} \textbf{Type 3}': & A \rightarrow B: & x, \mathcal{B}_x \\ & B \rightarrow A: & A, \langle x, \mathcal{B}_x \rangle_{k_{BA}} \\ & A \rightarrow B: & B, auth_{k_{AB}}(x, \mathcal{B}_x) \end{array}$$

Asymmetric Key Cryptosystem.

$$\begin{array}{lrc} \textbf{Type 1}': & A \rightarrow B: & B, \{\!|x, \mathcal{B}_x|\!\}_{ek_B} \\ & B \rightarrow A: & A, \langle x, \mathcal{B}_x \rangle \\ & A \rightarrow B: & B, auth_{sk_A}(x, \mathcal{B}_x) \end{array}$$

$$\begin{array}{lrc} \textbf{Type 2}': & A \rightarrow B: & B, \{\!|x, \mathcal{B}_x|\!\}_{ek_B} \\ & B \rightarrow A: & A, \langle x, \mathcal{B}_x \rangle_{sk_B} \\ & A \rightarrow B: & B, auth_{sk_A}(x, \mathcal{B}_x) \end{array}$$

$$\begin{array}{lrc} \textbf{Type 3}': & A \rightarrow B: & x, \mathcal{B}_x \\ & B \rightarrow A: & A, \langle x, \mathcal{B}_x \rangle_{sk_B} \\ & A \rightarrow B: & B, auth_{sk_A}(x, \mathcal{B}_x) \end{array}$$

The transformed primitives achieve the same properties as the original primitives do if there is no confusion between $\langle x, \mathcal{B}_x \rangle_k$ and $auth_k(x, \mathcal{B}_x)$. The only difference between these primitives is the order that goals are achieved, but more choices on primitives will enrich the design methodology.

Lemma 12 describes B's guarantee as a responder and Lemma 13 describes A's as an initiator, when they finish a run of a transformed primitive.

Lemma 12. *Whenever B finishes a run of a transformed primitive as a responder supposedly with A, i.e. if $\xi^-_{BA}(\omega) \in \psi$ for a bundle ψ, then $[\xi^+_{AB}(\omega)] \in \psi$.*

$$\forall \psi \bullet \xi^-_{BA}(\omega) \in \psi \Rightarrow [\xi^+_{AB}(\omega)] \in \psi$$

Proof. Let $\xi^-_{BA}(\omega) = |e^-(m_1), e^+(m_2), e^-(m_3)|_B$, where $m_1 = (B, \{|x, \mathcal{B}_x|\}_{k_1})$ or $(x, \mathcal{B}_x)$, $m_2 = (A, \langle x, \mathcal{B}_x \rangle_{k_3})$ and $m_3 = auth_{k_2}(x, \mathcal{B}_x)$ with $\mathcal{B}_x = (A, B)$.

We have to prove that $[\xi^+_{AB}(\omega)] = |e^+(m_1), e^-(m_2), e^+(m_3)|_A$, given $\xi^-_{BA}(\omega) \in \psi$. It is obvious that $e^+(m_3) \in \xi^+_{AB}(\omega)$ because only A can generate m_3 if the forms of m_3 and $auth_{k_2}(x, \mathcal{B}_x)$ do not cause any confusion. If A is honest, $e^-(m_2) \in \xi^+_{AB}(\omega)$ because A only generates $e^+(m_3)$ when there is a previous event $e^-(m_2)$, and also $e^+(m_1) \in \xi^+_{AB}(\omega)$.

Otherwise, if A is dishonest, A cannot generate any $auth_{k_2}(*)$ with its binding group $\mathcal{B}'_x = (A', B)$, where $A' \neq A$. If such a message exists, then it can only originate from A'. However, such a message can only make B to believe that he is talking with A' if he generated a message m_2 with the binding group $\mathcal{B}_x = (A', B)$. If B generated a message m_2 with the binding group $\mathcal{B}_x = (A, B)$, i.e. if A modified the binding group of the first message m_1 originating from A', to $\mathcal{B}_x = (A, B)$, B will not consider the run as successful before B receives message m_3 from A. Therefore, A' will not receive the authentication reply from B, and A' will not consider the run as successful either. If A generates m_3, then A finishes all necessary steps for $\xi^+_{AB}(\omega)$, even though she might not have generated x.

Hence,

$$[\xi^+_{AB}(\omega)] \in \psi$$

□

Similarly, A's guarantee as an initiator can be proven, that is, when A finishes a run of a transformed primitive, there is a corresponding run of B as a responder.

Lemma 13. *Whenever A finishes a run of a transformed primitive, as an initiator supposedly with B, i.e. if $\xi^{+}_{AB}(\omega) \in \psi$ for a bundle ψ, then $[\xi^{-}_{BA}(\omega)] \in \psi$.*

$$\forall \psi \bullet \xi^{+}_{AB}(\omega) \in \psi \Rightarrow [\xi^{-}_{BA}(\omega)] \in \psi$$

This proves that all proposed primitives and transformed primitives are agreement tests.

Theorem 2. *All proposed primitives are agreement tests.*

Proof. The theorem follows from Lemma 10, 11, 12, and 13. □

The next question is whether the primitives are composable. In other words, given that a primitive achieves the agreement property when it was executed alone, we want to know whether it still achieves the same property i.e. the agreement property remains valid when it is executed together with other primitives. If this is true, then there is no interaction between primitives, so running multiple primitives together does not destroy any properties achieved by a single primitive.

4.2 Composability

To make composition work, it should be verified that no interference occurs when several primitives are executed together. In other words, each goal implemented by each independent primitive should remain valid after composition.

Assume that $p_1(\omega_1)$ and $p_2(\omega_2)$ are primitives with parameters ω_1 and ω_2, respectively. Let $g_1(\omega_1)$ and $g_2(\omega_2)$ be the goals realised by each primitive, respectively, when executed alone. We use notation $p_1(\omega_1) \otimes p_2(\omega_2)$ to represent an entangled run of the two primitives. By entangled runs, we mean running $p_1(\omega_1)$ and $p_2(\omega_2)$ together without any order restrictions between them, and if the parameters are not important for understanding, we use p_1 and p_2 instead of $p_1(\omega_1)$ and $p_2(\omega_2)$. Even though we allow any orderings among the two entangled primitives, the order restrictions within each primitive should always be kept.

It is simple to check whether p_1's goal g_1 is affected by the execution of p_2. When p_2 is executed before p_1, if p_2 does not break any assumptions which p_1 relies on, then g_1 is unaffected by the execution of p_2.

Remark 1. *If any execution of p_2 does not break assumptions on which p_1 relies, then p_1's goal g_1 is non-destructive, i.e. the execution of p_2 has no effects on g_1.*

If the property of Remark 1 is true, then we say g_1 is *non-destructive* in $p_1 \otimes p_2$, similarly, g_2 can be non-destructive in $p_1 \otimes p_2$. If both g_1 and g_2 are non-destructive in $p_1 \otimes p_2$, then we can conclude that there is no interference between p_1 and p_2.

We have already shown the non-constructive property of authentication by adopting agreement tests as authentication mechanisms. If it is proven that authentication has the non-destructive property as well as the non-constructive property, then we can conclude that authentication is composable because authentication is not disturbed by simultaneous runs of other primitives. Composability of secrecy is divided into two sub-issues, composability of long-term secrets and short-term secrets.

4.2.1 Authentication

We first check whether the goals of authentication achieved by each primitive are non-destructive. The non-destructiveness depends on initial assumptions, so two cases need to be considered separately, one which uses signature operations for authentication (**Type 2** and **Type 3**) and the other which does not (**Type 1**). In the former case, any agreement properties achieved by a primitive remain true as long as signature keys remain secret.

Lemma 14. *Authentication goals accomplished by **Type 2** and **Type 3** primitives are non-destructive in composition if long-term keys are assumed safe after composition.*

Proof. In **Type 2** and **Type 3** primitives, the secrecy of nonce x has no effect on the authentication guarantee as long as A sees it as fresh. A value only becomes fresh to A when it was recently generated by her, so it is obvious to A whether a value is fresh or not. Therefore, multiple instances of any primitives which belong to this category do not interfere with each other if long-term keys used in primitives are safe. This means that any authentications realised by a primitive or primitives in this category remain valid after entanglement. This is true even if each primitive uses the same nonce to achieve different authentication goals, because nonces are not seen as secret, and a signature on a message unambiguously shows from where the message originates and the binding group within the message also says with whom the responder is talking.

Moreover, the initiator will only accept a signed message from the responder whom she initially planned to authenticate with the nonce. Spy can generate a signature but the initiator will not accept it if Spy is not the member of the binding group of the nonce, i.e. fake messages from Spy do not cause any harm. Therefore, all goals achieved by the primitives belonging to this category are non-destructive. □

Primitives in **Type 1** assume that a nonce is secret, i.e. it is only known to the two participants, e.g. A and B. Therefore, any events which reveal the nonce to some other participants can destroy the authentication implemented by the primitives belonging to this category. The following example shows a case when two **Type 1** primitives which use the same nonce to achieve different agreement goals are mixed together.

$$\begin{array}{lll} \text{Message 1} & A \rightarrow B: & B, \{\!|x, \mathcal{B}_x|\!\}_{ek_B}, auth_{sk_A}(x, \mathcal{B}_x) \\ \text{Message 2} & A \leftarrow B: & A, \langle x, \mathcal{B}_x \rangle \\ & & \\ \text{Message 1}' & A \rightarrow C: & C, \{\!|x, \mathcal{B}'_x|\!\}_{ek_C}, auth_{sk_A}(x, \mathcal{B}'_x) \\ \text{Message 2}' & A \leftarrow C: & A, \langle x, \mathcal{B}'_x \rangle \end{array}$$

As can be seen from the given example, using the same nonce to achieve agreements with two different participants is problematic because both B and C can generate Message 2′ and Message 2, and A has no way to differentiate them. Therefore, when A finished a protocol run, there is no guarantee of a matching run. This occurred because the authentication guarantees achieved by the primitives in this category rely on the fact that a nonce x is secret between two participants. Hence, for authentication goals achieved by **Type 1** primitives to be non-destructive, the following should be met.

Lemma 15. *Authentication goals accomplished by* ***Type 1*** *primitives are non-destructive in composition if the same nonce is not used for the authentication of more than two different participants.*

Proof. Let $p_1(x; \mathcal{B}_x)$ and $p_2(x'; \mathcal{B}_{x'})$ be two primitives of **Type 1**. If $\mathcal{B}_x \cap \mathcal{B}_{x'} = \emptyset$, then the possibility that $x = x'$, i.e. the chance that two independent participants generate the same nonce is negligible. Therefore, it can be assumed that $x \neq x'$ in this case. Two primitives $p_1(x; \mathcal{B}_x)$ and $p_2(x'; \mathcal{B}_{x'})$ with $x \neq x'$ and $\mathcal{B}_x \cap \mathcal{B}_{x'} = \emptyset$ do not interfere each other, so the goals achieved by one primitive are not affected by the execution of the other. Therefore, the authentication property in this case is non-destructive. The other case left is when $\mathcal{B}_x \cap \mathcal{B}_{x'} \neq \emptyset$. Suppose $\mathcal{B}_x = (A, B)$. The primitive $p_1(x; \mathcal{B}_x)$ is discreet, so the only way that a participant C ($\neq A, B$) to know x is to get it from either A or B. The participant A can forward the secret x to C, but this does not give any benefit to A. Therefore, we can assume that A does not use a secret for different purposes. When one participant B forwards the secret x to a third participant C, the replies which B and C generate are different, so B cannot simply forward C's reply to make A believe that the authentication is successful. In other words, A's goal cannot be realised without a direct reply from B. Hence, whenever A finishes a run successfully with B, there is a matching record of the run of B. □

This is a local restriction, i.e. each honest participant can place the restriction and check it by himself, so it is easy to apply.

From Lemma 14 and 15 the following can be derived.

Theorem 3. *If long-term keys are assumed safe, then authentications realised by primitives are non-destructive as long as the same nonce is not used for the authentication of a different participant.*

Theorem 3 can be rephrased as follows.

Lemma 16. *If there is the guarantee that each participant uses a different nonce for a different authentication goal, the authentication goals realised by primitives p_1 and p_2 remain valid after composition, i.e. the authentication goals g_1 and g_2 are still valid in $p_1 \otimes p_2$.*

According to Theorem 3, complex authentication goals of a protocol can be decomposed into a set of simple authentication goals, and each simple goal can be realised using a protocol primitive. The authentication goals accomplished by a set of independent primitives remain valid even in the presence of other primitives, as long as the condition is met for each participant. This does not necessarily mean that the protocol achieved this way is correct, because there is no guarantee that all goals are achieved in the same session. However, the theorem is very important in the sense that it enables protocol designers to achieve authentication goals by composition.

4.2.2 Secrecy

The other important goals of protocols concern secrecy. There are two types of secrets, long-term and short-term. Secrecy of short-term secrets is generally dependent on secrecy of long-term secrets. Until now, long-term secrets such as signature keys or decryption keys are assumed to be secret without any proof. Proof of secrecy of long-term secrets is easy. If a protocol is regular, then it does not introduce any long-term secrets as parts of messages by an honest participant.

Lemma 17. *All proposed primitives are regular.*

Proof. This is obvious from the message forms of the primitives. Long-term keys are not parts of any messages introduced by any primitive. □

Is a composition of primitives regular? Suppose that $\mathcal{P}$ be a composition of regular primitives, i.e. $\mathcal{P} = p_1 \otimes p_2$. Let K_{L_1} and K_{L_2} be those sets of long-term keys used in p_1 and p_2, respectively. By composition no new message is introduced; therefore, the regularity of $\mathcal{P}$ depends on p_1 and p_2.

Lemma 18. *A composition of primitives $p_1 \otimes p_2$ is regular if the following conditions hold.*

1. *p_1 and p_2 are regular.*
2. *$\forall k_1 \in K_{L_1}, \forall m_2 \bullet k_1 \not\sqsubseteq m_2$, where m_2 is a message generated by p_2.*
3. *$\forall k_2 \in K_{L_2}, \forall m_1 \bullet k_2 \not\sqsubseteq m_1$, where m_1 is a message generated by p_1.*

Proof. From the regularity of p_1 and p_2,

$$\forall k_1 \in K_{L_1}, \forall m_1 \bullet k_1 \not\sqsubseteq m_1$$

$$\forall k_2 \in K_{L_2}, \forall m_2 \bullet k_2 \not\sqsubseteq m_2$$

Let $K_L = K_{L_1} \cup K_{L_2}$, then from Condition 2 and 3,

$$\forall k \in K_L, \forall m_1, m_2 \bullet k \not\sqsubseteq m_1 \wedge k \not\sqsubseteq m_2$$

Therefore, $p_1 \otimes p_2$ is regular. □

Lemma 18 provides a way to build a regular protocol by composition. If long-term keys are never used as nonces or short-term secrets, then protocols created through composition of primitives are obviously regular, so the following is clear.

Lemma 19. *A composition of regular primitives is regular if* $\mathsf{K}_L \cap \mathsf{V} = \emptyset$*, where* K_L *is the set of long-term keys, and* V *the set of variables.*

Proof. A composition of regular primitives which do not use long-term keys as nonces or short-term secrets satisfy the conditions of Lemma 18. □

A proof on short-term secrets is more difficult than one on long-term secrets, because it depends on the structure and order of messages within a protocol. If the encryptions in primitives are for confidentiality, then it has to be shown that x, either in **Type 1** or **Type 2** is secret. Let I be a set of terms known to Spy, and $\mathcal{S}_x$ be a protective domain of x, i.e. a set of terms to be protected from Spy for x to remain secret, $\mathcal{S}_x = \{x\} \cup \{\mathsf{prv}(Y) | Y \in \mathcal{B}_x\} \cup \{\mathsf{shr}(Y) | Y \in \mathcal{B}_x\}$. Assume that $\mathcal{S}_x$ is I-compatible, i.e. $\mathcal{I}[\mathcal{S}_x] \cap \mathsf{Parts}(\mathsf{I}) = \emptyset$. This is true if x is freshly generated by an honest participant. If $\mathsf{Spy} \in \mathcal{B}_x$, then there is nothing to prove because x is already known to Spy, so $\mathsf{Spy} \notin \mathcal{B}_x$ for the proof of secrecy of x.

Lemma 20. *Suppose that x is a short term secret. If its protective domain $\mathcal{S}_x$ is* I*-compatible, then all primitives are discreet.*

Proof. Let p be a primitive and e be an event generated by p.

When p is a **Type 3** primitive: There is no short-term secret to be protected, so $\mathcal{S}_* = \emptyset$ and $\mathcal{I}[\mathcal{S}_*] = \emptyset$. Obviously, $\forall e \bullet m(e) \subseteq \mathcal{C}[\mathcal{S}_*]$. Therefore, all primitives belonging to this type are discreet.

When p is a **Type 1** or **Type 2** primitive: Let x be a secret to be protected and its binding group be $\mathcal{B}_x = (A, B)$. If no keys are shared between A and B, then $\mathcal{S}_x = \{x, \mathsf{prv}(A), \mathsf{prv}(B)\}$. Let $(\mathcal{TR}, e) \in p$ and $x \in \mathsf{Unused}(\mathcal{TR})$, where $\mathcal{TR}$ is I-discreet. Since $\mathcal{TR}$ and $\mathcal{S}_x$ are I-discreet, $\mathsf{Analz}(\mathcal{TR} \cup \mathsf{I}) \subseteq \mathcal{C}[\mathcal{S}_x]$. We have to show that $\forall e \bullet m(e) \subseteq \mathcal{C}[\mathcal{S}_x]$. We only prove a simplified version of **Type 2** in asymmetric cryptosystems but all others can be derived similarly[2].

[2]From the definition of the ideal, the hashed output always belongs to the coideal, so we will drop $auth_k(*)$.

Two message rules in **Type 2** are:

$$\begin{array}{ll}\text{Rule 1:} & \exists A, B, x \bullet (\nu x : \mathcal{B}_x) \in \mathcal{TR} \wedge \mathcal{B}_x = \{A, B\} \\ & \wedge\, m = A' \rightarrow B : B, \{\!|x, \mathcal{B}_x|\!\}_{ek_B} \\ \text{Rule 2:} & \exists A, B, A', x \bullet (\nu x : \mathcal{B}_x) \in \mathcal{TR} \\ & \wedge\, m = A' \rightarrow B : B, \{\!|x, \mathcal{B}_x|\!\}_{ek_B} \in \mathcal{TR} \\ & \wedge\, m' = B \rightarrow A : A, \langle x, \mathcal{B}_x \rangle_{sk_B}\end{array}$$

Case 1: Rule 1

$$m = A' \rightarrow B : B, \{\!|x, \mathcal{B}_x|\!\}_{ek_B}$$

and $(\nu x : \mathcal{B}'_x) \in \mathcal{TR}$. If $B \in \mathcal{B}'_x$, then $\mathsf{prv}(B) \in \mathcal{S}_x$ and the encrypted term is in the coideal. Otherwise, if $B \notin \mathcal{B}'_x$, then $\mathsf{prv}(B) \notin \mathcal{S}_x$, so the encrypted term is in the coideal. This fact, together with $A \notin \mathcal{I}[\mathcal{S}_x]$ yields $m \notin \mathcal{I}[\mathcal{S}_x]$.

Case 2: Rule 2

$$m' = B \rightarrow A : A, \langle x, \mathcal{B}_x \rangle_{sk_B}$$

and there must exist

$$m = A' \rightarrow B : B, \{\!|x, \mathcal{B}_x|\!\}_{ek_B}$$

If $A \in \mathcal{B}_x$ then $m' \notin \mathcal{I}[\mathcal{S}_x]$, and we are done. Suppose that $A \notin \mathcal{B}_x$. Then we must show that $x \in \mathcal{C}[\mathcal{S}_x]$. It is trivial for x if $B \notin \mathcal{B}_x$ because x is then exposed in m and $\mathcal{TR}$ is I-discreet. We must show that $x \notin \mathcal{S}_x$ if we assume that $B \in \mathcal{B}_x$ and $A \notin \mathcal{B}_x$. Find the earliest occurrence of the subterm $m = B, \{\!|x, \mathcal{B}_x|\!\}_{ek_B}$. That is, there is a message m'' whose content has m as a subterm, and m is not a part of the prior trace $\mathcal{TR}'$.

m'' might be either faked or honest. If m'' is faked, then

$$m'' = \mathsf{Parts}(\mathsf{Fake}(m(\mathcal{TR}') \cup \mathsf{I})) = \mathsf{Parts}(m(\mathcal{TR}') \cup \mathsf{I}) \cup \mathsf{Fake}(m(\mathcal{TR}') \cup \mathsf{I})$$

Since $m \notin \mathsf{Parts}(m(\mathcal{TR}') \cup \mathsf{I})$, it must have been synthesised, meaning $x, \mathcal{B}_x \in \mathsf{Fake}(m(\mathcal{TR}') \cup \mathsf{I}) \subseteq \mathcal{C}[\mathcal{S}_x]$, so $x \notin \mathcal{S}_x$.

If m'' is honest, inspection of the rules and the message component types shows that $m'' = m$ and Rule 1 holds. But the analysis of Rule 1 has already been covered in the first case.

This shows that all primitives are discreet. $\square$

From Lemma 17, 20 and Theorem 1, the following is derivable.

Theorem 4. *Given a primitive p, every trace in* $\mathsf{Runs}(p)$ *is* I*-discreet.*

Special attention should be paid to Lemma 20 regarding the agreement of a secret value achieved by primitives. We only showed that every message belongs to the coideal, but we have not shown that A knows the value itself after finishing a run of the suggested primitives. For example, Spy can generate a nonce x and Spy can then send a message containing the nonce to B with a fake identification in it, e.g. $\mathcal{B}_x = (A, B)$. In this case, even if a reliable network environment is assumed, where every message finally arrives to its destination, it is still impossible for B to prove that A knows the value before she demonstrates it. However, this is not the case with the suggested primitives because of $auth_k(*)$, which is ignored in the proof above. Spy cannot generate the encrypted form and $auth_k(*)$ together with a fake identification without the knowledge of k. Hence, if we assume a reliable network environment, at least we can say that A knows the value itself with the proposed primitives.

In case of a short-term secret, we do not assume that there are more than two participants who know the secret. This holds for most published protocols. It might be possible to share a secret among three participants, such as a group key found in many group protocols, but we mainly focus on applications where a short-term secret is shared between two participants, except for the case when a trusted third party distributes a shared key. For example, in the Otway-Rees protocol, a shared key always becomes known to A, B, and Srv. However, this is a special case because we fully trust Srv, but under asymmetric key environments, it seems to be more prudent to use different shared keys between different participants.

We have seen that a single primitive is discreet when it is executed alone. Two primitives running together do not seem to interfere with each other if one does not reveal any secret that the other primitive is using. Let x_1 be a secret established by p_1, and x_2 be a secret established by p_2. Depending on the primitives used, either x_1 or x_2 may not be a secret, but for simplicity, both of them are assumed to be secrets. For a composition $p_1 \otimes p_2$ to be discreet, one primitive should not reveal the secret which the other primitive is using, i.e. the following conditions should be satisfied.

Lemma 21. *Let x_1 and x_2 be secrets established by p_1 and p_2, respectively. A composition of primitives $p_1 \otimes p_2$ is discreet, if the following conditions hold:*

1. *p_1 and p_2 are discreet.*

2. *$\forall m_2 \bullet m_2 \subseteq \mathcal{C}[\mathcal{S}_{x_1}]$, where m_2 is a message generated by p_2.*

3. *$\forall m_1 \bullet m_1 \subseteq \mathcal{C}[\mathcal{S}_{x_2}]$, where m_1 is a message generated by p_1.*

The second and third conditions are obvious, otherwise, x_1 might be revealed by some message of p_2 or similarly x_2 by some message of p_1. These conditions can be easily enforced if a different secret is used for each different run, but there are cases where the same secret should be used again, for example, in re-authentication protocols. In case of two-party protocols, only two cases need to be considered, when $\mathcal{B}_{x_1} = \mathcal{B}_{x_2}$ and when $\mathcal{B}_{x_1} = (A, B)$, $\mathcal{B}_{x_2} = (B, A)$. With multi-party protocols, it is more difficult to check this property but the cases when $\mathcal{B}_{x_1} \cap \mathcal{B}_{x_2} = \emptyset$ can be ignored, because if x_1 and x_2 are assumed to be freshly generated, then these cases are probabilistically negligible, that is to say, it is almost impossible for two participants to generate the same secret.

4.2.3 Adding New Primitives

A new protocol primitive can be added to an existing set of protocol primitives. In this subsection, we identify conditions that the new primitive should satisfy in order to make composition remain safe after its addition to the set.

Let $S = \{p_1, p_2, \ldots, p_n\}$ be a set of protocol primitives which are proven to be safe in composition, i.e., any composition of the primitives in the set should be both non-destructive and non-constructive. Let p be a primitive which claims to achieve certain goals, say a secrecy goal g_1 and an authentication goal g_2. Two different types of goals are defined in a primitive. A goal can be a property which all protocol steps in the primitive should satisfy like secrecy goals, or it can be a property which becomes true by a specific event or by a set of events of the primitive like authentication goals. Any primitives which achieve these types of goals can be considered as possible candidates for additions. However, not all protocol steps can be defined as a protocol primitive. We only consider pairwise goals in designing a protocol primitive, so the first condition that a protocol primitive should satisfy is the agreement property. For every goal of a primitive it should be clear to each participant who is his corresponding partner in achieving the goal. Otherwise, there is a possibility that a participant believes he achieved a certain goal even though there is no corresponding partner to that goal, so the goal does not satisfy the non-constructive property. The concept of binding groups for nonces and secrets is introduced in the previous section to avoid such confusion.

Requirement 1. *A protocol primitive should satisfy the agreement property.*

Whether the primitive p actually achieves g_1 and g_2 is out of the scope of this thesis. Protocol primitives are generally smaller in size than protocols themselves, so these proofs can comparably be easier to do with existing proof tools like Athena and NRL analyser.

After the verification of the claimed goals of p, it needs to be shown that the goals of the primitive p do not interfere with the goals in S. In other words, the goals of p should remain valid in composition with any primitives in S, and the goals of the primitives in S should remain valid in composition with p. The non-interference can be achieved by the non-destructive property and the non-constructive property. Different types of goals have different requirements to satisfy these properties. Here we only consider secrecy and authentication goals.

In case of secrecy goals, the non-constructive property implies that a secret value cannot suddenly be shared between two participants from nothing, i.e. a secret can only be created from secrets or nonces. This is also true in Diffie-Hellman based schemes because x is a secret in g^x, so without the secret x, a common secret g^{xy} cannot be constructed. The non-destructive property of secrecy goals is related with the regularity and the discreetness. By the regularity, no long-term secrets should be used as part of messages in a protocol.

Requirement 2. *A primitive should be regular, i.e. no long-term secrets should be used as a part of messages in the primitive.*

If long-term secrets of p are a subset of long-term secrets of S, then the regularity of $\{p\} \cup S$ is guaranteed by the regularity of p and S by Lemma 18. If long-term secrets of p are not a subset of long-term secrets of S, it has to be shown that $\{p\} \cup S$ is regular with respect to the union of long-term secrets of p and S. The proof of the regularity can be done by inspecting whether any message in p or S uses a long-term secret as a part of the message.

Requirement 3. *A primitive should be discreet with respect to the set of existing primitives and vice versa.*

The discreetness property of the primitive p is more difficult to prove. Two things need to be considered. First, a secret is assumed to be a shared value between two participants. By "shared" we mean that at least each participant knows with whom he is supposed to share the value. Otherwise, the discreetness of the primitive cannot be verified. This is one of the reasons for using the concept of binding groups. Second, the discreetness of p should be shown with respect to its own secret as well as secrets in S. It first has to be shown that p is discreet and then for each primitive in S, it has to be shown that Lemma 21 holds. Alternatively, it can be shown that messages in $S \cup p$ belong to the intersection of all coideals of short-term secrets $\{x_1, x_2, \ldots, x_n, x'\}$ in S, where x_i is a secret used in primitive p_i and x' is a secret used in p. In other words, $m \subseteq \mathcal{C}[S_x]$, where m is a message generated by $S \cup p$ and $\mathcal{C}[S_x] = \mathcal{C}[S_{x_1}] \cap \mathcal{C}[S_{x_2}] \cap \cdots \cap \mathcal{C}[S_{x_n}] \cap \mathcal{C}[S_{x'}]$. By showing that p is regular and discreet with respect to S, the non-destructive property and the non-constructive property of g_1 of p are proved.

The non-constructive property of authentication goals implies that no event in S should materialise the goal g_2 of p. In other words, without a certain e of p, the goal g_2 should not be achieved, i.e. e should be the only event that materialises the goal g_2. Ensuring the freshness of a challenge usually does not provide this property because the reply may not contain information showing for whom the reply is made. Similar message forms among primitives will also make it difficult to avoid attacks such as cut and paste. Therefore, it is necessary to differentiate between authentication replies coming from different participants and to differentiate between replies belonging to different sessions. Former can be done by adding the binding group of a challenge in each reply and latter can be done by adding an index or a session identifier after composition.

The non-destructive property of authentication goals is related to the secrecy of long-term and short-term values used for the authentication. Therefore, the regularity and the discreetness of those secrets should also be shown. However, this alone is not enough because using the same nonce twice can also destroy an authentication goal as shown in the example of Section 4.2.1. So no nonces should be used for the authentication of more than one participant.

Requirement 4. *No nonces should be used for the authentication of more than one participant either in a primitive or in composition.*

When a primitive satisfies Requirement 1, 2, 3, and 4, it has been shown in the previous subsections that the primitive has the non-interference property in composition.

4.3 Extending the Primitives

Only a limited number of primitives are presented in this thesis. However, protocol designers do not need to stick to only these primitives. Any protocol steps can be served as a primitive, if they satisfy the agreement, the discreetness, and the regularity properties. A new family of primitives can be generated and used in design, if it can be verified that they satisfy the required properties. In this way, designers can build a library of verified primitives, and can reference them whenever necessary. This section discusses a generalisation of primitives, and its advantages and limitations.

4.3.1 Flexibility and Limitations

Different applications may require different primitives, and many different techniques are available to achieve the same aim. In some applications, instead of using a long-term key for authentication, a new short-term key can be generated from a shared secret and then this key can be used to authenticate the other participant. In the following example, f is a pseudo-random function or a key generation function and c is an agreed constant term among the participants.

$$
\begin{array}{lcl}
A \to B: & B, \{\!|x, \mathcal{B}_x|\!\}_{ek_B}, auth_{sk_A}(x, \mathcal{B}_x) & \quad A \to B: & B, \{\!|x, \mathcal{B}_x|\!\}_{ek_B}, auth_{sk_A}(x, \mathcal{B}_x) \\
B \to A: & A, \langle x, \mathcal{B}_x \rangle & \quad B \to A: & A, \langle x, \mathcal{B}_x \rangle_{sk_B} \\
 & \Updownarrow & & \Updownarrow \\
A \to B: & B, \{\!|x, \mathcal{B}_x|\!\}_{ek_B}, auth_{sk_A}(x, \mathcal{B}_x) & \quad A \to B: & B, \{\!|x, \mathcal{B}_x|\!\}_{ek_B}, auth_{sk_A}(x, \mathcal{B}_x) \\
B \to A: & A, \langle \mathcal{B}_x \rangle_{f(x)} & \quad B \to A: & A, \langle c \rangle_{f(x, \mathcal{B}_x)}
\end{array}
$$

Under the ideal cryptography assumption, all the primitives above might be the same according to strand space theory, i.e. in terms of the guarantees they provide to each participant. Therefore, one might be flexibly exchangeable with the others if only correspondence among participants is considered. In other words, a protocol designed using the upper primitives can be transformed to use the lower ones. This exchangeability between primitives provides more diverse options in protocol design. However, in real applications where the ideal cryptography assumption does not hold; there might be a difference between them. For example, if x is a weak secret, which usually holds in password authentication, the primitive in the bottom second column suffers dictionary attacks, even though the one above does not.

Generally speaking, the choice of primitives depends on many factors such as applications, costs of cryptographic operations, and environments where the protocol is supposed to run. Generalising all these factors and finding the optimal primitive for all applications is difficult. It is also difficult to suggest every possible mechanism for each application. The decision which one fits better should be made case by case. Starting from a small number of primitives, designers can later add a new primitive and prove it correct using existing verification methods. Primitive verification is usually easier than protocol verification in terms of complexity. Moreover, these newly introduced primitives can later be reused in designing a new application. This provides some level of flexibility in selecting appropriate primitives in design, and enables reusability of design and implementation.

4.3.2 Generalisation of Primitives

Until now, a nonce is assumed to be a single variable, but it can be extended to a vector, or more precisely an ordered set of terms, by replacing x by a set of terms $\vec{x}$ without changing $\mathcal{B}_x$. This way, generalised versions of the primitives can be obtained, and the following is one example.

$$\begin{array}{llll} p\text{:} & M_1 & A \rightarrow B: & \vec{x}, \mathcal{B}_{\vec{x}}, auth_{sk_A}(\vec{x}, \mathcal{B}_{\vec{x}}) \\ & M_2 & B \rightarrow A: & A, \langle \vec{x}, \mathcal{B}_{\vec{x}} \rangle_{sk_B} \end{array}$$

Not all terms in the vector need to be variables: constant terms can be members of the vector too. However, in order to keep the regularity property of the primitives, long-term keys must not be members of the vector. It is also assumed that no agent name appears in the vector, since it can be included in the binding group instead of the vector if necessary. This way, we separate agent names from any other terms. As an example, $\vec{x} = (x_1, x_2)$ can be used as a parameter of a primitive, where x_1 may be a nonce and x_2 may be a participant's set of preferences for encryption and hashing. If x is a member of a vector $\vec{x}$, it is written as $x \in \vec{x}$. How to interpret each term in the vector should be specified by the protocol specification.

Replacing a variable in a primitive by a vector does not change any properties which the primitive achieves and the extension is defined as follows.

Definiton 26. *The extension of $p(x;\mathcal{B}_x)$, namely $p(\vec{x};\mathcal{B}_{\vec{x}})$ is defined as $p(\vec{x}/x;\mathcal{B}_x)$, where each variable x in $p(x;\mathcal{B}_x)$ is substituted by $\vec{x}$, and $\mathcal{B}_x = \mathcal{B}_{\vec{x}}$. $\vec{x}$ must not contain any long-term safe keys or participant names.*

When we mention a primitive, we now mean this generalised version, and we assume two participants engaged in a run are A (as an initiator) and B (as a responder), so we use the notation $p(\vec{x})$ instead of $p(\vec{x};\mathcal{B}_{\vec{x}})$.

4.4 Concurrent Goals

Previous subsections showed that authentication and secrecy goals achieved by primitives are preserved after composition. For a composition to be correct, all protocol goals accomplished by the primitives used in the composition should happen in a single session. This subsection discusses how to ensure that all these independent goals happen in a single session.

4.4.1 Multi-run Interference

It is previously shown that each primitive, as a single protocol, does not interfere with other primitives. However, due to the similarity in the forms of messages in primitives, the non-interference property does not hold when more than two primitives are regarded as a single protocol. Let $\mathcal{P}$ be a composition of two primitives p_1 and p_2, i.e. $\mathcal{P} = p_1 \otimes p_2$, where p_1 and p_2 are shown as below, with $x \neq y$, $\mathcal{B}_x = (A,B)$ and $\mathcal{B}_y = (B,A)$.

$$
\begin{array}{llll}
p_1\text{:} & M_1 & A \rightarrow B: & x, \mathcal{B}_x, auth_{sk_A}(x,\mathcal{B}_x) \\
 & M_2 & B \rightarrow A: & A, \langle x, \mathcal{B}_x \rangle_{sk_B} \\
p_2\text{:} & N_1 & B \rightarrow A: & y, \mathcal{B}_y, auth_{sk_B}(y,\mathcal{B}_y) \\
 & N_2 & A \rightarrow B: & B, \langle y, \mathcal{B}_y \rangle_{sk_A}
\end{array}
$$

Even though p_1 or p_2 as a single independent protocol is non-interfering, as a part of $\mathcal{P}$, one does interfere with the other. When A is running multiple instances of $\mathcal{P}$ with

B, playing a different role in each run, if A receives a message of the form either M_2 or N_2, this message can be interpreted either as a message of a run of primitive p_1 where she plays an initiator role or as a message of a run of primitive p_2 where she plays a responder role. This is an inherent problem in the proposed method because the method repeatedly uses protocol primitives which have structurally similar message forms. Generally speaking, in a composition $\mathcal{P} = p_1 \otimes p_2 \otimes \cdots \otimes p_n$, messages of p_i might be wrongly identified as messages of p_j, where $i \neq j$. This type of confusion should be prevented, otherwise it allows Spy to mount attacks such as cut and paste attacks. Various approaches can be used to tackle the problem. One way is to use an index, which differentiates each primitive from the others as in the following example.

$$
\begin{array}{llll}
p_1\text{:} & M_1 & A \rightarrow B: & x, \mathcal{B}_x, auth_{sk_A}(1, x, \mathcal{B}_x) \\
 & M_2 & B \rightarrow A: & A, \langle 1, x, \mathcal{B}_x \rangle_{sk_B} \\
p_2\text{:} & N_1 & B \rightarrow A: & y, \mathcal{B}_y, auth_{sk_B}(2, y, \mathcal{B}_y) \\
 & N_2 & A \rightarrow B: & B, \langle 2, y, \mathcal{B}_y \rangle_{sk_A}
\end{array}
$$

This solution is very simple and it works regardless of the number of primitives used in the protocol. An index in each message indicates to which primitive the message belongs, so it clears the confusion among similar messages. The scheme is similar to the method which puts a message number in each message in a protocol. Notice that an index needs to be inserted only in cryptographically transformed messages, because putting an index in cleartext messages does not serve any purpose.

Changing message forms in each primitive can also be a solution: more redundancy can be added (actually adding an index belongs to this, since it increases redundancy), or terms within a message can be reordered. Alternatively, each primitive can use a different hash function to produce its messages. For example, a hash function h_1 can be used for the first primitive, h_2 for the second primitive, etc. In environments where only one hash function is available, each hash function can still be generated by using methods such as $h_i = \langle i, * \rangle_*$. If an index in a message is interpreted as a hash function number in the previous example, the hashing method is also derivable from the vector $\vec{x}_i = (i, x)$.

Consequently, regardless of the solution used to avoid confusion, a vector $\vec{x}_i = (i, \vec{x})$ can be used as a parameter to generate each primitive. This enables designers to separate design from implementation to some degree. In the design stage, designers can ignore which method will be adopted to solve the confusion problem, and later in the implementation stage, the decision can be made without damaging any properties achieved in the design stage. For example, if communication costs are more expensive than computation costs, the scheme which generates a smaller size of the total messages might be preferable. Our interest lies in design rather than implementation, so the indexing method is used here, without specifying the actual implementation. From now on, $\vec{x}_i$ means $(i, \vec{x})$, i.e. $p(\vec{x}_i)$ means a primitive p with parameter $(i, \vec{x})$.

The non-interference property of indexed primitives can be proven using the strand formalism. A run of a primitive produces a bundle. Suppose $\psi_i(\omega_i)$ is a bundle produced by a run of a primitive $p_i(\omega_i)$ with a parameter $\omega_i = \vec{x}_i, \mathcal{B}_{\vec{x}}(= \{A, B\})$. Let $\xi^i_{AB}(\omega_i)$ and $\xi^i_{BA}(\omega_i)$ be the parts of the bundle that A and B see, respectively. That is, $\psi_i(\omega_i) = \{\xi^i_{AB}(\omega_i), \xi^i_{BA}(\omega_i)\}$, or more specifically $\psi_i(\omega_i) = \{\xi^{i+}_{AB}(\omega_i), \xi^{i-}_{BA}(\omega_i)\}$ if A is an initiator and B is a responder within the run. Under the assumption that each primitive has an index number, the agreement property can be extended as follows. The lemma means that there is no confusion between runs formed by different indexed primitives.

Lemma 22. *If $p_i \otimes p_j$ is a composition of indexed primitives, the following holds, where ψ_i is a bundle formed by a run of p_i.*

$$\forall \psi_i \bullet \xi^{i+}_{AB}(\omega_i) \in \psi_i \Rightarrow [\xi^{i-}_{BA}(\omega_i)] \in \psi_i$$
$$\forall \psi_i \bullet \xi^{i-}_{BA}(\omega_i) \in \psi_i \Rightarrow [\xi^{i+}_{AB}(\omega_i)] \in \psi_i$$

Proof. It suffices to show two things: 1. Given a strand $\xi^{i+}_{AB}(\omega_i)$ or $\xi^{i-}_{BA}(\omega_i)$, there exists a corresponding strand, i.e. $[\xi^{k-}_{BA}(\omega_i)]$ or $[\xi^{k+}_{AB}(\omega_i)]$, respectively. 2. These two strands are in the same bundle which is formed by p_i.

From the agreement property of primitives, i.e. from Lemma 9 and Theorem 2, the following is true.

$$\forall \psi_i \bullet \xi^{i+}_{AB}(\omega_i) \in \psi_i \Rightarrow \exists k \bullet [\xi^{k-}_{BA}(\omega_i)] \in \psi_i$$
$$\forall \psi_i \bullet \xi^{i-}_{BA}(\omega_i) \in \psi_i \Rightarrow \exists k \bullet [\xi^{k+}_{AB}(\omega_i)] \in \psi_i$$

Let $\psi = \{\xi^{i+}_{AB}(\omega_i), \xi^{k-}_{BA}(\omega_i)\}$ be a bundle formed by two strands, $\xi^{i+}_{AB}(\omega_i)$ and $\xi^{k-}_{BA}(\omega_i)$. ψ is a bundle since it is closed under the $\rightarrow$ and $\Downarrow$ relations of strand space theory. Therefore, for a send event $e^+ \in \psi$, there should be a receive event $e^- \in \psi$, where $m(e^+) = m(e^-)$. If $i \sqsubseteq m(e^+)$, then $i \sqsubseteq m(e^-)$. Hence $i = k$. $\square$

From Lemma 22, the following theorem is derivable.

Theorem 5. *If $p_1 \otimes p_2 \otimes \cdots \otimes p_n$ is a composition of indexed primitives, the following holds, where ψ_i is a bundle formed by primitive p_i.*

$$\forall \psi_i \bullet \xi^{i+}_{AB}(\omega_i) \in \psi_i \Rightarrow [\xi^{i-}_{BA}(\omega_i)] \in \psi_i$$
$$\forall \psi_i \bullet \xi^{i-}_{BA}(\omega_i) \in \psi_i \Rightarrow [\xi^{i+}_{AB}(\omega_i)] \in \psi_i$$

The theorem above is a generalisation of Lemma 22. It basically means that in a protocol composed of a set of indexed primitives, each primitive does not interfere with the others.

4.4.2 Goals and Bindings

In composition, a protocol is defined as a set of primitives with a predefined total order, and each primitive is defined by its parameter, i.e. a variable or a set of variables and its binding group. For a protocol produced by composition to be correct, goals accomplished by independent primitives should be inseparable from each other or the existence of one goal should guarantee the existence of some other goal. The simplest way to achieve such inseparability or dependency of goals is using a unique session identifier: adding a unique number to all the messages of the primitives that should occur together in a single session. In two party protocols, all terms generated by a protocol run are visible to both participants. Therefore, a unique session identifier will provide enough information for the participants to decide to which session a primitive run belongs. However, in some cases, such as protocols where a trusted third party is involved, some terms are not visible to some participants. Therefore, adding a unique identifier may not be sufficient.

Suppose $\mathcal{P} = p_i \otimes p_j$ is a composition of two primitives. Let g_i and g_j be the goals accomplished by the primitives p_i and p_j, respectively. A run of $\mathcal{P}$ is composed of a run of p_i and a run of p_j. Let $p_i(\vec{x}_i; \mathcal{B}_{\vec{x}})$ and $p_j(\vec{y}_j; \mathcal{B}_{\vec{y}})$ be the runs which should be linked together. The goals accomplished by $p_i(\vec{x}_i; \mathcal{B}_{\vec{x}})$ and $p_j(\vec{y}_j; \mathcal{B}_{\vec{y}})$ are $g_i(\vec{x}_i; \mathcal{B}_{\vec{x}})$ and $g_j(\vec{y}_j; \mathcal{B}_{\vec{y}})$, respectively. As a protocol is assumed to have a total order, there exists an order among the goals. Assume that $g_i(\vec{x}_i; \mathcal{B}_{\vec{x}}) \leq g_j(\vec{y}_j; \mathcal{B}_{\vec{y}})$. In order to make the two goals happen in a single session, some extra information is necessary. The extra information is called the *binding information* of the two primitives. A term which contains the binding information is called a *binder*. The binding information is generally defined as the union of the parameters of the primitives which need to happen together in a single run. For example, the binding information of the two primitives above can be defined as $(\vec{x}, \vec{y}, \mathcal{B}_{\vec{x}} \cup \mathcal{B}_{\vec{y}})$. A binder of the binding information can be any term(s) which delivers the binding information. For example, $\langle \vec{x}, \vec{y}, \mathcal{B}_{\vec{x}} \cup \mathcal{B}_{\vec{y}} \rangle$ can be a binder of $(\vec{x}, \vec{y}, \mathcal{B}_{\vec{x}} \cup \mathcal{B}_{\vec{y}})$. If more than one binder are necessary in composing a run, each binder needs to include an index to differentiate it from the others, i.e. $(k, \vec{x}, \vec{y}, \mathcal{B}_{\vec{x}} \cup \mathcal{B}_{\vec{y}})$. If necessary, a term or terms in a binder can be hashed or rearranged, e.g. $(k, \langle \vec{x} \rangle, \vec{y}, \mathcal{B}_{\vec{x}} \cup \mathcal{B}_{\vec{y}})$, $(k, \langle \vec{x} \rangle, \langle \vec{y} \rangle, \mathcal{B}_{\vec{x}} \cup \mathcal{B}_{\vec{y}})$, $(k, \langle \vec{x} \rangle, \vec{y}, \langle \mathcal{B}_{\vec{x}} \cup \mathcal{B}_{\vec{y}} \rangle)$, or $(k, \vec{x}, \mathcal{B}_{\vec{x}}, \vec{y}, \mathcal{B}_{\vec{y}})$.

A hash output of the binding information is generally preferable as a binder since it protects secrets which may be part of the binding information. In this case, to prevent any confusion which might happen between authentication replies and binders, hash functions are divided into two disjoint sets, H_1 and H_2. Since both binders and authentication use hash functions to generate their values, it is assumed that binders are generated by a hash function in H_2 and authentication replies by a hash function in H_1. This does not necessarily mean that different hash functions should be used in an implementation. Binders generally contain more terms than authentication replies, so the confusion does not happen. Alternatively, a binder can replace an authentication reply if the binder includes all the information of the authentication reply.

The ways of adding binders depend on the binding groups in primitives. If $\mathcal{B}_{\vec{y}}$ has no intersection with $\mathcal{B}_{\vec{x}}$, then the goals g_i and g_j are regarded as independent. Without any interaction between the members in $\mathcal{B}_{\vec{x}}$ and $\mathcal{B}_{\vec{y}}$, it is impossible to bind the two primitives together. Therefore, the two goals are seen as independent. We divide the cases to be considered into two groups, depending on the number of principals in composition, i.e. two-party and three-party cases. We do not consider cases of more than three principals.

Let e_i and e_j be the events which make $g_i(\vec{x}_i; \mathcal{B}_{\vec{x}})$ and $g_j(\vec{y}_j; \mathcal{B}_{\vec{y}})$ true, respectively. Assume $g_i \leq g_j$, i.e. $e_i \leq e_j$. Let $\mathcal{B}_{\vec{x}}$ be (A, B). Two subclasses of composition are as follows.

Two-party Composition

Given that $\mathcal{B}_{\vec{x}} = (A, B)$, two cases need to be considered when the binding groups in $\mathcal{P}$ are composed of the same members, i.e. either when $\mathcal{B}_{\vec{y}} = (A, B)$ or when $\mathcal{B}_{\vec{y}} = (B, A)$.

When $\mathcal{B}_{\vec{y}} = (A, B)$: The binding information is $\mathtt{c} = (\vec{x}, \vec{y}, \mathcal{B}_{\vec{x}})$ since $\mathcal{B}_{\vec{x}} = \mathcal{B}_{\vec{y}}$. From the binding information $\mathtt{c}$, both participants know the runs which constitute the binding information, i.e. $p_i(\vec{x}; \mathcal{B}_{\vec{x}})$ and $p_j(\vec{y}; \mathcal{B}_{\vec{y}})$. Only A or B should be allowed to send a binder. Therefore, the message m sent by one of the participants, which contains the binding information, will be either $\langle \mathtt{c} \rangle_{sk_A}$ or $\langle \mathtt{c} \rangle_{sk_B}$ in asymmetric key cryptosystems, and $\langle \mathtt{c} \rangle_{k_{AB}}$ or $\langle \mathtt{c} \rangle_{k_{BA}}$ in symmetric key cryptosystems.

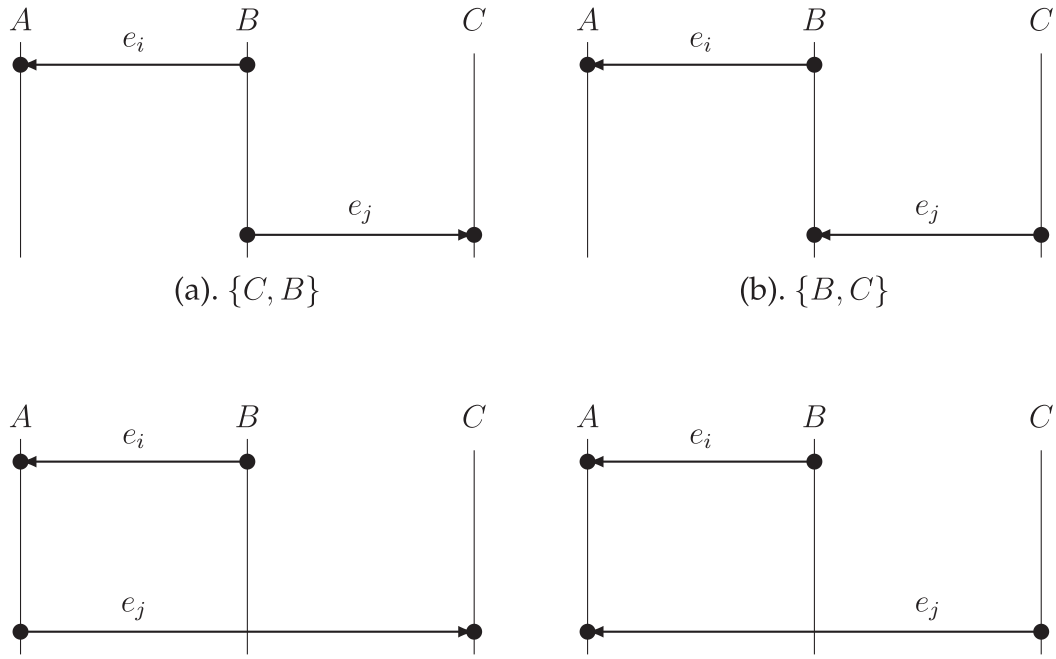
The position of m within the protocol sequence should be decided carefully. It might be possible to add m after e_j, but it is generally preferable to add m no later than e_j, i.e. $e(m) \leq e_j$, since it enables each participant to deduce the event e_j from m. Each participant agrees on the events belonging to the same run before the event e_j. If m is added after e_j, then there is a possibility that one participant, the one who sends m, finishes a run successfully, but the other participant may not know which two primitive runs constitutes a single session, since m may not be delivered to the other participant due to some network problem. This is generally not a desirable situation. More detailed explanations together with applications will be given in Chapter 5.

When $\mathcal{B}_{\vec{y}} = (B, A)$: This case is the same as the previous case, if $\mathtt{c} = (\vec{x}, \vec{y}, \mathcal{B}_{\vec{x}})$ is interpreted as the binding information of the two parameters, $(\vec{x}; \mathcal{B}_{\vec{x}})$ and $(\vec{y}; \mathcal{B}_{\vec{y}})$.

It is obvious that given the binder defined above, the two participants can decide which two primitive runs belong to the same session.

Three-party Composition

Three-party composition is more complicated than two-party composition. Many cases need to be considered to cover all possible combinations of primitives. Some possible cases of $\mathcal{B}_{\vec{y}}$ are shown in Figure 4.1. The remaining cases of three-party composition can be obtained from the figure by reversing the directions of the primitive p_i, i.e. $p_i(\vec{x}; \mathcal{B}_{\vec{x}})$ where $\mathcal{B}_{\vec{x}} = \{B, A\}$.



(a). $\{C, B\}$ (b). $\{B, C\}$

(c). $\{A, C\}$ (d). $\{C, A\}$

Figure 4.1: **Three-party Composition**

In two-party composition, both participants know all the events that occurred during a run based on the events they performed. However, this is not the case with three-party composition. Those events in which a participant is not directly engaged, might not be visible to the participant. For example, in case (a), C might not know whether e_i has happened between A and B. In other words, the existence of some events is not clearly visible to some participants.

In three-party composition of Figure 4.1, only one participant sees both events, and the other two participants do not. Let p_i be a part of a protocol which A and B are engaged in. After finishing a run of p_i with a parameter ω, one or both of the participants A and B might contact C to achieve some other goals. For example, A may send to B some information, which B cannot understand without the help of C, or which B can use to communicate with C. Sometimes, C might be interested in the existence of some specific interactions between A and B. Especially when the participant who sees all the events is a trusted authentication server, the binding problem of three-party composition becomes much easier. We will discuss this problem in Chapter 6.

Except for the cases when some trust relations exist among participants or when a secret is shared between two participants who are not directly engaged in a run, it is very difficult to prove the existence of a certain event, since some events are invisible to some participants. For example, adding a unique identifier does not provide any evidence to C that some events have happened between A and B even though it might piece some runs of primitives together. The cases when C plays a role of a verifier or C shares a secret with either of A or B will be described through some interesting examples in Chapter 7 .

5

Two-party Protocols

Two-party security protocols are one of the most common and basic forms of security protocols found in published literature [11]. As demonstrated by Guttman [24], three-party protocols can be implemented using two-party protocols. However, in spite of a long history of research on the subject, little is known about their design principles. This chapter demonstrates how to construct two-party authentication protocols by composition of protocol primitives using the theories presented in the previous chapter. Section 5.1 explains design by composition. Section 5.2 shows how to maintain the discreetness property of composition and how to compose a single agreed run from any composition of primitives in two-party environments. Finally, Section 5.3 shows examples of applications.

5.1 Design By Composition

Regardless of design methodologies, it is important for protocol designers to know exactly what they are planning to achieve in a new protocol because any mistakes in the process lead to flaws in the protocol obtained. Authentication and secrecy are generally considered as the two most important protocol goals. Non-repudiation is another goal to be considered in some cases. Protocol goals are usually given to designers either as protocol specifications or requirements.

Suppose that a protocol needs to accomplish a set of goals. There are various ways to implement these goals. Conventional design methodologies usually try to tackle all these goals as a whole instead of solving them one by one. However, it is not easy to find a protocol which satisfies all the goals at the same time. As the number of goals increases, it becomes more difficult and takes more time to find a correct protocol using conventional methodologies. On top of that, the correctness of the protocols produced by conventional methodologies is normally not guaranteed. In order to solve these problems of conventional approaches, we provide a way in which designers can work with each goal individually, one by one, without concerning themselves with the other goals. We call this type of design methodology *design by composition* or in short *composition*. Devising solutions for one goal at a time is obviously much easier for designers than providing solutions serving multiple goals. In composition, for the given set of goals, each goal in the set is implemented individually using a non-interfering mechanism, which is proven to be correct, and then all these mechanisms are tied together using binders to make all the goals happen at the same time. In the design, each authentication or secrecy goal is implemented by protocol primitives and under the assumption that long-term keys are safe from Spy. Authentication subgoals are considered before secrecy subgoals

Protocol design by composition is similar to modular approaches or top-down methodologies which are commonly used in software design. That is to say, protocol composition not only provides a way of building up complex goals from a group of simple goals, making it easier to verify their correctness, but also guarantees the correctness of the protocol constructed by the approach.

5.2 Two-party Protocol Composition

This section explains how to compose a two-party protocol from protocol primitives and also describes how to bind these protocol primitives together.

5.2.1 p-Protocols and Semi-bundles

Suppose that $\mathcal{G} = \{g_1, g_2, \ldots, g_n\}$ is the set of goals which a protocol should implement, where g_i is an authentication or secrecy goal achieved by a pair of participants, A and B. The first step of composition is to select an appropriate primitive which achieves each subgoal in $\mathcal{G}$. Let $\mathcal{P}_\mathcal{G} = \{p_1, p_2, \ldots, p_n\}$ be a set of primitives, which are selected to implement the protocol, i.e. a set of primitives which cover all the goals in $\mathcal{G}$. Let $\mathcal{P} = p_1 \otimes p_2 \otimes \cdots \otimes p_n$. The events in each primitive p_i have a particular order $o(p_i)$ that the composition $\mathcal{P}$ cannot violate. A protocol is a set of message rules agreed between participants, so the composition $\mathcal{P}$ as a protocol also has an order $o(\mathcal{P})$. The order $o(\mathcal{P})$ is generally defined by two things, the orders of each primitive, $o(p_1)$, $o(p_2)$, ..., $o(p_n)$, and the order of the goals $o(\mathcal{G})$. If secrecy goals are ignored in $\mathcal{G}$, then a goal g_i is accomplished by a specific event of p_i. Therefore, the order of the goals to be achieved determines the order of the events in the composition. In other words, the order of the goals can be understood as design constraints imposed on the order of the events. Putting more constraints on the order of the events reduces the complexity of the design. Under the assumption that a protocol has a total order, $o(\mathcal{G})$ can initially be introduced by the protocol requirements or it can later be added by the designer to provide a total order to the protocol.

Assume that the order $o(\mathcal{P})$ is commonly understood among participants as an agreement before the start of a protocol run, then a set of primitives together with the order $o(\mathcal{P})$ define a *primitive protocol* or *p-protocol*.

Definiton 27. (p-protocols) *Given a set of primitives $\mathcal{P}_g = \{p_1, p_2, \ldots, p_n\}$ with some predefined order $o(\mathcal{P})$, a p-protocol $\mathcal{P}$ of $\mathcal{P}_g$ is defined as a composition of all the primitives in the set, which does not violate any orders $o(p_i)$, and is denoted as follows.*

$$\mathcal{P} = |p_1 \otimes p_2 \otimes \cdots \otimes p_n|$$

A p-protocol can be understood as a collection of actions which might happen if all the primitives in the set are executed by the order $o(\mathcal{P})$. In case of two-party protocols, the order is assumed to be a total order. A p-protocol is an intermediate form defining the basic structure of the final protocol.

Let $p_i(\omega_i)$ be a primitive with $\omega_i = (a_i; \mathcal{B}_a)$. Even though each participant may play a different role in a different primitive, as a whole he only plays one role, say A an initiator and B a responder. Therefore, a binding group $\mathcal{B}_{AB} = (A, B)$ can replace all the binding groups in the primitives, i.e. $p_i(a_i; \mathcal{B}_{AB})$. From this point forward, when a p-protocol is mentioned, it is assumed that this step is already applied to the p-protocol, and the notation $p_i(a_i)$ is used instead of $p_i(a_i; \mathcal{B}_{AB})$.

Each primitive p_i of $\mathcal{P}$ will produce a bundle composed of two strands, denoted as $\psi_i(a_i) = \{\xi^{p_i}_{AB}(a_i), \xi^{p_i}_{BA}(a_i)\}$. At the end of a p-protocol run, each participant has a view of the protocol run, formulated as a sequence of events in a strand. Let these strands be $\xi^{\mathcal{P}}_{AB}(\omega_A)$ and $\xi^{\mathcal{P}}_{BA}(\omega_B)$, respectively, i.e.

$$\xi^{\mathcal{P}}_{AB}(\omega_A) = \bigcup_i \xi^{p_i}_{AB}(a_i) \qquad \text{and} \qquad \xi^{\mathcal{P}}_{BA}(\omega_B) = \bigcup_i \xi^{p_i}_{BA}(b_i)$$

Technically speaking, these two strands together may not form a trace of a single run of $\mathcal{P}$ because each participant may see a different part as a part of a specific run when multiple instances of the protocol are executed concurrently. In other words, although each run of a primitive forms a bundle, $\mathcal{P}$ as a whole may not form a bundle since the parameters which each participant sees might not be the same. The collection of strands that each participant sees in a run of a p-protocol is called a *semi-bundle*. A semi-bundle shows how a participant understands a specific run among multiple runs.

Definiton 28. (Semi-bundles) *Given a p-protocol $\mathcal{P}$ between two agents A and B, where $\mathcal{P} = |p_1 \otimes p_2 \otimes \cdots \otimes p_n|$, a semi-bundle of each participant in a run of $\mathcal{P}$ is defined as follows, where $|\ldots|_A$ represents events on strand A.*

$$\xi^{\mathcal{P}}_{AB}(\omega_A) = |p_1(a_1) \otimes p_2(a_2) \otimes \cdots \otimes p_n(a_n)|_A$$
$$\xi^{\mathcal{P}}_{BA}(\omega_B) = |p_1(b_1) \otimes p_2(b_2) \otimes \cdots \otimes p_n(b_n)|_B$$

where ω_A and ω_B are parameters which participants use to define a single run, and for each a_i and b_i, it is not necessary $a_i = b_i$.

Remark 2. *Notice that we need not have $\omega = (a_1, a_2, \ldots, a_n, \mathcal{B}_{AB})$ because some redundant elements can be removed from each a_i.*

A p-protocol is regular if no long-term safe keys are used as a part of protocol messages. Therefore, a p-protocol obtained from a set of regular primitives is regular. Some primitives in $\mathcal{P}$ may generate short-term secrets. Let $\mathcal{P}_{\mathcal{S}} = \{p'_1, p'_2, \ldots, p'_m\}$ be the set of primitives which generate short-term secrets i.e. $\mathcal{P}_{\mathcal{S}} \subseteq \mathcal{P}_{\mathcal{G}}$, and $\mathsf{S}_{\mathcal{P}} = \{s_1, s_2, \ldots, s_m\}$ be the set of those short-term secrets generated by $\mathcal{P_S}$. Let $\mathcal{S}_{s_i}$ be the protective domain of a secret s_i, and $\mathcal{S_P}$ be the union of all the protective domains of $\mathcal{P}$, i.e.

$$\mathcal{S_P} = \bigcup_i \mathcal{S}_{s_i} = \mathsf{S}_{\mathcal{P}} \cup \{\mathsf{prv}(X)|X \in \mathcal{B}_{AB}\} \cup \{\mathsf{shr}(X)|X \in \mathcal{B}_{AB},\}^1$$

If $\mathcal{P}$ is discreet, then $\mathcal{P}$ does not emit a secret to a participant C ($\neq A, B$). A discreet status of $\mathcal{P}$ would result in every trace of a run of the protocol being discreet, and also in every trace of multiple runs of the protocol being discreet. Notice that the discreetness property of a protocol is not particularly related with a single run. From Lemma 21, the following should hold for a p-protocol to be discreet.

1. For each $p_i \in \mathcal{P_S}$, p_i is discreet.
2. For every message m_j generated by a primitive p_j ($\neq p_i$), $m_j \subseteq \mathcal{C}(\mathcal{S_P})$.

Each primitive in $\mathcal{P_S}$ is supposed to be discreet by the choice of the primitive, so, the first condition is true. The second condition is generally true, if any short-term secret used in p_i is not used again as a parameter of any other primitive in $\mathcal{P_N}$ ($= \mathcal{P_G} - \mathcal{P_S}$). More specifically, any term in the ideal of s_i should not be used as a parameter of any primitive of $\mathcal{P_G}$. However, if it is assumed that two participants of the protocol are honest and they only use basic terms as nonces or secrets, then not using a secret again as a parameter of a different primitive is sufficient to guarantee the discreetness property of the protocol.

Lemma 23. *A p-protocol $\mathcal{P}$ of two-party composition is discreet, if two participants of the protocol only use basic terms as nonces or secrets, and if for each secret $s_i \in \mathsf{S}_{\mathcal{P}}$ and for all w_j, s_i is not used in w_j, where w_j is the parameter of primitive p_j ($j \neq i$).*

Proof. Assume that $\mathsf{S}_{\mathcal{P}}$ is I-compatible, where I is an initial knowledge set of Spy. Otherwise, there is no secret to be protected. Suppose that $\mathcal{P}$ is a composition of two different types of primitives, $\mathcal{P_S}$ and $\mathcal{P_N}$, as previously defined. The primitives in $\mathcal{P_S}$ do not reveal any secret due to the discreetness property of the primitives. Each primitive p_i in $\mathcal{P_S}$ generates two types of messages, encrypted or hashed. From the definition of the

[1] When an agent has more than one private or secret key, a private or secret key whose inverse key is not used to encrypt any secret in $\mathsf{S}_{\mathcal{P}}$ does not need to be included in $\mathcal{S_P}$.

ideal, all hashed messages automatically belong to $\mathcal{C}(\mathcal{S}_\mathcal{P})$. An encrypted message $\{|x|\}_k$ belongs to $\mathcal{I}[\mathcal{S}_\mathcal{P}]$, either when $\{|x|\}_k \in \mathcal{I}[\mathcal{S}_\mathcal{P}]$, or when $x \in \mathcal{I}[\mathcal{S}_\mathcal{P}]$ and $k^{-1} \notin \mathcal{I}[\mathcal{S}_\mathcal{P}]$. The first case does not happen because only basic terms are allowed as secrets. The second case is impossible because long-term keys are assumed safe from Spy and for each encrypted term, a corresponding binding group is included in the term. Therefore, all messages generated by the primitives in $\mathcal{P}_\mathcal{S}$ are discreet.

Moreover, if these secrets are not used as a part of the parameter of p_i in $\mathcal{P}_\mathcal{N}$, then no elements in $\mathcal{S}_\mathcal{P}$ are revealed by messages of $\mathcal{P}_\mathcal{N}$. When a secret s_i is initially unknown to Spy, the only way for Spy to get it is to possess a term $x \in \mathcal{I}[s_i]$. To show that terms generated by $\mathcal{P}_\mathcal{N}$ belong to $\mathcal{C}[\mathcal{S}_\mathcal{P}]$, assume that x is one of those terms generated by $\mathcal{P}_\mathcal{N}$ but belongs to the ideal. Messages generated by $\mathcal{P}_\mathcal{N}$ are either hashed or plain. Hashed terms do not belong to the ideal, so x is not a hashed term. The term x belongs to $\mathcal{I}[\mathcal{S}_\mathcal{P}]$, either when $x = a \cdot b$ and $a \in \mathcal{I}[\mathcal{S}_\mathcal{P}]$ or $b \in \mathcal{I}[\mathcal{S}_\mathcal{P}]$, or when $x = \{|y|\}_k$ and $y \in \mathcal{I}[\mathcal{S}_\mathcal{P}]$ and $k^{-1} \notin \mathcal{I}[\mathcal{S}_\mathcal{P}]$. The first case contradicts the assumption that secrets are not used as a part of the parameters of the primitives in $\mathcal{P}_\mathcal{N}$ by honest participants, and Spy does not know the secrets. For the second case to happen, an honest participant who knows the secrets should generate $\{|y|\}_k$ as part of the messages of the primitives in $\mathcal{P}_\mathcal{N}$. However, this is impossible if only basic terms are used as nonces by honest participants. Moreover, for any given challenge, the primitives in $\mathcal{P}_\mathcal{N}$ do not generate any encrypted terms as an output, so Spy cannot use the primitives as an oracle. Hence, $\mathcal{P}$ is discreet. $\square$

In conclusion, composition of primitives is discreet if there is no duplicate use of a variable, especially a secret. Moreover, adding any terms in the coideal to $\mathcal{P}$ does not break the discreetness property of $\mathcal{P}$, so the following holds.

Lemma 24. *Given a discreet p-protocol $\mathcal{P}$ and its* I*-compatible set of secrets* $\mathsf{S}_\mathcal{P}$, *a protocol $\mathcal{P}'$ which obtained by adding events e to $\mathcal{P}$ is discreet if $m(e) \subseteq \mathcal{C}(\mathcal{S}_\mathcal{P})$.*

Basically, any terms belonging to the coideal can be added to $\mathcal{P}$ to generate a new discreet protocol, but we put some restrictions on this in generating a protocol $\mathcal{P}'$. First, we only add terms which are produced by hash functions. Signatures without message recovery feature are assumed to be produced after hashing, therefore, adding these terms does not destroy the discreetness property. Second, we do not add terms randomly. For example, if a certain term is unknown, its hashing cannot be calculated. Generally speaking, a term y which needs a term x as its hash input comes after x.

However, if it is assumed that every term x originating from A is known to A at the start of a run, then using a hash value as in the protocols of Anderson et al. [3], that is, sending a hash output before sending its input, is not a violation of the restriction. However, if the hash output needs an input from B, then the output can be available after a receiving event of the input from B.

5.2.2 Constituting a Session

As shown in the previous subsection, a semi-bundle does not form a bundle, so it does not guarantee the existence of its corresponding semi-bundle. This happens because a p-protocol composed of more than two primitives lacks information on which primitive runs are directly related. Therefore, binding information is necessary to help both participants agree on the parameter of a run. In two-party protocols, all messages generated in a run are visible to both participants, so the simplest solution is to add a global session identifier, a unique number which defines a specific session. Suppose s_{id} is a unique number generated by a participant A. There is no need for s_{id} to be a random number: it can be a sequential number. Let ω_i be a parameter of a primitive p_i, i.e. $\omega_i = (a_i, \mathcal{B}_{a_i})$. Replace a_i with $a'_i = (s_{id}, a_i)$. Now, each message contains a session identifier, and each participant can identify primitive runs belonging to the same session. Notice that the integrity of the session identifier is protected by $auth_k(a'_i, \mathcal{B}_{a'_i})$ message.

The other solution is to add an extra message, called a *binder*, containing the necessary binding information. For example, in order to bind primitive runs $p_1(a_1; \mathcal{B})$, $p_2(a_2; \mathcal{B})$, ..., $p_n(a_n; \mathcal{B})$, a message delivering $\mathtt{c} = (a_1, a_2, \ldots, a_n, \mathcal{B})$ can be added. This message can originate from either participant, A or B. The origination does not matter as long as it comes from one of the two participants involved in a protocol run. However, the message might reveal some secrets, so instead of $\mathtt{c}$, its hash is generally preferred. For two-party protocol composition, either $\langle \mathtt{c} \rangle_{sk_A}$ or $\langle \mathtt{c} \rangle_{sk_B}$ will serve as the binder.

The position of a binder in a p-protocol $\mathcal{P}$ should be decided carefully. The events in $\mathcal{P}$ are supposed to have a total order, $o(\mathcal{P})$. Therefore, the events can be sorted according to the order $o(\mathcal{P})$. If $o(\mathcal{P})$ is not defined yet, then any order can be used provided the order of each primitive $o(p_i)$ is not violated. For example, if $\mathcal{P} = p_1 \otimes p_2$, where p_i is composed of events e_{i1} and e_{i2}, then Figure 5.1 shows all possible orderings when merging of messages is not considered. Two adjacent events may be merged if they originate from the same participant, but this does not make any difference in the discussion.

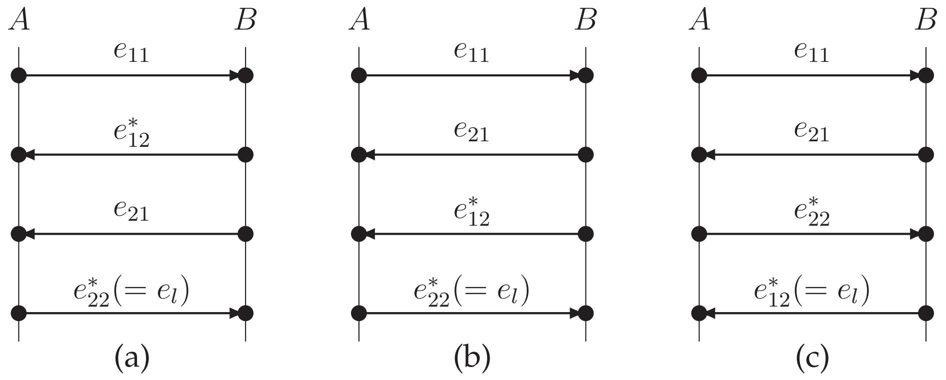


Figure 5.1: **Possible Orderings of Two Primitives**

The starred events are authentication replies. The last starred event e_l in each ordering is important because it finishes a protocol run. A binder m_c can be added any place between events in the orderings above, but special attention needs to be paid to the following cases.

$e_l^- < e^+(m_c)$ **and** e_l^-**,** $e^+(m_c)$ **occur in the same participant** : The last event e_l^- is assumed to finish a run. Therefore, if $e^+(m_c)$ comes after e_l^-, then the participant A performing the event e_l^- achieves all the goals in the session defined by the binder m_c, but the other participant B may not know the binding before he receives m_c even though he might have performed all corresponding events of the session. Moreover, the message m_c can be lost during its transmission. This is not a favourable situation even though it does not necessarily lead to flaws in the protocol. In order to avoid this situation, the order restriction $e^+(m_c) \leq e_l^+$ is added to $o(\mathcal{P})$, which makes the event e_l^+ happen only after $e^-(m_c)$ occurred, i.e. the sender of the last message does not send the last message before he receives the message m_c. For example, in the Needham-Schroeder public key protocol, A sends the message $\{|N_B|\}_{ek_B}$ only after she received $\{|N_B, N_A|\}_{ek_A}$, where $\{|N_B, N_A|\}_{ek_A}$ is m_c.

$e^+(m_c) \leq e_l^+$ **and** e_l^+**,** $e^+(m_c)$ **occur in the same participant** : $e^+(m_c)$ happens before e_l^+. Assume that both are the events of A communicating with B. If there is no receive event between them, i.e. A sends $e^+(m_c)$ and then sends e_l^+ without receiving any message from B, then their corresponding events, $e^-(m_c)$, e_l^-, may not occur in the same order. This gives a possibility that $e^-(m_c)$ happens after e_l^- on B. Therefore, the situation previously described may happen in this case, too. When indexes in the messages are ignored, generally $m(e^l) \sqsubseteq m_c$ and both events e_l^+, $e^+(m_c)$ originate from the same participant. Therefore, m_c is enough to deliver all necessary information of $m(e_l)$, that is, e_l can be deleted from the p-protocol.

The two rules above ensure that either A or B performs events in the order, either $e^-(m_c) < e_l^+$ or $e^-(m_c) \leq e_l^-$. Generally speaking, the two events, $e^-(m_c)$ and the last event which lies on the same strand as $e^-(m_c)$, i.e. either e_l^- or e_l^+, define the order restriction. The order of the protocol after adding the binder is obtained by adding the order restriction to $o(\mathcal{P})$.

Let $\mathcal{P}'$ be the protocol obtained by adding binding events to $\mathcal{P}$. The following is derivable.

Lemma 25. *If $\mathcal{P}' = |p_1 \otimes p_2 \otimes \cdots \otimes p_n \otimes p_o|$, where p_o is a set of binding events added using the rules above, then $\mathcal{P}'$ satisfies the agreement property.*

$$\forall \psi \bullet \xi_{AB}^{\mathcal{P}'}(\omega) \in \psi \Rightarrow [\xi_{BA}^{\mathcal{P}'}(\omega)] \in \psi$$
$$\forall \psi \bullet \xi_{BA}^{\mathcal{P}'}(\omega) \in \psi \Rightarrow [\xi_{AB}^{\mathcal{P}'}(\omega)] \in \psi$$

where $\xi_{AB}^{\mathcal{P}'}(\omega) = |p_1 \otimes p_2 \otimes \cdots \otimes p_n \otimes p_o|_A$ and $\xi_{BA}^{\mathcal{P}'}(\omega) = |p_1 \otimes p_2 \otimes \cdots \otimes p_n \otimes p_o|_B$.

Proof. From Theorem 5, the following holds for each run of primitive p_i.

$$\forall \psi_i \cdot \xi_{AB}^{p_i}(a_i) \in \psi_i \Rightarrow [\xi_{BA}^{p_i}(a_i)] \in \psi_i$$

$$\forall \psi_i \cdot \xi_{BA}^{p_i}(a_i) \in \psi_i \Rightarrow [\xi_{AB}^{p_i}(a_i)] \in \psi_i$$

For a participant A to finish a run successfully, all actions she should perform have to be done, if she is honest. The last action she performs can be either a receive event or a send event.

Last event is a receive event e_l^- : if a binder m_c is added by A, then $e^+(m_c) < e_l^+$ due to the first rule above. Therefore, B receives m_c before he performs e_l^+. If m_c

originates from B, then the second rule forces $e^-(m_c) \leq e_l^-$. Therefore, the binder always becomes known before the last event and both participants can agree on a specific run.

Last event is a send event e_l^+ : if a binder m_c is added by A, then e_l^+ is replaced with $e^+(m_c)$, so either B finishes a successful run when m_c is delivered to B, or he does not achieve all of his goal when m_c is not delivered. If m_c is added by B, then it comes before the event e_l^+, otherwise, A will not generate e_l^+. Therefore, both agrees on a specific run.

Therefore, both participants always agree on a specific parameter before they finish a run, and the parameter links all primitive runs belonging to the run, i.e. the agreement property is satisfied. □

Lemma 25 enables designers to compose a protocol which satisfies the agreement property. Sometimes, it is preferable to put a binder no later than the first authentication reply, because this enables both participants to agree on the parameter of a specific run as early as possible. This kind of binding is called *early binding*. Early binding is not always possible, but it is recommended because it provides the binding information to both participants before the first authentication reply in a composition, so that both participants know which primitives belong to the same session before the first authentication reply arrives.

5.3 Design Example

This section shows how to use the proposed method through examples of applications such as one-way authentication, two-way authentication and SSL/TLS-like protocols.

5.3.1 One-way Authentication

One-way authentication is the simplest form of authentication. It only authenticates one party, for example, a wireless access point will authenticate a user, but not the other way around. Several applications such as password authentication protocols use this type of authentication. One disadvantage of one-way authentication is that it leaves an unauthorised party as a potential launch pad for denial of service attacks. Transforming a protocol to make it resilient to denial of service attacks is usually possible by introducing mechanisms such as client puzzles [5,14,29], which lies outside of the scope of this thesis. One-way authentication is interesting as a building block for more complicated protocols. Understanding how to compose a compound one-way authentication protocol from simple primitives will help assessing the feasibility of more complex composition. This subsection shows some interesting examples of one-way authentication composition.

A primitive as an authentication test can implement one-way authentication. Most security protocols that rely on a simple challenge and response mechanism can be implemented with a single primitive. There are, however, other applications that require a more complex mechanism for one-way authentication which may not be achievable with a single primitive, such as re-authentication. One way of generating those complex mechanisms is to use a composition of primitives. Any two or more primitives can be combined together to generate a new mechanism for one-way authentication.

Let p_1 and p_2 be the two primitives as shown below, where $\mathcal{B} = (A, B)$.

$$\begin{array}{llll} p_1\text{:} & M_1 & A \rightarrow B: & \mathcal{B}, \{\!|1, x, \mathcal{B}|\!\}_{ek_B}, auth_{sk_A}(1, x, \mathcal{B}) \\ & M_2 & B \rightarrow A: & A, \langle 1, x, \mathcal{B}\rangle_{sk_B} \\ p_2\text{:} & N_1 & A \rightarrow B: & y, \mathcal{B}, auth_{sk_A}(2, y, \mathcal{B}) \\ & N_2 & B \rightarrow A: & A, \langle 2, y, \mathcal{B}\rangle_{sk_B} \end{array}$$

Notice that the binding group $\mathcal{B}$ in $\{\!|1, x, \mathcal{B}|\!\}_{ek_B}$ can be removed because it is included in $auth_{sk_A}(1, x, \mathcal{B})$. The events within each primitive have a specific order, i.e.

$$\begin{array}{lll} o(p_1) & : & e^+(M_1) < e^-(M_1) < e^+(M_2) < e^-(M_2) \\ o(p_2) & : & e^+(N_1) < e^-(N_1) < e^+(N_2) < e^-(N_2) \end{array}$$

These orders cannot be violated after composition. The events of p_1 are independent of the events of p_2. Let $\mathcal{P}$ be a composition of p_1 and p_2, that is, $\mathcal{P} = p_1 \otimes p_2$. Any rearrangement of the events of $\mathcal{P}$ is possible provided it does not violate the order of each primitive. For example, the messages with the same direction can be merged together to produce the following $\mathcal{P}$.

$$\begin{array}{llll} \mathcal{P}\text{:} & M_1' & A \rightarrow B: & y, \mathcal{B}, \{\!|1, x, \mathcal{B}|\!\}_{ek_B}, auth_{sk_A}(1, x, \mathcal{B}), auth_{sk_A}(2, y, \mathcal{B}) \\ & M_2' & B \rightarrow A: & A, \langle 1, x, \mathcal{B}\rangle_{sk_B}, \langle 2, y, \mathcal{B}\rangle_{sk_B} \end{array}$$

In order to piece the two primitives together, a binder should be added. The union of the parameters is the binding information, so $\mathtt{c} = (x, y, \mathcal{B})$. An index is not necessary because only one binder is necessary in this example. Two different versions of the binding are possible depending on who is responsible for the binding. As mentioned in the previous chapter, all events are visible to both participants in this case, so the binder can be added using the rules presented earlier.

When A is responsible for the binding

$$\begin{array}{llll} \mathcal{P}_1\colon & M_1' & A \to B: & y, \mathcal{B}, \{\!|1, x, \mathcal{B}|\!\}_{ek_B}, auth_{sk_A}(1, x, \mathcal{B}), auth_{sk_A}(2, y, \mathcal{B}) \\ & M_1'' & A \to B: & \langle \mathtt{c} \rangle_{sk_A} \\ & M_2' & B \to A: & A, \langle 1, x, \mathcal{B} \rangle_{sk_B}, \langle 2, y, \mathcal{B} \rangle_{sk_B} \end{array}$$

When B is responsible for the binding

$$\begin{array}{llll} \mathcal{P}_2\colon & M_1' & A \to B: & y, \mathcal{B}, \{\!|1, x, \mathcal{B}|\!\}_{ek_B}, auth_{sk_A}(1, x, \mathcal{B}), auth_{sk_A}(2, y, \mathcal{B}) \\ & M_2'' & B \to A: & \langle \mathtt{c} \rangle_{sk_B} \\ & M_2' & B \to A: & A, \langle 1, x, \mathcal{B} \rangle_{sk_B}, \langle 2, y, \mathcal{B} \rangle_{sk_B} \end{array}$$

Given $\mathtt{c}$, either as $\langle \mathtt{c} \rangle_{sk_A}$ or as $\langle \mathtt{c} \rangle_{sk_B}$, both participants can determine which two primitive runs belong together, so the composition together with the binder satisfies the agreement property. However, the second example needs more explanation since it does not remove the possibility that the binder arrives at B later than the second authentication reply. Therefore, it might be better to remove the second authentication reply to avoid such a situation. If indexes are ignored, then $x, \mathcal{B} \sqsubseteq \mathtt{c}$ and $y, \mathcal{B} \sqsubseteq \mathtt{c}$, so these terms can be removed if necessary, i.e. the binder can replace any of these authentication replies as well as any $auth$ messages if the binder and these messages have the same direction. Generally, this kind of removal produces more efficient protocols but it is up to designers to decide whether to keep these redundant terms or not.

The addition of the binding events puts more order restrictions on $o(\mathcal{P})$. For example, if M_2' is assumed to be the last message of the protocol $\mathcal{P}_1$, the order restriction $e^-(\langle \mathtt{c} \rangle_{sk_A}) < e^+(M_2')$, meaning that B generates M_2' after receiving $\langle \mathtt{c} \rangle_{sk_A}$, should be added to $o(\mathcal{P})$ to produce $o(\mathcal{P}_1)$. Both $\mathcal{P}_1$ and $\mathcal{P}_2$ can be rearranged if the rearrangement does not violate the orders $o(\mathcal{P}_1)$ and $o(\mathcal{P}_2)$, respectively.

When A is responsible for the binding

$$\begin{array}{llll} \mathcal{P}'_1\text{:} & M_1 & A \to B: & y, \mathcal{B}, \{\!|1, x, \mathcal{B}|\!\}_{ek_B}, \langle \mathsf{c} \rangle_{sk_A} \\ & M_2 & B \to A: & A, \langle 1, x, \mathcal{B} \rangle_{sk_B}, \langle 2, y, \mathcal{B} \rangle_{sk_B} \end{array}$$

When B is responsible for the binding

$$\begin{array}{llll} \mathcal{P}'_2\text{:} & M_1 & A \to B: & y, \mathcal{B}, \{\!|1, x, \mathcal{B}|\!\}_{ek_B}, auth_{sk_A}(1, x, \mathcal{B}), auth_{sk_A}(2, y, \mathcal{B}) \\ & M_2 & B \to A: & A, \langle 1, x, \mathcal{B} \rangle_{sk_B}, \langle \mathsf{c} \rangle_{sk_B} \text{ or } A, \langle \mathsf{c} \rangle_{sk_B} \end{array}$$

This kind of composition allows designers to generate more complicated primitives. Primitives with more than two variables, each serving different purposes can be produced. In the example, x can be used as a secret and y as a nonce. The structures of the protocol can further be changed if necessary. The following is one example which does not violate the order $o(\mathcal{P}_1)$. Either by merging or reordering messages, many different protocols can be produced without worrying about their security guarantees.

$$\begin{array}{llll} \mathcal{P}''_1\text{:} & M'_1 & A \to B: & y, \mathcal{B}, \langle \mathsf{c} \rangle_{sk_A} \\ & M_1 & A \to B: & \{\!|1, x, \mathcal{B}|\!\}_{ek_B} \\ & M_2 & B \to A: & A, \langle 1, x, \mathcal{B} \rangle_{sk_B}, \langle 2, y, \mathcal{B} \rangle_{sk_B} \end{array}$$

5.3.2 Two-way Authentication

Two-way authentication, or mutual authentication, is a common two party protocol. The design of two-way authentication protocols is similar to the previous example of one-way authentication. Any two or more primitives can be combined to generate a two-way authentication protocol. Suppose p_1 and p_2 are the selected primitives which are shown below, where A is an initiator and B a responder.

$$\begin{array}{llll} p_1\text{:} & M_1 & A \to B: & \mathcal{B}_x, \{\!|1, x, \mathcal{B}_x|\!\}_{ek_B}, auth_{sk_A}(1, x, \mathcal{B}_x) \\ & M_2 & B \to A: & A, \langle 1, x, \mathcal{B}_x \rangle_{sk_B} \\ p_2\text{:} & N_1 & B \to A: & y, \mathcal{B}_y, auth_{sk_B}(2, y, \mathcal{B}_y) \\ & N_2 & A \to B: & B, \langle 2, y, \mathcal{B}_y \rangle_{sk_A} \end{array}$$

Let $\mathcal{P} = p_1 \otimes p_2$, where both $\mathcal{B}_x$ and $\mathcal{B}_y$ are replaced by $\mathcal{B} = (A, B)$. The events within each primitive have a specific order, and the event $e^+(M_1)$ should precede any other events because A is an initiator, i.e.

$$e^+(M_1) < e^-(M_1) < e^+(M_2) < e^-(M_2)$$

$$e^+(N_1) < e^-(N_1) < e^+(N_2) < e^-(N_2)$$
$$e^+(M_1) < e^+(N_1)$$

As shown in Figure 5.1, three different orderings are possible. However, (a) and (b) can be regarded as the same case by merging two events e_{12} and e_{21}, i.e. by removing the order restriction, $e_{12} \leq e_{21}$ or $e_{21} \leq e_{12}$. The two events can later be rearranged, i.e. the order restriction can later be added. The way a binder is inserted is the same as in one-way authentication examples. In one-way authentication examples, two authentication replies are merged, which removes any order restriction between them. However, in two-party protocols, that is not possible. Therefore, the last event or last message of the protocol should be decided before adding a binder. The starred events are $e(M_2)$ and $e(N_2)$. Therefore, the following cases need to be considered separately.

When $e(M_2) < e(N_2)$

$$\begin{array}{llll} \mathcal{P}: & M_1 & A \rightarrow B: & \mathcal{B}, \{\!|1, x, \mathcal{B}|\!\}_{ek_B}, auth_{sk_A}(1, x, \mathcal{B}) \\ & M' & B \rightarrow A: & y, \mathcal{B}, \langle 1, x, \mathcal{B}\rangle_{sk_B}, auth_{sk_B}(2, y, \mathcal{B}) \\ & N_2 & A \rightarrow B: & B, \langle 2, y, \mathcal{B}\rangle_{sk_A} \end{array}$$

When $e(N_2) < e(M_2)$

$$\begin{array}{llll} \mathcal{P}: & M_1 & A \rightarrow B: & \mathcal{B}, \{\!|1, x, \mathcal{B}|\!\}_{ek_B}, auth_{sk_A}(1, x, \mathcal{B}) \\ & N_1 & B \rightarrow A: & y, \mathcal{B}, auth_{sk_B}(2, y, \mathcal{B}) \\ & N_2 & A \rightarrow B: & B, \langle 2, y, \mathcal{B}\rangle_{sk_A} \\ & M_2 & B \rightarrow A: & A, \langle 1, x, \mathcal{B}\rangle_{sk_B} \end{array}$$

Let c be the binding information which binds the two primitives, i.e.$\mathsf{c} = (x, y, \mathcal{B})$. The following protocols are some examples of early binding. In the first protocol, a binder can be added by A, but A cannot generate c without B's challenge, so for early binding, the binder should be added by B. In the second protocol, both participants can add the binder, but the example only shows the case when the binder is added by A.

When $e(M_2) < e(N_2)$

$$
\begin{array}{llll}
\mathcal{P}\colon & M_1 & A \to B: & \mathcal{B}, \{\!|1, x, \mathcal{B}|\!\}_{ek_B}, auth_{sk_A}(1, x, \mathcal{B}) \\
 & M_2' & B \to A: & \langle \mathsf{c} \rangle_{sk_B} \\
 & M_2 & B \to A: & y, \mathcal{B}, \langle 1, x, \mathcal{B} \rangle_{sk_B}, auth_{sk_B}(2, y, \mathcal{B}) \\
 & N_2 & A \to B: & B, \langle 2, y, \mathcal{B} \rangle_{sk_A}
\end{array}
$$

When $e(N_2) < e(M_2)$

$$
\begin{array}{llll}
\mathcal{P}\colon & M_1 & A \to B: & \mathcal{B}, \{\!|1, x, \mathcal{B}|\!\}_{ek_B}, auth_{sk_A}(1, x, \mathcal{B}) \\
 & N_1 & B \to A: & y, \mathcal{B}, auth_{sk_B}(2, y, \mathcal{B}) \\
 & N_2' & A \to B: & \langle \mathsf{c} \rangle_{sk_A} \\
 & N_2 & A \to B: & B, \langle 2, y, \mathcal{B} \rangle_{sk_A} \\
 & M_2 & B \to A: & A, \langle 1, x, \mathcal{B} \rangle_{sk_B}
\end{array}
$$

Two messages can be merged together if there is no order restriction which prevents the merge. If necessary, c can replace the authentication reply if they are the same direction, since $x, \mathcal{B} \sqsubseteq \mathsf{c}$ and $y, \mathcal{B} \sqsubseteq \mathsf{c}$

When $e(M_2) < e(N_2)$

$$
\begin{array}{llll}
\mathcal{P}\colon & M_1 & A \to B: & \mathcal{B}, \{\!|1, x, \mathcal{B}|\!\}_{ek_B}, auth_{sk_A}(1, x, \mathcal{B}) \\
 & M_2'' & B \to A: & y, \mathcal{B}, \langle \mathsf{c} \rangle_{sk_B} \\
 & N_2 & A \to B: & B, \langle 2, y, \mathcal{B} \rangle_{sk_A}
\end{array}
$$

When $e(N_2) < e(M_2)$

$$
\begin{array}{llll}
\mathcal{P}\colon & M_1 & A \to B: & \mathcal{B}, \{\!|1, x, \mathcal{B}|\!\}_{ek_B}, auth_{sk_A}(1, x, \mathcal{B}) \\
 & N_1 & B \to A: & y, \mathcal{B}, auth_{sk_B}(2, y, \mathcal{B}) \\
 & N_2'' & A \to B: & B, \langle \mathsf{c} \rangle_{sk_A} \\
 & M_2 & B \to A: & A, \langle 1, x, \mathcal{B} \rangle_{sk_B}
\end{array}
$$

In this example, further optimisation is possible because there are dependencies between events. For example, in the first case, an honest participant A generates the last message N_2 only when she performed a previous event $e(M_1)$. Therefore, it is possible to remove $auth_{sk_A}(1, x, \mathcal{B})$ from the protocol. A does not generate the last message without having a previous event $e(M_1)$, and Spy cannot generate the last message because Spy does not own the key sk_A. Therefore, when B receives the last message he can conclude that it comes from A. However, this protocol does not necessarily achieve the same properties which the original one does, so in applying this optimisation, designers need to be careful. The protocols can be rearranged if necessary for the same reason as in the previous example.

5.3.3 The SSL and TLS Protocols

Secure web servers, and many other kinds of servers that want to protect data from prying eyes during transmission, often use the Secure Socket Layer (SSL), originally developed by Netscape. Transport Layer Security (TLS) is a refinement of SSL, which ensures privacy between communicating applications and their users on the Internet. When a server and client communicate, TLS guarantees that no `Spy` may eavesdrop or tamper with any message. TLS works by using a private key to encrypt data that should be transferred over the Internet. Both Netscape Navigator and Internet Explorer support this type of protocol, and many Web sites use the protocol to obtain confidential user information, such as credit card numbers. The goals verified in [42] with regard to TLS are as follows.

1. 'The peer's identity can be authenticated ... using public key cryptography'
2. 'The negotiated secret is unavailable to eavesdroppers, ... and for any authenticated connection the secret cannot be obtained, even by an attacker who can place himself in the middle of the connection'
3. 'No attacker can modify the negotiation communication without being detected by the parties'

These are the main goals of the protocol but they do not cover all philosophies behind the design. Therefore, with the goals in mind only, it is not easy to come up with a protocol like TLS. Moreover, it is very difficult to figure out all the minor details in TLS. Instead we start from the given main goals and then modify the protocol obtained in order to get a protocol like TLS. Here, we only show the version of TLS which requires certificates from both client C and server S. The other versions can be obtained in similar ways. Let $[C, ek_C]_{sk_{CA}}$ and $[S, ek_S]_{sk_{CA}}$ be the certificates of the client and the server, respectively, and assume that these certificates are known to each other. This assumption can later be removed by adding a message which contains each certificate before any use of the key. In TLS, the server is assumed not to reveal any secret received from clients. If not, the protocol cannot guarantee what it claims [42]. This is a strong assumption because any message secretly delivered to the server can be used to generate a secret short-term key, which is not generally true. Based on this assumption, the following message can deliver a secret to the server. Remind that $\vec{x}_i = (i, \vec{x})$.

$$p_1\colon \quad L_1 \quad C \rightarrow S: \quad \mathcal{B}_{\vec{x}}, \{\!|\vec{x}_1, \mathcal{B}_{\vec{x}}|\!\}_{ek_S}, auth_{sk_C}(\vec{x}_1, \mathcal{B}_{\vec{x}})$$

Due to the $auth$ message, the delivery is authenticated. A dishonest client cannot just forward the encrypted term coming from other participant with a changed $auth$ message to the server because of $\mathcal{B}$. In this case, the binding group inside the encrypted term will not include the client, so the server will not accept it as coming from the client. Therefore, to be accepted by the server, the client should be included in the binding group, which means the client knows the secret. Under this circumstance, after the delivery of a secret from a client to the server, the client and the server can use it to authenticate each other. Let p_1 and p_2 be the primitives shown below, where f_c and f_s are two different pseudo random number generators taking $\vec{x}$ as their inputs, $\mathcal{B}_{\vec{y}} = (C, S)$ and $\mathcal{B}_{\vec{z}} = (S, C)$.

$$\begin{array}{llll}
p_2\colon & M_1 & C \rightarrow S: & \mathcal{B}_{\vec{y}}, \vec{y} \\
& M_2 & S \rightarrow C: & \langle \vec{y}_2, \mathcal{B}_{\vec{y}} \rangle_{f_s(\vec{x})} \\
& M_3 & C \rightarrow S: & auth_{f_c(\vec{x})}(\vec{y}_2, \mathcal{B}_{\vec{y}}) \\
p_3\colon & N_1 & S \rightarrow C: & \mathcal{B}_{\vec{z}}, \vec{z} \\
& N_2 & C \rightarrow S: & \langle \vec{z}_3, \mathcal{B}_{\vec{z}} \rangle_{f_c(\vec{x})} \\
& N_3 & S \rightarrow C: & auth_{f_s(\vec{x})}(\vec{z}_3, \mathcal{B}_{\vec{z}})
\end{array}$$

Assume that $\vec{x}$ is a shared secret between C and S, the primitives p_2 and p_3 are regular and discreet, and they are agreement tests. To merge the two primitives, replace the binding groups in each primitive with $\mathcal{B} = (C, S)$ and define an order among messages in the composition $p_2 \otimes p_3$. The following shows one possible ordering.

$$\begin{array}{llll}
p_2 \otimes p_3\colon & M_1 & C \rightarrow S: & \mathcal{B}, \vec{y} \\
& N_1 & S \rightarrow C: & \mathcal{B}, \vec{z} \\
& N_2 & C \rightarrow S: & \langle \vec{z}, \mathcal{B} \rangle_{f_c(\vec{x})} \\
& M_2 \& N_3 & S \rightarrow C: & \langle \vec{y}, \mathcal{B} \rangle_{f_s(\vec{x})}, auth_{f_s(\vec{x})}(\vec{z}_3, \mathcal{B}) \\
& M_3 & C \rightarrow S: & auth_{f_c(\vec{x})}(\vec{y}_2, \mathcal{B})
\end{array}$$

The binding information of the two primitives is $\mathsf{c} = \vec{y}, \vec{z}, \mathcal{B}$. Either C or S can add a binder to link relevant primitives. As mentioned earlier, any term which can be interpreted as c can be a binder. For this example, $\langle \mathsf{c} \rangle_{f_c(\vec{x})}$ and $\langle \mathsf{c} \rangle_{f_s(\vec{x})}$ will be used as the binders. Adding $\langle \mathsf{c} \rangle_{f_c(\vec{x})}$ will produce the following protocol.

$$\begin{array}{llll}
\mathcal{P}\colon & M_1 & C \to S: & \mathcal{B}, \vec{y} \\
& N_1 & S \to C: & \mathcal{B}, \vec{z} \\
& N_2 & C \to S: & \langle \mathsf{c} \rangle_{f_c(\vec{x})} \\
& M_2' & S \to C: & \langle \vec{y}, \mathcal{B} \rangle_{f_s(\vec{x})}, auth_{f_s(\vec{x})}(\vec{z}_3, \mathcal{B}) \\
& M_3 & C \to S: & auth_{f_c(\vec{x})}(\vec{y}_2, \mathcal{B})
\end{array}$$

If $\langle \mathsf{c} \rangle_{f_s(\vec{x})}$ is added again to $\mathcal{P}$, which is generally unnecessary because the two primitives are already bound, but it does not cause any harm either, then the following protocol is produced.

$$\begin{array}{llll}
\mathcal{P}\colon & M_1 & C \to S: & \mathcal{B}, \vec{y} \\
& N_1 & S \to C: & \mathcal{B}, \vec{z} \\
& N_2 & C \to S: & \langle \mathsf{c} \rangle_{f_c(\vec{x})} \\
& M_2' & S \to C: & \langle \mathsf{c} \rangle_{f_s(\vec{x})} \\
& M_3 & C \to S: & auth_{f_c(\vec{x})}(\vec{y}_2, \mathcal{B})
\end{array}$$

M_3 is mainly used to prove that $\vec{y}$ originates from C, but which can be shown by $\langle \mathsf{c} \rangle_{f_c(\vec{x})}$, so it is possible to remove M_3. $\mathcal{P}$ together with the delivery of the secret produces the following protocol. Notice that L_1 can be placed anywhere before M_2' because $\vec{x}$ is first used by S in M_2'.

$$\begin{array}{llll}
\mathcal{P} \otimes p_1\colon & L_1 & C \to S: & \mathcal{B}, \{\!|\vec{x}_1, \mathcal{B}|\!\}_{ek_S}, auth_{f_c(\vec{x})}(\vec{x}_1, \mathcal{B}) \\
& M_1 & C \to S: & \mathcal{B}, \vec{y} \\
& N_1 & S \to C: & \mathcal{B}, \vec{z} \\
& N_2 & C \to S: & \langle \mathsf{c} \rangle_{f_c(\vec{x})} \\
& M_2' & S \to C: & \langle \mathsf{c} \rangle_{f_s(\vec{x})}
\end{array}$$

Any rearrangement which does not violate the orders of each primitive is allowed. However, for this example, $\vec{x}$ should be known before it is used as the input of the functions, f_c and f_s. A rearrangement similar to TLS will produce the following protocol, which does not violate any order restrictions. The correctness of the protocol follows from the theories presented in previous sections.

$$
\begin{array}{llccl}
\mathcal{P}_f\text{:} & M_1' & C \to S: & \mathcal{B}, \vec{y} & \text{Client Hello} \\
& N_1 & S \to C: & \mathcal{B}, \vec{z} & \text{Server Hello} \\
& L_1 & C \to S: & \mathcal{B}, \{|\vec{x}_1, \mathcal{B}|\}_{ek_S} & \text{Client Key Exchange} \\
& M_1''\&N_2 & C \to S: & auth_{f_c(\vec{x})}(\vec{x}_1, \mathcal{B}), \langle \mathsf{c} \rangle_{f_c(\vec{x})} & \text{Client Finished} \\
& M_2' & S \to C: & \langle \mathsf{c} \rangle_{f_s(\vec{x})} & \text{Server Finished}
\end{array}
$$

This protocol is not exactly the same as the TLS Handshake protocol, but shares many of its features. Notice that the protocol achieves all the goals listed in [42]. If a session identifier is added in each message of the protocol, the protocol also allows the client to resume a secure session with the server like the TLS Handshake protocol.

6

Symmetric Protocols

Many authentication protocols based on symmetric key cryptography have been proposed after the seminal work of Needham and Schroeder [39,40]. The client and server based distributed computing environments of the mid 80's and early 90's created many demands on authentication protocols using symmetric key cryptography. Symmetric key cryptography is very easy to use, and usually very fast too. On the other hand, symmetric keys must be kept secure. However, the distributed computing environment has dramatically changed in the past few years by the introduction of fast CPUs, mobile ad-hoc networks, peer to peer, and ubiquitous computing. These changes have made authentication protocols based on public key cryptography preferable in many applications because public keys are safe to be published anywhere. Nevertheless, public key cryptography has its limitations: it is slower and requires larger keys than symmetric key cryptography, and involves CPU-intensive computations, which makes it less suitable for small, battery powered devices. This chapter discusses design issues on protocols based on symmetric key cryptography, in short, symmetric protocols. The outline of the chapter is as follows. Section 6.1 investigates the distinctive features of symmetric protocols, mainly regarding issues concerning design by composition. We inspect the structural problem occurring in a symmetric protocol due to its symmetric use of keys, and suggest a possible solution for the problem so that the proposed composition methodology can work in this environment. Section 6.2 describes what trust means in symmetric protocols using authentication servers. Section 6.3 examines several design rules useful for the composition of symmetric protocols. Finally, Section 6.4 demonstrates the utility of the proposed methodology through design examples of several applications.

6.1 Symmetric Protocol Design

Symmetric key authentication protocols can be divided into two categories depending on how the freshness of key distribution messages is determined. One category uses challenge and response, and the other is based on timestamps [15]. Protocols using timestamps need fewer messages than the ones based on nonces [21], the downside being that they require synchronised clocks. On the other hand, protocols based on nonces require good random number generators and state storage, in order to prevent certain types of attacks such as reflection and replay. The protocols presented in this chapter mainly make use of nonces since nonce-based protocols are easily transformable into timestamp-based ones if synchronised clocks are provided. Symmetric key protocols can also be divided into two categories depending on whether they need authentication servers for a protocol run. Symmetric key environments have a disadvantage in key management since every pair of participants should share a secret key to communicate with each other. To overcome this disadvantage, authentication servers are usually introduced so that each participant shares a secret key with the servers instead, and the servers intervene in communication between participants.

The design of symmetric protocols is structurally very similar to that of asymmetric protocols. However, due to the different assumptions on the underlying cryptosystem, some features are distinctive. It is important to know the differences, especially the differences which need to be considered regarding protocol composition. First, *role asymmetry* is not guaranteed in symmetric environments due to the symmetric use of the same secret key. Role asymmetry means that two participants playing different roles should generate distinctive cryptographically transformed terms. In asymmetric environments where long-term private keys are not shared between participants, there is less confusion regarding the origin of cryptographically transformed terms, whereas in symmetric environments where private keys are shared among participants, such confusion is unavoidable without extra information. In Woo and Lam's Π protocol [53], for example, the origins of the two messages M_3 and M_5 cannot be easily determined due to the overlapped use of shared keys for different purposes.

$$
\begin{array}{lll}
M_1 & A \rightarrow B\colon & A \\
M_2 & B \rightarrow A\colon & N_b \\
M_3 & A \rightarrow B\colon & \{\!|N_b|\!\}_{k_{AS}} \\
M_4 & B \rightarrow S\colon & \{\!|A, \{\!|N_b|\!\}_{k_{AS}}|\!\}_{k_{BS}} \\
M_5 & S \rightarrow B\colon & \{\!|N_b|\!\}_{k_{BS}}
\end{array}
$$

Although the same problem can occur in asymmetric environments too, it is more likely to occur in symmetric environments. In order to avoid such confusion and to simulate role asymmetry, we adopt the recommendation of `RFC1994` [49] and use a different key for each participant, i.e. if k is a shared key between A and B, then A uses k_{AB} and B uses k_{BA} instead of k, where k_{AB} and k_{BA} are different and derivable from k by both participants. One example of the key derivation function for a participant A is $k_{A\beta} = h(A, k)$, where k is a shared key between participants A and β.

Second, authentication servers are generally trusted to behave honestly. What and how much participants should trust the authentication servers may differ from application to application. However, a certain level of trust is always required in design and verification of protocols. This issue is discussed in the next section.

6.2 Authentication Servers and Trust

This section discusses trust on authentication servers from a design point of view, and shows how it affects a design.

6.2.1 Trust

The meaning of trust differs from application to application, and it is difficult to formalise correctly, even though it plays an important role in protocol verification and design. In protocols that require authentication servers, protocol correctness requires more than the existence of communication channels between participants and the appropriate authentication servers. Correctness is critically dependent on the ability of the servers to faithfully follow the protocols. Each participant bases its judgment on its own observations made from messages sent and received, and its trust in the server's judgment.

Authentication servers in symmetric protocols are assumed to behave in a particular way in a protocol run. Among those behaviours are "secrecy" and "faithfulness". Authentication servers are trusted not to divulge the secrets of participants. If every term that an authentication server sends is regular, long-term secret keys are safe from Spy. That is to say, the only way that Spy gets to know a long-term secret key is to get it directly from the participant who shares the key with the authentication server. However, this is unlikely to happen unless the participant is compromised. Authentication servers sometimes send a short-term secret to particular participants, helping them to create a common secure channel. In this case, authentication servers are not allowed to send the secret to any other participants. In other words, for a short-term secret originating from the authentication servers to be secure, every protocol step that the authentication servers take has to be discreet. Consequently, by the secrecy of authentication servers, every protocol step that authentication servers perform is required to be regular as well as discreet.

Secrecy Condition: All outputs of an authentication sever should be discreet and regular.

$$
\begin{array}{lll}
M_1 & A \to B: & M, A, B, \{\!|N_a, M, A, B|\!\}_{k_{AS}} \\
M_2 & B \to S: & M, A, B, \{\!|N_a, M, A, B|\!\}_{k_{AS}}, \{\!|N_b, M, A, B|\!\}_{k_{BS}} \\
M_3 & S \to B: & M, \{\!|N_a, k|\!\}_{k_{AS}}, \{\!|N_b, k|\!\}_{k_{BS}} \\
M_4 & B \to A: & M, \{\!|N_a, k|\!\}_{k_{AS}}
\end{array}
$$

Figure 6.1: **Otway-Rees Protocol**

Authentication servers are also trusted to faithfully follow the protocol specification. First, authentication servers should check all relevant information inside an input message before generating a meaningful output message. As Boyd and Mao pointed out, the responsibility of authentication servers regarding input validation is usually not described clearly, and this lack of detailed explanation sometimes invites unwanted attacks [34]. They note that in the Otway-Rees authentication protocol, the authentication server should not only check the form of the input message M_2 but also check whether all the plaintext information inside the encrypted terms of M_2 match each other. In other words, the authentication server should accept a message of the form $(X_1, A_1, B_1, \{|Y_1, Y_2, A_2, B_2|\}_{k_1}, \{|Z_1, Z_2, A_3, B_3|\}_{k_2})$ as an input message only if $k_1 = k_{A_1S}$, $k_2 = k_{B_1S}$, $X_1 = Y_2 = Z_2$, $A_1 = A_2 = A_3$, and $B_1 = B_2 = B_3$. Therefore, for any given output of the authentication server, if the authentication server is faithful, there exists a corresponding input. For example, when A receives M_4, she can infer from M_4 that a certain input has happened in the authentication server. Second, all outputs the authentication server generates should have the specific form defined by the protocol specification. For example, the output $(M, \{|N_a, k|\}_{k_{AS}}, \{|N_b, k|\}_{k_{BS}})$ always comes as one unit, i.e. the server's output always consists of one basic term and two encrypted terms. Hence, when A receives the message M_4, she can conclude that the authentication server has generated a term $\{|x, k|\}_{k_{B'S}}$ for some x, because M_4 should be a part of some output generated by the server. Third, the authentication server is assumed to keep the binding group of an input message in the output it generates. Based on this assumption, A can conclude that $B' = B$, i.e. $\{|x, k|\}_{k_{BS}}$ is the message that the authentication server produced, because the binding group of the nonce N_a includes B. If A binds N_a to more than one different binding group, then the authentication server will generate outputs accordingly with different binding group. This way, the authentication server does not need to maintain a record of all nonces. An honest participant should not bind a nonce to more than one different binding group to avoid the confusion which might happen due to multiple bindings of a single nonce. Finally, any secrets generated by the authentication server should be fresh.

In order to provide a specification of a faithful authentication server in composition, the authentication server is regarded as a program, which has input and output conditions specified by the protocol specification. In conclusion, in protocol composition, a faithful authentication server is required to satisfy the following conditions.

Definiton 29. (Faithful Authentication Servers) *Suppose S be an authentication server of a protocol $\mathcal{P}$. Let i_S be the set of input terms which satisfy the input conditions of the protocol, and o_S be the set of possible output terms of the authentication server. If S is faithful, then the following statements hold.*

1. *S produces an output only for an input x satisfying the input condition, i.e. $x \in i_S$, therefore, for an output of S, there always exists a corresponding input.*

2. *If y_i is a component of a term that can be verified to originate from S, then there exists an output $y \in o_S$ containing y_i as a component.*

3. *S does not change the binding group in the input messages when it generates an output.*

4. *All secrets generated by S are fresh.*

6.3 Composition

The composition of two-party protocols is basically the same as that of asymmetric protocols, if each participant sharing a secret key k is assumed to use a different key derived from k. Our interest is mainly in the composition issues regarding the protocols using an authentication server. The composition can be divided into two different stages. One is before having any common secret between two participants and the other is after sharing a secret. The latter is basically the same as two-party protocols sharing a secret, which are already discussed in the previous chapter, therefore, the main focus on this section will be the former.

Let p_i be an indexed protocol primitive with the following protocol steps, where $\vec{x}_i = (i, \vec{x})$.

$$\begin{array}{llll} p_i & M_1 & A \rightarrow S: & \mathcal{B}_{\vec{x}}, \{\!|\vec{x}_i, \mathcal{B}_{\vec{x}}|\!\}_{k_{AS}}, auth_{k_{AS}}(\vec{x}_i, \mathcal{B}_{\vec{x}}) \\ & M_2 & A \leftarrow S: & A, \langle \vec{x}_i, \mathcal{B}_{\vec{x}} \rangle_{k_{SA}} \end{array}$$

or

$$\begin{array}{llll} p_i & M_1 & A \rightarrow S: & \mathcal{B}_{\vec{x}}, \vec{x}, auth_{k_{AS}}(\vec{x}_i, \mathcal{B}_{\vec{x}}) \\ & M_2 & A \leftarrow S: & A, \langle \vec{x}_i, \mathcal{B}_{\vec{x}} \rangle_{k_{SA}} \end{array}$$

These primitives are designed to communicate with the authentication server S, since both participants cannot directly talk to each other before they agree on a secret. Figure 6.2 shows the difference between two-party protocols and the protocols with an authentication server. In (a), A directly communicates with B, so A can check whether a reply from B is fresh or not, whereas in (b), A does not directly communicate with B, so she has no way to check that. The same is true when B receives a reply. Therefore, in (b), when A finishes a run of the primitive, she cannot infer what B has done without the

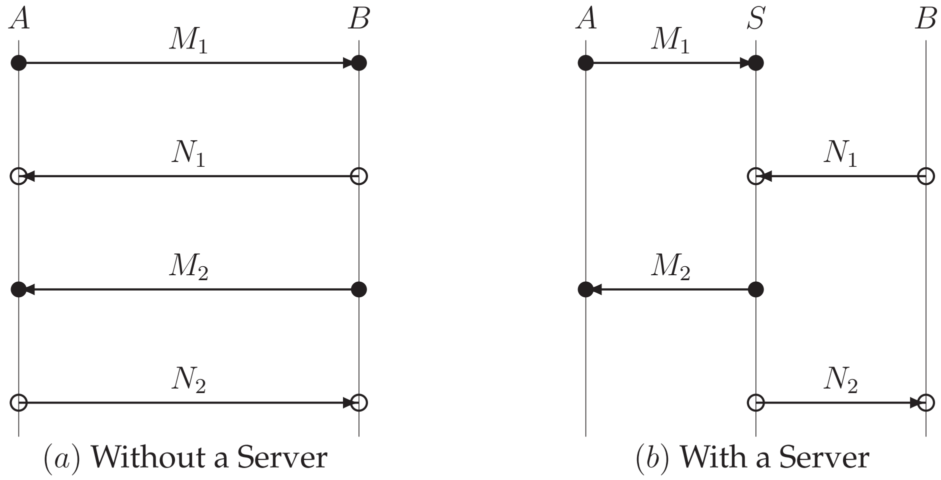


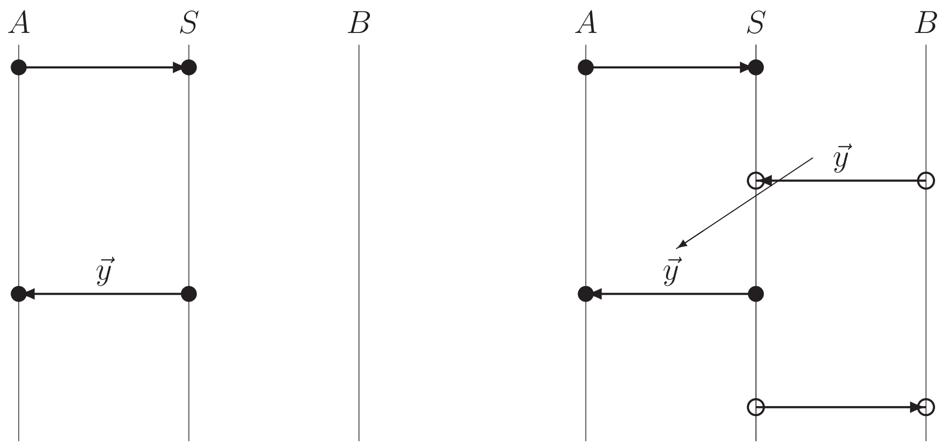
Figure 6.2: **Comparison of Composition**

help of S. This section shows how to compose a protocol in such an environment, using the agreement property together with the faithfulness assumption of the authentication server.

6.3.1 Authenticated Delivery

An authentication server plays many different roles. It delivers a message coming from one participant to the other, and it generates a fresh key and distributes it to a group of participants. Moreover, the authentication server is the only entity which can see all parameters used in a single run. For example, in the Otway-Rees protocol, the authentication server knows that N_a and N_b are the variables of a specific run. There are two different types of information, one originating from the authentication server itself and the other from protocol participants. The key k in $\{|N_a, k|\}_{k_{AS}}$ of the Otway-Rees protocol is an example of the former and N_b in $\{|B, k, N_a, N_b|\}_{k_{AS}}$ of the Yahalom protocol is an example of the latter. Let $\vec{y}$ be the information delivered by the authentication server, possibly in an encrypted form if the information is secret. Figure 6.4 shows the cases when $\vec{y}$ originates from two different sources, S and B.

Assuming that the delivered information is secret, the delivery mechanism of the authentication server in both cases is the same, i.e. the server will send A an encrypted term such as $\{| \cdots \vec{y} \cdots |\}_{k_{SA}}$. The following shows an unsolicited test implementing the



(a)
created by S

(b)
created by B

Figure 6.3: **Information Delivery**

delivery.

$$p' \quad M \quad A \leftarrow S: \quad \{|\vec{y}|\}_{k_{SA}}$$

This protocol step p' is regular, but it is not discreet because the message M does not say to A with whom $\vec{y}$ is shared. To make it discreet, S should provide the binding group of $\vec{y}$ as shown below.

$$p' \quad M \quad A \leftarrow S: \quad \{|\vec{y}|\}_{k_{SA}}, auth_{k_{SA}}(\vec{y}, \mathcal{B}_{\vec{y}})$$

Based on the assumption that the authentication server is faithful, the authentication server will only send the secret $\vec{y}$ to the participants in $\mathcal{B}_{\vec{y}}$. If $\mathcal{B}_{\vec{y}}$ is provided by A in (a) and by B in (b), then it can be proven that Spy cannot get the secret unless Spy is a member of $\mathcal{B}_{\vec{y}}$, which shows that the delivery is discreet.

The delivery mechanism alone is not useful, because the recipient of the information has no idea whether the information is fresh or not. The information delivery mechanism p' and a protocol primitive p_i can be merged together to produce an authenticated delivery mechanism. The primitive p_i is the only protocol primitive needed for the example, so the index is unnecessary. Suppose $\mathcal{B} = \mathcal{B}_{\vec{x}} = \mathcal{B}_{\vec{y}}$.

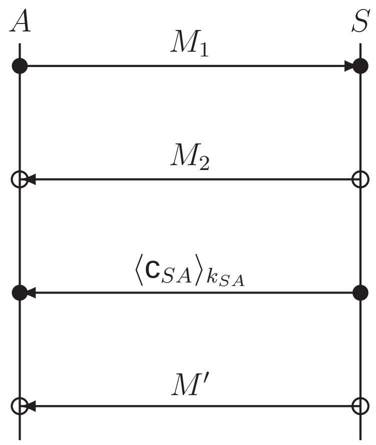


Figure 6.4: **Authenticated Delivery**

$$
\begin{array}{llll}
p_i & M_1 & A \rightarrow S: & \mathcal{B}, \vec{x}, auth_{k_{AS}}(\vec{x}, \mathcal{B}) \\
 & M_2 & A \leftarrow S: & A, \langle \vec{x}, \mathcal{B} \rangle_{k_{SA}} \\
p' & M' & A \leftarrow S: & \{\!|\vec{y}|\!\}_{k_{SA}}, auth_{k_{SA}}(\vec{y}, \mathcal{B}) \\
 & & & \Downarrow \\
p'_i & M_1 & A \rightarrow S: & \mathcal{B}, \vec{x}, auth_{k_{AS}}(\vec{x}, \mathcal{B}) \\
 & M_2\&M' & A \leftarrow S: & A, \langle \vec{x}, \mathcal{B} \rangle_{k_{SA}}, \{\!|\vec{y}|\!\}_{k_{SA}}, auth_{k_{SA}}(\vec{y}, \mathcal{B}), \langle \mathtt{c}_{SA} \rangle_{k_{SA}} \\
 & & & \Downarrow \\
p'_i & M_1 & A \rightarrow S: & \mathcal{B}, \vec{x}, auth_{k_{AS}}(\vec{x}, \mathcal{B}) \\
 & M_2 & A \leftarrow S: & A, \{\!|\vec{y}|\!\}_{k_{SA}}, \langle \mathtt{c}_{SA} \rangle_{k_{SA}}
\end{array}
$$

Notice that this is an instance of two-party composition, so the method proposed in the previous chapter can be applied here. Message M' and M_2 can be merged together because their directions are the same. Let $\mathtt{c}_{SA}$ be the binding information which pieces p_i and p' together, i.e. $\mathtt{c}_{SA} = (\vec{x}, \vec{y}, \mathcal{B})$. Early binding will put a binder of the binding information from S to A, e.g. $\langle \mathtt{c}_{SA} \rangle_{k_{SA}}$ because A cannot produce the binder before she receives the message M'. After adding $\langle \mathtt{c}_{SA} \rangle_{k_{SA}}$, the redundant terms $\langle \vec{x}, \mathcal{B} \rangle_{k_{SA}}$ and $auth_{k_{SA}}(\vec{y}, \mathcal{B})$ can be removed since $\vec{x}, \mathcal{B} \sqsubseteq \mathtt{c}_{SA}$ and $\vec{y}, \mathcal{B} \sqsubseteq \mathtt{c}_{SA}$. Similarly, the following primitive with an index can be produced using composition

$$
\begin{array}{llll}
p'_i & M_1 & A \rightarrow S: & \mathcal{B}, \{\!|\vec{x}_i, \mathcal{B}|\!\}_{k_{AS}}, auth_{k_{AS}}(\vec{x}_i, \mathcal{B}) \\
 & M_2 & A \leftarrow S: & A, \{\!|\vec{y}|\!\}_{k_{SA}}, \langle \vec{x}_i, \vec{y}, \mathcal{B} \rangle_{k_{SA}}
\end{array}
$$

Both primitives above are agreement tests, so they can be used as protocol primitives. These primitives are especially useful in designing key distribution protocols and re-authentication protocols. Key distribution protocols can use the primitives to deliver a secret key, and re-authentication protocols can use the primitives to deliver a token containing a key and a timestamp (the key's lifetime). Notice that if $\vec{y} = \texttt{null}$, then primitive p'_i is equal to p_i.

Regarding the freshness of the information, there is a difference between the cases (a) and (b) of Figure 6.4. In case of (a), if A trusts that S only generates fresh information, then she can verify that $\vec{y}$ is fresh from the variable she knows, i.e. from the nonce included in $\vec{x}$. However, this is not true in case of (b), because the composition of primitives does not guarantee that the information was recently sent by B. How to ensure that this information is freshly delivered by B will be discussed in Section 6.3.3.

6.3.2 Correspondence

In two-party protocols, when A finishes her part of a run, from the actions she has done, she knows the corresponding actions performed by her partner. However, in protocols with an authentication server, this is not the case. For example, in the Otway-Rees protocol, what A and B see at the end of a protocol run might be the following series of events if messages unrecognisable by each participant are ignored.

$$\begin{array}{lll} M_1 & A \rightarrow S: & M, A, B, \{\!|N_a, M, A, B|\!\}_{k_{AS}} \\ M_4 & A \leftarrow S: & M, \{\!|N_a, k|\!\}_{k_{AS}} \\ & & \\ M_2 & B \rightarrow S: & M, A, B, \{\!|N_b, M, A, B|\!\}_{k_{BS}} \\ M_3 & B \leftarrow S: & M, \{\!|N_b, k|\!\}_{k_{BS}} \end{array}$$

With the events above alone, it is impossible for A and B to derive all properties that the protocol achieves. This is the same in case of composition. Putting several primitives together does not automatically guarantee that all goals of the primitives happen together. In order to verify the correctness of the Otway-Rees protocol, the authentication server should be faithful, i.e. the server accepts an input of the form $M_1 \cup M_2$ and produces an output of the form $M_3 \cup M_4$. These input and output forms are specified by the protocol specification, which designers have to produce by composition. In other words, the protocol specification is unknown during composition, so it is impossible to define input and output forms of the authentication server. In composition, to tackle this problem, all messages destined to the authentication server are initially regarded as the input and all message originating from the server as the output. Later, if there is a change in the input, then it can accordingly be updated in the protocol specification.

Let $\mathcal{P}_{\mathcal{G}} = \{p_1, p_2, \ldots, p_n\}$ be a set of indexed primitives used to compose a protocol, where n different participants communicate with an authentication server. A p-protocol of $\mathcal{P}_{\mathcal{G}}$, namely $\mathcal{P} = |p_1 \otimes p_2 \otimes \cdots \otimes p_n|$, is defined as the set of primitives together with a predefined order $o(\mathcal{P})$. Suppose $\mathcal{B} = (A_1, A_2, \ldots, A_n, S)$ is the binding group of $\mathcal{P}$, and let each participant A_i playing a role R_i execute $p_i(\vec{a}_i; \mathcal{B})$. Notice that S is included in the binding. Generally, replacing the binding group $\mathcal{B}_x$ of p_i in $\mathcal{P}$ with $\mathcal{B}$ can affect the discreetness property of the primitive since $\mathcal{B}$ and $\mathcal{B}_x$ are not the same. However, the discreetness property of the primitive can be proven before the replacement. In other words, if $p_i(x; \mathcal{B}_x)$ is discreet, then $p_i(x; \mathcal{B})$ is also discreet, but if $p_i(x; \mathcal{B})$ is discreet, then $p_i(x; \mathcal{B}_x)$ is not necessarily discreet. Therefore, adding extra terms to the binding group of the primitive, which is already proven discreet, does not cause any harm to the secret x. Under the assumption that the authentication server is faithful, the input of the authentication server is defined as the union of the messages originating from each participant A_i, and the output of the authentication as the union of the messages originating from the server. To simplify the discussion, assume that $n = 2$, i.e. $\mathcal{P} = p_1 \otimes p_2$, and each primitive p_i is shown as below, where $\vec{\alpha}_i = (i, \vec{\alpha})$, $\mathcal{B} = (A, B, S)$, $\mathsf{c}_{SA} = (\vec{a}, \vec{c}, \mathcal{B})$, $\mathsf{c}_{SB} = (\vec{b}, \vec{d}, \mathcal{B})$. The input of the authentication server is $M_1 \cup N_1$ and the output $M_2 \cup N_2$.

$$
\begin{array}{llll}
p_1 & M_1 & A \rightarrow S: & \mathcal{B}, \{\!|\vec{a}_1, \mathcal{B}|\!\}_{k_{AS}}, auth_{k_{AS}}(\vec{a}_1, \mathcal{B}) \\
 & M_2 & A \leftarrow S: & A, \{\!|\vec{c}_1|\!\}_{k_{SA}}, \langle \mathsf{c}_{SA} \rangle_{k_{SA}} \\
p_2 & N_1 & B \rightarrow S: & \mathcal{B}, \vec{b}, auth_{k_{BS}}(\vec{b}_2, \mathcal{B}) \\
 & N_2 & B \leftarrow S: & B, \{\!|\vec{d}_2|\!\}_{k_{SB}}, \langle \mathsf{c}_{SB} \rangle_{k_{SB}}
\end{array}
$$

At the end of a run each participant has its own view of the protocol run, formulated as a sequence of events in a strand, and let these strands be $\xi_{AS}^{\mathcal{P},1}(\omega; \mathcal{B})$ and $\xi_{BS}^{\mathcal{P},2}(\mu; \mathcal{B})$ for A and B, respectively. Each participant has its own corresponding run of the authentication server due to the agreement property of primitives.

Lemma 26. *If $\mathcal{P} = |p_1 \otimes p_2|$, ψ is a bundle formed by $\mathcal{P}$, and the authentication server is faithful, then*

$$\forall \psi \bullet \xi_{AS}^{\mathcal{P},1}(\omega; \mathcal{B}) \in \psi \Rightarrow [\xi_{SA}^{\mathcal{P},1}(\omega; \mathcal{B})] \in \psi$$

$$\forall\psi \bullet \xi^{\mathcal{P},2}_{BS}(\mu;\mathcal{B}) \in \psi \Rightarrow [\xi^{\mathcal{P},2}_{SB}(\mu;\mathcal{B})] \in \psi$$

Proof. From the agreement property of indexed primitives,

$$\forall\psi \bullet \xi^{\mathcal{P},1}_{AS}(\omega;\mathcal{B}) \in \psi \Rightarrow [\xi^{\mathcal{P},1}_{SA}(\omega;(A,B',S))] \in \psi$$

$$\forall\psi \bullet \xi^{\mathcal{P},2}_{BS}(\mu;\mathcal{B}) \in \psi \Rightarrow [\xi^{\mathcal{P},2}_{SB}(\mu;(A',B,S))] \in \psi$$

Assume that $B' \neq B$. The authentication server is faithful, so for S to generate an output containing $\vec{a}$ with $\mathcal{B}' = (A, B', S)$, there should be an input originating from A, which contains $\vec{a}$ and $\mathcal{B}'$. However, this contradicts the fact that A finished a run with S, i.e. the existence of $\xi^{\mathcal{P},1}_{AS}(\omega;\mathcal{B})$ means that S generated the output term $\langle \mathsf{c}_{SA}\rangle_{k_{SA}}$, where $x, \mathcal{B} \sqsubseteq \mathsf{c}_{SA}$. Similarly, B's guarantee can be proven. □

In the opposite direction, from a run of the authentication server, its corresponding strand can be derived. In other words, when the authentication server has finished its part of a run, there exists its corresponding run of each participant engaged in the run.

Lemma 27. *If $\mathcal{P} = |p_1 \otimes p_2|$, ψ is a bundle formed by $\mathcal{P}$, and the authentication server is faithful, then*

$$\forall\psi \bullet \xi^{\mathcal{P},1}_{SA}(\omega;\mathcal{B}) \in \psi \Rightarrow [\xi^{\mathcal{P},1}_{AS}(\omega;\mathcal{B})] \in \psi$$
$$\forall\psi \bullet \xi^{\mathcal{P},2}_{SB}(\mu;\mathcal{B}) \in \psi \Rightarrow [\xi^{\mathcal{P},2}_{BS}(\mu;\mathcal{B})] \in \psi$$

A faithful authentication server generates an output only when a correct input was given. Therefore, if there is a corresponding run of an initiator in the authentication server, then there should be a corresponding run of a responder in the authentication server, too.

Lemma 28. *If the authentication server is faithful, then*

$$\forall\psi \bullet (\xi^{\mathcal{P},1}_{SB}(\omega;\mathcal{B}) \in \psi \Rightarrow \exists\mu \bullet \xi^{\mathcal{P},2}_{SB}(\mu;\mathcal{B}) \in \psi)$$

$$\forall\psi \bullet (\xi^{\mathcal{P},2}_{SB}(\mu;\mathcal{B}) \in \psi \Rightarrow \exists\omega \bullet \xi^{\mathcal{P},1}_{SA}(\omega;\mathcal{B}) \in \psi)$$

From Lemma 26, 27 and 28 together with the faithful assumption of the authentication server, each participant can now infer its corresponding run(s) of the other participant(s) from the existence of its matching run(s) of the authentication server.

Theorem 6. *If $\mathcal{P} = |p_1 \otimes p_2 \otimes \cdots \otimes p_n|$, ψ is a bundle formed by $\mathcal{P}$, and the authentication server is faithful, then*

$$\forall \psi \bullet (\xi^{\mathcal{P},i}_{A_i S}(\omega_i; \mathcal{B}) \in \psi \Rightarrow \forall A_j \in \mathcal{B}, \exists \mu_j \bullet [\xi^{\mathcal{P},j}_{A_j S}(\mu_j; \mathcal{B})] \in \psi)$$

This is only obtainable due to the agreement property of protocol primitives and the assumption on the server's faithful behaviour. Sometimes, a weaker result, which can be obtained from some primitives satisfying authentication tests not agreement tests, is also useful in designing a protocol. Let p'_i be a protocol primitive which does not contain term $auth_{k_{S*}}(\cdot)$, i.e.:

$$\begin{array}{llll} p'_i & M_1 & A \rightarrow S: & \mathcal{B}_{\vec{a}}, \vec{a} \\ & M_2 & A \leftarrow S: & A, \langle \vec{a}_i, \mathcal{B}_{\vec{a}} \rangle_{k_{SA}} \end{array}$$

Lemma 27 and 28 do not hold in this case, even though Lemma 26 is true. Notice that Theorem 6 only guarantees the existence of a certain set of strands, but the theorem does not guarantee the recentness of those strands, since each A_i does not generate any cryptographically transformed term containing a nonce coming from the other participants.

A Secrecy Condition of authentication servers can be proven if the authentication server is faithful.

Lemma 29. *If $\xi^{\mathcal{P}}_S = \bigcup_i \xi^{\mathcal{P}_i}_{SA_i}(*; \mathcal{B})$ is the union of actions performed by the faithful authentication server in $\mathcal{P} = |p_1 \otimes p_2 \otimes \cdots \otimes p_n|$, then $\xi^{\mathcal{P}}_S$ is discreet and regular.*

Proof. The regularity of $\xi^{\mathcal{P}}_S$ follows from the regularity of each primitive p_i used in the composition. Let $\mathsf{S}_{\mathcal{P}}$ be a set of short term secrets generated by S, then its protective domain is $\mathcal{S}_{\mathcal{P}} = \mathsf{S}_{\mathcal{P}} \cup \{k_{SX}, k_{XS} | X \in \mathcal{B}\}$. Two types of messages are generated by S in $\mathcal{P}$, either hashed or encrypted ones. Hashed terms belong to the coideal of $\mathcal{S}_{\mathcal{P}}$. For an encrypted term $\{|\vec{b}|\}_{k_{SX}}$, there always exists a hashed term $\langle \vec{a}, \vec{b}, \mathcal{B}' \rangle_{k_{SX}}$. From the assumption that S is faithful, for S to generate the output $\langle \vec{a}, \vec{b}, \mathcal{B}' \rangle_{k_{SX}}$, there should be a corresponding input message originating from X, which contains $\vec{a}$ and its binding group $\mathcal{B}'$. Suppose that $\mathcal{B} \neq \mathcal{B}'$, since a variable can be bound to more than one different binding group by a participant. Let Y be a participant who belongs to $\mathcal{B}'$ but does not belong to $\mathcal{B}$, then Y cannot get any term $x \in \mathsf{S}_{\mathcal{P}}$, because S's output always happens together with a hashed term containing the binding group $\mathcal{B}$. This shows that $\xi^{\mathcal{P}}_S$ is discreet. □

6.3.3 Recentness

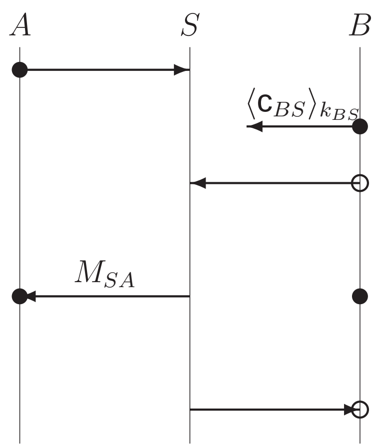


Figure 6.5: **Recentness**

The composition of primitives in the proposed way does not provide any guarantee that the participants were recently running a protocol. The guarantee on a participant's recent involvement in a run can be achieved in two different ways. One participant can directly check the other participant's involvement using a challenge and response mechanism after both agree on a shared key, or the goal can be accomplished through the help of the authentication server.

Let $\vec{a}$ be a nonce generated by A and M_{SA} be its reply from S; then any cryptographically transformed term containing $\vec{a}$ and originating from B can be a proof of B's recentness. Such a term can only be checked by the authentication server. Let $\mathtt{c}_{BS}$ be binding information such that $\vec{a}, \mathcal{B}_{\vec{a}} \sqsubseteq \mathtt{c}_{BS}$ ($\mathtt{c}_{BS}$ needs to include the binding group to make it distinct from other $\mathtt{c}'_{BS}$ generated by B playing a different role). When S receives a term such as $\langle \mathtt{c}_{BS} \rangle_{k_{BS}}$ from B, S can check whether the variable and its binding group in the term match the challenge $\vec{a}$ coming from A. If they match, then S generates its reply M_{SA} to A. Therefore, A only receives her reply when there exists a term from B containing a fresh nonce she generated. Two things need to be considered regarding $\mathtt{c}_{BS}$. Notice that if a nonce $\vec{b}_j$ created by B is included in $\mathtt{c}_{BS}$, i.e. $\vec{a}_i, \vec{b}_j, \mathcal{B}_{\vec{a}_i} \sqsubseteq \mathtt{c}_{BS}$, then $\mathtt{c}_{BS}$ becomes the binding information of two primitives, $p_i(\vec{a}_i; (A, S))$ and $p_j(\vec{b}_j; (B, S))$. Some care needs to be taken when the message coming from A in the example above is encrypted. In this case, B cannot see the content of the message, so he cannot create $\mathtt{c}_{BS}$. It is still possible to generate a cryptographically transformed term from the encrypted message, but we do not use such a scheme because it has many drawbacks. To solve the problem, we use a primitive with two nonces. The primitive replaces the original one whenever the situation mentioned above happens.

$$\begin{array}{llll} p & M_1 & A \rightarrow S: & \mathcal{B}, a', \{\!|\vec{a}_i, \mathcal{B}|\!\}_{k_{AS}}, auth_{k_{AS}}(a', \vec{a}_i, \mathcal{B}) \\ & M_2 & A \leftarrow S: & A, \langle a', \vec{a}_i, \mathcal{B}\rangle_{k_{SA}} \text{ or } A, \langle \vec{a}_i, \mathcal{B}\rangle_{k_{SA}} \end{array}$$

The reason of using the primitive above is to make the binding easier. A provides a plaintext nonce a' together with an encrypted variable, so when B needs to generate the binding information $\mathtt{c}_{BS}$, he can use a' instead of the encrypted term, i.e. $a', \mathcal{B}_{a'} \sqsubseteq \mathtt{c}_{BS}$ for the recentness guarantee of B's involvement, or $a', \vec{b}_j, \mathcal{B}_{a'} \sqsubseteq \mathtt{c}_{BS}$ for the recentness guarantee of B's involvement together with the binding.

6.4 Design Examples

This section shows how the proposed method can be applied to mutual authentication, key transport and agreement, and key distribution.

6.4.1 Mutual Authentication

The design of two-party mutual authentication protocols without an authentication server is basically the same as that of asymmetric protocols, if two distinctive keys are used by each participant. All properties verified in the previous chapter hold here, hence, nothing more needs to be mentioned.

6.4.2 Protocols with Authentication Servers

Symmetric protocols with authentication servers or trusted third parties are one of the most common forms found in published literature. Some examples are the Needham-Schroeder symmetric key protocol, the Otway-Rees protocol, and the Yahalom protocol. These protocols are structurally different but share many common features. This subsection shows how to design and derive such protocols starting with a simple composition of two primitives.

p_{11}	M_1	$A \to S:$	$A, B, S, \{\!|\vec{a}_1, A, B, S|\!\}_{k_{AS}}, auth_{k_{AS}}(\vec{a}_1, A, B, S)$
	M_2	$A \leftarrow S:$	$A, \{\!|\vec{c}_1|\!\}_{k_{SA}}, \langle \mathsf{c}_{SA} \rangle_{k_{SA}}$
p_{12}	N_1	$B \to S$:	$A, B, S, \{\!|\vec{b}_2, A, B, S|\!\}_{k_{BS}}, auth_{k_{BS}}(\vec{b}_2, A, B, S)$
	N_2	$B \leftarrow S:$	$B, \{\!|\vec{d}_2|\!\}_{k_{SB}}, \langle \mathsf{c}_{SB} \rangle_{k_{SB}}$
p_{21}	M_1	$A \to S:$	$A, B, S, \vec{a}, auth_{k_{AS}}(\vec{a}_1, A, B, S)$
	M_2	$A \leftarrow S:$	$A, \{\!|\vec{c}_1|\!\}_{k_{SA}}, \langle \mathsf{c}_{SA} \rangle_{k_{SA}}$
p_{22}	N_1	$B \to S:$	$A, B, S, \vec{b}, auth_{k_{BS}}(\vec{b}_2, A, B, S)$
	N_2	$B \leftarrow S:$	$B, \{\!|\vec{d}_2|\!\}_{k_{SB}}, \langle \mathsf{c}_{SB} \rangle_{k_{SB}}$
p_{31}	M_1	$A \to S:$	$A, B, S, \{\!|\vec{a}_1, A, B, S|\!\}_{k_{AS}}, auth_{k_{AS}}(\vec{a}_1, A, B, S)$
	M_2	$A \leftarrow S:$	$A, \langle \mathsf{c}'_{SA} \rangle_{k_{SA}}$
p_{32}	N_1	$B \to S:$	$A, B, S, \{\!|\vec{b}_2, A, B, S|\!\}_{k_{BS}}, auth_{k_{BS}}(\vec{b}_2, A, B, S)$
	N_2	$B \leftarrow S:$	$B, \langle \mathsf{c}'_{SB} \rangle_{k_{SB}}$
p_{41}	M_1	$A \to S:$	$A, B, S, \vec{a}, auth_{k_{AS}}(\vec{a}_1, A, B, S)$
	M_2	$A \leftarrow S:$	$A, \langle \mathsf{c}'_{SA} \rangle_{k_{SA}}$
p_{42}	N_1	$B \to S:$	$A, B, S, \vec{b}, auth_{k_{BS}}(\vec{b}_2, A, B, S)$
	N_2	$B \leftarrow S:$	$B, \langle \mathsf{c}'_{SB} \rangle_{k_{SB}}$

Table 6.1: Primitives

Assume that S is an authentication server with faithful behaviour and each participant shares a secret key with S. Any two primitives using symmetric keys can be combined together to generate a new protocol. However, it is very important to specify the cardinality of the binding group in each primitive before any composition, since the authentication server's behaviour depends on it. We can simply assume that each binding group's cardinality is the same. It is possible to design a protocol without making them same but it does not seem to have any special advantages, so we simply assume them equal and denote the size as n. For this example, we assume that $n = 3$, i.e. two participants and an authentication server (the design of protocols where $n > 3$ can be done similarly). Any primitives can be used but for the examples, the primitives shown in Table 6.1 are mainly used, where an indexed vector $\vec{\alpha}_i$ denotes $(i, \vec{\alpha})$, and the subscript i in p_{*i} represents the participant's location (or role) within the binding group, i.e. p_{*1} and p_{*2} represent $p_{*1}(\vec{a}_1; (A, \cdot, S))$ and $p_{*2}(\vec{b}_2; (\cdot, B, S))$ respectively. Let $\mathcal{B} = (A, B, S)$, $\mathsf{c}_{SA} = (\vec{a}_1, \vec{c}, \mathcal{B})$, $\mathsf{c}_{SB} = (\vec{b}_2, \vec{d}, \mathcal{B})$, $\mathsf{c}'_{SA} = (\vec{a}_1, \mathcal{B})$ and $\mathsf{c}'_{SB} = (\vec{b}_2, \mathcal{B})$. For notational convenience, we use p_i to denote p_{ji}.

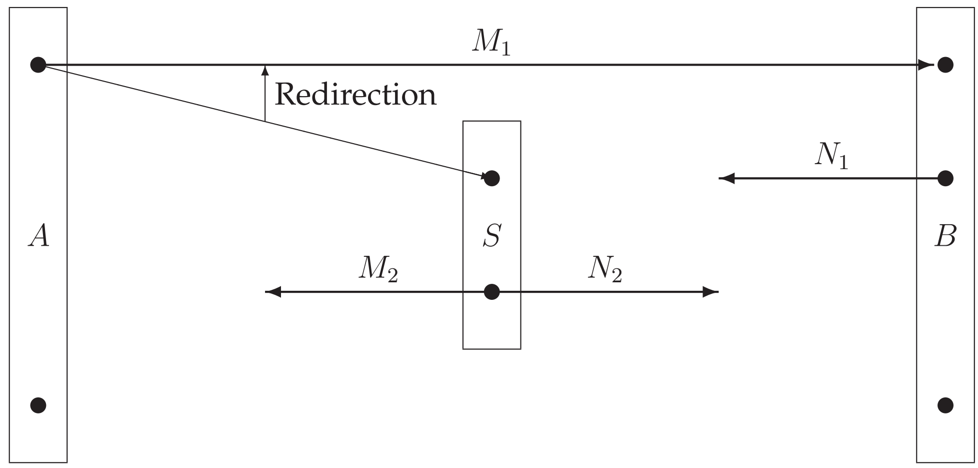


Figure 6.6: **p-protocol**

Let $\mathcal{P} = |p_1 \otimes p_2|$ be a simple composition of any two primitives in the table. For $\mathcal{P}$ to be a p-protocol, $o(\mathcal{P})$ should be determined. Except M_1, which should be the first message, all other events can be arranged as designers want if they do not violate each local order $o(p_i)$. Our assumption on the authentication server also limits the choice of possible $o(\mathcal{P})$. The authentication server is supposed to generate an output message after receiving all input messages, so $e(N_1) \leq e(M_2)$. A p-protocol satisfying these restrictions structurally looks like Figure 6.6, where M_1 is redirected to B instead of S. The redirection is up to the designers' decision, but if the redirection is not performed, then one more message should be added in order to notify B that A wants to start a run.

A protocol consists of a simple composition, therefore all properties verified in the previous section hold here too. That is, when A finishes a run of the protocol, it is guaranteed that a corresponding strand exists in B, but it is not guaranteed that the strand is recent. Depending on protocol needs, the simple protocol can be enhanced by adding events delivering extra or hidden information.

Key Establishment: Key Transport or Key Agreement

Key establishment either by *key transport* or *key agreement* is a common goal of protocols using an authentication server. A key transport mechanism is a key establishment technique where one participant creates or otherwise obtains a secret value, and securely transfers it to the other(s) [37]. Basically, one participant sends a secret to the other via the authentication server, and the authentication server delivers the secret to the other participant. The wide-mouthed frog protocol is a famous example of this kind. Either of the participants can play the role of a secret sender, but we assume that B plays that role in the example given below. Assuming that A and B are two participants, p_{21} and p_{32} might be a desirable choice of the primitives for the key transport because only B needs to send a secret to S.

Let $\mathcal{P} = |p_{21}(\vec{a}_1, \vec{c}_1; \mathcal{B}) \otimes p_{32}(\vec{b}_2; \mathcal{B})|$, where $\mathcal{B} = (A, B, S)$. Let $\vec{c} = \vec{b}$, which means the authentication server will deliver $\vec{b}$ through $\vec{c}$ to A. The values in $\vec{c}_1$ will be replaced with the corresponding values $\vec{b}_2$ except the index of $\vec{c}_1$. At the end of a run, the authentication server will have $\vec{b}$ delivered from B. Moreover, this information comes with its binding group information, i.e. $\mathcal{B}$. Delivering $\vec{b}$ securely to a participant in the binding group is not difficult. Adding an encrypted term from the authentication server to the participant might be enough, but would not provide any evidence to A that the delivered information originated recently from B. Therefore, a cryptographically transformed term of $\vec{a}$ originating from B should be included before the server delivers $\vec{b}$. A term $\langle \mathtt{c}_{BS} \rangle_{k_{BS}}$ will do the job, where $\vec{a}, \mathcal{B} \sqsubseteq \mathtt{c}_{BS}$ or $\vec{a}, \vec{b}, \mathcal{B} \sqsubseteq \mathtt{c}_{BS}$

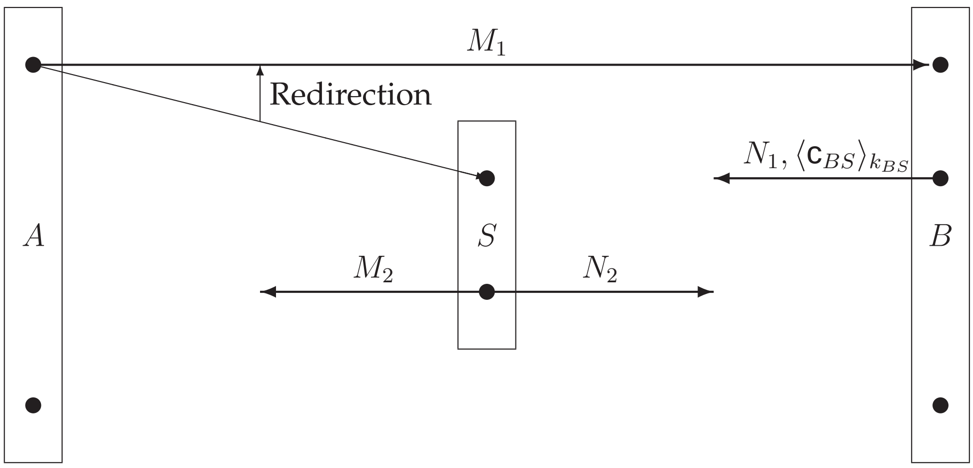


Figure 6.7: **p-protocol after Adding an Extra Term**

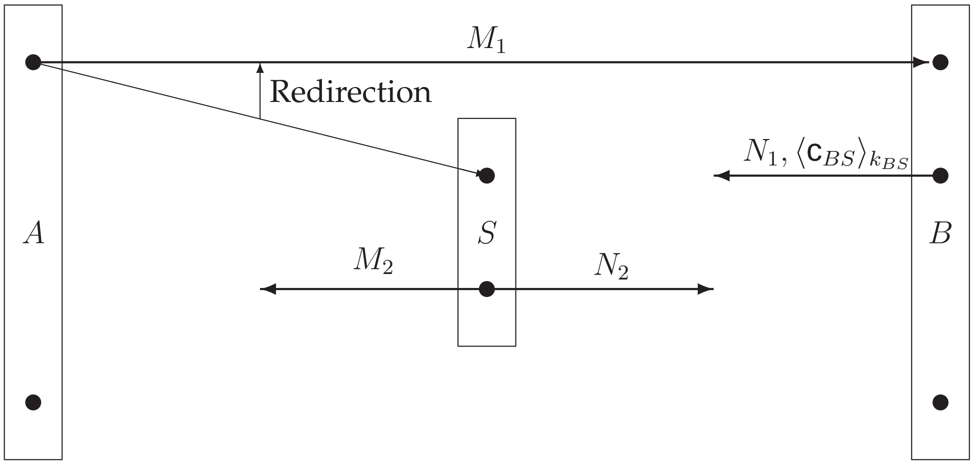
Figure 6.7 shows the p-protocols after adding an extra term, and the following shows a possible rearrangement of messages in the figure. The last two messages originating from the authentication server can change their positions if necessary.

$$
\begin{array}{lll}
\mathcal{P} & A \rightarrow S: & \mathcal{B}, \vec{a}, auth_{k_{AS}}(\vec{a}_1, \mathcal{B}) \\
& B \rightarrow S: & \mathcal{B}, \{\!|\vec{b}_2, \mathcal{B}|\!\}_{k_{BS}}, auth_{k_{BS}}(\vec{b}_2, \mathcal{B}), \langle \mathsf{c}_{BS} \rangle_{k_{BS}} \\
& * \leftarrow S: & A, B, \{\!|\vec{c}_1|\!\}_{k_{SA}}, \langle \mathsf{c}_{SA} \rangle_{k_{SA}}, \langle \mathsf{c}'_{SB} \rangle_{k_{SB}} \\
& & \Downarrow \quad \vec{b} = \vec{c} \quad \text{and if } \mathsf{c}_{BS} = \vec{a}, \vec{b}, \mathcal{B} \\
\mathcal{P} & A \rightarrow *: & \mathcal{B}, \vec{a}, auth_{k_{AS}}(\vec{a}_1, \mathcal{B}) \\
& B \rightarrow *: & \mathcal{B}, \{\!|\vec{b}_2, \mathcal{B}|\!\}_{k_{BS}}, \langle \vec{a}, \vec{b}, \mathcal{B} \rangle_{k_{BS}} \\
& * \leftarrow S: & A, B, \{\!|\vec{b}_1|\!\}_{k_{SA}}, \langle \vec{a}_1, \vec{b}, \mathcal{B} \rangle_{k_{SA}}, \langle \vec{b}_2, \mathcal{B} \rangle_{k_{SB}} \\
& & \Downarrow \\
\mathcal{P} & A \rightarrow B: & A, B, S, \vec{a}, auth_{k_{AS}}(\vec{a}_1, A, B, S) \\
& B \rightarrow S: & A, B, S, \vec{a}, auth_{k_{AS}}(\vec{a}_1, A, B, S), \{\!|\vec{b}_2, A, B, S|\!\}_{k_{BS}}, \langle \vec{a}, \vec{b}, A, B, S \rangle_{k_{BS}} \\
& S \rightarrow B: & B, \langle \vec{b}_2, A, B, S \rangle_{k_{SB}} \\
& S \rightarrow A: & A, \{\!|\vec{b}_1|\!\}_{k_{SA}}, \langle \vec{a}_1, \vec{b}, A, B, S \rangle_{k_{SA}}
\end{array}
$$

Key agreement is a key establishment technique in which a shared secret is derived by two (or more) participants as a function of information contributed by each of these, ideally such that no participant can predetermine the resulting value. Key agreement can be seen as an extension of the key transport protocol proposed above. Similarly, p_{11} and p_{12} might be a preferable choice for the purpose, since both participants send secrets. The same order restrictions applied in the previous example hold here. Let $\vec{a} = \vec{d}$ and $\vec{b} = \vec{c}$. That is, the values in $\vec{d}_2$ will be replaced with the corresponding values $\vec{a}_1$ except the index of $\vec{d}_2$, and the values in $\vec{c}_1$ will be replaced with the corresponding values $\vec{b}_2$ except the index of $\vec{c}_1$. The primitive p_{11} is replaced with the following primitive to make binding simpler before composition since the participant B cannot see the nonce coming from A.

$$
\begin{array}{llll}
p'_{11} & M_1 & A \rightarrow S: & \mathcal{B}, a', \{\!|\vec{a}_1, \mathcal{B}|\!\}_{k_{AS}}, auth_{k_{AS}}(a', \vec{a}_1, \mathcal{B}) \\
& M_2 & A \leftarrow S: & A, \{\!|\vec{c}_1|\!\}_{k_{SA}}, \langle \mathsf{c}_{SA} \rangle_{k_{SA}}
\end{array}
$$

To show that A was recently involved in the run, A has to generate a term either using a newly agreed key generated through the key generation function or using the nonce originating from B and sends it to B. Assuming that A knows a secret key, a one-way authentication procedure which authenticates A can be added to finish the protocol as shown in Figure 6.8. Notice that message M'_1 can be removed if $\vec{b}$ contains a nonce (not a secret) which can later be used as a challenge. The following is a possible rearrangement of messages in the figure, where M'_1 and M'_2 can be any one-way authentication protocol between A and B using the newly agreed key.



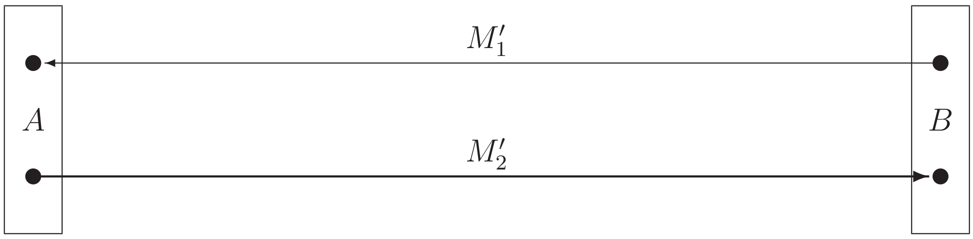


Figure 6.8: **p-protocol for Key Agreement**

$$
\begin{array}{lll}
\mathcal{P} & A \to S: & \mathcal{B}, \{\!|\vec{a}_1, \mathcal{B}|\!\}_{k_{AS}}, auth_{k_{AS}}(\vec{a}_1, \mathcal{B}) \\
 & B \to S: & \mathcal{B}, \{\!|\vec{b}_2, \mathcal{B}|\!\}_{k_{BS}}, auth_{k_{BS}}(\vec{b}_2, \mathcal{B}), \langle \mathsf{c}_{BS} \rangle_{k_{BS}} \\
 & B \leftarrow S: & A, B, \{\!|\vec{c}_1|\!\}_{k_{SA}}, \langle \mathsf{c}_{SA} \rangle_{k_{SA}}, \{\!|\vec{d}_2|\!\}_{k_{SB}}, \langle \mathsf{c}_{SB} \rangle_{k_{SB}} \\
 & B \to A: & M'_1 \\
 & A \to B: & M'_2 \\
 & & \Downarrow \vec{c} = \vec{b}, \vec{d} = \vec{a} \\
\mathcal{P} & A \to S: & \mathcal{B}, \{\!|\vec{a}_1, \mathcal{B}|\!\}_{k_{AS}}, auth_{k_{AS}}(\vec{a}_1, \mathcal{B}) \\
 & B \to S: & \mathcal{B}, \{\!|\vec{b}_2, \mathcal{B}|\!\}_{k_{BS}}, \langle \mathsf{c}_{BS} \rangle_{k_{BS}} \\
 & B \leftarrow S: & A, B, \{\!|\vec{b}_1|\!\}_{k_{SA}}, \langle \mathsf{c}_{SA} \rangle_{k_{SA}}, \{\!|\vec{a}_2|\!\}_{k_{SB}}, \langle \mathsf{c}_{SB} \rangle_{k_{SB}} \\
 & B \to A: & M'_1 \\
 & A \to B: & M'_2 \\
 & & \Downarrow \\
\mathcal{P} & A \to B: & A, B, S, \{\!|\vec{a}_1, A, B, S|\!\}_{k_{AS}}, auth_{k_{AS}}(\vec{a}_1, A, B, S) \\
 & B \to S: & A, B, S, \{\!|\vec{a}_1, A, B, S|\!\}_{k_{AS}}, auth_{k_{AS}}(\vec{a}_1, A, B, S), \\
 & & \{\!|\vec{b}_2, A, B, S|\!\}_{k_{BS}}, \langle \vec{a}, \vec{b}, A, B, S \rangle_{k_{BS}} \\
 & S \to B: & A, B, \{\!|\vec{b}_1|\!\}_{k_{SA}}, \langle \vec{a}, \vec{b}, A, B, S \rangle_{k_{SA}}, \{\!|\vec{a}_2|\!\}_{k_{SB}}, \langle \vec{a}, \vec{b}, A, B, S \rangle_{k_{SB}} \\
 & B \to A: & M'_1 \\
 & A \to B: & M'_2
\end{array}
$$

One-way Authentication through the Authentication Server

In environments where each participant shares a secret key with an authentication server, an authentication procedure between participants goes through the authentication server. The authentication can be performed with the help of the authentication server, or the authentication server can deliver a secret so that both participants can communicate directly to authenticate each other. The former is the focus of the discussion here, and the latter will be covered below, under key distribution. Woo and Lam's Π protocol series implements a one-way authentication mechanism in such an environment. One participant A wants to authenticate another participant B. The important thing is that A should be guaranteed that B was recently involved in the run by the authentication server. The primitive p_{41} might be a desirable choice for the purpose since no secret needs to be sent by A.

In order to provide that B was recently engaged in a run, there should be a term originating from B containing A's challenge, and the authentication server will reply back to A only when A's challenge is contained in B's reply. This can be easily achieved by adding a term $\vec{a}, \mathcal{B} \sqsubseteq \mathtt{c}_{BS}$ in p_{41}, which results in a protocol like the following.

$$
\begin{array}{lrl}
\mathcal{P} & A \rightarrow S: & \mathcal{B}, \vec{a}, auth_{k_{AS}}(\vec{a}_1, \mathcal{B}) \\
 & B \rightarrow S: & \langle \mathtt{c}_{BS} \rangle_{k_{BS}} \\
 & A \leftarrow S: & A, \langle \mathtt{c}'_{SA} \rangle_{k_{SA}} \\
 & & \Downarrow \\
\mathcal{P} & A \rightarrow B: & \mathcal{B}, \vec{a}, auth_{k_{AS}}(\vec{a}_1, \mathcal{B}) \\
 & B \rightarrow S: & \mathcal{B}, \vec{a}, auth_{k_{AS}}(\vec{a}_1, \mathcal{B}), \langle \vec{a}, \mathcal{B} \rangle_{k_{BS}} \\
 & S \rightarrow A: & A, \langle \vec{a}_1, \mathcal{B} \rangle_{k_{SA}}
\end{array}
$$

Key Distribution

Key distribution is very similar to key agreement, except that a shared key originates from the authentication server. This can be easily achievable from the key agreement protocol above by sending a new key k through $\{\!|\vec{c}|\!\}_{k_{SA}}$ and $\{\!|\vec{d}|\!\}_{k_{SB}}$, i.e. $\vec{c} = \vec{d} = k$. Each participant receives the shared key as a response to his challenge, so the freshness of the key is guaranteed.

$$
\begin{array}{lrl}
\mathcal{P} & A \rightarrow B: & \mathcal{B}, \{\!|\vec{a}_1, \mathcal{B}|\!\}_{k_{AS}}, auth_{k_{AS}}(\vec{a}_1, \mathcal{B}) \\
 & B \rightarrow S: & \mathcal{B}, \{\!|\vec{a}_1, \mathcal{B}|\!\}_{k_{AS}}, auth_{k_{AS}}(\vec{a}_1, \mathcal{B}), \{\!|\vec{b}_2, \mathcal{B}|\!\}_{k_{BS}}, \langle \mathtt{c}_{BS} \rangle_{k_{BS}} \\
 & S \rightarrow A: & A, B, \{\!|\vec{k}_1|\!\}_{k_{SA}}, \langle \mathtt{c}_{SA} \rangle_{k_{SA}}, \{\!|\vec{k}_2|\!\}_{k_{SB}}, \langle \mathtt{c}_{SB} \rangle_{k_{SB}}
\end{array}
$$

Re-authentication

The design of re-authentication or repeated authentication protocols can be done through composition by merging the key transport (or the key agreement) protocol and mutual authentication protocol proposed above. However, most cases of re-authentication protocols, a shared key has a limited lifetime specified by a timestamp, which is defined by either B or S. Therefore, when the authentication server generates a shared key, a timestamp has to be bound with it and should be part of the messages which the server sends to each participant, e.g. $\vec{c} = \vec{d} = (k, t)$, where t is a timestamp. The server will send a key with a timestamp to A, together with a token. A token is an encrypted message from which B can recover the key and the timestamp. If the timestamp is provided by B instead of S, then p_{12} can be extended to include the timestamp, i.e.

$$\begin{array}{llll} p_{12} & N_1 & B \rightarrow S: & \mathcal{B}, \{\!|\vec{b}_2, t, \mathcal{B}|\!\}_{k_{BS}}, auth_{k_{BS}}(\vec{b}_2, t, \mathcal{B}) \\ & N_2 & B \leftarrow S: & B, \{\!|\vec{d}_2, t|\!\}_{k_{SB}}, \langle \mathtt{c}_{SB} \rangle_{k_{SB}} \end{array}$$

The binding information $\mathtt{c}_{SB}$ should also include the timestamp. Then the primitive can be merged with the extended p_{21}, where $\mathtt{c}_{SA}$ also includes t.

$$\begin{array}{llll} p_{21} & M_1 & A \rightarrow S: & \mathcal{B}, \vec{a}, auth_{k_{AS}}(\vec{a}_1, \mathcal{B}) \\ & M_2 & A \leftarrow S: & B, \{\!|\vec{c}_1, t|\!\}_{k_{SA}}, \langle \mathtt{c}_{SA} \rangle_{k_{SA}} \end{array}$$

Putting everything together produces the following protocol.

$$\begin{array}{llll} \mathcal{P} & M_1 & A \rightarrow S: & \mathcal{B}, \vec{a}, auth_{k_{AS}}(\vec{a}_1, \mathcal{B}) \\ & N_1 & B \rightarrow S: & \mathcal{B}, \{\!|\vec{b}_2, t, \mathcal{B}|\!\}_{k_{BS}}, auth_{k_{BS}}(\vec{b}_2, t, \mathcal{B}) \\ & M_2 & A \leftarrow S: & B, \{\!|\vec{c}_1, t|\!\}_{k_{SA}}, \langle \mathtt{c}_{SA} \rangle_{k_{SA}} \\ & N_2 & B \leftarrow S: & B, \{\!|\vec{d}_2, t|\!\}_{k_{SB}}, \langle \mathtt{c}_{SB} \rangle_{k_{SB}} \end{array}$$

The authentication server has to deliver a fresh key, the timestamp, and B's nonce, and a token to A, so $\vec{c}_1 = k, \vec{b}$, and $\vec{d}_2 = k$. The addition of a binder $\langle \vec{a}, \vec{b}, t, \mathcal{B} \rangle_{k_{BS}}$ by B produces the following protocol.

$$\begin{array}{lll} \mathcal{P} & A \rightarrow B: & \mathcal{B}, \vec{a}, auth_{k_{AS}}(\vec{a}_1, \mathcal{B}) \\ & B \rightarrow S: & \mathcal{B}, \vec{a}, auth_{k_{AS}}(\vec{a}_1, \mathcal{B}), \{\!|\vec{b}_2, t, \mathcal{B}|\!\}_{k_{BS}}, \langle \vec{a}, \vec{b}, t, \mathcal{B} \rangle_{k_{BS}} \\ & S \rightarrow A: & A, \{\!|\vec{b}_1, k, t|\!\}_{k_{SA}}, \langle \vec{a}_1, \vec{b}, k, \mathcal{B} \rangle_{k_{SA}}, \{\!|k, t|\!\}_{k_{SB}}, \langle \vec{b}_2, k, t, \mathcal{B} \rangle_{k_{SB}} \\ & A \rightarrow B: & B, \{\!|k, t|\!\}_{k_{SB}}, \langle \vec{b}_2, k, t, \mathcal{B} \rangle_{k_{SB}} \end{array}$$

For the repeated authentication part, any two-party symmetric authentication protocol using the key k (two different keys should be derived from k to provide role asymmetry) and preferably the nonce $\vec{b}$ as a challenge can be merged to the protocol above. If the first message of the added two-party symmetric authentication protocol has the same direction as the last message of the protocol above, then both messages can be merged together to reduce the number of message rounds of the final protocol.

In this chapter, we showed how to use the proposed composition methodology in the design of authentication protocols where an authentication server is involved. As can be seen from examples, the composition method makes the design of those protocols much easier to handle.

7

Complex Protocols

This chapter discusses complex design problems such as wireless authentication protocols and secure payments protocols. The design of these protocols demands new features, namely anonymity and accountability. There exist many different ways of implementing these services. For example, anonymity can be implemented through pseudonym schemes or blind group signatures. The main focus of this chapter is to show that the proposed composition theory is unaffected by the integration of these services into protocol primitives. In this way, the properties proven in previous chapters can be used for the design of protocols requiring these new features.

The outline of the chapter is as follows. Section 7.1 shows how to incorporate anonymity and accountability schemes into the proposed methodology without destroying the established theory. Section 7.2 investigates a common message structure which appears in many examples, and proposes more efficient ways to utilise the structure. Section 7.3 discusses how to design authentication protocols in environments where users have mobility. Many different levels of anonymity are identified and the protocols which satisfy the anonymity requirements are presented. Section 7.4 extends the applicability of the proposed methodology in the design of secure electronic payment protocols.

7.1 New Requirements for Protocol Primitives

So far, our interest has been in the implementation of mechanisms for authentication and secrecy. However, some protocols need goals which do not belong to authentication and secrecy. Anonymity and accountability are among them. This section explains how to incorporate these goals into the proposed methodology.

7.1.1 Anonymity

Anonymity is the state of being not identifiable within a set of subjects, namely the anonymity set, the set of all possible subjects who might cause an action [45]. A sender may be anonymous within a set of potential senders, his sender anonymity set, and a recipient may be anonymous within his recipient anonymity set. Both anonymity sets may be disjoint, be the same, or they may overlap. Given the items of interests (IOIs), anonymity can also be defined as *unlinkability* between IOIs and a certain anonymity set. In protocol design, the IOIs are user identities and anonymity is to achieve some kind of transaction or exchange of messages without revealing the identity of some or all of the participants, i.e. *sender anonymity*, *recipient anonymity*, or *relationship anonymity*. Technically speaking, anonymity in security protocols means not only that the name of the participants should not appear directly, but also that the identities of the participants should not be deducible from the information that is available.

Maintaining anonymity is desirable in a variety of electronic commerce applications. For example, if someone were to vote electronically, he probably would not want anyone to know the candidate for whom he voted, or if one were to use electronic cash to purchase a product, he may not want his identity to be known since this information could be used to trace his spending patterns, and perhaps spam him with junk mail. Although achieving anonymity can be an important design criterion in cryptographic systems, it comes at a cost. If the systems are not carefully designed, the overall security of the system could be compromised.

Unlike secrecy, not the messages themselves but their association with particular participants needs to be protected. In the proposed methodology, anonymity means that a subset of the elements in the binding group of the event is unlinkable to the event. A simple way to implement this unlinkability is using encryption. Like secrecy, user identities in each message, or the binding group can be encrypted in order to implement the required anonymity service. However, it is impossible to implement a high level of anonymity using this scheme only. More viable solutions are pseudonym schemes.

Pseudonym Schemes

Pseudonym systems allow users to interact with multiple organisations anonymously, using pseudonyms. The pseudonyms cannot be linked, but are formed in such a way that a user can prove to one organisation a statement about his relationship with another. Such a statement is called a credential. *A single-use credential* is a credential that a user may safely use once. *A multiple-use credential* may safely be transferred to as many organisations as the user wishes without having to interact further with the issuing organisation. We assume that there exists an infrastructure where users remain anonymous to organisations. The infrastructure is composed of users, issuing organisations and verifying organisations. Users are known to each organisation under a different pseudonym, indeed possibly under multiple pseudonyms. The infrastructure includes procedures by which a user and an organisation establish a new pseudonym, *pseudonym establishment protocols*, and allow users to obtain credentials, *credential issuing protocols* and to show them, *credential verification protocols*. For a higher level of anonymity, especially for protection against traffic analysis attacks, not only pseudonyms, but also schemes such as DC-net and MIX-net [9,10,16,27,28,46] should be provided by the underlying systems.

Using pseudonym systems for anonymity services has several advantages for designers. A protocol can initially be designed without considering the anonymity service and then pseudonym schemes can later replace real user identities with their pseudonyms to provide the service. This provides some degree of separation between anonymity services and authentication and secrecy services in the design. Importantly, replacing real identities of users with their corresponding pseudonyms does not break any agreement properties already verified before the replacement, including authentication and secrecy. Therefore, we can always add anonymity services at the end of the design. However, designers should know what kind of anonymity services the protocol should implement, such as sender anonymity, recipient anonymity or relationship anonymity.

Sometimes, hiding real user identities is not enough to provide anonymity, because there can be a protocol step which reveals user sensitive information. For example, it is possible to trace a specific user from a sequence number in a protocol run, if each user uses a distinctive sequence number. Let $\mathcal{S}_U$ be the set of those terms which can reveal the real identity of user U. Suppose V_o is the verifying organisation. Let k_V be a secret key either known to both U and V_o or only to V_o. Any pseudonym U' of U should belong to the coideal generated by $\mathcal{S}_U \cup \{k_V\}$ in order to hide the real identify of U, that is, $U' \in \mathcal{C}[\mathcal{S}_U \cup \{k_V\}]$. Notice that the protocol obtained after replacing U' with U is discreet with respect to $\mathcal{S}_U \cup \{k_V\}$. Moreover, if it is required for two pseudonyms U' and U'' of U to be unlinkable, then each pseudonym should include some random information.

7.1.2 Accountability

While anonymity can protect an individual, there are also quite legitimate reasons for identifying people, especially where security and the risk of abusive behaviour are involved. This property of identifying responsibility to someone or for some activity is called *accountability*. With the advent of electronic commerce, cryptographic protocols are being adapted for implementing commercial transactions, and may need to provide accountability for protocol participants.

Definiton 30. (Accountability) *Accountability of a principal A for a statement $\varphi(\omega)$ regarding a principal B is the ability of B to make a third participant C conclude that $\varphi(\omega)$ is true and it originates from A.*

The digital signature is the most common way to implement accountability. In order to make A responsible for a statement $\varphi(\omega)$, A needs to sign the statement with her signing key, i.e. $[\varphi(\omega)]_{sk_A}$. Sometimes, not only the identity of the principal responsible for the statement, but also some extra information such as to whom and for what the principal is responsible, or the period of the responsibility needs to be included in the statement. To correctly implement accountability, all the necessary information which constitutes the responsibility should be explicitly included in the statement.

Accountability cannot simply be implemented by putting several primitives together. However, accountability can be added to the protocol at the end of composition. This can be done either by adding an extra message $[\varphi(\omega)]_{sk_A}$ for each statement or by signing existing terms.

7.2 Composition

This section examines a common structure which often appears in the design of multi-party protocols and suggests the concept of mutual equations to help the design of these protocols.

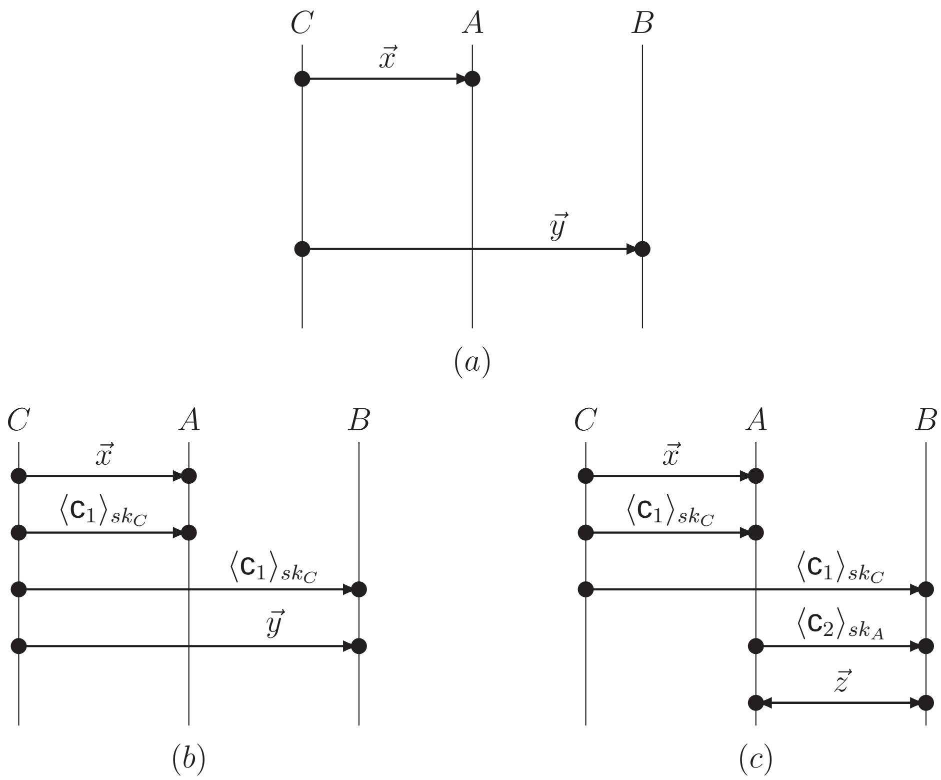


Figure 7.1: **Three-party Composition**

7.2.1 A Common Structure

In many examples of three-party protocol design, we face situations where two participants' authentication or agreement procedure depends on the information coming from another participant as shown in Figure 7.1. Generally speaking, one participant C says something $\vec{x}$ to A and $\vec{y}$ to B. Suppose that A and B can prove that $\vec{x}$ and $\vec{y}$ originate from C, respectively. If A and B trust C, then this situation becomes similar to the situation where two participants communicate via an authentication server, such as the Key Distribution protocol where a trusted server sends a secret key to both participants. In mobile network environments, a mobile user will trust its home domain, and a foreign domain and the home domain will trust each other after authentication. If A and B do not trust C, then they might be interested in knowing whether $\vec{x}$ and $\vec{y}$ match each other. For example, in secure payment systems, a customer sends order information $\vec{x}$ to a merchant and payment information $\vec{y}$ to an acquirer during a purchase procedure, and the merchant and the acquirer later check whether $\vec{x}$ and $\vec{y}$ are consistent. In other words, given the two subprotocols shown in Figure 7.1 (a), if C is trusted by both participants A and B or if C is the person who has to show either A or B that $\vec{x}$ and $\vec{y}$ are related, then C can send both A and C some extra information which binds $\vec{x}$ and $\vec{y}$. The binding information will be the union of the parameters i.e. $\mathtt{c} = \vec{x} \cup \vec{y}$ and it should originate from C. Generally, $\langle \mathtt{c} \rangle_{sk_C}$ can be the binder. However, sometimes, this binder might not provide enough information about the binding. For example, when $\vec{x}$ should be hidden from B, B cannot check $\mathtt{c}$ because $\vec{x}$ is unknown to him. To solve this problem, a term which contains the information $\vec{x}$ and belongs to the coideal of $\vec{x}$ can be used instead of $\vec{x}$, e.g. $\mathtt{c} = \langle \vec{x} \rangle \cup \vec{y}$. In this case, C should deliver $\langle \mathtt{c} \rangle_{sk_C}$ and $\langle \vec{x} \rangle_{sk_C}$ to B for the binding.

Assume that $\mathtt{c}_1 = \vec{x} \cup \vec{y}$ is used as the binding information. Figure 7.1 (b) shows the structure after adding the binders to both A and B. The binder should be sent to all participants who should know the binding information. Notice that the position of the binder comes before the second subprotocol as mentioned in Chapter 5. From the binder, each participant knows the parameter of the run which he was not involved in. By comparing the information they received from C, A and B can check whether $\vec{x}$ and $\vec{y}$ are related, if none of them are in collaboration with C. After receiving the binders, A and B can use $\mathtt{c}_1$ to link another subprotocol between them. Assume that a subprotocol between A and B should be part of the whole protocol, and let $\vec{z}$ be the parameter of the run of the subprotocol. This subprotocol can be bound to the other subprotocols using the binding information $\mathtt{c}_2 = \mathtt{c}_1 \cup \vec{z}$. The binding information is sufficient enough to bind the subprotocols, i.e. it can be interpreted as the binding information of two primitives between A and B, one with $\mathtt{c}_1$ and the other $\vec{z}$. Therefore, the case becomes the same as two-party composition and all rules of two-party composition are valid here, too, including the proof that the two subprotocols are bound by the binder. Either of the participants A and B can add the binder. In case of A, the binder will be $\langle \mathtt{c}_2 \rangle_{sk_A}$ as shown in (c).

7.2.2 Mutual Equations

In order to utilise the structure of Figure 7.1 (a) more efficiently, let us introduce the concept of mutual equations. When $\vec{x} = \vec{y}$, it is easy to prove that C sent the same information to both participants when they finish a successful run. Instead of sending the same information, some complicated equation(s) can be agreed by the participants. When the participants receive some data from C, they can evaluate the equations with the data they received, and then check whether the outputs match each other. We call these equations *mutual equations*.

Definiton 31. (Mutual Equations) *Let f be an equation $f_A(\vec{x'}_A) = f_B(\vec{y'}_B)$ agreed by participants. The equation f is a mutual equation (of A by B) if it satisfies the following.*

1. *$(\vec{y}_B)_{1\cdots n}$ and $(\vec{x}_A)_{1\cdots n}$ are either known or provable to originate from C. These values can be delivered in secure ways if necessary.*
2. *When a set of fresh values $(\vec{y}_B)_{1\cdots n}$ are given to B, B can generate their outputs $(f_B(\vec{y'}_B))_{1\cdots n}$, where $\vec{y}_B \subseteq \vec{y'}_B$.*
3. *Given $y' \in \vec{y}_B$, A can calculate a fresh output $f_A(\vec{x'}_A)(= f_B(\vec{y'}_B))$, where $\vec{x}_A \subseteq \vec{x'}_A$*
4. *It is impossible to calculate $f_A(x)$ or $f_B(x)$ without x.*

Any equations can be mutual equations, but for all mutual equations, it is required that the output $f_i(x)$ should be impossible to calculate without its input x. Therefore, if x or a part of x is a secret, then only the participant possessing x can calculate the output $f_i(x)$. Hash and signature functions are commonly used to form these equations e.g. $y_B = \langle x_A \rangle$, $y_B = \langle x_A \rangle_{sk_C}$. The identity function can also be used to generate these equations, if $\vec{x}_A$ and $\vec{y}_B$ are delivered from C in secure ways e.g. $y_B = x_A$. Two or more functions can be combined to create these equations e.g. if $f_{A1}(x_A) = f_{B1}(y_B)$ and $f_{A2}(x'_A) = f_{B2}(y'_B)$ are mutual equations, then $f_{A1}(x_A) \cdot f_{A2}(x'_A) = f_{B1}(y_B) \cdot f_{B2}(y'_B)$ and $\langle f_{A1}(x_A) \rangle \cdot f_{A2}(x'_A) = \langle f_{B1}(y_B) \rangle \cdot f_{B2}(y'_B)$ are also mutual equations. Encryptions are not used as parts of mutual equations because mutual equations are proposed as

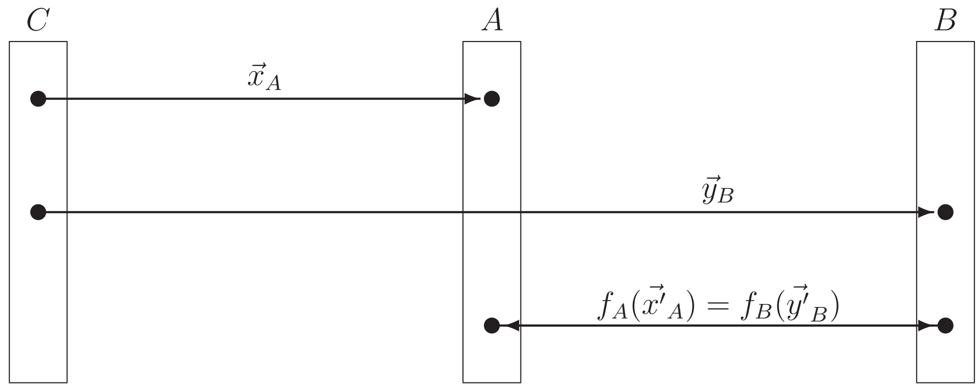


Figure 7.2: **Mutual Equations**

a checking mechanism for information rather than as a hiding mechanism. The reason for $\vec{x}_A \subseteq \vec{x'}_A$ or $\vec{y}_B \subseteq \vec{y'}_B$ is to allow each participant to use its own private information in generating the output.

Mutual equations are especially useful to check whether the information received by A or B from C are equivalent or related, i.e. both participants can confirm that the other participant possesses the same information using the equation they agreed on. Mutual equations also guarantee that the existence of a certain event means the existence of its corresponding event. For example, in Figure 7.2, if the equation matches, then A knows that a send event $e^+(\vec{y}_B)$ originating from C happened in the recent time period defined by $\vec{x}_A$. Well defined mutual equations can simplify the design of three-party protocols. These equations can also be used as authentication tests. Technically speaking, any instance of authentication tests can be transformed into mutual equation(s). Some examples of mutual equations found in published protocols are as follows.

Key Distribution

$$id(k) = id(k)$$

where id is the identity function which always returns the same value that is used as its argument. In this case $\vec{x}_A = \vec{y}_B = k$.

Global Systems for Mobile Communication (GSM)

$$\mathsf{A3}(k_U, \mathsf{RAND}) = \mathsf{XRES}$$

where A3 is the authentication algorithm used in GSM, k_U is a long-term secret key, which is unique and only known to the mobile user U and his home authority, and RAND is a random number generated by the AuC (Authentication Center) to produce the SRES (Signed Response) or the XRES (Expected Response). The AuC generates a set of (RAND, XRES) pairs and passes them to V, i.e. $\vec{x}_U = \emptyset$ and $\vec{y}_V = (\mathtt{RAND}, \mathtt{XRES})_{1 \cdots n}$. To authenticate U, V sends RAND to U and U generates SRES using the function A3 with inputs k_U and RAND, and returns SRES to V where it is compared with the XRES. If they match, the authentication is successful.

Universal Mobile Telecommunications System (UMTS)

$$
\begin{aligned}
f_1(k_U, \mathtt{AMF}, \mathtt{SQN}) &= \mathtt{MAC} \\
f_2(k_U, \mathtt{RAND}) &= \mathtt{XRES} \\
f_3(k_U, \mathtt{RAND}) &= \mathtt{CK} \\
f_4(k_U, \mathtt{RAND}) &= \mathtt{IK} \\
f_5(k_U, \mathtt{RAND}) &= \mathtt{AK}
\end{aligned}
$$

where f_1 and f_2 are message authentication functions, network and user, respectively; f_3 and f_4 are key generation functions, cipher key and integrity key, respectively; f_5 is the anonymity key derivation function; AMF is an authenticated management field; SQN is a sequence number; CK is a cipher key, IK is an integrity key, and AK is an anonymity key, respectively. AUTN (Authentication Token) is defined (SQN $\oplus$ AK, AMF, MAC) and AV (Authentication Vector) (RAND, XRES, CK, CK, IK, AUTN). In this case, $\vec{x}_U = \emptyset$ and $\vec{y}_V = (\mathtt{AV})_{1 \cdots n}$, i.e. U's home network sends nothing to U but sends V a set of $(\mathtt{AV})_{1 \cdots n}$, which V can use to authenticate U. For the authentication, V sends the parameters RAND and AUTN to U. U checks whether AUTN can be accepted and, if so, produces a response SRES which is sent back to V. V compares the received SRES with XRES, and if they match then V considers the authentication and the key agreement exchange to be successfully finished.

Secure Electronic Transactions

$$
f_1(\mathtt{OD}) \cdot \langle \mathtt{PI} \rangle = \langle \mathtt{OD} \rangle \cdot f_1(\mathtt{PI}) \quad (= f_2([\langle \mathtt{OD} \rangle \cdot \langle \mathtt{PI} \rangle]_{sk_C}))
$$

where f_1 is a simple hash function and f_2 is a signature verification function, $\cdot$ represents concatenation, OD represents an order description and PI payment information. Let $z = [\langle \mathtt{OD} \rangle \cdot \langle \mathtt{PI} \rangle]_{sk_C}$. Customer C sends $(\mathtt{OD}, \langle \mathtt{PI} \rangle, z)$ to the merchant M and $(\langle \mathtt{OD} \rangle, \mathtt{PI}, z)$ to the acquirer A, i.e. $\vec{x}_A = (\langle \mathtt{OD} \rangle, \mathtt{PI}, z)$ and $\vec{y}_M =$

$(\texttt{OD}, \langle\texttt{PI}\rangle, z)$. The acquirer verifies the merchant if the merchant provides x' and the hash output of x' equals the first term of $f_2(z)$ i.e. $\langle x'\rangle = \langle\texttt{OD}\rangle$.

Mutual equations provide more diverse ways of implementing protocols. As can be seen in the GSM example, mutual equations also provide schemes to authenticate two participants by supplying a set of question and answer pairs to particular participants instead of sending them a secret key.

7.3 Wireless Authentication Protocols

Wireless networks can be an effective way to extend network access. However, they are not simply a continuation of a wired Local Area Network (LAN). On a wireless network everyone can hear from everyone else, so wireless LANs or mobile network environments add a new level of threat to network security such as privacy of communications, accountability for use and availability of service. Generally, the security of wireless communications can be compromised much more easily than that of wired communications. The situation gets further complicated if the users are allowed to cross security domains. Mobile networked computing is also raising some important questions on anonymity and privacy issues. For example, being reachable at any location and at any time creates concern about privacy issues among the potential users. Hence, these issues need to be properly dealt with in a design by composition.

The section is concerned with the design of authentication protocols for mobile networked computing environments. It develops mobile user authentication protocols in intra and inter domain situations using symmetric-key cryptosystems. The protocols provide varying degrees of anonymity of the communicating users to other system users.

7.3.1 System Model and Assumptions

A mobile user has its home where it is registered on a long-term basis. Users of a given domain are registered with the Authentication Server (AS) of that domain. The AS of a domain can be replicated or partitioned within the domain but the set of all partitioned and duplicated ASs represents a single domain-level authority, called the *home* *authority* or the *home domain*. Assume that each mobile user has a universal identity, which is stored in a personal device such as a Subscriber Identity Module (SIM) card. The universal identity of a mobile user is only known to some organisation which is assumed not to reveal the identity. Generally, the mobile user's home domain can play the role of the organisation, if the identity does not need to be protected from its home domain. A mobile user can move to a domain outside its home domain. This domain is called a *foreign authority* or a *foreign (or visiting) domain*. When accessing the network in a foreign domain, a mobile user needs to be authenticated to the foreign domain (generally through the home domain) before using any services.

A mobile user U and its home domain H share secret keys k_{UH} and k_{HU}, respectively and a foreign domain V and H share k_{VH} and k_{HV}, respectively.

7.3.2 Security Requirements

The security goals which may need to be achieved after the successful execution of a wireless authentication protocol are as follows:

A1 Entity authentication: A mobile user and its home domain need to authenticate each other (or only the mobile user) before the home domain grants the mobile user to access any services, and a foreign domain should similarly authenticate a mobile user, either with or without the help of the mobile user's home domain.

A2 Mutual agreement on data for further authentication: For communication either between a mobile user and its home domain or between a mobile user and a foreign domain, some fresh data value such as a secret key needs to be mutually agreed among the participants who need further communication at the end of a protocol run.

In addition to the authentication requirements, wireless authentication protocols have further goals such as privacy.

Privacy

Privacy is one of the most important issues in the design of wireless authentication protocols. The required level of privacy depends on various factors such as the cost incurred by providing the service. The subjects of privacy are a mobile user U, his home domain H, a visiting (or foreign) domain V, legitimate network entities such as other authorised third parties involved in a transaction, and Spy. The objects of privacy are the identity of the user, the identity of its home domain, the identity of its visiting domain, etc.

C1 Hiding User Identity from Spy: This is the simplest privacy requirement. For example, the current location of a mobile user should not be directly associated to the user's identity. Generally speaking, the basic requirement is that analysis of successive aliases should not lead to the disclosure of the real user's identity.

C2 Hiding User Identity from Foreign Authorities: There is no need for a foreign authority to know the real identity of the mobile user. What it needs is only a proof of the solvency of the entity accessing the service and enough information to bill the user's home authority.

C3 Hiding Relationship between the User and Authorities: A stronger privacy requirement is to hide the existing relationship between a mobile user and its home domain from other entities in order to prevent the disclosure of the user's identity by inference. For example, each time the user accesses the network, if the identity of its home domain is not protected, then information about the user's real identity may be inferred by analysing the traffic between the foreign and the home authorities.

7.3.3 Composition

Our design of the protocols starts from a two-party authentication protocol between a mobile user U and its home network H, and then merges the protocol with another two-party authentication protocol and generates a three-party protocol. User authentication protocols based on symmetric key cryptography are considered here. Symmetric key cryptography is particularly suitable for situations where minimal computer power and less computational time are required, which are usually the characteristics of mobile environments.

Intra-Domain Protocol

The intra-domain protocol is for authentication between a mobile user U and its home network H. This protocol can provide either one-way or mutual authentication. When a mobile user is in its home network and wants to use the services provided by the home network, this protocol is used. After the required authentication, H may send a secret key or a set of secret keys to U, so that U can use it for future conversations.

For mutual authentication of U and H, any two protocol primitives mentioned in the previous chapter can be selected. The authentication of U by H can be implemented in many different ways. This example uses a timestamp or a single-use pseudonym. Both of them use unsolicited tests with time-sensitive values, in order to reduce the number of message rounds for the authentication procedure. Let U' and H' be identities of U and H, either real or temporary, and let the binding group $\mathcal{B}$ be (U', H'). An unsolicited test using a timestamp is shown below, where t is a timestamp.

$$p_1 \quad L_1 \quad U \rightarrow H: \quad \mathcal{B}, t, auth_{k_{UH}}(t, \mathcal{B})$$

Alternatively, a single-use pseudonym, which is an output of a time-sensitive value such as $U' = f(t, U)$, can replace the timestamp. From now on, let the primitive p_1 represent either of the two primitives. For the authentication of H by U, we can use the following protocol primitive. No secret needs to be sent from U to H, so the choice of the primitive is justified. The vector $\vec{k}_1$ is a set of secret keys (or a single key) generated by H in response to U's request.

$$\begin{array}{llll} p_2 & M_1 & U \rightarrow H: & \mathcal{B}, \vec{x}, auth_{k_{UH}}(\vec{x}_1, \mathcal{B}) \\ & M_2 & U \leftarrow H: & U', \{\!|\vec{k_1}|\!\}_{k_{HU}}, \langle \vec{x}_1, \vec{k}, \mathcal{B} \rangle_{k_{HU}} \end{array}$$

Merging the two primitives together produces the following p-protocol.

$$\begin{array}{llll} p_1 & L_1 & U \rightarrow H: & \mathcal{B}, t, auth_{k_{UH}}(t, \mathcal{B}) \\ p_2 & M_1 & U \rightarrow H: & \mathcal{B}, \vec{x}, auth_{k_{UH}}(\vec{x}_1, \mathcal{B}) \\ & M_2 & U \leftarrow H: & U', \{\!|\vec{k_1}|\!\}_{k_{HU}}, \langle \vec{x}_1, \vec{k}, \mathcal{B} \rangle_{k_{HU}} \end{array}$$

The two primitives can be bound by adding the binding information, say $\mathtt{c}$. The union of the parameters of p_1 and p_2 is $(t, \vec{x}_1, \mathcal{B})$, so $\mathtt{c} = (t, \vec{x}, \mathcal{B})$. Early binding puts the binder $\langle \mathtt{c} \rangle_{k_{UH}}$ from U to H, and the binder replaces $auth_{k_{UH}}(t, \mathcal{B})$ and $auth_{k_{UH}}(\vec{x}_1, \mathcal{B})$ since $t, \mathcal{B} \sqsubseteq \mathtt{c}$ and $\vec{x}_1, \mathcal{B} \sqsubseteq \mathtt{c}$.

$$
\begin{array}{llll}
p_1 \otimes p_2 & L_1 \& M_1 & U \to H: & \mathcal{B}, t, \vec{x}, auth_{k_{UH}}(t, \mathcal{B}), auth_{k_{UH}}(\vec{x}_1, \mathcal{B}), \langle \mathtt{c} \rangle_{k_{UH}} \\
 & M_2 & U \leftarrow H: & U', \{\!|\vec{k_1}|\!\}_{k_{HU}}, \langle \vec{x}_1, \vec{k}, \mathcal{B} \rangle_{k_{HU}} \\
 & & & \Downarrow \\
p_1 \otimes p_2 & M_1' & U \to H: & \mathcal{B}, t, \vec{x}, \langle t, \vec{x}, \mathcal{B} \rangle_{k_{UH}} \\
 & M_2 & U \leftarrow H: & U', \{\!|\vec{k_1}|\!\}_{k_{HU}}, \langle \vec{x}_1, \vec{k}, \mathcal{B} \rangle_{k_{HU}}
\end{array}
$$

When U receives the last message M_2, due to the agreement property of the primitive p_2 together with the binder $\mathtt{c}$, its corresponding record of a run exists on H. Similarly a receiving event $e^-(M_1')$ lies on H. This is a two-party composition, so the binder added in the protocol links the two primitives.

Inter-Domain Protocol

When a mobile user U travels to a foreign domain V, and requests a service in V, V needs to authenticate U before providing the service. The first authentication is usually done through the help of the mobile user's home authority H, since V cannot verify the mobile user directly. After the first authentication, in order to make consecutive runs more efficient, the home network can send a set of secret keys or formulas to the foreign domain and the mobile user, respectively.

To pick the proper protocol primitives for the design, each participant's guarantee needs to be examined carefully. Assuming that the home domain sends a set of secret keys only to the verified participants, it is easily derivable that two authentication protocol primitives are necessary for the design of the protocol: one between U and H and the other between V and H. The authentication between U and H can be achieved by the intra-domain protocol. To clarify whose service the mobile user is trying to use, all participants need to be mentioned, possibly in each message. This is accomplished by extending $\mathcal{B}$ to (U', V', H'). This extension should be done carefully. If there exists a secret x between U and H, which should not be revealed to V, the discreetness property of x should be proven before the extension.

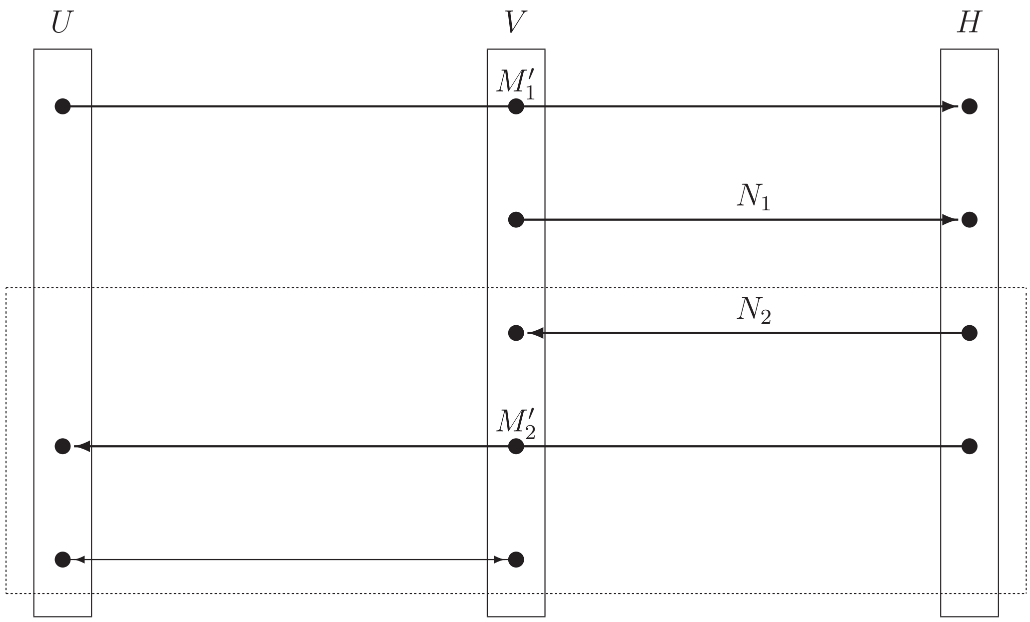


Figure 7.3: **Protocol Structure**

The authentication between V and H can be achieved in a similar way. Any protocol primitive can be selected for the purpose. Let p_2 be the protocol primitive selected. Let $\vec{y}_2$ be a nonce with an index, chosen by V to authenticate H with its binding group $\mathcal{B}$, and $\vec{k}_2$ be a set of secret keys or formulas generated by H in response to V's request.

$$\begin{array}{llll} p_2 & N_1 & V \rightarrow H: & \mathcal{B}, \vec{y}, auth_{k_{VH}}(\vec{y}_2, \mathcal{B}) \\ & N_2 & V \leftarrow H: & V', \{\!|\vec{k}_2|\!\}_{k_{HV}}, \langle \vec{y}_2, \vec{k}, \mathcal{B} \rangle_{k_{HV}} \end{array}$$

Putting the two protocols together with some order relations among events in the primitives forms a simple p-protocol $\mathcal{P}$ shown in Figure 7.3. Notice that every message from and to U should go through V due to the structural restrictions imposed on the network environment, i.e. the mobile user is in the foreign domain. The protocol steps of $\mathcal{P}$ are as follows.

$$\begin{array}{llll} \mathcal{P} & M_1' & U \rightarrow V: & \mathcal{B}, t, \vec{x}, \langle t, \vec{x}_1, \mathcal{B} \rangle_{k_{UH}} \\ & N_1 & V \rightarrow H: & \mathcal{B}, \vec{y}, auth_{k_{VH}}(\vec{y}_2, \mathcal{B}) \\ & N_2 & V \leftarrow H: & V', \{\!|\vec{k}_2|\!\}_{k_{HV}}, \langle \vec{y}_2, \vec{k}, \mathcal{B} \rangle_{k_{HV}} \\ & M_2' & U \leftarrow V: & U', \{\!|\vec{k}_1|\!\}_{k_{HU}}, \langle \vec{x}_1, \vec{k}, \mathcal{B} \rangle_{k_{HU}} \end{array}$$

As shown in the previous chapter, if $\vec{k}_1$ and $\vec{k}_2$ share a common element, for example, such as a shared key, then the two primitives are inseparable, i.e. the existence of one primitive guarantees the existence of the other, and vice versa.

Each participant's pseudonym can replace their real identity for the privacy services. The pseudonym of U should belong to the coideal of U. In order to provide **C1** privacy, $\mathcal{S}_U = \{U, k_{HU}, k_{UH}, k_{HV}, k_{VH}\}$ should be protected. Any pseudonym U', which $U' \subseteq \mathcal{C}[\mathcal{S}_U]$ and is recognisable by H can be used, such as $\{\!|U|\!\}_{k_{UH}}$, or $\{\!|U|\!\}_{k_{UV}}$ between U and V and then $\{\!|U|\!\}_{k_{VH}}$ between V and H. For **C2** privacy, $\mathcal{S}'_U = \{U, k_{HU}, k_{UH}\}$ should be protected. Any pseudonym U', which $U' \subseteq \mathcal{C}[\mathcal{S}'_U]$ and is recognisable by H can be used, such as $\langle U \rangle_{k_{UH}}$. Generally, a random number r is added in generating a pseudonym to make it harder to link each session to a specific user, i.e. $\{\!|U, r|\!\}_{k_{UH}}$. Assuming that underlying infrastructure provides mechanisms such as onion routing or mix-nets, **C3** privacy can be provided by using pseudonyms for both U and H such as $U' = \langle U, r \rangle_{k_{UH}}$ and $H' = \langle H, r \rangle_{k_{HU}}$.

The protocol is structurally the same as the Key Distribution protocol of the previous chapter. There are various ways to make V to authenticate U. The simplest way is to send a secret key or keys which can be used between V and U as most published protocols in this application do. The other way is to formulate a mutual equation and send corresponding input data of the equation to each participant since N_2 and M_2 satisfy the structural requirements of the mutual equation. The application of the GSM equation produces the following protocol.

$$\begin{array}{llll} \mathcal{P} & M'_1 & U \rightarrow V: & \mathcal{B}, t, \vec{x}, \langle t, \vec{x}_1, \mathcal{B} \rangle_{k_{UH}} \\ & N_1 & V \rightarrow H: & \mathcal{B}, \vec{y}, auth_{k_{VH}}(\langle \vec{y}_2, \mathcal{B} \rangle) \\ & N'_2 & V \leftarrow H: & V', \{\!|(\mathsf{AV})_{1\cdots n}|\!\}_{k_{HV}}, \langle \vec{y}_2, (\mathsf{AV})_{1\cdots n}, \mathcal{B} \rangle_{k_{HV}} \\ & M''_2 & U \leftarrow V: & U', \langle \vec{x}_1, \mathcal{B} \rangle_{k_{HU}} \end{array}$$

7.4 Secure Payments Protocols

From 1994 to 1996, payment protocols for all kinds of payment models were proposed, including the Secure Electronic Transactions (SET) protocol. However, today only two approaches for secure payments over the Internet are practically relevant: the SET protocol and encryption of credit card data via SSL or its successor TLS. The SET protocol was initiated by the two largest credit card companies, MasterCard and VISA. Its main goal is to develop a very secure protocol to make it absolutely safe to buy products and services on the Internet. According to the Business Description of the SET protocol, it aims to provide confidentiality of customer's account details and purchases, ensure payment integrity and accountability, and authenticate both merchants and customers [35].

This section identifies security requirements for general electronic payment protocols and shows how to implement a payment protocol similar to the SET protocol using the methodology proposed.

7.4.1 System Model and Assumptions

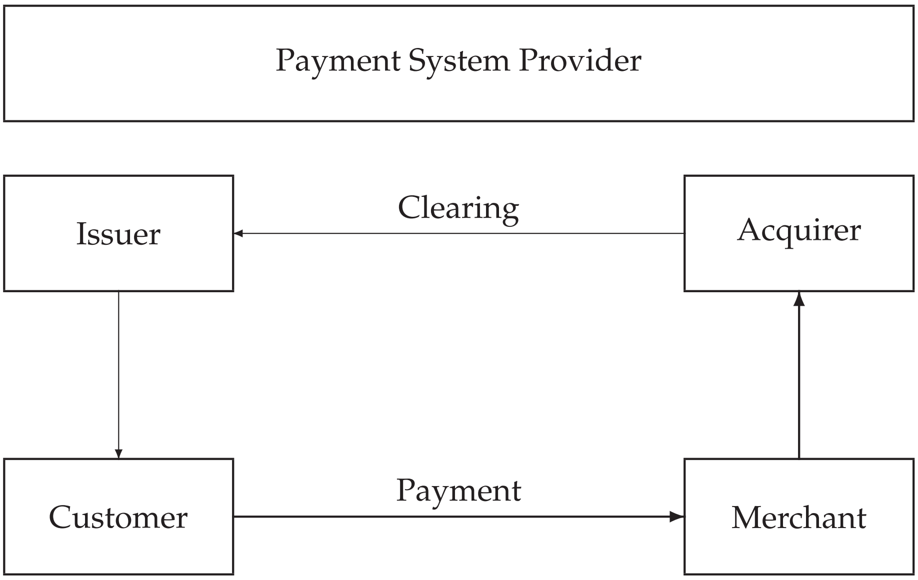


Figure 7.4: **Generic Model of a Payment System**

The generic model of the payment system is shown in Figure 7.4. The payment system is operated by a payment system provider who maintains a fixed business relationship with a number of banks. Banks act as credit card account issuers to customers, and as acquirers of payment records from merchants. Each issuer has a Bank Identification Number (BIN) assigned when an issuer signs up with a payment system provider. Each customer receives his credit card from an issuer, and is assigned a Personal Identification Number (PIN). It is also assumed that the customer is using a computer to execute the payment protocol A merchant signs up with the payment system provider and with a specific bank, called an *acquirer*, to accept deposits. Like a customer, a merchant needs a secure device that stores the merchant's secret keys and performs the payment protocol. Clearing between acquirers and issuers is done using the existing financial networks. Two important procedures for the electronic payment are *authorisation* and *settlement*.

Authorisation: The process by which a customer's credit card is verified as valid and that it has the credit available to make a transaction. Authorisation also verifies that the billing information the customer has provided matches with the information in his credit card company.

Settlement: The process by which transactions with authorisation codes are sent to the merchant. Settlement is a sort of electronic bookkeeping procedure that causes all funds from captured transactions to be routed to the merchant's acquiring bank for deposit.

7.4.2 Security Requirements

This section considers a range of security requirements for each participant involved in the payment process: issuer, acquirer, customer and merchant. They range from mandatory security requirements to optional features.

Issuer/Acquirer Requirements

The issuer and the acquirer are assumed to enjoy some degree of mutual trust. Moreover, it is assumed that an infrastructure enabling secure communication between these participants is already in place. Therefore, we see them as one unit and unify their respective requirements.

A1 *Proof of transaction authorisation by customer*: When the acquirer debits a certain credit card account by a certain amount, the acquirer must be in possession of an unforgeable proof that the owner of the credit card has authorised this payment. This proof must not be replayable or usable as a proof for some other transaction. This means it must certify at least the amount, currency, goods description, merchant identification, and delivery address Note also that in this context the merchant may be an adversary, and even such a merchant must not be able to generate a fake debit.

A2 *Proof of transaction authorisation by merchant*: When the acquirer authorises a payment to a certain merchant, the acquirer must be in possession of an unforgeable proof that this merchant has asked that this payment be made to him.

Merchant Requirements

M1 *Proof of transaction authorisation by acquirer*: The merchant needs an unforgeable proof that the acquirer has authorised the payment. Note that the amount and currency, the time and date, and information to identify the transaction must be certified.

M2 *Proof of transaction authorisation by customer*: Before the merchant receives the transaction authorisation from the acquirer, the merchant might need an unforgeable proof that the customer has authenticated it.

Customer Requirements

C1 *Impossibility of unauthorised payment*: It must be impossible to charge a customer's credit card without possession of the credit card number, PIN and the customer's secret signature key. This must remain the case even if the customer has engaged in many prior legitimate transactions. Two requirements are:

- **Impossibility:** Unauthorised payments are impossible provided the acquirer is honest and his secret key is not available to Spy.
- **Disputability:** The customer can prove not having authorised a payment even if the acquirer's secret key is available to Spy.

C2 *Proof of transaction authorisation by acquirer*: The customer might need to have proof that the acquirer authorised the transaction. This receipt from the acquirer is not essential, but might be convenient.

C3 *Receipt from merchant*: The customer might need a proof that the merchant who has previously made the offer has received payment and promised to deliver the goods. This takes the form of undeniable receipt.

The following requirements might also be desirable.

P1 *Privacy*: Customers might require privacy of order and payment information. For example an investor purchasing information on certain stocks may not want competitors to know which stocks he is interested in.

P2 *Anonymity*: Customers might also want anonymity from Spy as well as the merchant.

7.4.3 Composition

Each participant A possesses two pairs of keys: (ek_A, dk_A) for encryption and decryption, (sk_A, vk_A) for signature and verification. Each participant also holds two corresponding certificates for the public keys. A public key certificate of ek_A, namely, $\mathsf{Cert}(A, ek_A)$ $(= [A, ek_A]_{sk_{CA}})$ is issued by a certification authority and it is assumed that there exists a public key infrastructure which supports the process of certification and verification of this certificate. A customer C also has a credit card data and its PIN.

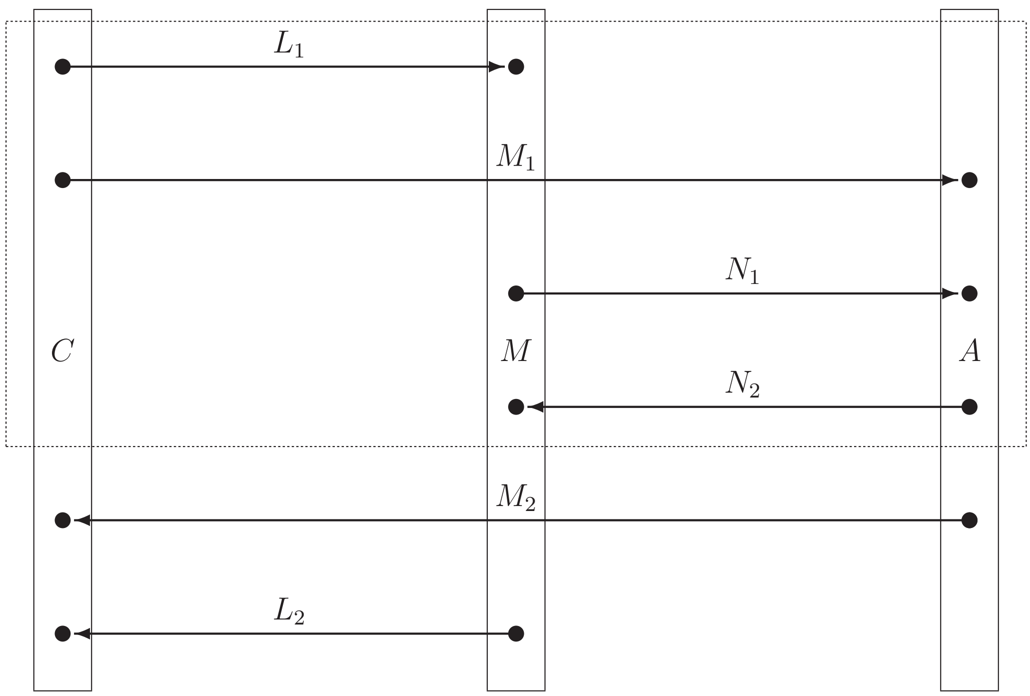


Figure 7.5: **A Purchase Procedure**

A normal purchase procedure might go as follows: C browses M's website and decides on items to purchase. C sends his order description possibly with the payment information to M. C may also send the payment information directly to A, so that

A can check the validity of the payment information that he received from M. For authorisation, M forwards the payment information to A, since M cannot check its validity due to requirement **P1**. A checks the payment information and if it is correct, sends an authorisation code to M. A also needs to send the authorisation code to C in order to inform C that the purchase is authorised. M completes the order by sending the goods ordered and the confirmation to C. Finally, M captures the transaction from his bank and C's bank sends a credit card bill to C. The procedure is shown in Figure 7.5.

The procedure can be divided into three subprotocols, based on the interactions among the participants: three two-party protocols, i.e. one between C and M, another C and A, and the other M and A, respectively. Each message is labelled accordingly in Figure 7.5 Alternative orderings are also possible. For example, N_1 can be located any place between L_1 and N_2. Notice that the messages within the dashed box in Figure 7.5, i.e. L_1, M_1, N_1, and N_2, satisfy the structural requirement of mutual equations. To utilise this structure, mutual equations can be formulated to implement the subprotocol composed of the messages N_1 and N_2, if necessary.

To implement the first subprotocol (the protocol between C and M), an appropriate primitive needs to be carefully selected. The selection is based on the needs for delivery of secrets. For this subprotocol, C needs to send his order description secretly to M but M does not need to send any secret to C. Therefore, the following primitive is suitable, where $\mathcal{B} = (C, M, A)$:

$$\begin{array}{llll} p_1 & L_1 & C \rightarrow M: & \mathcal{B}, \{\!|\vec{x}_1, \mathcal{B}|\!\}_{ek_M}, auth_{sk_C}(\vec{x}_1, \mathcal{B}) \\ & L_2 & M \rightarrow C: & C, \langle \vec{x}_1, \mathcal{B} \rangle_{sk_M} \end{array}$$

The values of $\vec{x}$ are closely related to the order description, which might include values such as the items selected by C, amount of each item, total price, currency, date, the merchant's identity, and delivery address. Any omission of a critical value in $\vec{x}$ can cause serious problems on the protocol implemented, so all required data values should be included. To trace and record each transaction, a unique transaction number may need to be added in $\vec{x}$. The transaction number makes the composition of the subprotocols easier. Let `OD` be the order description which includes all this information, i.e. OD $= (\vec{x}_1, \mathcal{B})$, then

$$
\begin{array}{llll}
p_1 & L_1 & C \to M: & \mathcal{B}, \{|\mathtt{OD}|\}_{ek_M}, auth_{sk_C}(\mathtt{OD}) \\
 & L_2 & M \to C: & C, \langle \mathtt{OD} \rangle_{sk_M}
\end{array}
$$

To implement the second subprotocol (the protocol between C and A), the following primitive can be used. The selection of the primitive is sound since C should send the payment information secretly to A, possibly with some extra information, and A does not need to send any secret to C.

$$
\begin{array}{llll}
p_2 & M_1 & C \to A: & \mathcal{B}, \{|\vec{y}_2, \mathcal{B}|\}_{ek_A}, auth_{sk_C}(\vec{y}_2, \mathcal{B}) \\
 & M_2 & A \to C: & C, \langle \vec{y}_2, \mathcal{B} \rangle_{sk_A}
\end{array}
$$

The values of $\vec{y}$ are mainly connected with the payment information, which includes total price, currency, date, the merchant's identity, the customer's credit card number, its expiration date and PIN. Let $\mathtt{PI}$ be the payment information, i.e. $\mathtt{PI} = (\vec{y}_2, \mathcal{B})$, then

$$
\begin{array}{llll}
p_2 & M_1 & C \to A: & \mathcal{B}, \{|\mathtt{PI}|\}_{ek_A}, auth_{sk_C}(\mathtt{PI}) \\
 & M_2 & A \to C: & C, \langle \mathtt{PI} \rangle_{sk_A}
\end{array}
$$

In other to combine the primitives p_1 and p_2, a binder should be added. The binding information of p_1 and p_2 is $\mathtt{c}_1 = (\vec{x}, \vec{y}, \mathcal{B})$. Unfortunately, $\langle \mathtt{c}_1 \rangle_{sk_C}$ does not work in this case: M and A have no way to verify the binder due to the privacy requirements. Some of possible binders which are understandable by both participants M and A and which satisfy the privacy requirements are $\langle\langle \vec{x} \rangle, \langle \mathtt{PI} \rangle, \mathcal{B}\rangle_{sk_C}$, $\langle\langle \mathtt{OD} \rangle, \langle \vec{y} \rangle, \mathcal{B}\rangle_{sk_C}$, and $\langle\langle \mathtt{OD} \rangle, \langle \mathtt{PI} \rangle\rangle_{sk_C}$. These solutions require extra terms to be sent to a certain participant. For example, if the binder is $\langle\langle \vec{x} \rangle, \langle \mathtt{PI} \rangle, \mathcal{B}\rangle_{sk_C}$, then $\langle x \rangle_{sk_C}$ should be sent to A to help A's verification of the binder. Adding one of the binders above produces the following interim protocol, where $\mathtt{c}_1 = (\langle \mathtt{OD} \rangle, \langle \mathtt{PI} \rangle)$:

$$
\begin{array}{llll}
\mathcal{P} & L_1 & C \to M: & \mathcal{B}, \{|\mathtt{OD}|\}_{ek_M}, auth_{sk_C}(\mathtt{OD}), \langle \mathtt{c}_1 \rangle_{sk_C}, \langle \mathtt{PI} \rangle_{sk_C} \\
 & M_1 & C \to A: & \mathcal{B}, \{|\mathtt{PI}|\}_{ek_A}, auth_{sk_C}(\mathtt{PI}), \langle \mathtt{c}_1 \rangle_{sk_C}, \langle \mathtt{OD} \rangle_{sk_C} \\
 & M_2 & A \to C: & C, \langle \mathtt{OD} \rangle_{sk_A} \\
 & L_2 & M \to C: & C, \langle \mathtt{PI} \rangle_{sk_M}
\end{array}
$$

For the last subprotocol, the following primitive can be used, where $\vec{z}_3$ represents a nonce with a corresponding index:

$$
\begin{array}{llll}
p_3 & N_1 & M \to A: & \mathcal{B}, \{|\vec{z}_3, \mathcal{B}|\}_{ek_A}, auth_{sk_M}(\langle \vec{z}_3, \mathcal{B} \rangle) \\
 & N_2 & A \to M: & M, \langle \vec{z}_3, \mathcal{B} \rangle_{sk_A}
\end{array}
$$

Notice that a mutual equation can be formed to implement this subprotocol. In this example, formulating a mutual equation is not simple because of the privacy requirement **P1**. Generally speaking, the simplest way to formulate an mutual equation is to simulate the Key Distribution equation. In other words, a third party just sends a common value to the two participants. For this example, the common value need not be a secret because it is only used for the verification of what C said. Let χ be the intersection of $\texttt{OD}$ and $\texttt{PI}$. If χ is not empty, and contains enough information to verify both $\texttt{OD}$ and $\texttt{PI}$, then χ can be used as the common value. For example, χ contains a unique transaction number, date, the customer identity, the merchant identity, etc. Alternatively, we can generate an artificial common value by combining each hash output of $\texttt{OD}$ and $\texttt{PI}$, i.e. $\chi = \langle \texttt{OD} \rangle \cdot \langle \texttt{PI} \rangle$ or $\langle \langle \texttt{OD} \rangle \cdot \langle \texttt{PI} \rangle \rangle$. This scheme has several advantages: The scheme can be used for the cases where more than two participants need to generate a formula. The scheme keeps all the information in $\texttt{OD}$ and $\texttt{PI}$. M can generate $\langle \texttt{OD} \rangle$ from $\texttt{OD}$ and A can generate $\langle \texttt{PI} \rangle$ from $\texttt{PI}$, so each needs the other participant to finish the verification of the equation. The introduction of this mutual equation produces a dual signature similar to that used in the SET protocol.

Alternatively, the binding information can be added by M in order to bind p_2 and p_3 through a binder $= \langle \texttt{OD}, \texttt{PI}, \vec{z}_3 \rangle_{sk_M}$. However, $\texttt{PI}$ is unaccessible by M and $\texttt{OD}$ should be protected from A so, the binder will be $\langle \langle \texttt{OD} \rangle, \langle \texttt{PI} \rangle, \vec{z}_3 \rangle_{sk_M}$. Adding the binder produces the following interim protocol, where $\texttt{c}_2 = (\langle \texttt{OD} \rangle, \langle \texttt{PI} \rangle, \vec{z}_3)$:

$$
\begin{array}{llll}
\mathcal{P} & L_1 & C \to M: & \mathcal{B}, \{|\texttt{OD}|\}_{ek_M}, auth_{sk_C}(\texttt{OD}), \langle \texttt{c}_1 \rangle_{sk_C}, \langle \texttt{PI} \rangle_{sk_C} \\
 & M_1 & C \to A: & \mathcal{B}, \{|\texttt{PI}|\}_{ek_A}, auth_{sk_C}(\texttt{OD}), \langle \texttt{c}_1 \rangle_{sk_C}, \langle \texttt{OD} \rangle_{sk_C} \\
 & N_1 & M \to A: & \mathcal{B}, \{|\vec{z}_3, \mathcal{B}|\}_{ek_A}, auth_{sk_M}(\vec{z}_3, \mathcal{B}), \langle \texttt{c}_2 \rangle_{sk_M} \\
 & N_2 & A \to M: & M, \langle \vec{z}_3, \mathcal{B} \rangle_{sk_A} \\
 & M_2 & A \to C: & C, \langle \texttt{PI} \rangle_{sk_A} \\
 & L_2 & M \to C: & C, \langle \texttt{OD} \rangle_{sk_M}
\end{array}
$$

From $\texttt{c}_2$, A and M know that $p_3(\vec{z}_3; \mathcal{B})$ belongs to the same session where $p_1(\texttt{OD})$ and $p_2(\texttt{PI})$ belong. The binding information $\texttt{c}_2$ does not need to be delivered to C because C does not need to know the binding.

The composition only provides ways of combining primitives, i.e. it provides guarantees for the existence of a certain event. However, it does not implement the accountability required. For the enforcement of accountability, extra signature messages imposing the accountability on the associated participant can be added, or terms in the interim protocol can be changed. Generally speaking, the second approach requires fewer messages in the final protocol. The security requirements of each participant are achieved as follows;

Issuer/Accquirer

A1: implemented by $\langle \mathtt{c}_1 \rangle_{sk_C}$, possibly together with $\{\!|\mathtt{PI}|\!\}_{ek_A}$ and $\langle \vec{x}_1 \rangle_{sk_C}$.

A2: implemented by $\langle \mathtt{c}_2 \rangle_{sk_M}$, possibly together with $\{\!|\vec{z}_3, \mathcal{B}|\!\}_{ek_A}$

Merchant

M1: not implemented.

M2: implemented by $\langle \mathtt{c}_1 \rangle_{sk_C}$.

Customer

C1: implemented by $\langle \mathtt{c}_1 \rangle_{sk_C}$.

C2: implemented by $\langle \mathtt{PI} \rangle_{sk_A}$.

C3: implemented by $\langle \mathtt{OD} \rangle_{sk_M}$.

For **M1**, the following change can be made in a message.

$$\langle \vec{z}_3, \mathcal{B} \rangle_{sk_A} \Rightarrow \langle \langle \mathtt{OD} \rangle, \{\!|\mathtt{PI}|\!\}_{ek_A}, \langle \vec{z}, \mathcal{B} \rangle \rangle_{sk_A}$$

The privacy requirement **P1** is implemented by the protocol and **P2** can be implemented if the customer uses a pseudonym, which is only identifiable by Issuer/Acquirer, instead of his real identity.

The final protocol after redirections or merges of messages, and removals of redundant messages might be the following:

$$\begin{array}{llll}
\mathcal{P}' & M_1 & C \to M: & \mathcal{B}, \{\!|\mathtt{OD}|\!\}_{ek_M}, \{\!|\mathtt{PI}|\!\}_{ek_A}, \langle \mathtt{c}_1 \rangle_{sk_C}, \langle \mathtt{OD} \rangle_{sk_C}, \langle \mathtt{PI} \rangle_{sk_C} \\
& M_2 & M \to A: & \mathcal{B}, \{\!|\mathtt{PI}|\!\}_{ek_A}, \{\!|\vec{z}_3, \mathcal{B}|\!\}_{ek_A}, \langle \mathtt{c}_1 \rangle_{sk_C}, \langle \mathtt{OD} \rangle_{sk_C}, \langle \mathtt{c}_2 \rangle_{sk_M} \\
& M_3 & A \to M: & M, \langle \langle \mathtt{OD} \rangle, \{\!|\mathtt{PI}|\!\}_{ek_A}, \langle \vec{z}, \mathcal{B} \rangle \rangle_{sk_A}, \langle \mathtt{PI} \rangle_{sk_A} \\
& M_4 & M \to C: & C, \langle \mathtt{OD} \rangle_{sk_M}, \langle \mathtt{PI} \rangle_{sk_A}
\end{array}$$

The protocol above implements the SET purchase procedure. The main benefit of design by composition is that a complex protocol design problem can be decomposed into a set of simpler protocol design problems, which are usually much easier to solve and to verify their correctness.

8

Conclusions and Further Work

This dissertation has proposed a new methodology for security protocol design and has demonstrated its utility through several applications. All of the techniques have been validated with mathematical proofs. Section 8.1 discusses the main achievements of this research and Section 8.2 identifies further avenues in which the work should be extended in the future.

8.1 Conclusions

The main achievements of the research are summarised in the following statements.

Authentication and secrecy, which are two most important goals of security protocols, are shown to be composable. First, protocol primitives are presented to implement composable authentication and secrecy, and their properties such as agreement, regularity, and discreetness are verified using strand spaces. Second, as a new design method, design by composition is proposed to exploit the composability of authentication and secrecy provided by protocol primitives. Composition is a technique for implementing a more complex protocol using simpler protocols. The composition framework allows the specification of a complex protocol to be decomposed into the specifications of simpler components, which makes the design and verification of the protocol easier to handle. Benefits of this approach are similar to those gained when using a modular approach to software development. On the one hand, complex reasoning about an overall protocol can be reduced to simpler reasoning about a collection of components, and on the other hand, complex goals can be built up from a group of simple goals by adding binding information to those simple goals. Protocol composition in this way makes protocol design simpler and safer than most existing design methodologies, because it is systematic. Protocol composition also provides reusability in protocol design, in the sense that verified primitives or protocols of one application can be reused as parts of another application. Finally, based on the understanding of the composition, many interesting examples, including mobile authentication protocols and secure payment protocols, are demonstrated to be composable and their correctness is verified.

8.2 Further Work

A number of areas for further work have emerged from this research.

Message Refinement Rules

Composition rules alone are not satisfactory to explain certain design mechanisms found in the literature. Many existing protocols show that a complicated protocol can be reached from a simple protocol by gradually strengthening either contents or structures of messages starting from a simple protocol. This design process is called *protocol derivation by refinement*. Intuitively, refinement means either a straight substitution of a new message for an existing message or a reordering of messages, and derivation is a way of getting a new protocol $\mathcal{P}'$ from an existing protocol $\mathcal{P}$ by the application of refinements on $\mathcal{P}$. Not all refinements on a protocol can be justified, and some of them are useless if the strength of the protocol becomes weakened after the refinements. To keep the security properties after executing several refinements, each refinement rule should be accumulative. Accumulative refinement rules can be identified through the examination of published protocols and their correctness can also be verified. These rules together with composition rules will make the composition scheme a more powerful tool to use in protocol design.

Message Optimisation Rules

The protocols produced by composition are generally not optimal. To generate more efficient protocols, some message terms can be combined or redundant messages can be removed from the protocols during composition. For example, in protocols using a faithful authentication sever, some binding groups do not need to be included in generating authentication replies by the sever because the server does not change the binding group received from a participant. When the participant receives the authentication reply from the server, he knows the binding group even though it is not included in the reply. Moreover, when two or more participants trust each other, some components of messages do not need to be delivered to the other participants. A better understanding of these message optimisation rules will help the composition approach to produce more efficient protocols.

Bibliography

[1] ABADI, M., AND GORDON, A. D. A calculus for cryptographic protocols: The Spi calculus. In *Proceedings of the 4th ACM Conference on Computer and Communications Security* (1997), ACM Press, pp. 36–47.

[2] ABADI, M., AND NEEDHAM, R. Prudent engineering practice for cryptographic protocols. *IEEE Transactions on Software Engineering 22*, 1 (1996), 6–15.

[3] ANDERSON, R., BERGADANO, F., CRISPO, B., LEE, J.-H., MANIFAVAS, C., AND NEEDHAM, R. A new family of authentication protocols. *ACM Operating Systems Review 32*, 4 (1998), 9–20.

[4] ANDERSON, R., AND NEEDHAM, R. Robustness principles for public key protocols. In *Proceedings of CRYPTO* (1995), pp. 236–247.

[5] AURA, T., NIKANDER, P., AND LEIWO, J. DoS-resistant authentication with client puzzles. In *Proceedings of International Security Protocols Workshop* (2001), Springer-Verlag, pp. 170–177.

[6] BIRD, R., GOPAL, I., HERZBERG, A., JANSON, P., KUTTEN, S., MOLVA, R., AND YUNG, M. Systematic design of two-party authentication protocols. In *Proceedings of CRYPTO* (1991), pp. 44–61.

[7] BURROWS, M., ABADI, M., AND NEEDHAM, R. A logic of authentication. *Proceedings of the Royal Society 426*, 1871 (1989), 233–271.

[8] BUTTYAN, L., STAAMANN, S., AND WILHELM, U. A simple logic for authentication protocol design. In *Proceedings of the 11th Computer Security Foundations Workshop* (1998), IEEE Computer Society Press, pp. 153–162.

[9] CHAUM, D. L. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM 24*, 2 (1981), 88 – 84.

[10] CHAUM, D. L. The dining cryptographers problem: Unconditional sender and recipient untraceability. *Journal of Cryptology 1*, 1 (1988), 65 – 75.

[11] CLARK, J. A., AND JACOB, J. L. A survey of authentication protocol literature. Tech. Rep. 1.0, 1997.

[12] CLARK, J. A., AND JACOB, J. L. Searching for a solution: Engineering tradeoffs and the evolution of provably secure protocols. In *Proceedings of IEEE Symposium on Security and Privacy* (Oakland, CA, 2000), IEEE Computer Society Press, pp. 14–17.

[13] DATTA, A., DEREK, A., MITCHELL, J. C., AND PAVLOVIC, D. A derivation system for security protocols and its logical formalization. In *Proceedings of the 16th Computer Security Foundation Workshop* (Pacific Grove, CA, 2003), IEEE Computer Society Press, pp. 109 – 125.

[14] DEAN, D., AND STUBBLEFIELD, A. Using client puzzles to protect TLS. In *Proceedings of the 10th USENIX Security Symposium* (2001), pp. 1 – 8.

[15] DENNING, D., AND SACCO, G. Timestamps in key distribution protocols. *Communications of the ACM 24*, 8 (1981), 533–536.

[16] DESMEDT, Y., AND KUROSAWA, K. How to break a practical MIX and design a new one. In *Proceedings of EUROCRYPTO* (2000), vol. 1807 of *Lecture Notes in Computer Science*, pp. 557–572.

[17] DOLEV, D., AND YAO, A. On the security of public-key protocols. *IEEE Transaction on Information Theory 30*, 2 (1983), 198–208.

[18] FOLEY, S. N., AND ZHOU, H. Towards an architecure for autonomic security protocols. In *Proceedings of International Security Protocols Workshop* (Cambridge, England, 2003), Springer-Verlag, pp. 25–30.

[19] GOETHALS, J.-M., AND QUISQUATER, J.-J. Authentication procedures. In *Proceedings of the Workshop on Cryptography,* (1983), vol. 149 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 283–288.

[20] GOLLMANN, D. What do we mean by entity authentication. In *Proceedings of IEEE Symposium on Security and Privacy* (Oakland, CA, 1996), IEEE Computer Society Press, pp. 46–54.

[21] GONG, L. Lower bounds on messages and rounds for network authentication protocols. In *ACM Conference on Computer and Communications Security* (1993), ACM Press, pp. 26–37.

[22] GONG, L., NEEDHAM, R., AND YAHALOM, R. Reasoning About Belief in Cryptographic Protocols. In *Proceedings of IEEE Symposium on Research in Security and Privacy* (1990), IEEE Computer Society, pp. 234–248.

[23] GONG, L., AND SYVERSON, P. 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)* (1995), pp. 44–55.

[24] GUTTMAN, J. D. Security protocol design via authentication tests. In *IEEE Computer Security Foundation Workshop* (Nova Scotia, Canada, 2002), IEEE Computer Society Press, pp. 24–26.

[25] GUTTMAN, J. D., AND FÁBREGA, F. J. T. Authentication tests and the structure of bundles. *Theoretical Computer Science*.

[26] GUTTMAN, J. D., AND THAYER, F. J. Authentication tests. In *Proceedings of IEEE Symposium on Security and Privacy* (2000), pp. 96–109.

[27] JAKOBSSON, M. A practical mix. In *Proceedings of Eurocrypt* (1998), vol. 1403 of *Lecture Notes in Computer Science*, pp. 448–461.

[28] JAKOBSSON, M. Flash mixing. In *Symposium on Principles of Distributed Computing* (1999), pp. 83–89.

[29] JUELS, A., AND BRAINARD, J. Client puzzles: A cryptographic defense against connection depletion attacks. In *Proceedings of Networks and Distributed Security Systems* (1999), pp. 151–165.

[30] LAMPORT, L. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM 21*, 7 (1978), 558 – 565.

[31] LOWE, G. An attack on the Needham-Schroeder public-key authentication protocol. *Information Processing Letters 56*, 3 (1995), 131–133.

[32] LOWE, G. Breaking and fixing the Needham-Schroeder public-key protocol using FDR. In *Tools and Algorithms for the Construction and Analysis of Systems* (1996), vol. 1055 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 147–166.

[33] LOWE, G. A hierarchy of authentication specifications. In *Proceedings of the 10th IEEE Computer Security Foundations Workshop* (1997), IEEE Computer Society Press, pp. 31 – 43.

[34] MAO, W., AND BOYD, C. Towards the Formal Analysis of Security Protocols. In *Proceedings of the 6th Computer Security Foundations Workshop* (1993), IEEE Computer Society Press, pp. 147–158.

[35] MASTERCARD, AND VISA. SET Secure Electronic Transaction specification: Business description.

[36] MEADOWS, C. A. The NRL Protocol Analyzer: An Overview. *Journal of Logic Programming 26*, 2 (1996), 113–131.

[37] MENZIES, A. J., VON OORSCHOT, P. C., AND VANSTONE, S. A. *Handbook of Applied Cryptography*. CRC Press, New York, 1996.

[38] MILLEN, J. K., AND RUESS, H. Protocol-independent secrecy. In *Proceedings of IEEE Symposium on Security and Privacy* (2000), IEEE Computer Society Press, pp. 110 – 209.

[39] NEEDHAM, R. M., AND SCHROEDER, M. D. Using encryption for authentication in large networks of computers. *Communications of the ACM 21*, 12 (1978), 993–999.

[40] NEEDHAM, R. M., AND SCHROEDER, M. D. Authentication revisited. *ACM Operating System Review 21*, 7 (1987), 7–7.

[41] PAULSON, L. The inductive approach to verifing cryptographic protocols. *Journal of Computer Security 6*, 1 (1998), 85–128.

[42] PAULSON, L. C. Inductive analysis of the Internet protocol TLS. *ACM Transactions on Information and System Security 2*, 3 (1999), 332–351.

[43] PERRIG, A., AND SONG, D. A first step towards the automatic generation of security protocols. In *Proceedings of IEEE Symposium on Security and Privacy* (Oakland, CA, 1998), IEEE Computer Society Press, pp. 73–83.

[44] PERRIG, A., AND SONG, D. Looking for diamonds in the desert — extending automatic protocol generation to three-party authentication and key agreement protocols. In *Proceedings of the 13th IEEE Computer Security Foundation Workshop* (2000), IEEE Computer Society Press, pp. 64–76.

[45] PFITZMANN, A., AND KÖHNTOPP, M. Anonymity, unobservability, and pseudeonymity — a proposal for terminology. In *International Workshop on Designing Privacy Enhancing Technologies* (2001), Springer-Verlag, pp. 1–9.

[46] RACKOFF, C., AND SIMON, D. R. Cryptographic defense against traffic analysis. In *ACM Symposium on Theory of Computing* (1993), ACM Press, pp. 672 – 681.

[47] ROSCOE, A. W. Intensional Specifications of Security Protocols. In *Proceedings 9th IEEE Computer Security Foundations Workshop* (1996), IEEE Computer Society Press, pp. 28–38.

[48] SCHNEIDER, S. Using CSP for protocol analysis: the Needham-Schroeder public key protocol. Tech. rep., 1996.

[49] SIMPSON, W. PPP challenge handshake authentication protocol (chap).

[50] SYVERSON, P. Limitations on design principles for public key protocols. In *Proceedings of IEEE Symposium on Security and Privacy* (1996), IEEE Computer Society Press, pp. 62–73.

[51] THAYER, J., AND GUTTMAN, J. D. Honest ideals on strand spaces. In *Proceedings of the 11th Computer Security Foundations Workshop* (1998), IEEE Computer Society Press, pp. 66 – 78.

[52] THAYER, J., HERZOG, J., AND GUTTMAN, J. Strand spaces: Proving security protocols correct. *Journal of Computer Security 7*, 2-3 (1999), 191–230.

[53] WOO, T. Y. C., AND LAM, S. S. A lesson on authentication protocol design. *ACM Operating Systems Review 28*, 3 (1994), 24 – 37.