BUILDING BLOCKS FOR SECURE SERVICES: AUTHENTICATED KEY TRANSPORT AND RATIONAL EXCHANGE PROTOCOLS

THÈSE N° 2511 (2001)

PRÉSENTÉE AU DÉPARTEMENT DE SYSTÈMES DE COMMUNICATION

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

POUR L'OBTENTION DU GRADE DE DOCTEUR ÈS SCIENCES

PAR

Levente BUTTYÁN

Ingénieur informaticien diplômé, Université Technique de Budapest, Hongrie

de nationalité hongroise

acceptée sur proposition du jury:

Prof. J.-P. Hubaux, directeur de thèse
Dr. N. Asokan, corapporteur
Prof. R. Molva, corapporteur
Prof. M. Reiter, corapporteur

Lausanne, EPFL
2001

Abstract

This thesis is concerned with two security mechanisms: authenticated key transport and rational exchange protocols. These mechanisms are potential building blocks in the security architecture of a range of different services. Authenticated key transport protocols are used to build secure channels between entities, which protect their communications against eavesdropping and alteration by an outside attacker. In contrast, rational exchange protocols can be used to protect the entities involved in an exchange transaction from each other. This is important, because often the entities do not trust each other, and both fear that the other will gain an advantage by misbehaving. Rational exchange protocols alleviate this problem by ensuring that a misbehaving party cannot gain any advantages. This means that misbehavior becomes uninteresting and it should happen only rarely.

The thesis is focused on the construction of formal models for authenticated key transport and rational exchange protocols. In the first part of the thesis, we propose a formal model for key transport protocols, which is based on a logic of belief. Building on this model, we also propose an original systematic protocol construction approach. The main idea is that we reverse some implications that can be derived from the axioms of the logic, and turn them into synthesis rules. The synthesis rules can be used to construct a protocol and to derive a set of assumptions starting from a set of goals. The main advantage is that the resulting protocol is guaranteed to be correct in the sense that all the specified goals can be derived from the protocol and the assumptions using the underlying logic. Another important advantage is that all the assumptions upon which the correctness of the protocol depends are made explicit. The protocol obtained in the synthesis process is an abstract protocol, in which idealized messages that contain logical formulae are sent on channels with various access properties. The abstract protocol can then be implemented in several ways by replacing the idealized messages and the channels with appropriate bit strings and cryptographic primitives, respectively.

We illustrate the usage of the logic and the synthesis rules through an example: We analyze an authenticated key transport protocol proposed in the literature, identify several weaknesses, show how these can be exploited by various attacks, and finally, we redesign the protocol using the proposed systematic approach. We obtain a protocol that resists against the presented attacks, and in addition, it is simpler than the original one.

In the second part of the thesis, we propose an original formal model for exchange protocols, which is based on game theory. In this model, an exchange protocol is represented as a set of strategies in a game played by the protocol parties and the network that they use to communicate with each other. We give formal definitions for various properties of exchange protocols in this model, including rationality and fairness. Most importantly, rationality is defined in terms of a Nash equilibrium in the protocol game. The model and the formal definitions allow us to rigorously study the relationship between rational exchange and fair exchange, and to prove that fairness implies rationality (given that the protocol satisfies some

further usual properties), but the reverse is not true in general.

We illustrate how the formal model can be used for rigorous verification of existing protocols by analyzing two exchange protocols, and formally proving that they satisfy the definition of rational exchange. We also present an original application of rational exchange: We show how the concept of rationality can be used to improve a family of micropayment schemes with respect to fairness without substantial loss in efficiency.

Finally, in the third part of the thesis, we extend the concept of rational exchange, and describe how similar ideas can be used to stimulate the nodes of a self-organizing ad hoc network for cooperation. More precisely, we propose an original approach to stimulate the nodes for packet forwarding. Like in rational exchange protocols, our design does not guarantee that a node cannot deny packet forwarding, but it ensures that it cannot gain any advantages by doing so. We analyze the proposed solution analytically and by means of simulation.

Résumé

Cette thèse de doctorat porte sur l'étude de deux mécanismes de sécurité: *le transport authentifié de clés cryptographiques* et *les protocoles d'échange rationnel*. Ces mécanismes pourraient être utilisés comme blocs servant à construire des architectures sécuritaires pour différents types de services. Ainsi, les protocoles de transport authentifié de clés cryptographiques sont-ils employés pour construire des canaux sécurisés entre les entités, protégeant leurs transmissions des attaques extérieures aussi bien passives qu'actives. Les protocoles d'échange rationnel quant à eux peuvent être employés pour protéger les entités impliquées dans une transaction les uns des autres. Ce dernier point est important car, souvent, les différentes parties ne se font pas confiance, chacun craignant que l'autre triche pour augmenter ses gains. Les protocoles d'échange rationnel allègent ce problème en garantissant qu'un tricheur ne gagne rien à se comporter de la sorte. Ceci signifie que mal agir devient inintéressant et qu'il ne devrait donc se produire que rarement.

Dans la présente thèse, on va mettre l'accent sur la construction des modèles formels pour le transport authentifié de clés cryptographiques et les protocoles d'échange rationnel. Dans la première partie de cette thèse, nous proposons un modèle formel pour les protocoles de transport authentifié, basé sur une logique de croyance. En s'appuyant sur ce modèle, nous proposons également une approche originale pour la construction systématique de protocoles. L'idée principale consiste à inverser des implications qui peuvent être dérivées des axiomes de la logique et les transformer en règles de synthèse. Ces règles de synthèse peuvent être employées pour construire un protocole et pour obtenir des hypothèses à partir d'un ensemble de buts qu'on désire atteindre. L'avantage principal d'une telle méthode est l'assurance que le protocole résultant est correct dans le sens où tous les buts indiqués peuvent être déduits du protocole et des hypothèses en utilisant la logique que nous avions définie. Un autre avantage important réside dans le fait que les hypothèses permettant d'atteindre les buts recherchés sont explicites. Le protocole résultant est un protocole abstrait, dans lequel les messages idéalisés qui contiennent des formules logiques sont envoyés via des canaux ayant diverses propriétés d'accès. Le protocole abstrait peut alors être implémenté de plusieurs manières, en remplaçant les messages idéalisés et les canaux, respectivement, par les chaînes binaires et les primitives cryptographiques appropriées.

Nous illustrons l'utilisation de la logique et des règles de synthèse par un exemple: Nous analysons un protocole de transport de clés authentifié proposé dans la littérature, y identifions des faiblesses, montrons comment celles-ci peuvent être exploitées dans diverses attaques, et finalement remodelons le protocole en utilisant notre approche systématique. Nous obtenons un protocole qui résiste aux attaques précédemment identifiées, et qui plus est, représente un niveau de complexité inférieur au protocole initial.

Dans la deuxième partie de la thèse, nous proposons un modèle formel original pour les protocoles d'échange, basé sur la théorie des jeux. Dans ce modèle, un protocole d'échange

est représenté comme un ensemble de stratégies dans un jeu auquel participent les différentes parties impliquées dans le protocole ainsi que le réseau qu'ils utilisent pour communiquer entre eux. Nous donnons dans ce modèle des définitions formelles des différentes propriétés des protocoles d'échange telles que la rationalité et l'équité. La rationalité est définie en termes d'équilibre de Nash. Le modèle et les définitions formelles nous permettent d'étudier rigoureusement le rapport entre l'échange rationnel et l'échange équitable, et de montrer qu'équité implique rationalité (à condition que le protocole satisfasse certaines autres propriétés usuelles), mais que l'inverse n'est pas vrai en général.

Nous montrons comment le modèle formel peut être utilisé pour la vérification rigoureuse des protocoles existants en analysant deux protocoles d'échange, et en montrant formellement qu'ils satisfont la définition de l'échange rationnel. Nous présentons également une application originale d'échange rationnel: Nous montrons comment le concept de la rationalité peut être employé à améliorer une classe de schémas de micro-paiements, en termes l'équité, sans perte substantielle d'efficacité.

Finalement, dans la dernière partie de cette thèse, nous étendons le concept de l'échange rationnel et décrivons comment des idées similaires peuvent être employées pour stimuler la coopération dans les réseaux ad hoc auto-organisateurs. Plus précisément, nous proposons une approche originale pour inciter les nœuds à relayer les paquets dans les réseaux ad hoc. Comme pour les protocoles d'échange rationnel, notre conception ne garantit pas qu'un nœud accepte toujours de relayer les paquets pour les autres, mais il nous assure au moins que ce nœud ne gagne rien en agissant de la sorte. Nous étudions la solution proposée analytiquement et au moyen de simulations.

To Zita,
for her patience and love

Acknowledgements

I am grateful to Professor Jean-Pierre Hubaux, my academic supervisor, for giving me the possibility to come to EPFL and to conduct research in his laboratory. Jean-Pierre has been an excellent "boss" who always defended my interest in various situations. I also want to thank him for giving me a wide range of freedom in my research, and for maintaining the level of my self-esteem.

I am thankful to the members of my thesis committee, Dr. Asokan, Professor Refik Molva, and Professor Michael Reiter, as well as to the president of the committee, Professor André Schiper for the time and effort that they have invested in judging the contents of my thesis. Many thanks to Jean-Pierre Hubaux, Srdjan Čapkun, Naouel Ben Salem, and Uwe Wilhelm, who have spent a considerable amount of their precious time reading this text at various stages and suggesting improvements.

I am grateful to all the people with whom I worked together on papers and projects, in particular, Uwe Wilhelm, Sebastian Staamann, Jean-Pierre Hubaux, Srdjan Čapkun, István Vajda, Edwin Wiedmer, and Sean Boran, for all the interesting discussions that we had together. A special thanks goes to Uwe Wilhelm, who taught me the rigor of technical writing, and many other things. I am also thankful to the anonymous reviewers of my papers, who provided me with useful comments and helped to improve the quality of my papers, and so, indirectly, the quality of this thesis.

I want to thank all my past and present colleagues in the lab for making work a pleasure, especially Shawn Koppenhöfer, Constant Gbaguidi, and Srdjan Čapkun, with whom I shared my office. Special thanks to Shawn Koppenhöfer, from whom I learned many things about preparing a presentation, and to Olivier Verscheure and Srdjan Čapkun, whose enthusiasm has created a nice atmosphere in the lab. I am grateful to Bruno Dufresne, Jean-Pierre Dupertuis, and Marc-André Lüthi for keeping the computing infrastructure up and running, and to Danielle Alvarez, Angela Devenoge, and Holly Cogliati for taking care of all the administrative problems. I also want to thank to Naouel Ben Salem for translating the abstract of this thesis into French, to Danielle Alvarez for her help in organizing my thesis exam, and to Holly Cogliati for improving my English.

Most of all, I want to thank Zita for her love and patience. She made my private life joyful, which provided a solid background in my work. I am grateful to her for the confidence that she has put in me, for her encouragement, and for countless other things.

Contents

Introduction

Advances in communication technology, in particular, the Internet explosion and the widespread deployment of digital cellular systems provide opportunities for a vast range of new services for industry and commerce. Examples range from remote control of industrial processes to commercial transactions carried out over public communication networks. However, the electronic infrastructure that supports these services is known to be susceptible to misuse. Therefore, a crucial requirement for these services to be successful is security.

This thesis is concerned with two security mechanisms: authenticated key transport and rational exchange protocols. These mechanisms are potential building blocks in the security architecture of a range of different services. Authenticated key transport protocols are used to establish a shared secret key between two (or more) entities in a dynamic manner. The established key can then be used by the entities to achieve various security objectives. Intuitively, an authenticated key transport protocol can be thought of as a mechanism to build a secure (logical) channel between the entities, which protects their communication against eavesdropping and alteration by an outside attacker.

Entities involved in a commercial transaction often distrust each other. Protecting an honest entity from the other entities involved in the transaction is as important as protecting the entities from an outside attacker. A typical situation where this need arises is when two remote parties have to exchange some digital items via a communication network. Examples include electronic contract signing, certified e-mail, and purchase of network delivered services. An inherent problem in these applications is that a misbehaving party may bring the other, correctly behaving party in a disadvantageous situation. For instance, a service provider may deny service provision after receiving payment from a user. This may discourage the parties and hinder otherwise desired transactions.

The best known approach to solve this problem is to use a fair exchange protocol, which guarantees for a correctly behaving party that it cannot suffer any disadvantages. This thesis is not specifically concerned with this approach. It is about an alternative approach, called rational exchange. Roughly, a rational exchange protocol ensures that a misbehaving party cannot gain any advantages, therefore, misbehavior is not interesting, and should happen only rarely.

Scope of research

Designing good security mechanisms that fulfill their purpose is a difficult exercise. There are countless examples for security mechanisms proposed in the literature that were found to be flawed some time after their publication. This is true for cryptographic algorithms (e.g., encryption schemes) as well as cryptographic protocols (e.g., key transport protocols).

It is widely accepted that informal reasoning does not provide sufficient evidence of the correctness of even simple security mechanisms. This has led many researchers to construct formal models, in which security mechanisms can be represented and their properties can be rigorously investigated. In this thesis, we follow this approach and focus on the construction of formal models for authenticated key transport and rational exchange protocols.

Regarding authenticated key transport protocols, numerous formal models have already been proposed in the literature (see Section 1.5). Most of these models are tailored for the formal verification of existing protocols. Our goal is slightly different: We aim at a formal model that directly supports the construction of robust key transport protocols. More precisely, we want to develop a model, and on top of that, a protocol construction tool that enables the design of protocols that are guaranteed to be correct in the underlying model[1]. The advantage of such a tool is quite obvious: One can save numerous rounds of verification and redesign steps by constructing robust protocols in the first place.

As opposed to key transport protocols, there exist only a few formal models developed for exchange protocols (see Section 4.7), and no formal model for rational exchange in particular. Consequently, there exists no rigorous study of what exactly rational exchange protocols achieve and how they are related to fair exchange. Therefore, our goal is to come up with a formal model for exchange protocols, in which both rationality and fairness can be defined, and the relationship between the two concepts can be investigated. We expect that a formal definition of rational exchange and its comparison to fair exchange will help us to better understand the concept. In addition, the formal model can serve as the basis for rigorous verification of existing rational exchange protocols.

Outline of the thesis

Part I: Authenticated key transport protocols

In Chapter 1, we propose a formal model for key transport protocols, and based on that model, a systematic protocol construction approach. Our model is based on a logic of belief (similar to the well-known BAN logic [BAN90a]). The protocol construction approach is based on a set of synthesis rules that we obtain by reversing some implications that can easily be derived in the logic. The synthesis rules are used to construct a protocol and a set of assumptions starting from a set of goals in a systematic way. The main advantage of this approach is that the resulting protocol is guaranteed to be correct in the sense that all the specified goals can be derived from the protocol and the assumptions using the logic. Another advantage is that all the assumptions, upon which the correctness of the protocol depends, are made explicit, so someone who reviews the design can easily verify if they are acceptable in a given application or not.

The main notion of our logic is the *channel*. Channels are abstractions that we use to model, in a uniform manner, the multitude of ways in which principals can send messages to each other in a protocol. Channels can represent physical links, such as a wire or a network connection, but we mainly use them to represent cryptographic primitives, such as encryption with a given key or digital signature of a given user. As a consequence, our logic does not have any construct that explicitly deals with cryptographic operations; instead, we use only

[1] When we started to work on this problem in 1997 and published our first results [BSW98] in 1998, no such protocol construction tool existed that we were aware of. Since then, some other researchers have also stared to work on the same problem (see e.g., [PS00]).

channels with various access properties. This allows us to reason about protocols at a high abstraction level, without being concerned with actual implementation details. This feature of our model is particularly useful when designing protocols from scratch.

In Chapter 2, we illustrate the application of our approach through an example. We analyze an authenticated key transport protocol that was proposed in the literature for the so called global mobility network. Using our logic, we identify several weaknesses in the protocol, and show how these can be exploited by various, previously unknown attacks. Then, we redesign the protocol using our systematic protocol construction approach. The new protocol resists against the previously presented attacks, and it is simpler than the original one.

Part II: Rational exchange protocols

In Chapter 3, we introduce the concept of rational exchange in an informal way. We present an original rational payment protocol as an example, and show how the main idea of rationality can be used to improve a family of micropayment protocols by making cheating uninteresting at virtually no cost (loss in efficiency). Our goal with these examples is to illustrate the usefulness of the concept of rational exchange, and to prepare the grounds for the formal treatment.

In Chapter 4, we present an original formal model of exchange protocols, which is based on game theory. In this model, an exchange protocol is represented as a set of strategies in a game that is played by the protocol parties and the network that they use to communicate with each other. Then, we give formal definitions for various properties of exchange protocols, including rationality and fairness. Most importantly, rationality is defined in terms of a Nash equilibrium in the protocol game. The model and the formal definitions allow us to rigorously study the relationship between rationality and fairness. We prove that fairness implies rationality (given that the protocol satisfies some further usual properties), but the reverse is not true in general. This means that fair exchange protocols provide stronger guarantees than rational exchange protocols. Thus, one expects that rational exchange protocols are less complex than fair exchange protocols. This means that rational exchange can be viewed as a trade-off between true fairness and complexity, and as such, it may provide interesting solutions to the exchange problem in certain applications.

In Chapter 5, we illustrate how the formal model can be used for rigorous verification of existing rational exchange protocols: We analyze two exchange protocols, and formally prove that they satisfy our definition of rational exchange. The first protocol that we prove to be rational is the example payment protocol of Chapter 3; the second one is a protocol proposed by Syverson in [Syv98].

Part III: Incentives to cooperate in ad hoc networks

Finally, in Chapter 6, we extend the concept of rational exchange, and describe how similar ideas can be used to stimulate cooperation in self-organizing ad hoc networks. The functioning of such networks heavily depends on the cooperative behavior of the nodes. In military and rescue applications, the nodes are naturally motivated to cooperate, because they belong to a single authority and have a common goal. However, in civilian applications, which are expected in the future, the nodes have no good reason to cooperate. Indeed, in order to save their own resources (e.g., battery), they tend to be "selfish".

We focus on the stimulation of packet forwarding, which is a fundamental service that

the nodes should provide to each other in ad hoc networks. We propose an original approach that is based on a trusted and tamper resistant hardware module (called security module) in each node and cryptographic protection of packets. We present a protocol that requires that each packet (generated as well as received for forwarding) be passed to the security module. The security module maintains a counter, which is decreased when the node wants to send a packet as originator, and increased when the node forwards a packet. The counter must remain positive, which means that if the node wants to send its own packets, then it must forward packets for the benefit of others. Like in rational exchange protocols, a node can still misbehave (e.g., it can deny packet forwarding, or bypass the security module), however, our design ensures that it cannot gain any advantages by doing so. Therefore, misbehavior is uninteresting, and should happen only rarely. We analyze the proposed mechanism analytically and by means of simulations.

Part I

Authenticated key transport protocols

Chapter 1

A logic of channels

1.1 Introduction

In large scale, open systems (e.g., in the Internet), it is necessary to have protocols by which a pair of principals (users, computers, services) that originally do not share any secrets can establish a shared secret in a dynamic manner. Such protocols are usually referred to as *key establishment protocols*, and the established shared secret is called *session key* [MvOV97]. Once two principals have established a session key, they can use it to authenticate each other and to cryptographically protect their further communication.

In general, it is desired that each party in a key establishment protocol can determine the identity of the other party or parties that could have access to the established session key. In other words, each party should know with whom exactly the key was established. This is particularly important, if the session key is intended to be used for entity authentication or data origin authentication. This property is sometimes called *(implicit) key authentication*. If a key establishment protocol provides (implicit) key authentication, then it is called *authenticated key establishment protocol*.

Key establishment can be subdivided into *key agreement* and *key transport*. Key agreement is a key establishment technique where a session key is derived by two parties as a function of information contributed by, or associated with, each of them, such that ideally no party can predetermine the resulting value. In key transport, one party creates or otherwise obtains a session key, and securely transfers it to the other party. In this thesis, we are exclusively concerned with key transport protocols.

Despite their apparent simplicity, the design of authenticated key transport protocols is surprisingly error prone. This is illustrated by the fact that many protocols have been proposed that were found to be flawed later on. The most well-known example is probably the Needham-Schroeder symmetric key protocol [NS78], which was found by Denning and Sacco to allow an intruder to pass an old, possibly compromised session key as a new one to a legitimate party [DS81]. Many other examples for flawed protocols can be found in the literature (see e.g., [BAN90a, Low95, AN96, Aba97, CJ97]).

It is important to note that most of the flaws are related to weaknesses in the structure of the protocol, and independent of the underlying cryptographic primitives. This means that protocols can fail even if the cryptographic algorithms used within them are sound. In fact, most of the flaws leave the protocol unprotected from some form of *replay attack* [Syv94]. An important consequence of this observation is that it indeed makes sense to focus attention on

the structure of the protocols, and abstract away the details of the cryptographic primitives.

In order to support the design of robust key transport protocols, many researchers have proposed verification methods to be used to discover flaws in protocols. Since flaws are sometimes very subtle, they can easily be overlooked when reasoning about the protocol in an informal way. For this reason, most of the proposed verification methods are based on formal techniques, such as logics and algebrae (see Subsection 1.5.3 for an overview). Although formal techniques in general are often criticized to be applicable only to "toy examples", the techniques proposed for the formal verification of key transport protocols proved to be very useful by discovering many previously unknown flaws in existing protocols.

While they certainly have their merits, formal protocol verification techniques also have a major drawback: they are of little help in re-designing a flawed protocol or designing a new one from scratch. Even if the formal verification of a protocol suggests a possible correction of a flaw, one will certainly want to verify the modified protocol again, because the modification may introduce new flaws. This may lead to many rounds of verification and re-design steps. It is therefore desirable to have methods that allow the construction of robust protocols in the first place.

We address this problem by proposing a systematic approach to construct key transport protocols that is based on a logic of belief similar to the well-known BAN logic [BAN90b, BAN90a]. Our main idea is to reverse some implications that can be derived from the axioms of the logic and to turn them into *synthesis rules*. The synthesis rules can be used by the protocol designer to derive a protocol Π and a set of assumptions Λ starting from a set of goals Γ. Our approach has the advantage that it results in correct protocols in the sense that all the goals in Γ can be derived from Π and Λ using the logic. Another important advantage is that all the assumptions upon which the correctness of the protocol depends are made explicit, so someone who reviews the design can verify if they are acceptable in a given application.

We hasten to note that we do not claim a fully automated protocol synthesis method. In particular, the synthesis rules do not replace human creativity. They simply *help* the designer to construct protocols in a systematic way. We will illustrate this in detail through an example in Chapter 2. We do not claim either that the set of synthesis rules presented in this thesis is complete; one can derive new rules from our logic if this is deemed necessary in a particular application.

A fully automated protocol construction tool, APG, has recently been proposed in [PS00]. However, APG is based on a different approach: Instead of systematic construction of the protocol, a large number of protocols are generated in a random manner, and a fast automatic protocol verification tool, Athena [Son99], is used to filter the correct ones. At the time we worked out our approach (in 1998), neither APG nor any other protocol construction tool existed that we were aware of.

An approach similar to our synthesis rules is described in [AS97], where a weakest precondition calculus is used to generate the least restrictive set of required assumptions from the goals of the protocol and the protocol itself. Although the authors discuss the possibility of using their approach for protocol derivation, they do not actually use their results to construct protocols as we do. Instead, they use the weakest precondition calculus for protocol verification.

As we mentioned before, our approach is based on a logic of belief. While this logic has some similarities to the well-known BAN logic, it also differs from it in many ways. The

most striking difference is that our logic does not have any construct that explicitly deals with cryptographic primitives, such as encryption or digital signature. Instead, we model these primitives in a uniform manner as *channels* with various access properties. In addition to cryptographic primitives, channels can also represent physical links, such as a wire or a network connection between two devices, but we will not use this feature of our model in this thesis.

The use of channels makes our logic simple and compact. It also allows us to reason about protocols at a rather high abstraction level without being concerned with the actual cryptographic implementation details. We believe that this feature is useful when designing protocols, because it allows us to identify the required properties of a channel first, and decide about its implementation afterwards (possibly taking into consideration additional design criteria). Note also that similar kind of abstractions are considered to be useful in software engineering, and in particular, in distributed computing (e.g., socket, remote procedure call, etc.).

On the other hand, we admit that reasoning about the protocol at a high abstraction level has no real advantage in protocol verification. Indeed, by abstracting away some details, we reduce the class of attacks that our logic can discover. In particular, flaws that are related to the incorrect usage of cryptographic primitives remain hidden. We do not consider this to be a disadvantage of our approach, since our primary goal is not protocol verification.

We are not the first who use channel abstractions to represent cryptographic primitives. π-calculus and join-calculus channel primitives are used in a similar manner in [AG98] and [AFG98], respectively. However, those approaches are not based on a logic, but on process algebrae. In [BM94b, BM94a], channel abstractions are used to design and analyze key transport protocols, but again not within a logic. A logic that uses channels is introduced in [LABW92, WABL94] although in the somewhat different context of distributed operating systems. In addition, unlike our logic, that logic does not distinguish between past and present, and therefore, it cannot be used to discover vulnerabilities to replay attacks. On the other hand, the logic described in [LABW92, WABL94] deals with some other issues (e.g., roles and delegation), which are not addressed in our logic. Finally, we must also mention [ABKL92], in which a logic very similar to ours is described. In fact, that logic provided some inspiration in the development of our logic. The difference is that in [ABKL92], both channels and cryptographic primitives are used, and channels model only physical links. By abstracting away cryptographic primitives and using the channel abstraction in a more uniform way, our logic is more elegant and simpler.

Since our synthesis rules are based on our logic, they also involve channel formulae. As a consequence, the abstract protocol obtained in the synthesis process is described in terms of channels. The replacement of the channels with concrete cryptographic primitives (or physical links) (i.e., the implementation or refinement of the abstract protocol) is only partially supported by our method. Namely, we do not provide formal translation rules from channels to cryptographic primitives, but a set of required properties for the channels (in terms of beliefs held by various principals) are derived as assumptions during the synthesis. We believe that in most of the cases, this gives sufficient guidance to choose the appropriate implementation.

A formal treatment of secure implementation of channel abstractions is described in [AFG98] in the context of the join-calculus and the sjoin-calculus. In [Rue91] and [TB95], production rules are given for some types of channels that describe how to establish them. A similar approach is presented in [MS94]. In addition, a detailed treatment of channels that are implemented by encryption can be found in [LABW92].

Although replacing the channels with the appropriate cryptographic primitives is quite straightforward, we cannot exclude the possibility of introducing some flaws in the protocol during channel replacement. For this reason, we require that once the channels are replaced with their implementations, the protocol is verified with another tool in which cryptographic primitives are explicitly represented.

In other words, our approach can be viewed as part of a more general design principle known as *top-down* design. According to this design principle, the protocol designer should first use a relatively abstract model to specify, construct, and verify the protocol. If the protocol is correct at that level, then the designer can switch to a less abstract (more detailed) model, which refines the previous one. This means that the protocol should be translated into the notation of the less abstract model, and analyzed with the tools available at that level. Through the repeated execution of this process, a detailed description or even the actual protocol code can be produced.

The top-down approach bridges the abstraction gap between the informal protocol specification (e.g., plain English) and the final implementation (e.g., C source code) by introducing several smaller steps. We assume that it is easier at each level to produce a correct implementation of the previous level than to produce a final implementation of the protocol directly from the informal specification. It also makes it easier to change the implementation later on (which might be necessary because of modifications to the original design assumptions), because there is a chance that the protocol does not have to be completely redesigned, but it might be sufficient to propagate the changes through the different levels.

We believe that our logic of channels and the corresponding synthesis rules provide an abstract modeling tool that can be used at the beginning of the design process. However, according to the top-down approach, more detailed models and the corresponding analysis tools should also be used in subsequent phases of the design.

The rest of this chapter is organized as follows: In Section 1.2, we describe our logic. In section 1.3, we give some examples for channels. Based on the logic, in Section 1.4, we construct protocol synthesis rules. Finally, in Section 1.5, we give an overview of the formal models proposed for key transport protocols.

1.2 The logic

Our logic is a modal logic [HC96] of belief, similar to the well-known BAN logic [BAN90b, BAN90a]. It consists of a language, a set of axioms, and a few inference rules. The language is used to describe the protocol, its goals, and the assumptions made. The inference rules are used to derive statements about the protocol (more precisely, about the state of the protocol participants) from the axioms, the assumptions, and the protocol itself. The novelty of our logic is the use of channels. We commence the presentation of the logic by the introduction of this novel notion.

1.2.1 Channels

A channel is an abstraction that we mainly use to represent various cryptographic primitives, such as encryption and digital signature, in a uniform way. The main advantage of using channels is that we can reason about the protocol at a rather high abstraction level. We believe that this is a useful feature when designing protocols (although perhaps less interesting when verifying them), because it allows us to determine the required properties of a channel

first, and to decide about its implementation afterwards (possibly taking into consideration additional design criteria).

We could characterize a channel C with two sets: the *reader set* $r(C)$ and the *writer set* $w(C)$, where $r(C)$ contains those principals that can read from C, and $w(C)$ contains those principals that can write in C. However, instead of the writer set, we will use another notion: the *source set* $s(C)$, which contains those principals that could be the source of a message that arrived on C. Clearly, $s(C) \subseteq w(C)$, and intuitively, $w(C) \setminus s(C)$ contains those principals that could write in C but that are trusted not to do so. There is a good reason for focusing on the source set instead of the writer set, which we now explain.

Suppose that a principal P received a message X on a channel C. Ultimately, P's goal is to determine who could have sent X. It is, therefore, more important for P to know $s(C)$, than to know $w(C)$. Suppose, for instance, that C is implemented by encryption with a key that has been generated and distributed by a trusted principal S (e.g., a key server). This means that S may be able to write in C, but it is trusted for never doing so. Therefore, when P receives a message on C, it should not attribute the message to S.

The following example may shed more light on the difference between the writer set and the source set of a channel. Let us consider a slightly modified version of the Wide-mouthed-frog protocol described in [BAN90a]:

Modified Wide-mouthed-frog protocol		
(msg1)	$A \rightarrow S:$	$A,\ B$
(msg2)	$S \rightarrow A:$	$\{T_S,\ B,\ K_{AB}\}_{K_{AS}}$
(msg3)	$S \rightarrow B:$	$\{T_S,\ A,\ K_{AB}\}_{K_{BS}}$

The protocol is described as follows: When A wants to establish a session key with B, it sends a request to the key server S. The request simply contains the name of A and B in clear (msg1). When S receives the request, it generates a session key K_{AB} and sends it to A and B together with a timestamp T_S and the name of the other party. The message (msg2) intended for A is encrypted with the key K_{AS}, which is a long-term key shared by A and S. Similarly, the message (msg3) intended for B is encrypted with K_{BS}, which is a long-term key shared by B and S. When A and B receive their messages, they decrypt them and verify the timestamp. If the messages are timely, then both A and B accept K_{AB} as a session key to be used with the other party.

We want to model the encryption with K_{BS} in (msg3) as a channel C. It is clear that $r(C) = w(C) = \{B, S\}$, since both B and S *can* encrypt and decrypt with K_{BS}. However, notice that only the server encrypts messages with the long-term keys. In other words, when B receives (msg3), it knows that (1) only S and B itself can generate such a message, and (2) B itself actually never generates such a message. Therefore, B can safely conclude that only S can be the source of (msg3). This means that $s(C) = \{S\} \neq w(C)$.

Let us consider now the original Wide-mouthed-frog protocol:

Wide-mouthed-frog protocol		
(msg1)	$A \rightarrow S:$	$A,\ \{T_A,\ B,\ K_{AB}\}_{K_{AS}}$
(msg2)	$S \rightarrow B:$	$\{T_S,\ A,\ K_{AB}\}_{K_{BS}}$

In this protocol, the session key K_{AB} is not generated by the server, but the initiator party A. The server is used only to relay the key received from A to B. Otherwise, the messages are almost identical to those of the previous protocol. If we want to model again the encryption with K_{BS} in (msg2) as a channel C, then we have that $r(C) = w(C) = \{B, S\}$. In addition, $s(C) = \{B, S\}$ too, because both B and S use K_{BS} to encrypt messages for each other (B uses K_{BS} to send messages of the first type to S when it wants to setup a session key with someone), and the messages from S to B are indistinguishable from the messages from B to S. This means that when B receives (msg2) in the Wide-mouthed-frog protocol, it cannot determine the source of the message: it could have been generated by B itself, sent to S, and intercepted and reflected back to B by an attacker (within the validity interval of the timestamp). Of course, the attacker cannot find out the session key, or cannot coerce B to accept a compromised session key, and so the attack (if we can call this an attack at all) is not a very useful one. Our goal was simply to give an insight into the difference between the source set and the writer set of a channel.

As for the reader set of a channel C, it will represent those principals that can recognize the usage of C. This means that when a principal in $r(C)$ receives a message on C, it is able to determine that it has received a message on C (and not on another channel). We make this assumption to keep our model simple. Real implementations of channels can support this feature, for instance, by putting enough redundancy into the messages and using references to channels (e.g., key identifiers) as hints.

Another important characteristic of a channel is *timeliness*. A channel C is said to be timely (denoted by $\natural(C)$) if whenever a message arrives via C, we can conclude that the message has been recently sent by someone in $s(C)$. Timely channels are typically implemented by using freshly established session keys or including freshly generated nonces in cryptographically protected messages.

Now we turn our attention to the logic itself. Our presentation will follow the style of the Abadi-Tuttle logic described in [AT91].

1.2.2 Language

Like in [BAN90a], we want that in our model, principals can send logical formulae to each other in messages. In other words, we want to allow *idealized* messages. A message that contains logical formulae is called idealized, because, unlike a pure bit string representation, it explicitly describes the intended meaning of the message, and does not leave any ambiguities about its interpretation.

The need for idealization in BAN logic was criticized as being the source of potential errors in protocol verification (see e.g., [BM93]). This may indeed be true. The problem is that the person who verifies the protocol and translates the original protocol messages into idealized messages may not be the same person who designed the protocol, and so, she may misinterpret some messages. While idealization may be a drawback in protocol verification, we believe that it is extremely useful in protocol design, since the designer – at least in the first stage of the design – should not be concerned with bit strings, but should focus on the meaning of the messages, which can conveniently be expressed with logical formulae. Of course, the idealized messages then need to be translated into bit strings, but fortunately, this translation appears to be easier than the reverse operation.

In this subsection, we define a language $\mathcal{M}$ in which idealized messages can be written. We distinguish a sublanguage $\mathcal{F}$ of $\mathcal{M}$, which is called the (sub)language of formulae. Intuitively,

$\mathcal{F}$ contains those sentences of $\mathcal{M}$ to which we can assign a truth value, while $\mathcal{M} \setminus \mathcal{F}$ contains sentences, to which it does not make sense to assign a truth value (e.g., principal names). We will define $\mathcal{M}$ and $\mathcal{F}$ by mutual induction. However, before doing so, we must introduce some more definitions:

- *Primitive terms*: Let $\mathcal{T}$ be a set of primitive terms. Primitive terms are constant symbols that represent principal names, channels, keys, and things like nonces and other data items that might be used in protocols.
- *Principal lists*: Let $\mathcal{P}'$ be the language of principal lists defined over $\mathcal{T}$ by induction as follows:
  - P is a sentence in $\mathcal{P}'$ if $P \in \mathcal{T}$ is a principal name;
  - π', P is a sentence in $\mathcal{P}'$ if π' is a sentence in $\mathcal{P}'$ and $P \in \mathcal{T}$ is a principal name such that π' does not contain P.
- *Principal sets*: Let $\mathcal{P}$ be the language of principal sets defined over $\mathcal{P}'$ as follows: $\{\pi'\}$ is a sentence in $\mathcal{P}$ if π' is a sentence in $\mathcal{P}'$. This just means, that we will write a set of principals as a comma separated list of principal names in curly braces.

Now, we are ready to define $\mathcal{M}$ and $\mathcal{F}$. $\mathcal{M}$ is the smallest language that satisfies the following conditions:

- X is a sentence in $\mathcal{M}$ if $X \in \mathcal{T}$ (i.e., X is a primitive term);
- $X; Y$ is a sentence in $\mathcal{M}$ if X and Y are sentences in $\mathcal{M}$;
- $C(X)$ is a sentence in $\mathcal{M}$ if X is a sentence in $\mathcal{M}$ and $C \in \mathcal{T}$ is a channel;
- ϕ is a sentence in $\mathcal{M}$ if ϕ is a sentence in $\mathcal{F}$,

and $\mathcal{F}$ is the smallest language that satisfies the following conditions:

- $(r(C) = \pi)$, $(P \in r(C))$, $(s(C) = \pi)$, and $(P \in s(C))$ are sentences in $\mathcal{F}$ if $C \in \mathcal{T}$ is a channel, $P \in \mathcal{T}$ is a principal name, and π is a sentence in $\mathcal{P}$;
- $(K \sim \pi)$ is a sentence in $\mathcal{F}$ if $K \in \mathcal{T}$ is a key and π is a sentence in $\mathcal{P}$;
- $(P \models\!\!\models \phi)$ is a sentence in $\mathcal{F}$ if ϕ is a sentence in $\mathcal{F}$ and $P \in \mathcal{T}$ is a principal name;
- $(P \lhd X)$, $(P \mid\!\sim X)$, and $(P \mid\!\mid\!\sim X)$ are sentences in $\mathcal{F}$ if X is a sentence in $\mathcal{M}$ and $P \in \mathcal{T}$ is a principal name;
- $\sharp(X)$ is a sentence in $\mathcal{F}$ if X is a sentence in $\mathcal{M}$;
- $\natural(C)$ is a sentence in $\mathcal{F}$ if $C \in \mathcal{T}$ is a channel;
- finally, $\neg\phi$, $(\phi \wedge \psi)$, $(\phi \vee \psi)$, $(\phi \Rightarrow \psi)$, and $(\phi \Leftrightarrow \psi)$ are sentences in $\mathcal{F}$ if ϕ and ψ are sentences in $\mathcal{F}$.

The intuitive meaning of the main constructs of the language is the following:

- $X; Y$ means a compound message constructed from two (sub)messages X and Y by concatenation.

- $C(X)$ denotes message X in channel C.
- $(r(C) = \pi)$ represents the statement that the reader set of channel C is π. $(P \in r(C))$ represents the statement that principal P is in the reader set of channel C. $(s(C) = \pi)$ and $(P \in s(C))$ represent similar statements for the source set of channel C.
- $(K \sim \pi)$ represents the statement that the key K is a session key for the set π of principals. One may ask the question: Why do we have statements that involve keys, when the purpose of introducing channels was to eliminate them? In principle, we could replace a statement about a session key K with statements about channels, but the problem is that the protocol usually does not specify what kind of channels are supposed to be created from K. In other words, how the session key is used once it has been established depends on the application, and it is out of the scope of the protocol itself. Therefore, we need a formula that describes the general statement that a key is a session key for a set of principals without referring to any channels that may be build from K later.
- $(P \models \phi)$ represents the statement that principal P believes that the formula ϕ is true. This does not mean that ϕ is really true, but P acts as though.
- $(P \lhd X)$ means that principal P sees message X. This is possible, for instance, if someone has sent X or a message that contains X via a channel from which P can read.
- $(P \lhd C(X))$ means that principal P received message X on channel C. However, if P cannot read from C, then P cannot recognize that this is message X in channel C. Nevertheless, if C is not a physical channel, then we allow P to store $C(X)$ and to repeat it on another channel.
- $(P \mathrel{|\!\sim} X)$ means that P once said X. In other words, P at some time sent a message that contained X. It is not known exactly when the message was sent.
- $(P \mathrel{|\!|\!\sim} X)$ means that P has recently said X. More precisely, we distinguish two epochs: present and past. Every event that happened after the start of the current protocol run will be considered as an event happened in the present. $(P \mathrel{|\!|\!\sim} X)$ means that P uttered X in the present epoch (i.e., in the current protocol run).
- $\sharp(X)$ means that X is fresh (i.e., it has never been said before the present epoch). This is usually true for messages that contain nonces.
- $\natural(C)$ means that channel C is timely. As we mentioned before, if a channel is timely, then whenever a message arrives on it, we can conclude that it has been said recently (in the present epoch).
- $\neg\phi$, $(\phi \wedge \psi)$, $(\phi \vee \psi)$, $(\phi \Rightarrow \psi)$, and $(\phi \Leftrightarrow \psi)$ represent the usual connectives (negation, 'and' operation, 'or' operation, implication, and equivalence) from propositional logic.

We can demonstrate the expressive power of our language by showing how certain aspects of trust can be represented in it. For instance, the belief of principal P in the *honesty* of principal Q is described by the following formula:

$$P \models ((Q \mathrel{|\!|\!\sim} \phi) \Rightarrow (Q \models \phi))$$

This means that P believes that if Q has recently said ϕ, then Q believes ϕ. In other words, P believes that "Q says only what it believes". Note that P does not necessarily believe that what Q says is true; it only believes that Q believes it is true. However, it is possible that P does not share all of Q's beliefs.

Another aspect of trust, which we call *competency*, can be expressed by the following formula:

$$P \models ((Q \models \phi) \Rightarrow \phi)$$

This means that P believes that if Q believes ϕ, then ϕ is true. If we combine these two formulae, then we obtain the following:

$$P \models ((Q \mid\sim \phi) \Rightarrow \phi)$$

This means that P believes that if Q says ϕ, then ϕ is true. In other words, P believes "what Q says is true".

In fact, in the above formulae ϕ is meant to be universally quantified, and we could make this explicit by writing, for instance:

$$P \models \forall \phi : ((Q \mid\sim \phi) \Rightarrow \phi)$$

While in principle, it is necessary to make the quantifiers explicit in order to avoid ambiguities, in practice, this need arises only in complex statements. In the simple statements that we will use in this thesis, it will be clear from the context if we mean universal quantification. For this reason, we leave quantifiers implicit.

We can restrict the scope of the above beliefs by giving more details about ϕ. P can believe, for instance, that Q is competent only in attributing messages to R. We can express this with the following formula:

$$P \models ((Q \models (R \mid\sim X)) \Rightarrow (R \mid\sim X))$$

At the same time, P may not believe, for instance, that Q is competent in recognizing freshness, which means that

$$P \models ((Q \models \sharp(X)) \Rightarrow \sharp(X))$$

does not necessarily hold.

1.2.3 Axioms

The axioms are the heart of the logic. They capture the basic properties of channels, communication, time, and the concept of belief at an abstract level. In fact, the set of axioms describes a model, in which we can reason about the properties of key transport protocols.

The axioms of our logic are all the instances of tautologies of propositional logic, and the following axiom schemas:

(AX1) $P \lhd C(X) \ \wedge \ P \in r(C) \ \Rightarrow \ P \models (P \lhd C(X))$

(AX2) $P \lhd C(X) \ \wedge \ P \in r(C) \ \Rightarrow \ P \lhd X$

(AX3) $P \lhd X;Y \;\Rightarrow\; P \lhd X$ and $P \lhd X;Y \;\Rightarrow\; P \lhd Y$

(AX4) $P \lhd C(X) \;\wedge\; s(C) = \mathcal{S} \;\Rightarrow\; \bigvee_{Q \in \mathcal{S}} (Q \mid\sim X)$

(AX5) $P \lhd C(X) \;\wedge\; s(C) = \mathcal{S} \;\wedge\; \natural(C) \;\Rightarrow\; \bigvee_{Q \in \mathcal{S}} (Q \mid\!\sim X)$

(AX6) $Q \mid\sim X;Y \;\Rightarrow\; Q \mid\sim X$ and $Q \mid\sim X;Y \;\Rightarrow\; Q \mid\sim Y$

(AX7) $Q \mid\!\sim X;Y \;\Rightarrow\; Q \mid\!\sim X$ and $Q \mid\!\sim X;Y \;\Rightarrow\; Q \mid\!\sim Y$

(AX8) $Q \mid\sim X \;\wedge\; \sharp(X) \;\Rightarrow\; Q \mid\!\sim X$

(AX9) $\sharp(X) \;\Rightarrow\; \sharp(X;Y)$ and $\sharp(X) \;\Rightarrow\; \sharp(Y;X)$

(AX10) $P \models \phi \;\wedge\; P \models (\phi \Rightarrow \psi) \;\Rightarrow\; P \models \psi$

(AX11) $P \models \phi \;\Rightarrow\; P \models (P \models \phi)$

(AX12) $P \models \phi \;\wedge\; P \models \psi \;\Leftrightarrow\; P \models (\phi \;\wedge\; \psi)$

The meaning of most of the axioms should be clear; nevertheless, we provide explanations hereafter. Axioms (AX1–AX3) are concerned with *seeing*. (AX1) and (AX2) states that if a principal P receives a message X via a channel C, and P can read from C, then P recognizes that it received X via C, and it sees X. (AX3) says that if a principal sees a compound message, then it sees also parts of the message.

Axioms (AX4–AX5) are concerned with *source association*. (AX4) states that if a principal P sees $C(X)$, where the source set of C is $\mathcal{S}$, then someone from $\mathcal{S}$ must have said X. (AX5) states that if, in addition, C is timely, then someone from $\mathcal{S}$ must have recently said X.

Axioms (AX6–AX7) are concerned with *saying*. (AX6) states that if a principal said a (compound) message, then it said parts of the message as well. Similarly, (AX7) states that if a principal has recently said a (compound) message, then it has recently said parts of the message as well.

Axioms (AX8–AX9) deal with message *freshness*. (AX8) states that a fresh message must have been said recently, while (AX9) states that if part of a compound message is fresh, then the whole message is fresh.

Finally, Axioms (AX10–AX12) are concerned with the properties of the *belief* operator. (AX10) states that a principal believes all logical consequences of its beliefs. (AX11) states that a principal can tell what it believes. (AX12) states that believing ϕ and believing ψ is equivalent to believing $\phi \;\wedge\; \psi$.

1.2.4 Inference rules

The inference rules describe how to derive new formulae from already available formulae such as axioms, assumptions, or previously derived formulae. The main inference rule of our logic is *modus ponens*:

$$\frac{\phi,\; \phi \Rightarrow \psi}{\psi} \quad \text{(MP)}$$

This should be interpreted in the usual way: If both ϕ and $\phi \Rightarrow \psi$ are available, then we also have ψ.

In addition to the rule (MP), we have another inference rule that we call *necessitation*:

$$\frac{\vdash \phi}{\vdash P \models\!\models \phi} \quad \text{(Nec)}$$

This rule should be interpreted in the following way: If ϕ is a *theorem* (denoted by the $\vdash$ symbol in front of ϕ), then we have $P \models\!\models \phi$, and we consider it as a theorem as well. A formula ϕ is a theorem if there exists a sequence of formulae $\phi_1, \phi_2, \ldots, \phi_n$ such that $\phi_n = \phi$ and for each ϕ_i of the sequence at least one of the following holds:

- ϕ_i is an axiom of the logic;
- ϕ_i can be obtained from some ϕ_j and ϕ_k that occur earlier in the sequence (i.e., $j, k < i$) via the rule (MP) (i.e., $\phi_k = (\phi_j \Rightarrow \phi_i)$);
- ϕ_i can be obtained from some ϕ_j that occurs earlier in the sequence (i.e., $j < i$) via the rule (Nec) (i.e., $\phi_i = (P \models\!\models \phi_j)$ for some principal P).

A formula is thus a theorem if it can be derived only from the axioms using the inference rules (MP) and (Nec). However, we are mainly interested in the formulae that can be derived from a set of other formulae that represent the assumptions of a protocol and the protocol itself. Therefore, we introduce the *derivable* relation as follows: A formula ϕ is derivable from a set of formulae Φ, denoted by $\Phi \vdash \phi$ if there exists a sequence of formulae $\phi_1, \phi_2, \ldots, \phi_n$ such that $\phi_n = \phi$ and for each ϕ_i of the sequence at least one of the following holds:

- ϕ_i is a theorem (i.e., ϕ_i is an axiom or it can be derived from axioms only via (MP) and (Nec));
- $\phi_i \in \Phi$;
- ϕ_i can be obtained from some ϕ_j and ϕ_k that occur earlier in the sequence (i.e., $j, k < i$) via the rule (MP) (i.e., $\phi_k = (\phi_j \Rightarrow \phi_i)$).

Finally, we introduce the following inference rule, which we call *conjunction*:

$$\frac{\phi,\ \psi}{\phi\ \wedge\ \psi} \quad \text{(Con)}$$

The interpretation of (Con) is straightforward: If both ϕ and ψ are available, then we also have $\phi\ \wedge\ \psi$. Note that if ϕ and ψ are available, then using the axiom $\phi \Rightarrow (\psi \Rightarrow (\phi\ \wedge\ \psi))$ (which is a tautology), we can obtain $\phi \wedge \psi$ via two applications of the rule (MP). Therefore, strictly speaking, the rule (Con) is not necessary. We introduced it for convenience, and to make our proofs more intuitive.

In addition to the inference rules, we allow any derived formula or subformula of a derived formula to be replaced with an equivalent formula or subformula. For instance, $(\phi \Rightarrow \psi)$ can be replaced with $(\neg\phi\ \vee\ \psi)$.

1.2.5 Using the logic

One can use our logic of channels for analyzing existing key transport protocols. For this, the protocol description first have to be translated into our notation. This usually means that all the cryptographic operations, such as encryption and digital signature, have to be eliminated from the description, and must be replaced by channels. It is also required to interpret some messages, and to extend or replace certain parts of them by logical formulae when it is necessary to capture implicit assumptions behind protocol steps. This process is called idealization [BAN90a]. Finally, the usual $P \rightarrow Q : \ X$ notation, which is used to indicate that P sends message X to Q, must be replaced by the formula $Q \lhd \tilde{X}$, where $\tilde{X}$ is the idealized version of X. In short, if the original protocol description has the form:

$$\begin{array}{l} P \rightarrow Q : \ X_1 \\ Q \rightarrow P : \ X_2 \\ \ldots \end{array}$$

then the translation looks like this:

$$\begin{array}{l} Q \lhd \tilde{X}_1 \\ P \lhd \tilde{X}_2 \\ \ldots \end{array}$$

Note that, for the purpose of the analysis, the protocol is translated into a set of formulae.

The goal of the analysis is to prove that the formulae that represent the goal of the protocol can be derived from the assumptions and the protocol itself using the axioms and the inference rules of the logic. In other words, the goal of the analysis is to show that for every $\gamma \in \Gamma$, $\Lambda \cup \Pi \vdash \gamma$ holds, where Λ is the set of assumptions, Π is the set of formulae that represents the protocol, and Γ is the set of goals.

We consider that the protocol is correct if such a proof exists. The inability of constructing a proof indicates that the protocol may have flaws. Although, in this case, we cannot generate a complete attack scenario with our logic, the analysis reveals the possible weakness in the protocol, and usually we can construct an attack afterwards easily.

1.2.6 Limitations of the logic

We should mention that our logic has certain limitations. In order to use the logic in an appropriate way, it is important to understand these limitations. For this reason, we discuss them briefly in this section. Before starting, we note that these limitations are not unique to our logic; all the logics of the BAN family share them. Nevertheless, these logics are considered to be useful in protocol verification.

The first limitation is that the logic is based on a simple model of time, in which only two epochs are distinguished: past and present. Everything that happened before the current run of the protocol under examination is considered to be happened in the past; and everything that happened after the start of the current protocol run is considered to be happened in the present. This simplification in the model makes the logic itself simple. However, it also limits the class of attacks which can be described with the logic. More precisely, the logic is directed at *classic replays* [Syv94] (i.e., replays of messages that are sent before the current run of the protocol), but strictly speaking, it does not address *interleaving* attacks (i.e., attacks

that involve replays of messages from multiple simultaneous runs of the same protocol). The reason is that according to the simple model of time, the messages that are replayed in an interleaving attack may indeed be generated in the current epoch. This does not mean that the logic will never discover interleaving attacks, since such an attack may exploit a weakness in the protocol that can be uncovered by using the logic. The point is that there are no specific features of the logic that address such attacks.

The second limitation is that the logic cannot be used to reason about *secrecy*. We cannot explicitly prove, for instance, that an attacker does not know the session key that is established in a protocol. Usually, it is fairly easy to check if the protocol leaks some secret out in an obvious way (i.e., to check whether the protocol is resistant against passive eavesdropping). It is more difficult, however, to prove that the protocol keeps everything that it should secret if we admit active attackers. Our logic is not intended to be used for this purpose; secrecy properties of the protocol might be verified by other tools (e.g., the tools described in [KMM94]). This does not mean that the logic will never discover a subtle release of some secret, but again, there are no specific features in the logic that address this issue.

Considering these limitations, it is clear that our logic should not be used in isolation, but in conjunction with other tools that address the above mentioned issues.

1.3 Examples for channels

In this section, we give some examples how well known cryptographic primitives can be modeled as channels. These examples also suggest how channels can be implemented with cryptographic primitives.

Digital signatures

Sending a message together with a digital signature of principal P on the message can always be modeled as a channel C such that $r(C) = \Omega$ and $w(C) = s(C) = \{P\}$, where Ω is the set of all principals in the system.

Symmetric-key encryption

In general, encryption with a key K_{PQ} shared by two principals P and Q can be modeled as a channel C such that $r(C) = w(C) = s(C) = \{P, Q\}$. Such a channel is not very useful, because no single source can be associated to the messages that are received on it. In particular, when P receives a message m on C, it does not know whether m was sent by Q or P itself.

However, if P and Q can recognize their own messages from the structure of the message or the content of certain fields, then the encryption with K_{PQ} together with the features that allow P and Q to recognize their own messages can be modeled as two distinct channels C' and C'' such that $r(C') = w(C') = r(C'') = w(C'') = \{P, Q\}$ and $s(C') = \{P\}$ and $s(C'') = \{Q\}$.

Let us suppose, for instance, that every message that P and Q send to each other encrypted with K_{PQ} contains a direction bit. More precisely, when P wants to send a message m to Q, then it sends $\{d_{P\to Q}, m\}_{K_{PQ}}$, and when Q wants to send a message m' to P, then it sends $\{d_{Q\to P}, m'\}_{K_{PQ}}$, where $d_{P\to Q}$ is a bit that indicates that this is a message from P to Q and $d_{Q\to P}$ is a bit that indicates that this is a message from Q to P. We could then

model $\{d_{P\to Q}, m\}_{K_{PQ}}$ and $\{d_{Q\to P}, m'\}_{K_{PQ}}$ as $C'(m)$ and $C''(m')$, where $s(C') = \{P\}$ and $s(C'') = \{Q\}$, or at least $Q \models (s(C') = \{P\})$ and $P \models (s(C'') = \{Q\})$.

As an other example, let us consider the following message $\{k, P, Q, N\}_{K_{PQ}}$, and suppose that P sends this message in some protocol to Q in order to setup a new session key k between P and Q. N may denote a nonce (e.g., a time-stamp). Q may send a similar message $\{k', Q, P, N'\}_{K_{PQ}}$ to P in another run of the same protocol where the roles of P and Q are swapped. Note that in this example, the principal names in the second and the third fields of the messages may play two roles: first, their order (P, Q or Q, P) determines the direction of the message; second, they may also be intended to indicate who are associated with the new key k (or k'). If both roles were intended by the protocol design[1], then we cannot omit the names when replacing encryption with channels as we did in the previous example with the direction bits. This means, that we can model $\{k, P, Q, N\}_{K_{PQ}}$ and $\{k', Q, P, N'\}_{K_{PQ}}$ as $C'(k, P, Q, N)$ and $C''(k', P, Q, N')$ where $s(C') = \{P\}$ and $s(C'') = \{Q\}$, or at least $Q \models (s(C') = \{P\})$ and $P \models (s(C'') = \{Q\})$.

Asymmetric-key encryption

In general, encryption with the public key of principal P can be modeled as a channel C such that $r(C) = \{P\}$ and $w(C) = s(C) = \Omega$. Similarly to pure symmetric-key encryption, this channel is not very useful, because no single source can be associated to the messages that are received on it. Indeed, in this case, any principal of the system can be the source of those messages.

However, if the message that is encrypted with the public key of P contains a secret that P can uniquely associate to another principal Q and P itself, then we can model the encryption as a channel C' such that $r(C') = \{P\}$, $w(C') = \{P, Q\}$ and $s(C') = \{Q\}$ or at least $P \models (s(C') = \{Q\})$.

It is important that the feature of the message that identifies Q is also associated to P. Suppose for instance that the message contains the preimage x of the hash value $h(x)$ previously published by Q in an authentic way. P may believe that if it receives a message that contains x, then Q revealed x, but this is not enough to model $\{x, m\}_{K_P}$ as $C'(m)$ where $s(C') = \{Q\}$. The reason is that Q may innocently reveal x to another principal, say R, in a message $\{x, m'\}_{K_R}$, and R may send $\{x, m\}_{K_P}$ to P. On the other hand, if P and Q share a secret p, then we can model $\{p, m\}_{K_P}$ as $C'(m)$ where $s(C') = \{Q\}$, because it is reasonable to assume that Q never encrypts p with a public key other than the public key of P with whom it shares p.

Message authentication codes

Let us assume that two principals P and Q share a key K_{PQ}. They can use this key to protect the integrity of their communication by computing message authentication codes (MAC) on their messages, and sending those messages together with their MAC to each other. How can we model this scheme with channels?

Since the messages are sent in clear, one may want to model the above scheme as a channel C such that $r(C) = \Omega$. The problem is that in this case anybody can recognize messages that are sent on C (i.e., Axiom (AX1) can be applied for any principal). This does not

[1]We note that such overloading should be avoided and the role of each part of a message should be specified unambiguously.

reflect the reality, since principals other than P and Q should not be able to recognize these messages (i.e., they should not be able to associate any source to them). If we require that $r(C) = \{P, Q\}$, then we have another problem: only P and Q can see the content of the messages that are sent on C (i.e., we cannot apply Axiom (AX2) for principals other than P and Q). This does not reflect the reality either.

A solution is to model the above scheme with two distinct channels C_1 and C_2 such that $r(C_1) = w(C_1) = s(C_1) = \Omega$ and $r(C_2) = w(C_2) = s(C_2) = \{P, Q\}$, and to assume that P and Q send their messages on both channels. Then, anybody can see the messages on C_1 but no source can be associated to these messages. At the same time, only P and Q can read from C_2, and thus, only they can recognize the messages.

Similarly to symmetric-key encryption, if the messages from P to Q can be distinguished from the messages from Q to P, then we can model this with two channels C_2' and C_2'' such that $s(C_2') = \{P\}$ and $s(C_2'') = \{Q\}$.

1.4 Synthesis rules

As we mentioned earlier, our goal is protocol construction rather than protocol verification. Although as a conceptual model, the logic described above can be used in protocol construction as it is, we want to further enhance it to obtain a more handy tool. Our main idea is to reverse some implications that can easily be derived from the axioms of the logic in order to obtain *synthesis rules*. These rules can then be used to construct protocols in a systematic way.

In order to illustrate the idea, let us assume that the following two formulae have been derived in some proof:

$$P \models (P \lhd C(X)) \tag{1.1}$$

$$P \models (s(C) = \{Q\}) \tag{1.2}$$

Then, using the inference rule (Con), we can derive

$$P \models (P \lhd C(X)) \ \wedge \ P \models (s(C) = \{Q\}) \tag{1.3}$$

From (1.3) and the appropriate instance of Axiom (AX12) we obtain by rule (MP) that

$$P \models (P \lhd C(X) \ \wedge \ s(C) = \{Q\}) \tag{1.4}$$

In addition, from the appropriate instance of Axiom (AX4) and rule (Nec), we have

$$P \models (P \lhd C(X) \ \wedge \ s(C) = \{Q\} \Rightarrow Q \mathrel{|\sim} X) \tag{1.5}$$

Note that we could use (Nec) because (AX4) is an axiom, and so it is a theorem. Now, we can apply rule (Con) for (1.4) and (1.5), and from the resulting formula and the appropriate instance of Axiom (AX10) we obtain by rule (MP) that

$$P \models (Q \mathrel{|\sim} X) \tag{1.6}$$

We have just proved that given (1.1) and (1.2), one can always derive (1.6). This means that if (1.6) represented a goal during protocol construction, then it could be reached by reaching the goals (1.1) and (1.2). We record this fact in the following *synthesis rule*:

$$P \models (Q \mathrel{|\sim} X) \ \left\{ \begin{array}{l} P \models (P \lhd C(X)) \\ P \models (s(C) = \{Q\}) \end{array} \right.$$

In general, synthesis rules have the following form:

$$\psi \left\{ \begin{array}{l} \phi_1 \\ \phi_2 \\ \vdots \\ \phi_n \end{array} \right.$$

and they are interpreted as follows: If one encounters the goal ψ during the construction of a protocol, then one can replace ψ with the new goals ϕ_1, ϕ_2, ..., ϕ_n.

A (not necessarily complete) set of proposed synthesis rules can be found in Appendix A. Most of the rules presented there can be derived from the axioms of our logic in a similar way as we have illustrated above. There are, however, three synthesis rules (*S13), (*S14) and (*S15) that cannot be derived from the axioms. Below, we justify them by informal reasoning based on intuition. Their usefulness will be proved through the example of Chapter 2.

The synthesis process often leads to (sub)goals of the form $P \models (Q \models \phi)$. Clearly, we would not like to design a protocol, the correctness of which depends on false beliefs. Therefore, if $P \models (Q \models \phi)$ is required, then we want that $Q \models \phi$ holds. Note however, that $Q \models \phi$ is not sufficient to reach $P \models (Q \models \phi)$, and therefore, the goal $Q \models \phi$ cannot simply replace the goal $P \models (Q \models \phi)$ during protocol synthesis. Instead, it should be considered as an additional goal to be reached. This leads to the following synthesis rule:

$$P \models (Q \models \phi) \left\{ \begin{array}{l} P \models (Q \models \phi) \\ Q \models \phi \end{array} \right.$$

Similarly, when we have a (sub)goal of the form $P \models (Q \mid\!\sim X)$ or $P \models (Q \sim X)$, we want to ensure that Q was indeed able to say X. Suppose, for instance, that X is a nonce generated by P. In order for Q to be able to say X, it must first see X. But again, $Q \lhd X$ is not sufficient for $P \models (Q \mid\!\sim X)$ or $P \models (Q \sim X)$, and thus, the goal $Q \lhd X$ cannot simply replace them. These observations lead to the following two synthesis rules:

$$P \models (Q \mid\!\sim X) \left\{ \begin{array}{l} P \models (Q \mid\!\sim X) \\ Q \lhd X_1 \\ Q \lhd X_2 \\ \ldots \end{array} \right.$$

and

$$P \models (Q \sim X) \left\{ \begin{array}{l} P \models (Q \sim X) \\ Q \lhd X_1 \\ Q \lhd X_2 \\ \ldots \end{array} \right.$$

where $X_1, X_2, \ldots$ denote those parts of message X that cannot be computed by Q.

1.4.1 Using the synthesis rules

The intended purpose of the synthesis rules is to directly help the construction of robust protocols. In fact, they allow the protocol designer to construct a protocol in a systematic way, starting from the desired goals and possibly some initial assumptions that the design must respect.

The synthesis technique is the following: Given a (sub)goal G, one has to check first if any of the rules (*S13), (*S14) or (*S15) can be applied to G. If one of them can be applied, and the application results in new goals that have not been considered so far, then it must be applied. Otherwise, one can choose and apply any of the other synthesis rules that can be applied. When a synthesis rule is applied, G must be replaced with the new goals specified in the applied rule. By the repetition of the above process, one constantly generates new goals. The synthesis is finished, when a set of goals is reached such that each goal in the set can either be considered as an assumption or as a message sending step. We will illustrate this in detail through an example in Chapter 2.

Note that it is possible that several synthesis rules could be used to reach a given goal. Our method gives freedom to the designer to choose from these rules. Therefore, we still rely on human creativity. However, once a rule has been selected and applied, the new goals to be reached are explicitly listed for the designer. This reduces the chance that she overlooks something that she should not.

1.5 Related work on formal models for key transport protocols

In principle, formal models can be used in the specification, in the construction, and in the verification of key transport protocols. So far, the research community has mainly focused on the verification aspect. There is also some notable work on formal specification; however, using formal methods for protocol construction has received much less attention. In this section, we give an overview of the most significant results in the above mentioned three areas. Other surveys can be found in [RH93, Mea95, GNG97].

1.5.1 Specification

The design of a protocol should begin with the specification of the requirements that the designer wants the protocol to satisfy. This means that the designer should have a clear idea of what she wants the protocol to achieve, or in other words, what the correctness of the protocol means. However, expressing the correctness criteria of a protocol is not trivial. Early work in this field focused on secrecy as the main correctness criterion (i.e., ensuring that an attacker cannot learn the session key). Later, authenticity was realized to be an equally important criterion. The problem has been approached from several different angles, some with the aim of developing a set of criteria that can be applied to protocols in general, and others with the aim of developing ways to express criteria for a number of different types of protocols.

In [DvOW92], Diffie *et. al.* present informal requirements for the correctness of an authenticated key transport protocol. Briefly, they say that session keys should remain secret and that protocol runs should match. The latter means that if A and B run a protocol then A's record of messages received from B should match B's record of messages sent to A, and vice versa. This notion has been formalized by Bellare and Rogaway in [BR93] and [BR95], using a model based on communicating probabilistic Turing machines. The notion of matching runs has also been formalized by Syverson in his extension of the Abadi-Tuttle logic with temporal logic [Syv93]. In [WL93], Woo and Lam take a similar approach to defining the correctness of a protocol. They define a semantic model that is based on two basic security properties, correspondence and secrecy. The former requires that certain events can take place only if others have taken place previously. This is very similar to the notion of matching runs,

but it is broader, since the events do not have to be the sending and the receiving of the same message.

Another approach to specifying requirements for authenticated key transport protocols is presented in the requirement language developed for the NRL Protocol Analyzer [SM93]. The requirements specified in this language have a form similar to the notion of correspondence in [WL93] in the sense that the requirements are given on sequences of events. The difference is that, instead of giving general requirements for correspondence that applies to all protocols, the user of the language can specify requirements for protocols of a particular class. Thus, requirements can vary according to the intended function of the protocol. Furthermore, the action, by which the intruder learns a word (a data item) is modeled as an event too, thus, secrecy does not need to be defined as a separate part of the model. In [SM93], Syverson and Meadows give a set of requirements for various kinds of message authentication protocols, while in [SM95], they give requirements for key transport protocols.

1.5.2 Construction

Most of the existing work in the application of formal methods to cryptographic protocols has been focused on the formal verification of existing protocols. However, it would be more effective to use these methods directly in the construction of the protocol, since this would result in robust protocols in the first place, and thus, the expense of the re-design (correction of errors) could be reduced significantly. Although the use of formal methods for protocol construction seems to be a natural application of the technology, not much research has been done in this particular area.

One direction is to develop specific protocol construction methodologies so that the resulting protocols are more amenable to analysis by formal methods. This approach is followed by Heintze and Tygar in [HT96]. They develop a modular approach to design cryptographic protocols. They design a family of tools for reasoning about protocol security, and prove a composition theorem that allows them to state sufficient conditions on two secure protocols such that they may be combined to form a new secure protocol. They give counterexamples to show that when the conditions are not met, the new protocol may not be secure.

Another example of the same approach is described in [GS95], where Gong and Syverson present a new methodology to facilitate the design and analysis of secure key transport protocols. They suggest to restrict protocol designs to well defined practices, instead of ever increasing the complexity of protocol security analysis due to the endless variations in protocol construction. In particular, they introduce a novel notion of a fail-stop protocol, which automatically halts in response to any active attack, thus reducing protocol security analysis to that of passive attacks only. Excluding the possibility of an active attack makes the validation of the paradoxical secrecy assumption in BAN-like logics easier, and thus, puts modal logic based analysis methods (see next subsection) on a stronger footing. Gong and Syverson suggest types of protocols that are fail-stop, however, these may not be practical for some applications. In [KS96], Keromytis and Smith present a generic method for creating efficient fail-stop protocols.

A similar approach is proposed by Boyd and Mao in [BM94b, BM94a]. They define a methodology for designing key transport protocols in a restricted way by allowing only protocol messages that contain a small number of elements, which have a well defined purpose and meaning. They argue that the protocols designed in this way must be correct in the sense that a specified security criterion will not be violated if the protocol participants act

correctly. The protocols are defined at an abstract level, where cryptographic operations are represented as channels. To some extent this is similar to the way in which we use the channel abstraction, however, Boyd and Mao do not use the notion of channel within the context of a logic. Actually, their approach is not based on any formal logic. The abstract protocols can be made concrete in a variety of ways by implementing the channels with cryptographic primitives.

In [AN96], Abadi and Needham present principles for designing authenticated key transport protocols. Their principles are informal guidelines, but they can complement formal methods. The principles presented are neither necessary nor sufficient for correctness of protocols, however, they are helpful in the sense that a number of well-known errors can be avoided by adhering to them. Similar principles for public-key protocols are proposed in [AN95]. In [Syv96], Syverson criticizes this approach: He presents limitations and exceptions for some of the basic design principles, and gives examples for secure protocols that fail to meet the principles.

More recently, Perrig and Song propose an automatic protocol generation method in [PS00]. Their approach is based on random generation of a huge number of protocols that satisfy certain conditions (e.g., number of protocol participants and type of cryptographic primitives used are fixed), and filtering of the correct ones using a fast automatic protocol verification tool (Athena [Son99]).

1.5.3 Verification

Following [Mea95], we classify approaches to the formal verification of key transport protocols into four types:

- using verification tools not specifically developed for the analysis of cryptographic protocols;
- using expert systems to investigate different scenarios;
- using modal logics of knowledge and belief; and
- algebraic approaches.

Since we are mainly interested in the logic based design of protocols, we put the emphasis on the modal logic based approach.

Using general purpose verification tools

The main idea of this approach is to treat a key transport protocol as any other (distributed) program and attempt to prove its correctness. The first step is to specify the protocol and its correctness requirements so that the techniques apply. For this purpose, Varadharajan uses LOTOS [Var90], Kemmerer specifies the system in Ina Jo [Kem89], while others use even more general description techniques such as state machines [Var89] or Petri nets [NT92]. Once the protocol and its requirements are specified, it can be investigated by using the tools that are available in the formalism used.

In [Var90], Varadharajan studied how LOTOS can be used to analyze cryptographic protocols. He gives example specifications of protocols in LOTOS, but he cannot demonstrate any result in their analysis. The paper concludes by stating that LOTOS tools are not adequate for this kind of analysis.

In [Kem89] and [KMM94], Kemmerer uses an extension of first-order predicate calculus, a formal specification language called Ina Jo. Ina Jo was designed as a general purpose tool to support software development and correctness proofs. Kemmerer describes an example security system, and then gives an Ina Jo specification of it. He also specifies critical requirements that the system is to satisfy in all states. Once the specification is complete, Ina Jo generates theorems that can be used to verify if the critical requirements are satisfied. Kemmerer uncovers a weakness in his sample system, but the value of this method is limited, because the specification of the critical requirements requires that the designer knows the potential attacks in advance.

In [Var89], Varadharajan describes how to specify a protocol using state diagrams. Each party of the protocol is described by a state diagram, and then the protocol is represented as the cross product of the state diagrams for each individual parties. Once the protocol is described in this way, the designer can investigate various executions of the protocol by applying a technique known as the reachability analysis. The main problem of reachability analysis techniques is the quick growth of the number of the states (often referred to as the state explosion problem). In [SS98], Shmatikov and Stern address this problem in the context of security protocol verification. Their approach is based on the observation that given a state s, one can safely remove those states from the state graph in which the honest protocol participants have the same knowledge as in s and the intruder has less knowledge than in s. They propose two techniques that reduce the number of reachable states and thus allow the analysis of larger protocols. The first technique is to let the intruder always intercept messages sent by the honest participants. The second technique is to prevent the intruder from sending messages if at least one of the honest participants is able to send a message. Intuitively, both techniques increase the intruder's knowledge. They prove that the proposed state reduction techniques are sound (i.e., each protocol error that would have been discovered in the original state-graph will be discovered in the reduced graph). The techniques have been implemented in the Murφ verifier, and have reduced the execution time significantly.

In [NT92], Nieh and Tavares uses a Petri net based methodology for the formal modeling and analysis of cryptographic protocols. In particular, they use colored Petri nets to model protocols. Their model also includes a general intruder model that can be used to formulate intruder attacks and generate test cases. The analysis of the security properties of cryptographic protocols is based on an exhaustive penetration test that searches for scenarios that violate certain specified criteria. These criteria are defined in terms of requirements on Petri net states of the protocol. Although the use of colored Petri nets enable them to produce compact and manageable descriptions, tools that support the effective execution of an exhaustive search are still missing. One can translate these high level Petri nets into ordinary Petri nets, and use the tools available for those, but then one has to cope with the state explosion problem again.

A more recent approach models the protocol participants as Communicating Sequential Processes (CSP) [Hoa85] and uses the Failure Divergence's Refinement (FDR) checker, which is a general purpose tool that can be used to determine whether an implementation refines a specification. This approach has been successfully applied to the verification of key transport protocols by Roscoe [Ros95] and Lowe [Low96]. It has also been applied to other types of cryptographic protocols as well (e.g., a non-repudiation protocol has been analyzed by Schneider in [Sch98]).

Another recent approach is to use the Higher Order Logic (HOL) for stating and proving properties of cryptographic protocols [Sne95]. Tools that are based on HOL and used for

analyzing key transport protocols are include Convince [LHB96] and Isabelle [BP97, Pau98, Pau99].

Developing expert systems

The idea of this approach is to develop expert systems that the protocol designer can use to generate and investigate various scenarios. Most of these systems are based on an underlying, state machine based model of the protocol. But, as opposed to the state machine based analysis of protocols described in the previous subsection, these systems begin with an undesirable state and attempt to discover if this state is reachable from an initial state.

One of the earliest system of this approach is the Interrogator by Millen *et. al.* [MCF87]. In the Interrogator, protocol participants are modeled as communicating state machines whose messages are intercepted by the intruder who can either destroy messages, modify them, or let them pass through unmodified. Given a final state, in which the intruder knows some word which should remain secret, the Interrogator tries all possible ways to construct a path by which that state can be reached. If it finds such a path, then a security flaw is identified. The Interrogator has not yet found a previously unknown attack, but it has been able to reproduce a number of known attacks [KMM94].

The NRL Protocol Analyzer [Mea91, KMM94] is similar to the Interrogator: the designer specifies an insecure state and the Protocol Analyzer attempts to construct a path to that state from an initial state. Unlike the Interrogator, an unlimited number of protocol rounds are allowed in a single path. This allows the Protocol Analyzer to discover attacks that rely on the intruder's ability to weave several different runs of a protocol together. Also unlike the Interrogator, the emphasis is, not only on finding paths to insecure states, but also on proving that those states are unreachable. This is made possible by having the user prove that certain paths leading backwards from an insecure state go into infinite loops, thus, they never reach an initial state. Once these paths have been eliminated, the resulting search space is often small enough to search exhaustively. The proofs that paths lead into infinite loops are largely guided by the user, thus, the search is less automated than in the Interrogator. The NRL Protocol Analyzer has been successful in finding several previously unknown security flaws in protocols [Mea91] [Mea92].

In [LR92], Longley and Rigby use a rule based system that transforms goals into subgoals and can constantly continue this process. They use this rule based scheme to build a tree, in which each node represents a data item, and the children of a node represent those data items that are required for the knowledge of the data represented by the father node. In this way, they can construct a tree, in which the root node represents the data required by the intruder for an attack (e.g., a cryptographic key), and the leaves represent those data items that are required to know the root item. The Longley-Rigby tool allows the user to interact with the system. The user can determine whether a data can or cannot be found by the intruder. If the data is judged to be accessible, this information can be inserted into the system, and the generation of the tree can proceed. Longley and Rigby managed to find a subtle and previously unknown flaw in a hierarchical key management scheme.

The problem of expert systems developed specifically for the analysis of authentication protocols is that they are often inefficient because they perform exhaustive search. Sometimes they do not even halt and the results are inconclusive. To cope with these problems, they require human intervention. Their advantage is that if they discover a flaw, then the attack scenario that exploits the flaw is directly available, which is not the case, for instance, with

the more popular modal logic based approach described in the next subsection.

Modal logic based approach

The main idea of this approach is to use modal logics, similar to those that have been developed for the analysis of the evolution of knowledge and belief during the execution of distributed algorithms [HM90], for modeling and analyzing authenticated key transport protocols. After all, these protocols can be viewed as distributed algorithms. Such logics consists of a language, which is used to describe various statements about the protocol such as what the participants know or believe, and some inference rules, which are used to derive new statements from previously derived statements. The goal of the analysis is to derive a statement that represents the correctness condition of the protocol. The designer's inability to do so means that the protocol may not be correct. The analysis often reveals the possible weakness in the protocol, and an attack can be constructed easily afterwards. Our logic of channels is an example application of this approach.

The best known and most influential logic of this type is called BAN logic [BAN90a, BAN90b]. It can be used to describe the beliefs of trustworthy parties involved in key transport protocols and the evolution of these beliefs as a consequence of communication. It has been successfully applied to discover flaws in a variety of protocols and has also been helpful in the understanding of the basic concepts of authentication. BAN logic is simple, which might be one of the reasons for its popularity. The need for many universal assumptions in the underlying model, however, is a minor disadvantage. In BAN logic, it is assumed that principals of the system are trustworthy and do not release secrets (this has not always been understood with its full meaning [Nes90]); the applied encryption schemes are strong (i.e., encrypted messages can be decrypted only with the appropriate key); each encrypted message contains sufficient redundancy to allow a principal who decrypts it to verify that it has used the right key; and principals can recognize and ignore their own messages.

BAN logic does not attempt to model a protocol in a richness as other logics do. It does not attempt to model the distinction between seeing a message and understanding it; it does not attempt to model the revision of beliefs; it does not attempt to model trust or the lack of it; and finally, it does not attempt to model knowledge. The avoidance of these issues is intentional in BAN logic; this makes it simple and straightforward. However, this also means that these issues have to be addressed in the informal mapping from protocol specification to BAN specification. This mapping is called *idealization*. Burrows *et. al.* consider the idealized protocols as clearer and more complete specifications than the traditional descriptions found in the literature, which they view as implementation dependent encodings of the protocol. Although, this is true, no clear idealization method is presented, which led to misunderstandings and the misuse of the logic [Nes90, BM93].

To overcome these problems, Abadi and Tuttle reformulate the original BAN logic and provide a new semantics for it in [AT91]. Abadi and Tuttle identifies many sources of the confusion created by BAN logic. They remove some unnecessary mixing of semantic and implementation details in the original definitions and inference rules. They define all concepts, such as seeing, saying, believing, etc., independently rather than jointly with other concepts. They reformulate the set of inference rules as an axiomatization with modus ponens and necessitation as the only rules. These changes make the logic much simpler and allow them to dispense with the implicit assumption of honesty (it is not needed anymore that principals believe the messages they send). One of the greatest contributions of the paper is a formal

semantics, which defines belief as a form of resource-bounded, defeasible knowledge. Abadi and Tuttle claim that the reformulated logic is sound according to the new semantics defined. However, Syverson and van Oorschot point out in [SvO94] that one of the axioms is not sound. The presentation of our logic of channels is very similar to that of the Abadi-Tuttle logic.

BAN logic has been extended in many directions. In [GNY90], Gong *et. al.* propose a logic referenced later as GNY logic. In GNY logic, it is not required to assume that principals are trustworthy and redundancy is always explicitly present in encrypted messages. GNY logic distinguishes between what a principal can possess and what it can believe in. It enables to express different trust levels and implicit conditions behind protocol steps. This makes GNY logic a more realistic model for key transport protocols than BAN logic. However, GNY logic has more than 40 inference rules, which has led many to reject this approach.

Other extensions of BAN logic include [MB93], in which Mao and Boyd propose a logic with several improvements on the original BAN logic; [GS91], in which Gaarder and Snekkenes extend BAN logic with constructs to reason about time; [vO93], in which van Oorschot extends BAN and GNY to deal with key agreement protocols such as Diffie-Hellman; and [CSNP92], in which Campbell *et. al.* extend BAN logic using probabilistic reasoning to calculate a measure of trust rather than using complete trust. The list of extensions above is not complete; there are many other works in this field.

In [SvO94], Syverson and van Oorschot propose a logic, later referenced as SvO, that encompasses a unification of four of its predecessors in the BAN family, namely, GNY logic [GNY90], the Abadi-Tuttle logic [AT91], the logic proposed by van Oorschot in [vO93], and BAN itself [BAN90a]. The proposed logic captures all of the desirable features of its predecessors, nonetheless, it accomplishes this with no more axioms or rules than the simplest of its predecessors (i.e., BAN). Syverson and van Oorschot also present a model-theoretic semantics with respect to which the logic is sound.

The importance of having an alternative, independently motivated semantics is emphasized by Syverson in [Syv90a, Syv91b, Syv91a, Syv92]. A formal semantics provides a precise structure with respect to which soundness and completeness of the logic may be proven, and thus, it allows us to evaluate the logic. If a logic does not have an independently motivated semantics, then whatever assurances protocol analysis via such logic brings may prove illusory. Syverson also illustrates how the semantics itself can be used as a reasoning tool to discover results that would be very difficult or impossible to prove by reasoning syntactically.

There are a number of other logics that do not belong to the BAN family. These include Bieber's CKT5 [Bie90], Syverson's KPL [Syv90b], Moser's logic [Mos89], Rangan's logic [Ran88], and the system of Yahalom *et. al.* [YKB93]. Bieber's CKT5 and Syverson's KPL can be used to reason about the evolution of knowledge about words used in a protocol. They also make a distinction between seeing a message and understanding its significance. Moser's logic is particular, because it is the only non-monotonic logic. It can be used to reason about beliefs of protocol participants, and about how these beliefs change if, for instance, a key used in a secure communication becomes compromised. Rangan's logic can be used to reason about the effect of trust in the composition of secure communications channels, and provides a formal basis for the evolution of belief from trust. The system of Yahalom *et. al.* is used to derive information about the nature of the trust that parties in a protocol must have in each other in order for a protocol to operate correctly.

Using modal logics of knowledge and belief is the most popular approach to applying formal methods in the analysis of key transport protocols. The reason is, probably, that these

logics are easy to use and intuitive. However, one must be careful, because there are often subtle assumptions in the underlying model, getting away from which may lead to the misuse of the logic and errors in the analysis. Thus, the designer has to be aware of the scope of the logic and its limitations.

Algebraic approach

Another approach to apply formal methods to cryptographic protocol analysis is to model the protocol as an algebraic system. Algebraic models were successful to represent very subtle kinds of knowledge in cryptographic protocols. Furthermore, the fact that the objects modeled correspond strongly to entities and messages used in the tools based on state machines suggests that algebraic models could be used to provide the state machine tools with a stronger capability of modeling the knowledge that the intruder can gain. However, the question of how these algebras can be incorporated in state machine based analysis tools is still open [Mea95].

The first work that considers a cryptographic protocol to be an algebraic system is [DY83] by Dolev and Yao. In their model, Dolev and Yao assume that the network is under the control of the intruder who can read all traffic, alter and destroy messages, and perform any operation, such as encryption, that is available to legitimate users of the system. However, it is assumed that initially the intruder does not know any information that is to be kept secret, such as encryption keys belonging to legitimate users of the system. Since the intruder can prevent any message from reaching its destination, and since she can also create messages of her own, Dolev and Yao treat any message sent by a legitimate user as a message sent to the intruder and any message received by a legitimate user as a message received from the intruder. Thus, the system becomes a machine used by the intruder to generate words. These words obey certain rewrite rules, such as the rule that encryption and decryption with the same key cancel each other out. Thus, finally, the intruder manipulates a term rewriting system. If the goal of the intruder is to find out a word that is meant to be secret, then the problem of proving a protocol secure is equivalent to the problem of proving that a certain word cannot be generated in a term rewriting system. Dolev and Yao use this idea to investigate the security of certain classes of public key protocols. They define the notion of cascade protocols and name-stamp protocols. They give sufficient and necessary conditions for cascade protocols to be secure, and provide a polynomial time algorithm to decide if a given name-stamp protocol is secure.

The Dolev-Yao model is too restrictive to be useful for the analysis of most key transport protocols. It can only be used to detect failures of secrecy, and it does not allow participants to remember state information from one state to the next. Thus, most later works extend it to be capable for analyzing other classes of protocols. These works include [Mer83], in which Merritt generalizes the technique of Dolev and Yao to model diverse cryptographic systems and to formally state and prove security properties other than secrecy. In [Tou92], Toussaint describes a technique for deriving the complete knowledge of participants in cryptographic protocols that is based on the algebraic model of Merritt.

A more recent and successful work is [AG98], in which Abadi and Gordon use the π-calculus [Mil99] to describe protocols at an abstract level. They use the π-calculus primitives for channels, in particular the scoping rules, to model certain properties of cryptographic protocols. Then, they extend the π-calculus to what they call *spi*-calculus to analyze protocols at a less abstract level. The spi-calculus permits an explicit representation of the use of

cryptography in protocols. In the spi-calculus, security properties of the protocol are expressed as equivalences between spi-calculus processes (e.g., a protocol keeps secret a piece of data X if the protocol with X is equivalent to the protocol with X' for every X' and in every environment). The intruder is not explicitly modeled, and this is a great advantage of the approach. Modeling the intruder can be tedious and can lead to errors (e.g., it is very difficult to model that the intruder can invent random numbers but is not lucky enough to guess the random secrets of legitimate parties). Instead, the intruder is represented as an arbitrary spi-calculus process.

1.6 Summary

In this chapter, we proposed a systematic approach to construct key transport protocols that is based on a logic of belief. Our main idea was to reverse some implications that can be derived from the axioms of the logic and to turn them into synthesis rules. The synthesis rules can be used by the protocol designer to derive a protocol Π and a set of assumptions Λ starting from a set of goals Γ. Our approach has the advantage that it results in correct protocols in the sense that all the goals in Γ can be derived from Π and Λ using the logic. Another important advantage is that all the assumptions upon which the correctness of the protocol depends are made explicit, so someone who reviews the design can verify if they are acceptable in a given application. We are not aware of any similar approach that addresses the same problem in its full generality.

While the logic that underlies our approach is similar to the well-known BAN logic [BAN90a, BAN90b], it also differs from it in many ways. The most striking difference is that our logic does not have any construct that explicitly deals with cryptographic primitives, such as encryption or digital signature. Instead, we model these primitives in a uniform manner as *channels* with various access properties. The use of channels makes our logic simple and compact. It also allows us to reason about protocols at a rather high abstraction level without being concerned with the actual cryptographic implementation details. We believe that this feature is useful when designing protocols (although perhaps less interesting when verifying them), because it allows us to identify the required properties of a channel first, and decide about its implementation afterwards (possibly taking into consideration additional design criteria).

The result of the protocol synthesis process is an abstract description of the protocol, which involves idealized messages and channels. In order to implement it, the abstract description needs to be translated into a less abstract one, which involves bit strings and cryptographic primitives. This translation is only partially supported by our approach. Namely, we do not provide formal translation rules, but the requirements that the implementation of the channels must satisfy are identified in form of assumptions about the source set and the reader set of the channels. In practice, this is usually sufficient to choose an appropriate implementation.

Our approach has certain limitations. First, the synthesis is not fully automatic: It is possible that at a given point in the construction there are several synthesis rules that are available, and the designer is left with her intuition (or creativity) to choose among these. On the other hand, once a synthesis rule has been chosen and applied, the further steps are explicitly listed for the designer, which reduces the chance that she overlooks something that she should not. Another limitation stems from an inherent limitation of BAN-like logics, namely, that we cannot reason about secrecy directly within the logic. Also, because of

the rather simple way in which time is modeled, complex interleaving attacks cannot be represented in the logic. This means that our approach should not be used in isolation, but in conjunction with other tools that overcome these problems. While this is certainly a limitation, it is not a real drawback, since if a top-down design approach is used, then such tools are applied anyhow in later steps of the design.

Publication: [BSW98]

Chapter 2

Protocol construction with the logic of channels

2.1 Introduction

Our main goal in this chapter is to illustrate the usage of the logic of channels presented in the previous chapter, and in particular to demonstrate the usage of the synthesis rules and our systematic protocol construction approach. For this reason, we consider an authenticated key transport protocol proposed for the so called Global Mobility Network (GLOMONET) in [SN97]. We identify several weaknesses in this protocol, and explain how these weaknesses can be exploited by various attacks. Then, we redesign the protocol using the synthesis rules introduced in the previous chapter. The result will be a robust, and intuitively much clearer protocol.

2.1.1 The Suzuki-Nakada protocol

In [SN97], an authentication mechanism is proposed for the Global Mobility Network, which consists of two phases:

- *the roaming-service-setup phase*, in which authentication and key transport is performed by the *visited network*, the *home network*, and the *roaming user* in order to set up the roaming-service environment; and
- *the roaming-service-provision phase*, in which authentication is performed only by the visited network and the roaming user in order to provide the roaming service within the visited network.

The motivation for this two-phase model is to have the home network involved in the authentication process only once, during the roaming-service-setup phase. In this phase, a shared session key is established between the visited network and the roaming user with the help of the home network. This secret key is used later in the roaming-service-provision phase to authenticate the roaming user and the visited network to each other without any contribution from the home network. Thus, as long as the roaming user stays in the region of the visited network, authentication can be performed without contacting the home network of the roaming user (unlike, for instance, in the GSM system, where the visited network often

has to obtain challenge-response pairs from the home network in order to authenticate the roaming user [MG98]).

The following authenticated key transport protocol[1] is proposed for the establishment of the session key between the visited network and the roaming user, which we will call Suzuki-Nakada protocol after the authors:

Suzuki-Nakada protocol		
(msg1)	$U \to V:$	$request(U, H)$
(msg2)	$V \to H:$	r_1
(msg3)	$H \to V:$	$\{r_1\}_{K_{HV}},\ r_2$
(msg4)	$V \to H:$	$\{r_2\}_{K_{HV}},\ U,\ \{\{K_a\}_{K_t}\}_{K_{HV}}$
(msg5)	$H \to V:$	$\{\{K_a\}_{K_t}\}_{K_{UH}}$
(msg6)	$V \to U:$	$r_3,\ K_t,\ \{\{K_a\}_{K_t}\}_{K_{UH}}$
(msg7)	$U \to V:$	$\{r_3\}_{K_a}$
(msg8)	$V \to U:$	$\{\{r_3\}_{K_a}\}_{K_a}$

where U, V, and H denote the roaming user, the visited network, and the home network, respectively; r_1, r_2, and r_3 are random numbers used as nonces; K_{HV} is a long-term secret key shared by H and V; K_{UH} is a long-term secret key shared by U and H; and K_t and K_a are keys generated by V. The operation of the protocol is described as follows:

- The roaming user U sends a service request to the visited network V (msg1). We assume that the service request contains the identifiers of U and her home network H. The original description of the protocol in [SN97] does not make this explicit, but we prefer to do so, because it makes it easier to explain the attacks later.

- V generates a random number r_1, and sends it to the home network H of U (msg2). This is a challenge for authenticating H.

- H responds to V's challenge with the encrypted random number $\{r_1\}_{K_{HV}}$, and sends another random number r_2 to V (msg3). The number r_2 is a challenge for authenticating V.

- V verifies if it has received back its random number r_1 encrypted with the key K_{HV}. If so, then V believes that H is present. V generates the user authentication key K_a and the temporary cipher key K_t. Then, V responds to H's challenge with $\{r_2\}_{K_{HV}}$, and sends also the identifier of U and the ciphertext $\{\{K_a\}_{K_t}\}_{K_{HV}}$ to H (msg4). Sending the identifier of U is not made explicit in [SN97], but it is clear that H needs to know it in order to be able to use the appropriate key in (msg5) later.

- H verifies if it has received back its random number r_2 encrypted with the key K_{HV}. If so, then H believes that V is present. H decrypts $\{\{K_a\}_{K_t}\}_{K_{HV}}$ with K_{HV} and re-encrypts the result $\{K_a\}_{K_t}$ with the key K_{UH}. H sends $\{\{K_a\}_{K_t}\}_{K_{UH}}$ to V (msg5).

- V forwards $\{\{K_a\}_{K_t}\}_{K_{UH}}$ to U along with the key K_t and the random number r_3, which is a challenge for authenticating U (msg6).

[1] We modified the notation used in [SN97], in order to keep the presentation of the whole thesis uniform.

- U uses the long-term key K_{UH}, and the key K_t that she has just received to obtain the authentication key K_a. Then, U responds to V's challenge with $\{r_3\}_{K_a}$ (msg7).
- V verifies if it has received back its random number r_3 encrypted with the fresh authentication key K_a. If so, then V believes that U is present and that she has accepted the key K_a. V sends $\{\{r_3\}_{K_a}\}_{K_a}$ to U (msg8).
- U verifies if she has received back $\{r_3\}_{K_a}$ encrypted with K_a. If so, then U believes that V is present and knows the key K_a.

Conceptually, the Suzuki-Nakada protocol can be divided into three sub-protocols:

1. The first sub-protocol consists of (msg2), (msg3), and the first block of (msg4). The goal of this sub-protocol is mutual authentication of V and H.
2. The second sub-protocol consists of (msg1), the second part of (msg4), (msg5), and (msg6) (except the random number r_3). The goal of this sub-protocol is to establish a secret session key K_a between U and V. K_a can then be used by U and V to authenticate each other in the third sub-protocol and later in the roaming-service-provision phase.
3. The third sub-protocol consists of the first block of (msg6) (the random number r_3), (msg7), and (msg8). Similar sub-protocols are often used by authenticated key transport protocols to achieve explicit key confirmation. However, the authors of the protocol do not mention explicit key confirmation to be a goal of their protocol. Instead, they require the third sub-protocol for mutual authentication of U and V.

In fact, the protocol does not reach any of the above mentioned three goals. The weaknesses in the protocol are quite obvious, and could be explained without any formal method. Nevertheless, in the next section, we will explain them in terms of our logic of channels, in order to illustrate how the logic would discover them.

2.2 Analysis

As we said before, the goal of the first sub-protocol is mutual authentication of V and H. This goal can be represented by the logical formulae: $V \models (H \mid\sim r_1)$ and $H \models (V \mid\sim r_2)$. We can assume that $V \models \sharp(r_1)$ and $H \models \sharp(r_2)$, since r_1 and r_2 are generated by V and H, respectively. This means that it would be sufficient to show that the formulae $V \models (H \mid\sim r_1)$ and $H \models (V \mid\sim r_2)$ can be derived for the protocol.

The intention of encrypting r_1 and r_2 with K_{HV} is to send them through channels to which the intended receiver can associate a source. Ideally, we would like that $\{r_1\}_{K_{HV}}$ can be modeled as $C'_{HV}(r_1)$, and $\{r_2\}_{K_{HV}}$ can be modeled as $C''_{HV}(r_2)$, where $V \models (s(C'_{HV}) = \{H\})$ and $H \models (s(C''_{HV}) = \{V\})$ hold. The problem is that both V and H use K_{HV} to encrypt random numbers for the other, therefore, the encryption of these numbers with K_{HV} cannot be modeled as two channels C'_{HV} and C''_{HV} with distinct source sets. Indeed, when V receives $\{r_1\}_{K_{HV}}$, it does not know whether the message was generated by H or V, since there is nothing in the message that identifies its source. And similarly, when H receives $\{r_2\}_{K_{HV}}$, it does not know whether it was generated by V or H. In particular, if the roles of V and H can be swapped (i.e., if V can be a home network of a user, and H can be a visited network for

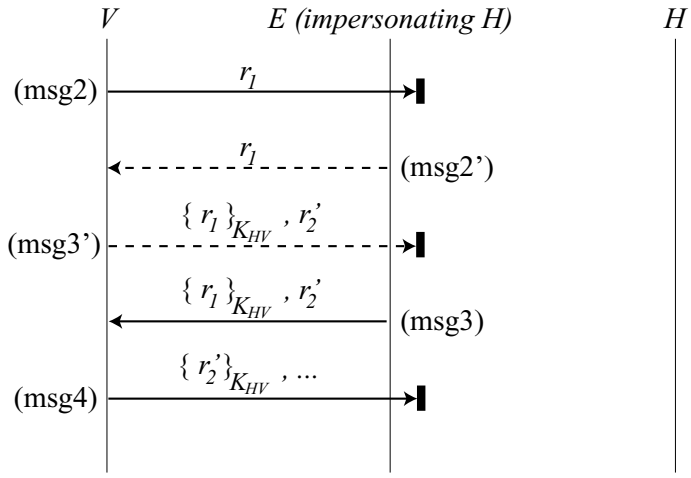

Figure 2.1: Illustration of a reflection attack on the first sub-protocol

the same user), then the first sub-protocol is vulnerable to the well-known *reflection attack* [BGH+93], which is illustrated in Figure 2.1.

The reflection attack is described as follows: The attacker E intercepts r_1 sent by V to H in one instance of the first sub-protocol (msg2). Then E starts another instance of the first sub-protocol with V pretending to be H and sending r_1 to V (msg2'). V probably does not check every challenge received from H against pending challenges that it sent to H, and therefore, it simply encrypts r_1 with K_{HV}, and sends $\{r_1\}_{K_{HV}}$ to H (msg3'). This message is intercepted by E, and $\{r_1\}_{K_{HV}}$ is reflected back to V as a response to V's challenge in the first instance of the sub-protocol (msg3). Note that from V's point of view, it successfully ran an instance of the first sub-protocol with H, although H may not be present at all.

The goal of the second sub-protocol is to setup a secret session key K_a between U and V. This can be represented by the following formulae in our logic: $U \models ((K_a \sim \{U,V\}) \wedge \sharp(K_a))$ and $V \models ((K_a \sim \{U,V\}) \wedge \sharp(K_a))$. Since K_a is generated by V, we can assume that $V \models ((K_a \sim \{U,V\}) \wedge \sharp(K_a))$ holds. The goal of the second sub-protocol is to establish the corresponding belief for U.

We can assume that U trusts V for generating good session keys, which can be represented by the following formulae:

$$U \models ((V \models ((K_a \sim \{U,V\}) \wedge \sharp(K_a))) \Rightarrow ((K_a \sim \{U,V\}) \wedge \sharp(K_a)))$$
$$U \models ((V \mid\sim ((K_a \sim \{U,V\}) \wedge \sharp(K_a))) \Rightarrow (V \models ((K_a \sim \{U,V\}) \wedge \sharp(K_a))))$$

Thus, it would be sufficient to show that $U \models (V \mid\sim ((K_a \sim \{U,V\}) \wedge \sharp(K_a)))$ can be derived. Note, however, that there is no direct channel between V and U through which V can send K_a and utter $(K_a \sim \{U,V\}) \wedge \sharp(K_a)$ to U. Therefore, V sends K_a to U via H. We can assume that U trusts H for reliably relaying messages between V and U, and perhaps even for recognizing freshness. This can be represented by the following formulae:

$$U \models ((H \models (V \mid\sim X)) \Rightarrow (V \mid\sim X))$$
$$U \models ((H \mid\sim (V \mid\sim X)) \Rightarrow (H \models (V \mid\sim X)))$$

On the other hand, H is not trusted for any statement regarding the session key K_a itself. In fact, the authors of the protocol even intended to hide K_a from H, that is why K_a is encrypted with K_t. This means that $U \models (V \mid\sim ((K_a \sim \{U, V\}) \wedge \sharp(K_a)))$ can only be derived via the formula $U \models (H \mid\sim (V \mid\sim ((K_a \sim \{U, V\}) \wedge \sharp(K_a))))$.

Now the problem is that the above formula cannot be derived because no timely channel exists on which H can send messages to U. There is a channel C_{UH} between H and U implemented by the encryption with the key K_{UH} for which we even have that $U \models (s(C_{UH}) = \{H\})$, since only ever H encrypts messages with K_{UH}. But this channel is not timely. In addition, the message sent by H to U through C_{UH} does not contain anything fresh for U. This means that all that we can derive are formulae like $U \models (H \mid\sim \ldots)$.

This weakness of the Suzuki-Nakada protocol is similar to the well-known weakness of the Needham-Schroeder protocol [NS78] discovered by Denning and Sacco in [DS81]. As a consequence, in the next section we will show that an attack similar to the attack against the Needham-Schroeder protocol can, indeed, be mounted against the Suzuki-Nakada protocol too.

Finally, we note that the third sub-protocol cannot reach its goal, since it depends on the result of the second sub-protocol. If the second sub-protocol was correct, then the single encryption with K_a in (msg7) and the double encryption with K_a in (msg8) could be modeled as two channels C'_a and C''_a such that $V \models ((s(C'_a) = \{U\}) \wedge \natural(C'_a))$ and $U \models ((s(C''_a) = \{V\}) \wedge \natural(C''_a))$, and we could derive that $V \models (U \mid\sim r_3)$ and $U \models (V \mid\sim r_3)$, which represent the goal of mutual authentication of U and V. But the second sub-protocol is not correct.

2.3 Attacks

In this section, we present three attacks that exploit the weaknesses identified in the previous section. The first two attacks exploit the vulnerability of the protocol to reflection attacks as described above. They enable a legitimate, but malicious user to obtain the authentication key K_a established between the roaming user and the visited network. In this way, she can impersonate the roaming user or the visited network. These attacks require that the attacker has access to the communication link between the visited network and the home network, and that she can intercept and modify messages. Sometimes, it can be assumed that this is not possible (or too expensive). The third attack that we present is valid even in this case. By exploiting the weakness that the user does not receive anything fresh in the protocol, it allows the attacker to feed the roaming user with a compromised, old authentication key, and thus, to masquerade as the visited network.

2.3.1 Attack 1

In this attack, the attacker E obtains the authentication key K_a of the roaming user U and the visited network V. We assume that E is a legitimate, but malicious user from the same home network H as the roaming user U, and that E eavesdropped and recorded the protocol run in which K_a has been established. Thus, E knows $\{\{K_a\}_{K_t}\}_{K_{HV}}$ and K_t. The attack scenario is illustrated in Figure 2.2.

Since E is a legitimate user, she can start the protocol with V. E lets the protocol run until (msg4) is sent. Then, E substitutes the second part of the message $\{\{K'_a\}_{K'_t}\}_{K_{HV}}$ with $\{\{K_a\}_{K_t}\}_{K_{HV}}$. H decrypts $\{\{K_a\}_{K_t}\}_{K_{HV}}$, and re-encrypts the result $\{K_a\}_{K_t}$ with K_{EH}, which is the long-term key shared by E and H. When E receives (msg6) from V, she obtains

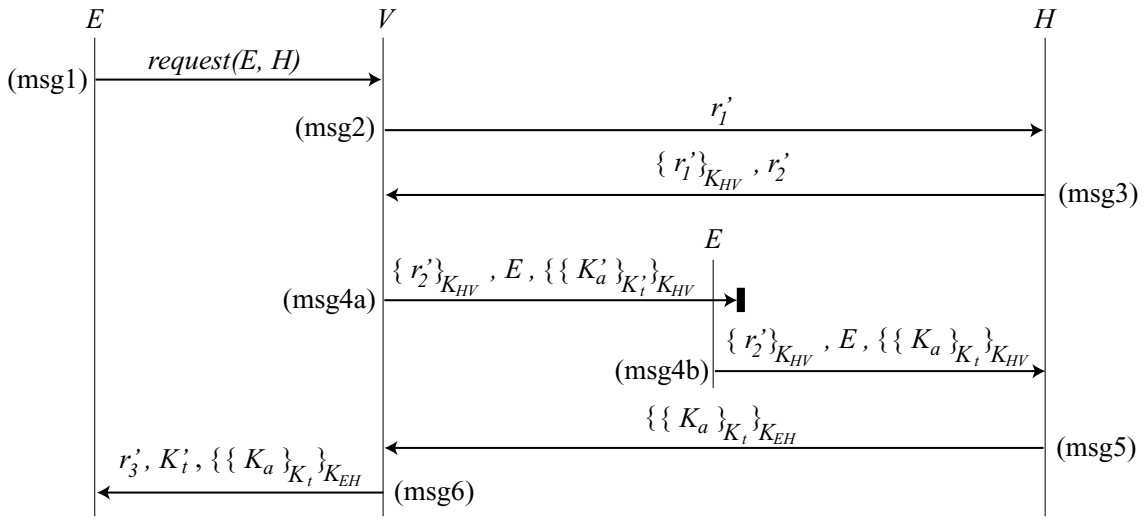

Figure 2.2: Scenario of Attack 1

K_a, since she knows both K_{EH} and K_t. Later, E can use K_a to impersonate U or V in the roaming-service-provision phase.

Obviously, Attack 1 is a coordinated activity of several physically dispersed malicious entities that we consider jointly to be the attacker. Two of these entities eavesdrop the communication between U and V, and between V and H, respectively, a third one starts the protocol with V, and a fourth one modifies (msg4) in transit between V and H. We assume that the attacker entities communicate with each other (possibly in a proprietary way). The attack requires that the intruder can associate messages, which are observed on different interfaces, to each other. In particular, E has to eavesdrop (msg4) of the original protocol run on the V–H interface, and (msg6) from the same protocol run on the V–U interface. Similarly, E has to catch and modify (msg4) on the V–H interface, which belongs to the protocol run initiated by E on the E–V interface. Considering that (msg4) contains the identifier of the initiator of the protocol, the problem of associating the right messages to each other does not seem to be too difficult.

Furthermore, the attack does not require the modification of messages on the air interface (between U and V), which would be quite difficult to do. Whereas, eavesdropping of messages sent over a wireless connection is considered to be rather simple, because of the broadcast nature of wireless communication. Eavesdropping and modifying messages that are sent between the visited network and the home network is technically possible, since these networks are usually connected via a fixed network, which is not necessarily secure[2]. Therefore, we believe that Attack 1 is valid and feasible.

However, the fixed network is often assumed to be physically protected and thus, the communication between the visited network and the home network is considered to be secure. If we accept this assumption, then neither eavesdropping nor modification of messages is possible between V and H, and Attack 1 no longer works. But we note that a small modification of the protocol would also prevent Attack 1, and we would not need the assumption of the security of the fixed network. We would prefer to make the protocol itself more robust than

[2]The authors of the protocol also considered this as a threat (see [SN97], Section II, threat 4).

to rely on strong assumptions.

2.3.2 Attack 2

Attack 2 is similar to Attack 1 with the difference that the home network H is not involved in it. In this attack too, the attacker E obtains the authentication key K_a of the roaming user U and the visited network V. Now, we assume that E is a legitimate, but malicious user, and that the home network of E is V. We also assume that E eavesdropped and recorded the protocol run, in which K_a has been established. Thus, E knows $\{\{K_a\}_{K_t}\}_{K_{HV}}$ and K_t. The attack scenario is illustrated in Figure 2.3.

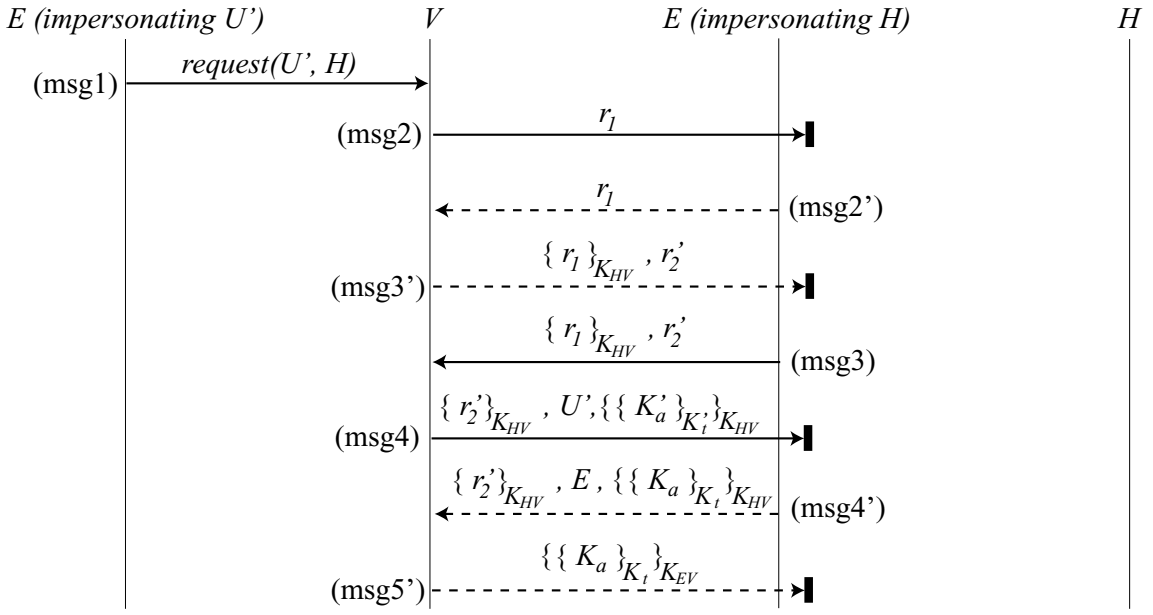

Figure 2.3: Scenario of Attack 2

The attacker E pretends to be a user U' from the home network H of the roaming user U. Note that U' is not necessarily equal to U. When V receives the false request of U' (msg1), it generates a random number r_1, and sends it to H as a challenge to authenticate H (msg2). This is intercepted and reflected back to V by E (msg2'). V interprets r_1 as the challenge of H to authenticate V in another session. Note that such challenges are regularly received by V form H, when users from the network of V roam in the network of H and request roaming service setup there. So V responds the challenge with $\{r_1\}_{K_{HV}}$ and sends another random number r_2' to H as a challenge to authenticate H (msg3'). This message is intercepted and reflected back to V by E as (msg3) of the first session. V accepts the message, since it is correct response to its challenge sent in (msg2). Then, V generates an authentication key K_a' and a temporary cipher key K_t' and sends (msg4) to H. E again intercepts the message, and reflects it back to V, but before doing so, it replaces U' with E, and $\{\{K_a'\}_{K_t'}\}_{K_{HV}}$ with $\{\{K_a\}_{K_t}\}_{K_{HV}}$ (msg4'). V interprets the message as the fourth message of the second session. Since it contains a correct response to its challenge sent in (msg3'), it accepts the message, and sends (msg5') to H. E can now decrypt message (msg5'), and obtain the key K_a. All the notes on the required capabilities of the attacker described in Attack 1 apply here as well.

2.3.3 Attack 3

This attack, which enables an attacker E to impersonate the visited network V to the roaming user U, exploits the fact that U does not receive any fresh message in the protocol. Attack 3 is essentially the same as the attack against the Needham-Schroeder protocol [NS78] described in [DS81]. In Attack 3, we assume that K_a^* is a compromised, old, authentication key, and that the attacker E recorded the protocol run that established K_a^*. Thus, E possesses the old authentication key K_a^*, the corresponding temporary cipher key K_t^*, and the ciphered message $\{\{K_a^*\}_{K_t^*}\}_{K_{UH}}$. The attack scenario is illustrated in Figure 2.4.

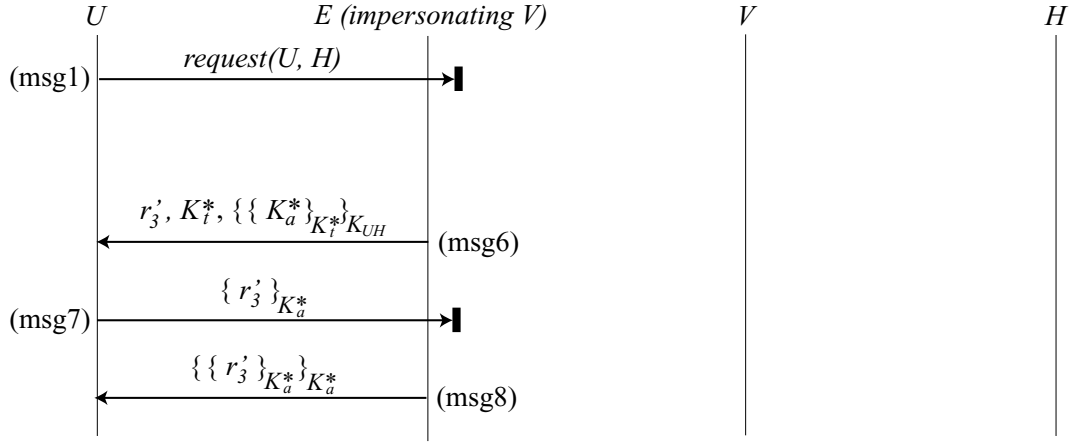

Figure 2.4: Scenario of Attack 3

When U starts a new instance of the protocol, E intercepts the request, and plays back (msg6) from the old protocol[3]. U thinks that the authentication key is K_a^*, so she sends $\{r_3'\}_{K_a^*}$ to V. This message is intercepted by E. E generates the last message $\{\{r_3'\}_{K_a^*}\}_{K_a^*}$, and sends it to U. Notice that neither the visited network nor the home network was involved in the attack.

In Attack 3, the attacker has to be able to play the role of the visited network and to make the roaming user send messages to her instead of the visited network. Although this requirement seems to be strong, satisfying it is not impossible. There are commercially available devices called "IMSI catchers" [Fed99], the functionality of which is very similar to that the attacker needs in Attack 3. From the side of the mobile phone, an IMSI catcher behaves as a base station of the mobile network. A mobile phone, which is closer to an IMSI catcher than to a base station, can be coerced by the IMSI catcher to establish a connection with it rather than with the base station. The mobile phone does not even know that it talks with an IMSI catcher instead of a base station. The IMSI catcher can relay communication between the mobile phone and the base station, and stay unnoticed. We believe that such a device would enable Attack 3.

[3] E may change the random number r_3 to r_3'.

2.4 Correction

In this section, we re-design the protocol. This will be a demonstration of the usage of the synthesis rules introduced in the previous chapter (see also Appendix A) and our systematic protocol construction approach. During the design, we will keep the assumptions identified in Section 2.2 in mind, although we will not strictly stick to them. We will use them to guide the selection of the next synthesis rule to be applied when there are several rules available. We give detailed illustration for the re-construction of the second sub-protocol (i.e., the setup of the session key), since this is the most important part of the protocol.

2.4.1 Re-construction of the second sub-protocol

As we said earlier, the goal of the second sub-protocol is to establish a secret session key K_a between U and V, which can be represented by the formulae: $U \models ((K_a \sim \{U, V\}) \wedge \sharp(K_a))$ and $V \models ((K_a \sim \{U, V\}) \wedge \sharp(K_a))$. Recall that in the Suzuki-Nakada protocol, V generates K_a, and thus, we can take $V \models ((K_a \sim \{U, V\}) \wedge \sharp(K_a))$ as an assumption. We will keep this feature of the protocol. This means that the goal of the second sub-protocol reduces to the single formula:

$$U \models \gamma \tag{2.1}$$

where $\gamma = ((K_a \sim \{U, V\}) \wedge \sharp(K_a))$.

We start the synthesis from the above goal (2.1). Using (S2), we replace it with the new goals:

$$U \models (V \models \gamma) \tag{2.2}$$

$$U \models ((V \models \gamma) \Rightarrow \gamma) \tag{2.3}$$

We will consider (2.3) as an assumption. This means that we assume that U believes that V is competent in generating the session key. This makes sense since V represents a network operator. In addition, the original protocol made the same assumption. We continue the synthesis with (2.2). Since (*S13) can be applied to this goal, we must apply it. The new goals are:

$$U \models (V \models \gamma) \tag{2.4}$$

$$V \models \gamma \tag{2.5}$$

We have already considered (2.5) as an assumption. From (2.4), using (S3), we obtain the following new goals:

$$U \models (V \mid\sim \gamma) \tag{2.6}$$

$$U \models ((V \mid\sim \gamma) \Rightarrow (V \models \gamma)) \tag{2.7}$$

We will consider (2.7) as an assumption. This means that we assume that U believes that V is honest with respect to statements about the session key. Again, the same assumption was made in the original protocol. Next, we should apply (*S14) to (2.6), but it does not result in any new goals, since V knows the identifiers of U and V, and the session key K_a (i.e., it can say γ). We apply (S5) to (2.6), and we obtain the following new goals:

$$U \models (V \mid\sim \gamma; r_0) \tag{2.8}$$

$$U \models \sharp(\gamma; r_0) \tag{2.9}$$

The goal (2.9) can be replaced with the new goal:

$$U \models \sharp(r_0) \tag{2.10}$$

which we will consider as an assumption. We continue with the goal (2.8), to which we must apply (*S15). We obtain the following new goals:

$$U \models (V \mid\sim \gamma; r_0) \tag{2.11}$$
$$V \lhd r_0 \tag{2.12}$$

We will consider (2.12) as a message sending step. We continue the synthesis with the other goal (2.11). Using (S2) again, we obtain the following new goals:

$$U \models (H \models (V \mid\sim \gamma; r_0)) \tag{2.13}$$
$$U \models ((H \models (V \mid\sim \gamma; r_0)) \Rightarrow (V \mid\sim \gamma; r_0)) \tag{2.14}$$

We will consider (2.14) as an assumption. This means that we assume that U believes that H is competent in recognizing messages that are sent by V. Next, we must apply (*S13) to (2.13). The new goals that we obtain are:

$$U \models (H \models (V \mid\sim \gamma; r_0)) \tag{2.15}$$
$$H \models (V \mid\sim \gamma; r_0) \tag{2.16}$$

For the moment we put aside the goal (2.16); we will return to it later. We continue the synthesis with the other goal (2.15), which we can replace with the following new goals using (S3):

$$U \models (H \mid\!\sim (V \mid\sim \gamma; r_0)) \tag{2.17}$$
$$U \models ((H \mid\!\sim (V \mid\sim \gamma; r_0)) \Rightarrow (H \models (V \mid\sim \gamma; r_0))) \tag{2.18}$$

We will consider (2.18) as an assumption. This means that we assume that U believes that H honestly relays messages between V and U. Next, we must apply (*S14) to the goal (2.17). We obtain the following new goals:

$$U \models (H \mid\!\sim (V \mid\sim \gamma; r_0)) \tag{2.19}$$
$$H \lhd \gamma \tag{2.20}$$
$$H \lhd r_0 \tag{2.21}$$

We will take care of the goals (2.20) and (2.21) a bit later. We continue the synthesis by replacing the goal (2.19) with the following new goals using (S5):

$$U \models (H \mid\sim (V \mid\sim \gamma; r_0); r_0) \tag{2.22}$$
$$U \models \sharp((V \mid\sim \gamma; r_0); r_0) \tag{2.23}$$

The goal (2.23) can be replaced by $U \models \sharp(r_0)$, which we have already considered as an assumption. Thus, we can continue with the goal (2.22). We should apply (*S15) to this goal, but it does not result in any new goals, since H knows the identifier of V, and we have

already considered $H \lhd \gamma$ and $H \lhd r_0$ as goals (see (2.20) and (2.21) above). So we replace (2.22) with the following new goals using (S8):

$$U \models (U \lhd C_{UH}((V \mid\sim \gamma; r_0); r_0)) \tag{2.24}$$

$$U \models (s(C_{UH}) = \{H\}) \tag{2.25}$$

We will take the goal (2.25) as an assumption. This means that we assume that there exist a channel C_{UH} such that U believes that only H sends messages through C_{UH}. Clearly, this assumption is related to the existence of the long-term key K_{UH} shared between U and H that can be used to implement C_{UH}. From (2.24), we obtain the following new goals using (S10):

$$U \lhd C_{UH}((V \mid\sim \gamma; r_0); r_0) \tag{2.26}$$

$$U \in r(C_{UH}) \tag{2.27}$$

Taking into account the discussion on K_{UH} and C_{UH} above, we can consider (2.27) as an assumption. As for (2.26), we will consider it as a message sending step.

Now, we return to the goals that we have put aside. These are (2.16), (2.20), and (2.21). We start with the goal (2.16). We should first apply (*S15), but it does not result in any new goals. Therefore, we can replace (2.16) with the following new goals obtained by using (S8):

$$H \models (H \lhd C_{HV}(\gamma; r_0)) \tag{2.28}$$

$$H \models (s(C_{HV}) = \{V\}) \tag{2.29}$$

The goal (2.29) will be considered as an assumption. Thus, we assume that there exists a channel C_{HV} such that H believes that only V sends messages on C_{HV}. Such a channel can easily be implemented based on the long-term key K_{HV} shared by H and V. Before going on with the goal (2.28), we consider the goals (2.20) and (2.21) that we have put aside. Using (S11), we can replace both with the goal:

$$H \lhd \gamma; r_0 \tag{2.30}$$

Now, both (2.28) and (2.30) can be replaced with the goals:

$$H \lhd C_{HV}(\gamma; r_0) \tag{2.31}$$

$$H \in r(C_{HV}) \tag{2.32}$$

by using either (S10) or (S12), respectively. We will consider the goal (2.32) as an assumption, while (2.31) will be a message sending step.

Since there is no more goal to be considered, the synthesis is finished. We obtained the following set of assumptions:

(A1) $V \models \gamma$ where $\gamma = ((K_a \sim \{U, V\}) \ \wedge \ \sharp(K_a))$

V believes that K_a is a fresh session key between U and V. This is a reasonable assumption, since V generates K_a.

(A2) $H \in r(C_{HV})$ and $H \models (s(C_{HV}) = \{V\})$

There exists a channel C_{HV} such that H can read from C_{HV}, and H believes that only V sends messages via C_{HV}. Such a channel can easily be implemented using the long-term key K_{HV} shared by H and V.

(A3) $U \in r(C_{UH})$ and $U \models (s(C_{UH}) = \{H\})$

There exists a channel C_{UH} such that U can read from C_{UH}, and U believes that only H sends messages via C_{UH}. Such a channel can easily be implemented using the long-term key K_{UH} shared by U and H.

(A4) $U \models ((V \models \gamma) \Rightarrow \gamma)$ and $U \models ((V \mid\sim \gamma) \Rightarrow (V \models \gamma))$

U believes that V is competent in generating session keys and honestly makes statements about those keys.

(A5) $U \models ((H \models (V \mid\sim X)) \Rightarrow (V \mid\sim X))$ and $U \models ((H \mid\sim (V \mid\sim X)) \Rightarrow (H \models (V \mid\sim X)))$

U believes that H is competent in recognizing messages sent by V and honestly relays those messages to U.

(A6) $U \models \sharp(r_0)$

U believes that r_0 is fresh. For instance, r_0 could be a random number freshly generated by U.

and the following set of message sending steps:

$$V \lhd r_0$$
$$H \lhd C_{HV}(\gamma; r_0)$$
$$U \lhd C_{UH}((V \mid\sim \gamma; r_0); r_0)$$

The above example illustrates the rather tedious process of using the synthesis rules. We will not detail further how the first and the third sub-protocol can be re-constructed. Instead, we re-state the goals of these sub-protocols, and give the assumptions and the message sending steps that we obtained during their synthesis.

2.4.2 Re-construction of the first sub-protocol

For the first sub-protocol, we keep the goals $V \models (H \mid\sim r_1)$ and $H \models (V \mid\sim r_2)$. This means that we only want the first sub-protocol to check the presence of the parties. It would be possible to tie mutual authentication between H and V to a particular protocol run[4], but this would not have any effect on the second sub-protocol, and therefore we do not pursue this possibility further.

The assumptions we obtained are:

(A7) $V \in r(C'_{HV})$ and $V \models (s(C'_{HV}) = \{H\})$

There exists a channel C'_{HV} such that V can read from C'_{HV} and V believes that only H sends messages via C'_{HV}. Such a channel can be implemented using the long-term key K_{HV} shared by H and V.

(A8) $V \models \sharp(r_1)$

V believes that r_1 is fresh. r_1 could be a random number freshly generated by V.

[4]One possible way to do so would be to change the goals to $V \models (H \mid\sim r)$ and $H \models (V \mid\sim r)$, where r is a value that identifies the particular protocol run. In practice, r can be computed from random values contributed by both parties.

(A9) $H \in r(C''_{HV})$ and $H \models (s(C''_{HV}) = \{V\})$

There exists a channel C''_{HV} such that H can read from C''_{HV} and H believes that only V sends messages via C''_{HV}. Such a channel can be implemented using the long-term key K_{HV} shared by H and V.

(A10) $H \models \sharp(r_2)$

H believes that r_2 is fresh. r_2 could be a random number freshly generated by H.

The message sending steps for the first sub-protocol are:

$$H \lhd r_1$$
$$V \lhd C'_{HV}(r_1); r_2$$
$$H \lhd C''_{HV}(r_2)$$

2.4.3 Re-construction of the third sub-protocol

For the third sub-protocol, we slightly change the goals. Recall that the purpose of the third sub-protocol of the original Suzuki-Nakada protocol was mutual authentication of U and V. This can be represented by logical formulae of the form $U \models (V \mid\sim \ldots)$ and $V \models (U \mid\sim \ldots)$. Note that $U \models (V \mid\sim \gamma)$ can be derived from the re-constructed second sub-protocol, which means that V is already authenticated to U, and there is no reason to authenticate it again in the third sub-protocol. On the other hand, no formula of the form $V \models (U \mid\sim \ldots)$ can be derived from the first and second sub-protocols. Therefore, we set the goal of the third sub-protocol to $V \models (U \mid\sim ack)$, where ack represents a value (acknowledgement) known to both U and V a priori. Starting from this goal, we obtained the following assumption:

(A11) $V \in r(C_a)$ and $V \models ((s(C_a) = \{U\}) \ \wedge \ \natural(C_a))$

There exists a channel C_a such that V can read from C_a, and V believes that only U sends messages via C_a and C_a is timely. This channel is implemented using the freshly established session key K_a.

and message sending step:

$$V \lhd C_a(ack)$$

Alternatively, we could consider that the goal of the third sub-protocol is explicit key confirmation. This goal can be represented by the formulae $U \models (V \models \gamma)$ and $V \models (U \models \gamma)$, where $\gamma = ((K_a \sim \{U, V\}) \ \wedge \ \sharp(K_a))$. Again, $U \models (V \models \gamma)$ can be derived from the re-constructed second sub-protocol, but $V \models (U \models \gamma)$ can be derived neither from the second nor from the first sub-protocol. Therefore, we set the goal of the third sub-protocol to $V \models (U \models \gamma)$. Staring from this goal, we obtained the following two assumptions:

(A11') $V \in r(C_a)$ and $V \models ((s(C_a) = \{U\}) \ \wedge \ \natural(C_a))$

This assumption is the same as (A11).

(A12') $V \models ((U \mid\sim \gamma) \Rightarrow (U \models \gamma))$

V believes that U honestly makes statements about the session key (i.e., if U says that K_a is a fresh session key between U and V, then U actually believes that).

The message sending step that we obtained for the alternative goal is the following:

$$V \lhd C_a(\gamma)$$

2.4.4 Implementation of the abstract protocol

Putting together the message sending steps that we obtained for the sub-protocols we arrive to the following abstract protocol:

Abstract corrected Suzuki-Nakada protocol	
(msg1)	$V \lhd r_0$
(msg2)	$H \lhd r_1$
(msg3)	$V \lhd C'_{HV}(r_1); r_2$
(msg4)	$H \lhd C''_{HV}(r_2); C_{HV}(\gamma; r_0)$
(msg5)	$U \lhd C_{UH}((V \mid\sim \gamma; r_0); r_0)$
(msg6)	$V \lhd C_a(ack)$ or $V \lhd C_a(\gamma)$

Given the abstract protocol and the set of assumptions about the channels that it uses, we can now suggest implementations. In fact, there are many ways to translate the channels into the use of cryptographic primitives. For instance, an implementation that uses symmetric-key encryption (and thus, in this sense, resembles the original protocol) could look like this:

A correction of the Suzuki-Nakada protocol		
(msg1)	$U \to V:$	$request(U, H),\ r_0$
(msg2)	$V \to H:$	r_1
(msg3)	$H \to V:$	$\{H,\ r_1\}_{K_{HV}},\ r_2$
(msg4)	$V \to H:$	$\{V,\ r_2\}_{K_{HV}},\ \{U,\ K_a,\ r_0\}_{K_{HV}}$
(msg5)	$H \to V:$	$\{V,\ K_a,\ r_0\}_{K_{UH}}$
(msg6)	$V \to U:$	$\{V,\ K_a,\ r_0\}_{K_{UH}}$
(msg7)	$U \to V:$	$\{ack\}_{K_a}$

Here, each channel is implemented by symmetric key encryption with some further constraint. A general assumption that we make is that the encryption primitive provides message authentication as well. In general this is not true, but it can be achieved by putting enough redundancy into the messages (e.g., by appending the hash value of the message to the message before encryption).

The channel C'_{HV} is realized by encryption with K_{HV} with the constraint that the first field of every encrypted message must be the name of H, and the second field must be a random number. Requiring that the first field is the name of H ensures that the response to the challenge of V in (msg3) cannot be confused with a response to a similar challenge of H in another run of the protocol. Thus, reflection attacks are excluded. Similarly, C''_{HV} is realized by encryption with K_{HV} with the constraint that the first field of every encrypted message must be the name of V, and the second field must be a random number.

The channel C_{HV} is implemented by encryption with K_{HV} with the constraint that the first field of every encrypted message must be the identifier of a user, the home network of

which is H, and the second and third fields must be a session key and a random number, respectively. This is fine as long as no user can have two home networks, which we take as an assumption.

C_{UH} is realized by encryption with K_{UH} with the constraint that only H encrypts messages with this key. Finally, C_a is realized by encryption with K_a with the constraint that only U encrypts messages with this key. Note, however, that since K_a is a session key between U and V, it is very unlikely that only U encrypts messages with K_a. Therefore, depending on the further usage of K_a, some additional constraint might be necessary. One option would be, for instance, to require that neither U nor V ever sends the special message ack to the other party encrypted with K_a after the completion of the protocol.

In addition to the channels, we should say some words about the implementation of the idealized messages that contain the formula γ. Actually, γ refers to U, V, and K_a, so it should be possible to associate these entities to the appropriate messages of the implementation. In the second part of (msg4), U and K_a are explicitly mentioned, and V can be inferred from the context, namely, from the source set of the channel, on which the message is sent. Similarly, in (msg5) and (msg6), V and K_a are explicitly mentioned, and U can be inferred from the context, namely, from the fact that the message is encrypted with K_{UH}, and the destination of such a message can only be U.

It is easy to verify that the above correction of the Suzuki-Nakada protocol resists against all the attacks described in Section 2.3. An interesting thing is that the correction that we obtained using the synthesis rules resists against Attack 1 and Attack 2, although these attacks are interleaving attacks (i.e., attacks that involve simultaneous runs of multiple instances of the same protocol), and they cannot be represented in our simple model, where we distinguish only two epochs: past and present. This illustrates that our approach is useful in designing protocols that resist against subtle attacks.

In addition, the correction is simpler than the original protocol in two senses. First, it uses one less message flows. Second, it does not involve the use of the temporary cipher key K_t. Note that the use of K_t in the original protocol is somewhat paradoxical. The authors of [SN97] mention that "it is possible that the entities concerned take illegal actions in roaming-service provision. Therefore, it is desirable not to leak the authentication keys needed for their authentication to the other networks." Thus, it seems that the role of K_t is to hide K_a from the home network H. However, K_t is sent in clear to U in a later step of the protocol. This means that if H wants, then it can easily obtain K_a: an agent can eavesdrop communications within the visited network, and send temporary cipher keys to the home network, which can use them to obtain authentication keys. This unclear feature is automatically eliminated from the corrected protocol.

Clearly, implementations different from the one described above are possible too. Note, for instance, that the channels C'_{HV}, C''_{HV}, and C_a do not carry any secret information. In addition, as long as the implementation of γ does not reveal the session key K_a, we do not need to restrict either who can read from C_{HV} and C_{UH}. These observations lead to the following implementation:

Another correction of the Suzuki-Nakada protocol		
(msg1)	$U \to V:$	$request(U, H),\ r_0$
(msg2)	$V \to H:$	r_1
(msg3)	$H \to V:$	$mac_{K_{HV}}(H,\ r_1),\ r_2$
(msg4)	$V \to H:$	$mac_{K_{HV}}(V,\ r_2),\ \{K_a\}_{K_{HV}},\ mac_{K_{HV}}(U,\ h(K_a),\ r_0)$
(msg5)	$H \to V:$	$\{K_a\}_{K_{UH}},\ mac_{K_{UH}}(V,\ h(K_a),\ r_0)$
(msg6)	$V \to U:$	$\{K_a\}_{K_{UH}},\ mac_{K_{UH}}(V,\ h(K_a),\ r_0)$
(msg7)	$U \to V:$	$mac_{K_a}(ack)$

where $mac_k(m)$ denotes message m (in clear) together with a message authentication code computed on m with the key k, and h is a cryptographic hash function. Due to the properties of cryptographic hash functions, $h(K_a)$ does not reveal K_a, and at the same time it is a good reference to K_a. The advantage of this implementation is that it uses less encryption operations. Another advantage is that only a short message (namely, the key K_a) is encrypted, thus the cipher can be used in electronic code book (ECB) mode [MvOV97], which might have been a design criterion for the original protocol, although it was not mentioned in [SN97].

2.5 Summary

The goal of this chapter was to demonstrate our systematic approach to protocol construction and the usage of the logic of channels introduced in Chapter 1. For this reason, we explained some weaknesses in an authenticated key transport protocol proposed in the literature in terms of our logic, and re-designed the protocol from scratch using our synthesis rules. We showed two possible implementations of the resulting abstract protocol. The protocols that we obtained are robust and resist against various attacks, including even interleaving attacks. Furthermore, they are intuitively simpler and use less messages than the original protocol. Another important output of our systematic protocol construction method was a set of assumptions upon which the correctness of the protocol depends.

Publication: [BGSW00]

Part II

Rational exchange protocols

Chapter 3

An informal overview of the concept of rational exchange

3.1 Introduction

There are many applications where two remote parties have to exchange digital items via a communication network. Examples include

- *electronic contract signing* – where the contracting parties have to exchange non-repudiable commitments (typically implemented by digital signatures) to the contract text;
- *certified electronic mail* – where the sender and the recipient have to exchange a mail for an acknowledgement of receipt; and
- *purchase of network delivered services* – where the user and the service provider have to exchange a payment for a service (e.g., a music file, news, etc.).

An inherent problem in these applications is that a misbehaving party may bring the other party in a disadvantageous situation. For instance, a service provider may deny service provision after receiving payment from a user. This may discourage the parties and hinder otherwise desired transactions. We will refer to this problem as the *exchange problem*.

The best known approach to solve the exchange problem is to use a two-party *fair exchange* protocol. Such a protocol guarantees for a correctly behaving party (i.e., a party that follows the protocol faithfully) that it cannot suffer any disadvantages – no matter whether the other party behaves correctly or tries to cheat. Thus, executing the protocol faithfully is safe for both parties.

Two-party fair exchange has been extensively studied by the research community. Early work has resulted in fair exchange protocols that are based on gradual secret release schemes (e.g., [Cle90]). In these protocols, the items of the parties are exchanged in small pieces, typically bit-by-bit in such a way that the computational effort required from the parties to obtain each other's remaining bits is approximately equal at any stage during the execution of the protocol. Although they are theoretically important, these protocols are typically not considered to be suitable for practical applications.

Practical fair exchange protocols use a trusted third party that assists the main parties to accomplish fair exchange. There are *off-line* and *on-line* protocols proposed in the literature. Off-line protocols (e.g., [ASW97, BDM98, ASW00]) require the trusted third party

to participate actively only when one of the main parties has decided to invoke the third party. These protocols are also called *optimistic*, because they are most efficient in the hoped-for case when the third party does not need to be invoked. On-line protocols (e.g., [Ket95, DGLW96, Tyg96, ZG96, FR97]) require the trusted third party to participate in every exchange.

In principle, fair exchange protocols provide an ultimate solution to the exchange problem. However, there are applications where fair exchange cannot be used for technical reasons. An example is when at least one of the parties is permanently disconnected from the trusted third party. Consider, for instance, a system which consists of a set of vending machines that sell electronic tickets (e.g., tickets for movies or public transportation) and a set of users that are equipped with small portable devices (e.g., smart-cards or PDAs) that can be used to buy and store those tickets. This system can work in the following way: Users can load their portable devices with electronic coins at home or in a bank. When a user wants to buy a ticket from a vending machine, the portable device of the user and the vending machine execute a transaction in which the appropriate number of coins are exchanged for the desired ticket. The ticket is then stored on the portable device, which can later output it if necessary. The communication between the portable device and the vending machine can be based on short range communication technology (e.g., infrared or Bluetooth, or if the portable device is a smart-card, then it can simply be inserted in the smart-card reader built into the vending machine). Now, assume that the portable device has no other communication capabilities (for smart-cards this is definitely the case, and for most of today's PDAs as well). This means that the portable device cannot directly connect to the network, but only via the vending machine. In other words, if a fair exchange protocol was used in this scenario, then the communication between the portable device and the trusted third party would be fully under the control of the vending machine. We are not aware of any fair exchange protocol that can provide fairness to the portable device in these circumstances.

Therefore, we believe that it makes sense to study alternative approaches to alleviate the exchange problem. Such an alternative approach is *rational exchange*. To the best of our knowledge, this term was first used by Syverson in [Syv98]. Roughly, a rational exchange protocol ensures that a misbehaving party cannot gain any advantages by the misbehavior. Therefore, rational (self-interested) parties have no reason to deviate from the protocol. This means that cheating should happen only rarely.

As opposed to fair exchange, rational exchange has received less attention from the research community. We are aware only of a few rational exchange protocols proposed in the literature [Jak95, San97, Syv98]. These protocols seem to provide weaker guarantees than fair exchange protocols, but at the same time, they are also less complex. Therefore, rational exchange protocols can be viewed as a trade-off between complexity and true fairness, and as such, they may provide interesting solutions to the exchange problem in certain applications.

Our goal in the second part of this thesis is to study the concept of rational exchange and its relationship with fair exchange. For this reason, in the next chapter, we will introduce a formal model of exchange protocols, which is based on game theory. We will use this model, to give a formal definition for rational exchange relating it to the concept of Nash equilibrium in games. Furthermore, we will show how the concept of fair exchange can be defined in our model. This will allow us to compare the two notions and to prove formally (within our model) that fair exchange implies rational exchange, but the reverse is not true. To the best of our knowledge, such a study has not been performed yet in the literature.

However, before starting with the formal treatment, in the rest of this chapter, we give an example for a rational exchange protocol. Furthermore, we show how the main idea of this protocol can be used to improve a family of micropayment schemes with respect to fairness without substantial loss in efficiency in most practical cases. Our goal with these examples is to further illustrate the usefulness of the concept of rational exchange and to prepare the grounds for the formal definitions of the next chapter.

3.2 An example: a rational payment protocol

Let us consider the following payment protocol, which can be used for transferring payment from a user U to a vendor V in exchange for some service provided by V to U. Besides the main parties U and V, the protocol uses a trusted third party, the bank B.

A rational payment protocol
$U \to V:\ m_1 = (U, V, tid, val, h(rnd), \sigma_U(U, V, tid, val, h(rnd)))$
$V \to U:\ m_2 = srv$
$U \to V:\ m_3 = rnd$
if V received m_1 and m_3:
$\quad V \to B:\ m_4 = (m_1, rnd, \sigma_V(m_1, rnd))$
if V received only m_1:
$\quad V \to B:\ m_4' = (m_1, \sigma_V(m_1))$

We assume that before starting the protocol, U and V have already agreed on the details of the transaction. In particular, we assume that U and V agreed on the value *val* of the payment that U is supposed to pay to V, and the description of the service *srv* that V is supposed to provide to U. In addition, we assume that U and V also agreed on a fresh transaction identifier *tid*.

U starts the protocol by generating a random number *rnd* and computing its hash value $h(rnd)$. Then, she generates the digital signature $\sigma_U(U, V, tid, val, h(rnd))$, and sends m_1 to V. When V receives m_1, it provides the service to U (represented by sending $m_2 = srv$). If U is satisfied, then she reveals the random number *rnd* to V. If V received m_1 and m_3, then it generates the digital signature $\sigma_V(m_1, rnd)$, and sends m_4 to B. If V received only m_1, then it generates the digital signature $\sigma_V(m_1)$, and sends m_4' to B.

Upon reception of m_4, B verifies that it has never processed a transaction between U and V with the transaction identifier *tid* before by looking up its internal database, where it logs all processed transactions. Then, it verifies that the hash value of *rnd* equals the hash value in m_1, and the digital signatures of U and V in the message are valid. If these verifications are successful, then it logs the transaction, and transfers the value *val* from the account of U to the account of V. Upon reception of m_4', B performs similar verifications, and if these are successful, then B logs the transaction, and it debits U's account with the value *val*, but it does *not* credit V's account. This leaves B with a surplus of the value *val*. This surplus is handled according to some policy (e.g., it can be distributed to charity purposes). This policy is verified and the respect for it is controlled by independent law enforcement organizations, thus rendering collusion between the user and the bank, as well as between the vendor and the bank difficult.

What does this protocol achieve? It is clear that it does not provide fairness, since any of the parties can bring the other, correctly behaving party in a disadvantageous situation. For instance, U can refuse to reveal *rnd* to V. In this case, V can send only m'_4 to B, which means that V does not get paid for the service that it provided. Similarly, V can refuse the provision of the service after having received m_1 from U. In this case, V can send m'_4 to B, which means that U's account is debited, although she did not receive any service.

On the other hand, note that none of the parties gain any (financial) advantages by cheating. The reason is that V cannot obtain any money without providing the service to U (since U reveals *rnd* only if she received the service); and U cannot receive any service without being charged (since V provides the service only if it received m_1, in which case it can send at least m'_4 to B). This means that none of the parties has an incentive to deviate from the protocol. In a word, the protocol seems to be a rational exchange protocol. Indeed, in Chapter 5, we will prove this formally within the model that we will introduce in Chapter 4.

We would also like to point out that the protocol uses only a trusted third party (the bank) that is needed anyway in order to maintain the accounts of the users and the vendors. In addition, the bank performs essentially the same operations as it would perform in any check based payment systems.

3.3 An application: Removing the financial incentive to cheat in micropayment schemes

Micropayment schemes (e.g., [Ped95, GMA+95, AMS95, RS96, HSW96] and more recently [MPM+98]) are electronic payment schemes explicitly developed for very low value payment transactions, such as payment for information on the World Wide Web and payment for each second of a phone call. The main design goal of micropayment protocols is efficiency. Reaching this goal requires that communication and processing costs of micropayments be kept as low as possible, otherwise these costs may exceed the value of the payment itself, and thus, applying the micropayment scheme would not be economical. Other properties, and in particular fairness, are sacrificed to efficiency.

Since micropayment schemes are not fair, cheating is always possible: If the payer has to move first, then the payee can cheat by not providing the service after a micropayment has been received, otherwise, if the payee has to move first, then the payer can cheat by not sending the micropayment for the received service. It is argued that this potential misbehavior of the parties is tolerable, since the potential loss is very low. While this is true considering a single transaction, it might be a problem considering the whole system and longer time periods. In order to illustrate this, let us consider a service provider that persistently cheats by stealing 1 cent in each transaction. This service provider can earn more than 1 million dollars in a year assuming that it has about 300000 transactions per day. As a rough guide, a rather small telecommunication network operator with 50000 subscribers processes 300000 phone calls per day. One can say that a service provider risks to obtain a bad reputation by such a misbehavior, and therefore, it will not cheat. This may be true, however, as we will show in this section, such a misbehavior can also be made uninteresting for the service provider based on a technical approach.

In principle, a fair exchange protocol could be used to exchange micropayments for services. However, this would be too expensive (inefficient) as a solution for micropayments. Now we show, how a rational exchange protocol may provide an elegant solution in this situa-

tion. We describe how a family of micropayment schemes can be improved and made rational (so that cheating becomes uninteresting for any of the parties) at virtually no cost and loss in efficiency. Our solution is based on the main idea of the rational payment protocol of the previous section.

3.3.1 Original micropayment scheme

We only consider micropayment schemes where payment is based on the successive release of elements in a chain of cryptographic hash values (e.g., [Ped95, AMS95, RS96, HSW96, MPM+98]). In particular, we will illustrate our ideas by extending the PayWord system [RS96]. Other members of the same family can be extended in a similar way.

There are three roles in PayWord: the user U, the vendor V, and the broker B. Each user is registered with at least one broker. This relationship is represented by a PayWord certificate issued by the broker to the user. User U's certificate C_U contains the broker's name B, the user's name U, a public key K_U, such that the corresponding private key K_U^{-1} is known exclusively to U, and other data that is not relevant to our discussion. C_U is signed by B.

When U wants to buy some services from V, she generates a fresh chain of hash values $w_0, w_1, \ldots, w_n$ by picking w_n at random and then computing $w_i = h(w_{i+1})$ for $i = n-1, n-2, \ldots, 0$, where h denotes a publicly known, cryptographically strong one-way hash function and n is chosen by U. $w_1, w_2, \ldots, w_n$ are called *paywords*. w_0 is not a payword itself; it is called the root of the payword chain. Then, U signs a commitment M to the payword chain with the private key K_U^{-1}. M contains the vendor's name V, the user's certificate C_U, the root of the payword chain w_0, and other data that is not relevant to our discussion. This commitment authorizes B to pay V for any of the paywords $w_1, w_2, \ldots, w_n$ that V redeems with B later. U sends M to V. After verification of the signature on M, V stores M for later use.

The i-th micropayment from U to V consists of the pair (w_i, i). This can be verified by V using w_{i-1}, which is known from the previous micropayment or from the commitment in case of $i = 1$. A service session consists of a sequence of micropayments:

Service session in PayWord	
$U \to V$:	$w_1, 1$
$V \to U$:	*first piece of the service*
$U \to V$:	$w_2, 2$
$V \to U$:	*second piece of the service*
...	
$U \to V$:	w_ℓ, ℓ
$V \to U$:	*last piece of the service*

where $\ell \leq n$.

After service provision, V contacts B, which it knows from C_U, and presents M and the last micropayment (w_ℓ, ℓ). B checks the signature on M, and verifies if ℓ applications of h on w_ℓ gives w_0, which is in M. If all the verifications are successful, then B pays V the amount corresponding to ℓ paywords and charges that amount to the billing account of U.

PayWord is very economical in terms of the number of public key operations performed. In particular, the user has to sign only one commitment, which in turn allows a potentially large number of self-authenticating micropayments to be made to the same vendor. This is achieved by the application of the cryptographically strong one-way hash function h, which guarantees that only the user can generate the next payword (assuming that the user keeps w_n secret, which is in her own interest). Furthermore, paywords do not need to be encrypted, since they cannot be re-used and they represent financial value only for the vendor specified in the commitment to which the paywords belong.

As discussed before, PayWord does not provide fairness. The vendor may cheat the user by sending an unexpected service or nothing at all. If such a misbehavior is detected by the user, then she can stop sending more paywords, but she still loses the last one already sent. Since a payword has a very low value, this does not cause too much damage for the user. A persistently cheating vendor, however, can earn a substantial amount of money in this way.

3.3.2 Improved micropayment scheme

We will now present our extension to PayWord that removes the financial incentive to cheat and, thus, makes the misbehavior described above practically futile. We modify the original scheme only slightly and show that efficiency does not decrease substantially in most practical cases. Our basic idea stems from the rational payment protocol described in Section 3.2: We double the size of the hash chain and let a payword consist of two consecutive hash values. Intuitively, these can be thought of as two half-paywords. The first half-payword is sent to the vendor before the service provision and the second half is sent after the service has been provided[1]. Thus, the vendor can redeem the full payword only if it has provided the service. Now, this gives an advantage to the user, who can refuse to send the second half-payword hoping that she can escape from paying for the received service. In order to deter the user from this misbehavior, we let the broker charge the full value of a payword to the user's account if the vendor presents the first half-payword (which leaves the broker with a surplus of the value of one payword). This makes cheating uninteresting to the user, because she has to pay, even though the vendor cannot get the money. The surplus of the broker is handled in a similar way as in the payment protocol of Section 3.2.

We modify only the micropayment protocol and the rules of the broker. When user U wants to buy some services from vendor V, she generates a fresh chain of hash values $w'_0, w_1, w'_1, w_2, w'_2, \ldots, w_n, w'_n$ by picking w'_n at random and then computing $w_i = h(w'_i)$ and $w'_{i-1} = h(w_i)$ for $i = n, n-1, \ldots, 1$. The root of the chain is now w'_0 and U puts this value in the commitment, which she constructs in the same way as in the original scheme and sends to V at the beginning of the service session.

The i-th micropayment has three steps. First U sends the pair $(w_i, 2i-1)$ to V (the first half-payword), then V provides the i-th piece of the service to U, and finally U sends the pair $(w'_i, 2i)$ to V (the second half-payword). Each half-payword can be checked by V using the previously received half-payword. It might seem that our scheme requires twice as many messages from U to V as the original one, but fortunately this is not true in most practical cases, because a service session consists of a series of consecutive micropayments, and U can send the second half-payword of the i-th payment $(w'_i, 2i)$ and the first half-payword of the $(i+1)$-st payment $(w_{i+1}, 2i+1)$ in a single message. In fact, since V can always compute w'_i

[1] This is similar to the way how ripped coins are used in [Jak95].

from w_{i+1}, only the second pair $(w_{i+1}, 2i+1)$ has to be sent. A service session, thus, looks like this:

Service session in the improved scheme	
$U \to V:$	$w_1, 1$
$V \to U:$	*first piece of the service*
$U \to V:$	$w_2, 3$
$V \to U:$	*second piece of the service*
...	
$U \to V:$	$w_\ell, 2\ell - 1$
$V \to U:$	*last piece of the service*
$U \to V:$	$w'_\ell, 2\ell$

which involves only one additional message compared to the original scheme.

If everything goes well, then V can present the commitment and the pair $(w'_\ell, 2\ell)$ to B, which performs the same verifications (with twice as many hash computations) as in the original scheme. If the verifications are successful, then B pays V the amount corresponding to ℓ paywords and charges the same amount to the account of U. If something goes wrong and the last half-payword is missing, then V can only present the commitment and the pair $(w_\ell, 2\ell-1)$ to B. After the verifications, B pays V the amount corresponding to $\ell-1$ paywords and charges U for the amount corresponding to ℓ paywords.

3.3.3 Brief analysis

Rationality

Just like in the rational payment protocol of Section 3.2, none of the parties gain any financial advantages by cheating. In PayWord, it is possible that the vendor provides services that are worth $\ell - 1$ paywords and redeems ℓ paywords. In our scheme, the vendor can never earn more than the value of the services it provided, since the user authorizes the broker to pay the vendor after she has received the services (by sending the second half-payword). Similarly, the user can never receive services that are worth more than that she is charged for, because she lets the broker charge her before receiving the service (when releasing the first half-payword).

Efficiency

In most practical cases, our scheme is not substantially less efficient than PayWord (and other similar micropayment schemes). First of all, the number of public key cryptographic operations (digital signature generation and verification) is the same as in PayWord. Although each sequence of consecutive micropayments in our scheme requires one additional message from the user to the vendor, this is negligible considering that such a sequence probably consists of hundreds of messages.

We require the hash chain to be twice as long as in the original scheme. This requires twice as many hash computations by the user (when the paywords are generated), by the vendor (when the paywords are verified), and by the broker (when the paywords are redeemed). The user and the broker cases are not real efficiency problems, because these computations can be performed off-line.

The size of the memory where the user stores the chain does not need to be doubled. It is more efficient to store only the first half-paywords (i.e., $w_1, w_2, \ldots, w_n$) and w'_n additionally, because the second half-paywords are usually not used in a sequence of consecutive micropayments. If a second half-payword is needed, then it can easily be computed by one application of the hash function on one of the stored values. The required memory sizes for the vendor and the broker are the same as in the original scheme.

Thus, the only factor that makes our scheme less efficient than the original one is that verification of the micropayments requires twice as many on-line hash computations by the vendor (i.e., two hash computations per micropayment). This is the "price" that we have to pay for the additional guarantees that our scheme provides[2].

3.4 Summary

In this chapter, we introduced the concept of rational exchange informally. We gave an example for a rational payment protocol, in which none of the parties can gain any financial advantages by cheating. Our protocol uses only a trusted third party that would be needed anyway in order to maintain the accounts of the users. Furthermore, the trusted third party performs essentially the same operations as it would perform in any check based payment system (in which cheating is not made uninteresting).

We used the main idea of the example rational payment protocol to improve a family of micropayment schemes. More precisely, we made the PayWord scheme rational at virtually no cost and loss in efficiency. This example shows that rational exchange protocols may provide interesting solutions in situations where a fair exchange protocol would be too expensive.

Publications: [But00, BB01]

[2]One of the reviewers of this thesis (Asokan) called our attention to another way of improving the original PayWord scheme that does not require the doubling of the size of the hash chain, and thus, fully retains the efficiency of the original scheme. In this improvement, always the vendor moves first by providing the next piece of the service, and the user moves next by sending the next payword in the chain. Thus, a service session looks like this:

$V \to U$: *first piece of the service*
$U \to V$: $w_1, 1$
$V \to U$: *second piece of the service*
$U \to V$: $w_2, 2$
...
$V \to U$: *last piece of the service*
$U \to V$: w_ℓ, ℓ

When the vendor redeems the paywords at the bank, it indicates whether it correctly received the last payword or not (by setting a boolean flag in the redemption message). If the bank receives w_ℓ and no indication of fault from the vendor, then it charges the user with ℓ units and credits the vendor with the same amount. Otherwise, if the bank receives w_ℓ and an indication of a fault, then it charges the user with $\ell + 1$ units and credits the vendor with ℓ units. This protocol is rational, because the user cannot gain anything by not sending the last payword w_ℓ, since the vendor can still make her charged with ℓ units, and because the vendor cannot gain anything by maliciously indicating a fault.

Chapter 4

Protocol games and a formal definition of rational exchange

4.1 Introduction

Our goal in this chapter is to give a formal definition for rational exchange. The value of a formal definition is threefold:

- First, attempting to give a formal definition itself may help us to better understand the concept.
- Second, it requires the construction of a mathematical model, in which other, similar concepts, such as fair exchange, could also be defined and compared to rational exchange. Such a comparison may also help us to better understand rational exchange.
- Third, a formal definition is indispensable to rigorous verification of rational exchange protocols.

In order to give a formal definition for rational exchange, first we need an informal characterization of it that we can formalize later on. A natural approach to obtain such a characterization is to start from an informal definition of fair exchange. We have already mentioned that a fair exchange protocol should guarantee for each correctly behaving party that it cannot suffer any disadvantages. More precisely, the following is required for a two-party fair exchange protocol:

- *Fairness*: If a party A behaves correctly, then the other party B cannot get the item of A unless A gets the item of B.

Furthermore, a useful fair exchange protocol should also satisfy the following requirements [Aso98, PVG01]:

- *Termination*: The protocol will eventually be completed (i.e., a correctly behaving party will terminate execution at a certain point in time).
- *Effectiveness*: If both parties behave correctly, then each will have access to the expected item when the protocol is completed.

Additional requirements, such as *non-repudiation*, might also be specified [Aso98]. However, these are not integral requirements for fair exchange, therefore, we do not consider them here.

Now, rational exchange protocols can be defined in a similar way, but instead of fairness, the following is required:

- *Rationality*: None of the parties are motivated to misbehave. In other words, if one of the parties misbehaves, then she may bring the other, correctly behaving party in a disadvantageous situation, but she cannot gain any advantages by the misbehavior.

The above characterization is sufficient for a general understanding of the concept of rational exchange, and it seems to be a good starting point for the formalization. Note however that it is certainly not precise enough for a rigorous verification of a rational exchange protocol. In addition, it does not shed light on the relationship between rational exchange and fair exchange. One might have an intuitive feeling that fairness is a stronger requirement than rationality, but it seems to be difficult to prove this using only the informal definitions above. These observations further justify our goal of defining rational exchange formally.

The mathematical model, in which we will give a formal definition for rational exchange, is based on game theory [OR94]. Game theory is a set of analytical tools developed to study situations in which self-interested parties (which want to maximize their own benefits) interact with each other according to certain rules. Since exactly this kind of situations occur in exchange protocols, game theory appears to be a natural choice.

Thus, we model the situation in which participants of a given exchange protocol find themselves as a game. Hereafter, we refer to this game as the *protocol game*. The protocol game encodes all the possible interactions of the protocol participants. The protocol participants are modeled as players. The protocol itself (as a set of rules) is represented as a set of strategies (one strategy for each protocol participant). Misbehavior means that a protocol participant follows a strategy that is different from its prescribed strategy.

We define the above described requirements for rational exchange protocols (i.e., termination, effectiveness, and rationality) in terms of properties of the protocol game and the prescribed strategies of the protocol participants. Most importantly, we have been inspired by the striking similarity between the informal definition of rationality above and the concept of Nash equilibrium in games. Therefore, we define rationality in terms of a Nash equilibrium in the protocol game.

In addition, we also give a formal definition for the fairness requirement in our model. Representing the concepts of rational exchange and fair exchange in the same model allows us to study their relationships. In particular, it allows us to prove that fairness implies rationality (assuming that the protocol satisfies certain additional requirements), but the reverse is not true in general. Thus, the result that we obtain from the model justifies our intuition that fairness is a stronger requirement than rationality.

Finally, defining a formal model for exchange protocols and giving a formal definition for rational exchange in this model allows us to rigorously verify existing rational exchange protocols. In order to illustrate this, in Chapter 5, we formally prove that the payment protocol described in Section 3.2 and Syverson's exchange protocol described in [Syv98] satisfy the definition of rational exchange.

To the best of our knowledge, we are the first who formalized the concept of rational exchange in its full generality, studied its relation to fair exchange, and provided rigorous proofs of rationality for existing rational exchange protocols. Although game theory has already been applied in the context of exchange protocols (see e.g., [San97, KR00]), we are

not aware of any formal models with the same precision and generality as our protocol game model.

The outline of this chapter is the following: In Section 4.2, we briefly introduce some basic notions from game theory that we will use in the development of our model. We present a general framework for the modeling of exchange protocols as games in Section 4.3. Based on this, in Section 4.4, we formally define various properties of exchange protocols including rationality and fairness. In Section 4.5, we study the relationship between rational exchange and fair exchange, and formally prove that fairness implies rationality, but the reverse is not true. In all the above mentioned sections, we assume that the network that is used by the protocol participants is reliable. In Section 4.6, we sketch how this assumption could be relaxed. Finally, in Section 4.7, we report on some related work.

4.2 Preliminaries

In this section, we briefly introduce some notions from game theory that we will use in the development of our model later. These notions include the definition of extensive games, strategies, and the concept of Nash equilibrium. Our presentation follows the presentation of [OR94], however, there are some differences in the definition of strategies, which we will explain at the appropriate point below.

4.2.1 Extensive games

An *extensive game with imperfect information* or shortly an *extensive game* is a tuple

$$\langle P, Q, p, (\mathcal{I}_i)_{i \in P}, (\preceq_i)_{i \in P} \rangle$$

where

- P is a set of *players*;
- Q is a set of *action sequences* that satisfies the following properties:
  - the empty sequence ϵ is a member of Q,
  - if $(a_k)_{k=1}^{w} \in Q$ and $0 < v < w$, then $(a_k)_{k=1}^{v} \in Q$,
  - if an infinite action sequence $(a_k)_{k=1}^{\infty}$ satisfies $(a_k)_{k=1}^{v} \in Q$ for every positive integer v, then $(a_k)_{k=1}^{\infty} \in Q$;

  If q is a finite action sequence and a is an action, then $q.a$ denotes the finite action sequence that consists of q followed by a. An action sequence $q \in Q$ is *terminal* if it is infinite or if there is no a such that $q.a \in Q$. The set of terminal action sequences is denoted by Z. For every non-terminal action sequence $q \in Q \setminus Z$, $A(q)$ denotes the set $\{a : q.a \in Q\}$ of *available actions* after q.
- p is a *player function* that assigns a player in P to every action sequence in $Q \setminus Z$;
- $\mathcal{I}_i$ is an *information partition* of player $i \in P$, which is a partition of the set $\{q \in Q \setminus Z : p(q) = i\}$ with the property that $A(q) = A(q')$ whenever q and q' are in the same *information set* $I_i \in \mathcal{I}_i$;

- $\preceq_i$ is a *preference relation* of player $i \in P$ on Z.

The interpretation of an extensive game is the following: Each action sequence in Q represents a possible history of the game. The action sequences that belong to the same information set $I_i \in \mathcal{I}_i$ are indistinguishable to player i. This means that i knows that the history of the game is an action sequence in I_i but she does not know which one. The empty sequence ϵ represents the starting point of the game. After any non-terminal action sequence $q \in Q \setminus Z$, player $p(q)$ chooses an action a from the set $A(q)$. Then q is extended with a, and the history of the game becomes $q.a$. The action sequences in Z represent the possible outcomes of the game. If $q, q' \in Z$ and $q \preceq_i q'$, then player i prefers the outcome q' to the outcome q.

The preference relations of the players are often represented in terms of *payoffs*: a vector $y(q) = (y_i(q))_{i \in P}$ of real numbers is assigned to every terminal action sequence $q \in Z$ in such a way that for any $q, q' \in Z$ and $i \in P$, $q \preceq_i q'$ iff $y_i(q) \leq y_i(q')$.

Small finite games can conveniently be represented by trees. The edges and the vertices of the tree correspond to actions and action sequences, respectively. A distinguished vertex, called the root, represents the empty sequence ϵ. Every other vertex u represents the sequence of the actions that belong to the edges of the path between the root and u. Let us call a vertex u terminal if the path between the root and u cannot be extended beyond u. Terminal vertices represent the terminal action sequences in the game. Each non-terminal vertex u is labeled by $p(q)$ where $q \in Q \setminus Z$ is the action sequence that belongs to u. Finally, the terminal vertices may be labeled with payoff vectors to represent the preference relations of the players.

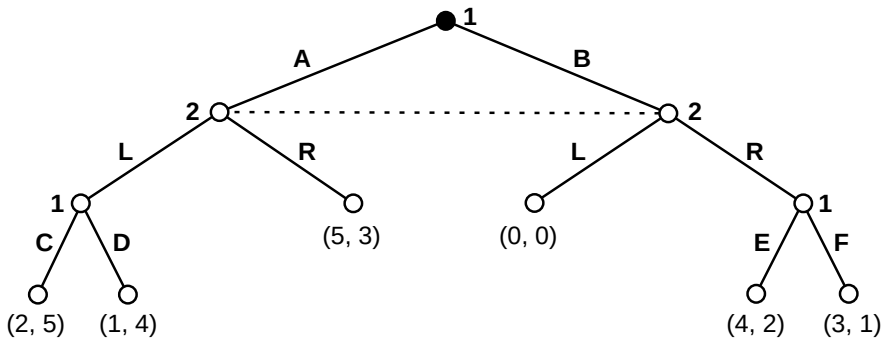

Figure 4.1: Tree representation of an extensive game

As an example let us consider Figure 4.1, which represents a game, where

- $P = \{1, 2\}$,
- $Q = \{\epsilon,\ \mathsf{A},\ \mathsf{B},\ \mathsf{A.L},\ \mathsf{A.R},\ \mathsf{B.L},\ \mathsf{B.R},\ \mathsf{A.L.C},\ \mathsf{A.L.D},\ \mathsf{B.R.E},\ \mathsf{B.R.F}\}$,
- $p(\epsilon) = 1,\ p(\mathsf{A}) = 2,\ p(\mathsf{B}) = 2,\ p(\mathsf{A.L}) = 1,\ p(\mathsf{B.R}) = 1$,
- $\mathcal{I}_1 = \{\{\epsilon\},\ \{\mathsf{A.L}\},\ \{\mathsf{B.R}\}\}$ and $\mathcal{I}_2 = \{\{\mathsf{A},\ \mathsf{B}\}\}$ (the nodes that represent action sequences that belong to the same information set are connected with a dashed line in Figure 4.1), and

- B.L $\preceq_1$ A.L.D $\preceq_1$ A.L.C $\preceq_1$ B.R.F $\preceq_1$ B.R.E $\preceq_1$ A.R and B.L $\preceq_2$ B.R.F $\preceq_2$ B.R.E $\preceq_2$ A.R $\preceq_2$ A.L.D $\preceq_2$ A.L.C.

4.2.2 Strategy

In [OR94], a strategy of player i is defined as a function s_i that assigns an action in $A(q)$ to each non-terminal action sequence $q \in Q \setminus Z$ for which $p(q) = i$, with the restriction that it assigns the same action to q and q' whenever q and q' are in the same information set of i. This means that s_i specifies the action to be chosen by i for *every* action sequence after which it is i's turn to move, even for action sequences that never occur if i follows s_i. In other words, the domain $\mathrm{dom}(s_i)$ of s_i is defined as $\mathrm{dom}(s_i) = \{q \in Q \setminus Z : p(q) = i\}$. The need for this definition of strategy arises in the concept of *subgame perfect equilibrium*. Since we will not use this equilibrium concept, we define strategies in a slightly different way.

In fact, our definition of strategy is similar to the above definition, with the difference that we define the domain of a strategy in a more restrictive way. For us, the domain $\mathrm{dom}(s_i)$ of a strategy s_i of player i contains only those non-terminal action sequences q for which $p(q) = i$ *and* q *is consistent with the moves prescribed by* s_i. Formally, we can define $\mathrm{dom}(s_i)$ in an inductive way as follows: A non-terminal action sequence $q = (a_k)_{k=1}^{w}$ is in $\mathrm{dom}(s_i)$ iff $p(q) = i$ and

- either there is no $0 \leq v < w$ such that $p((a_k)_{k=1}^{v}) = i$;
- or for all $0 \leq v < w$ such that $p((a_k)_{k=1}^{v}) = i$, $(a_k)_{k=1}^{v}$ is in $\mathrm{dom}(s_i)$ and $s_i((a_k)_{k=1}^{v}) = a_{v+1}$.

In [OR94], this notion is called *reduced strategy*.

We denote the set of all strategies of player i by S_i. Since a strategy s_i assigns the same action to every action sequence q that belongs to the same information set I, we sometimes write $s_i(I)$ instead of $s_i(q)$.

A *strategy profile* is a vector $(s_i)_{i \in P}$ of strategies, where each s_i is a member of S_i. Sometimes, we will write $(s_j, (s_i)_{i \in P \setminus \{j\}})$ instead of $(s_i)_{i \in P}$ in order to emphasize that the strategy profile specifies strategy s_j for player j.

In the game of Figure 4.1, player 1 has four strategies s_1^{AC}, s_1^{AD}, s_1^{BE}, and s_1^{BF}, such that

- the domain of s_1^{AC} is $\{\epsilon,\ \mathsf{A.L}\}$, and $s_1^{AC}(\epsilon) = \mathsf{A}$, and $s_1^{AC}(\mathsf{A.L}) = \mathsf{C}$;
- the domain of s_1^{AD} is $\{\epsilon,\ \mathsf{A.L}\}$, and $s_1^{AC}(\epsilon) = \mathsf{A}$, and $s_1^{AC}(\mathsf{A.L}) = \mathsf{D}$;
- the domain of s_1^{BE} is $\{\epsilon,\ \mathsf{B.R}\}$, and $s_1^{BE}(\epsilon) = \mathsf{B}$, and $s_1^{BE}(\mathsf{B.R}) = \mathsf{E}$;
- the domain of s_1^{BF} is $\{\epsilon,\ \mathsf{B.R}\}$, and $s_1^{BF}(\epsilon) = \mathsf{B}$, and $s_1^{BF}(\mathsf{B.R}) = \mathsf{F}$.

Note that, for instance, the action sequence B.R is not consistent with the strategy s_1^{AC}, because in B.R, ϵ is followed by B, whereas $s_1^{AC}(\epsilon) = \mathsf{A} \neq \mathsf{B}$. For this reason B.R is not in the domain of s_1^{AC} and s_1^{AD}. For similar reasons A.L is not in the domain of s_1^{BE} and s_1^{BF}.

Player 2 has only two strategies s_2^L and s_2^R, since the constraint that a strategy assigns the same action to every action sequence that belongs to the same information set must be respected. The domain of both strategies is $\{\mathsf{A},\ \mathsf{B}\}$, and we have that $s_2^L(\mathsf{A}) = s_2^L(\mathsf{B}) = \mathsf{L}$ and $s_2^R(\mathsf{A}) = s_2^R(\mathsf{B}) = \mathsf{R}$. In short, we could write $s_2^L(\{\mathsf{A}, \mathsf{B}\}) = \mathsf{L}$ and $s_2^R(\{\mathsf{A}, \mathsf{B}\}) = \mathsf{R}$.

4.2.3 Nash equilibrium

Let $o((s_i)_{i\in P})$ denote the resulting outcome when the players follow the strategies in the strategy profile $(s_i)_{i\in P}$. In other words, $o((s_i)_{i\in P})$ is the (possibly infinite) action sequence $(a_k)_{k=1}^{w} \in Z$ such that for every $0 \leq v < w$ we have that $s_{p((a_k)_{k=1}^{v})}((a_k)_{k=1}^{v}) = a_{v+1}$. A strategy profile $(s_i^*)_{i\in P}$ is a *Nash equilibrium* iff for every player $j \in P$ and every strategy $s_j \in S_j$ we have that

$$o(s_j, (s_i^*)_{i\in P\setminus\{j\}}) \preceq_j o(s_j^*, (s_i^*)_{i\in P\setminus\{j\}})$$

This means that if every player i other than j follows s_i^*, then player j is not motivated to deviate from s_j^*, because she does not gain anything by doing so.

The game of Figure 4.1 has a single Nash equilibrium, which is (s_1^{AC}, s_2^{L}). In order to see that this is a Nash equilibrium, let us first assume that player 1 plays s_1^{AC}. If player 2 now plays s_2^{L}, then her payoff is 5, whereas if she plays s_2^{R}, then her payoff is 3. Thus, player 2 is better off if she plays s_2^{L}. Now, let us assume that player 2 plays s_2^{L}. If player 1 plays s_1^{BE} or s_1^{BF}, then her payoff is 0. If she plays s_1^{AC} or s_1^{AD}, then her payoff is 2 or 1, respectively. Thus, the best response of player 1 to s_2^{L} is s_1^{AC}.

4.2.4 Restricted games

Let us consider an extensive game $G = \langle P, Q, p, (\mathcal{I}_i)_{i\in P}, (\preceq_i)_{i\in P}\rangle$. Let us divide the player set P into two disjoint subsets P_{free} and P_{fix}. Furthermore, let us fix a strategy $s_j \in S_j$ for each $j \in P_{fix}$, and let us denote the vector $(s_j)_{j\in P_{fix}}$ of fixed strategies by $\bar{s}_{fix}$. The *restricted game* $G_{|\bar{s}_{fix}}$ is the extensive game that is obtained from G by restricting each $j \in P_{fix}$ to follow the fixed strategy s_j. Formally, $G_{|\bar{s}_{fix}} = \langle P, Q_{|\bar{s}_{fix}}, p_{|\bar{s}_{fix}}, (\mathcal{I}_{i|\bar{s}_{fix}})_{i\in P}, (\preceq_{i|\bar{s}_{fix}})_{i\in P}\rangle$, where

- $Q_{|\bar{s}_{fix}}$ is the set of action sequences $(a_k)_{k=1}^{w} \in Q$ such that for every $0 \leq v < w$ for which $p((a_k)_{k=1}^{v}) = j \in P_{fix}$ we have that $s_j((a_k)_{k=1}^{v}) = a_{v+1}$;
- $p_{|\bar{s}_{fix}}(q) = p(q)$ for each $q \in Q_{|\bar{s}_{fix}}$;
- for any $q, q' \in Q_{|\bar{s}_{fix}}$, q and q' are in the same information set of a player i in $G_{|\bar{s}_{fix}}$ iff they are in the same information set of player i in G; formally:

$$\mathcal{I}_{i|\bar{s}_{fix}} = \bigcup_{I\in\mathcal{I}_i} \{I \cap Q_{|\bar{s}_{fix}}\}$$

- $q \preceq_{i|\bar{s}_{fix}} q'$ iff $q \preceq_i q'$ for any terminal action sequences $q, q' \in Q_{|\bar{s}_{fix}}$.

As an example, let us consider the game in Figure 4.2. Let $P_{free} = \{1, 3\}$ and $P_{fix} = \{2\}$, and let $\bar{s}_{fix} = (s_2^{LL})$, where s_2^{LL} is player 2's strategy that assigns L to both A and B. The restricted game that we obtain is illustrated in Figure 4.3.

If we set $P_{free} = \{2, 3\}$, $P_{fix} = \{1\}$, and $\bar{s}_{fix} = (s_1^{A})$, where s_1^{A} is player 1's strategy that assigns A to ϵ, then we obtain the restricted game depicted in Figure 4.4.

For any player $i \in P_{free}$ and for any strategy $s_i \in S_i$ of player i, let $s_{i|\bar{s}_{fix}}$ denote the strategy that s_i induces in the restricted game $G_{|\bar{s}_{fix}}$. For instance, in the game of Figure 4.2, player 3 has a strategy s_3^{XV} such that the domain of s_3^{XV} is {A.L, A.R, B.L, B.R}, and $s_3^{XV}(\mathsf{A.L}) = s_3^{XV}(\mathsf{A.R}) = \mathsf{X}$ and $s_3^{XV}(\mathsf{B.L}) = s_3^{XV}(\mathsf{B.R}) = \mathsf{V}$. In the restricted game of Figure 4.3, s_3^{XV}

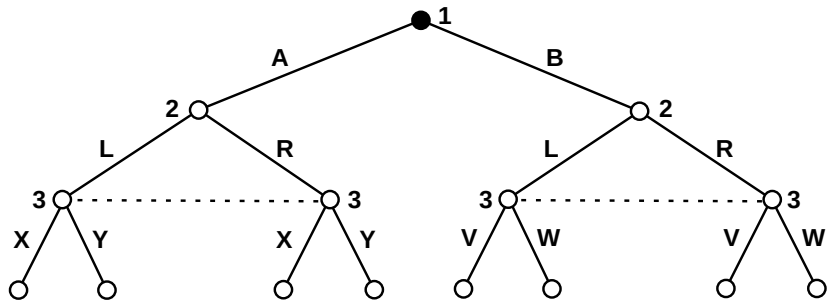

Figure 4.2: An extensive game

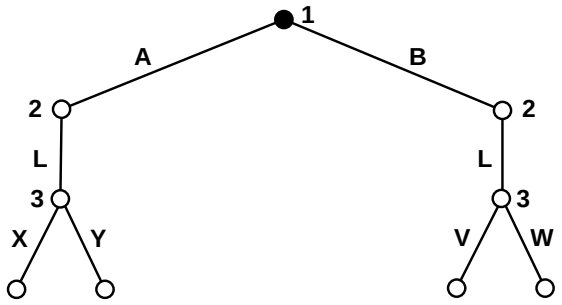

Figure 4.3: A restricted game obtained from the game of Figure 4.2 by restricting player 2 to follow the strategy s_2^{LL}

becomes $s^{XV}_{3|\bar{s}_{fix}}$, where the domain of $s^{XV}_{3|\bar{s}_{fix}}$ is $\{\mathsf{A.L},\ \mathsf{B.L}\}$, and $s^{XV}_{3|\bar{s}_{fix}}(\mathsf{A.L}) = \mathsf{X}$ and $s^{XV}_{3|\bar{s}_{fix}}(\mathsf{B.L}) = \mathsf{V}$.

Note that in $G_{|\bar{s}_{fix}}$, only the players in P_{free} can have several strategies; the players in P_{fix} are bound to the fixed strategies in $\bar{s}_{fix}$. This means that the outcome of $G_{|\bar{s}_{fix}}$ solely depends on what strategies are followed by the players in P_{free}. In other words, the players in P_{fix} become *pseudo* players, which are present, but do not have any influence on the outcome of the game.

The concept of Nash equilibrium in restricted games is defined as follows. Let us denote the resulting outcome in $G_{|\bar{s}_{fix}}$ when the players in P_{free} follow the strategies in the strategy profile $(s_{i|\bar{s}_{fix}})_{i\in P_{free}}$ by $o_{|\bar{s}_{fix}}((s_{i|\bar{s}_{fix}})_{i\in P_{free}})$. A strategy profile $(s^*_{i|\bar{s}_{fix}})_{i\in P_{free}}$ is a Nash equilibrium in the restricted game $G_{|\bar{s}_{fix}}$ iff for every player $k \in P_{free}$ and every strategy $s_{k|\bar{s}_{fix}}$ of k in $G_{|\bar{s}_{fix}}$ we have that

$$o_{|\bar{s}_{fix}}(s_{k|\bar{s}_{fix}}, (s^*_{i|\bar{s}_{fix}})_{i\in P_{free}\setminus\{k\}}) \preceq_{k|\bar{s}_{fix}} o_{|\bar{s}_{fix}}(s^*_{k|\bar{s}_{fix}}, (s^*_{i|\bar{s}_{fix}})_{i\in P_{free}\setminus\{k\}})$$

4.3 Protocol games

In this section, we describe a general framework for the construction of protocol games. This involves the definition of the players, the information sets of the players, the available

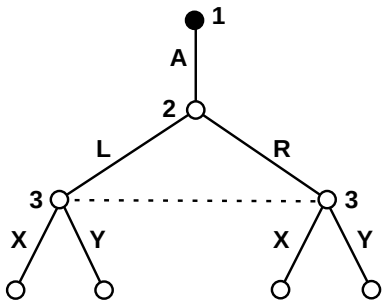

Figure 4.4: A restricted game obtained from the game of Figure 4.2 by restricting player 1 to follow the strategy s_1^A

actions in each information set, the set of action sequences of the game, and the payoffs. Before starting the description of these elements, we introduce our system model and some limitations on the possible misbehavior of the protocol participants.

We should note that we consider only two-party exchange protocols (i.e., protocols that involve only two main parties and possibly a trusted third party) for two reasons. First, we want to make the presentation easier. Second, most of the exchange protocols proposed in the literature are two-party exchange protocols. However, our model can easily be extended to multi-party exchange protocols as well.

4.3.1 System model

We assume that the network that is used by the protocol participants to communicate with each other is reliable, which means that it delivers messages to their intended destinations within a constant time interval. Such a network allows the protocol participants to run the protocol in a synchronous fashion. We will model this by assuming that the protocol participants interact with each other in *rounds*, where each round consists of the following two phases:

1. each participant generates some messages based on her current state, and sends them to some other participants;

2. each participant receives the messages that were sent to her in the current round, and performs a state transition based on her current state and the received messages.

We adopted this approach from [Lyn96], where the same model is used to study the properties of distributed algorithms in a synchronous network system. In Section 4.6, we sketch how this assumption could be relaxed and how asynchronous systems could be modeled as games.

4.3.2 Limitations on misbehavior

We want that the protocol game of an exchange protocol models all the possible ways in which the protocol participants can misbehave *within the context of the protocol*. The crucial point here is to make the difference between misbehavior within the context of the protocol and misbehavior in general. Letting the protocol participants misbehave in any way they can

would lead to a game that would allow interactions that have nothing to do with the protocol being studied. Clearly, such a game would not be a good model of the protocol, because it would be far too rich, and we suspect that it would be difficult, if not impossible, to analyze. Therefore, we want to limit the possible misbehavior of the protocol participants. However, we must do so in such a way that we do not lose generality. Essentially, the limitation that we will impose on protocol participants is that they can send only messages that are *compatible* with the protocol. We make this precise in the following paragraphs of this subsection.

We consider an exchange protocol to be a description $\pi(L)$ of a distributed computation on a set L of parameters, where L usually contains the identifiers of the executing parties, the items to be exchanged, the description of the items, and cryptographic parameters, such as keys, random numbers, etc. $\pi(L)$ consists of a set $\{\pi_1(L_1), \pi_2(L_2), \ \ldots\}$ of descriptions of local computations, where each L_k is a subset of L. We call these descriptions of local computations shortly *programs*. Each program $\pi_k(L_k)$ is meant to be executed by a protocol participant, and each L_k contains those parameters that are known to the participant that executes $\pi_k(L_k)$. Typically, each $\pi_k(L_k)$ contains instructions to wait for messages that satisfy certain conditions. When such an instruction is reached, the local computation can proceed only if a message that satisfies the required condition is provided (or a timeout occurs). We call a message m compatible with $\pi_k(L_k)$ if the local computation described by $\pi_k(L_k)$ can reach a state in which a message is expected and m would be accepted. Let us denote the set of messages that are compatible with $\pi_k(L_k)$ by $M_{\pi_k}(L_k)$. Then, the set of messages that are compatible with the protocol is defined as $M_\pi(L) = \cup_k M_{\pi_k}(L_k)$.

Note that each $M_{\pi_k}(L_k)$, and thus, $M_\pi(L)$ too may contain a large number of messages. The simple reason is that the condition upon which a message is accepted by $\pi_k(L_k)$ may be satisfied by a large number of messages. Let us consider, for instance, a protocol, in which a protocol participant should receive a message that contains data encrypted with a key that is unknown to the participant. The condition upon which this message is accepted cannot depend on the data within the encryption, because it is hidden from the participant. Therefore, the participant should accept any message that contains arbitrary data within the encryption but otherwise satisfies the required condition.

Apart from requiring the protocol participants to send messages that are compatible with the protocol, we do not impose further limitations on their behavior. In particular, we allow the protocol participants to quit the protocol at any time, or to wait for some time without any activity. Furthermore, the protocol participants can send any messages (compatible with the protocol) that they are able to compute in a given state. This also means that the protocol participants may alter the prescribed order of the protocol messages (if this is not prevented deliberately by the design of the protocol).

4.3.3 Players

We model each protocol participant (i.e., the two main parties and the trusted third party if there is any) as a player. In addition, we model the communication network that is used by the protocol participants to communicate with each other as a player too. Therefore, the player set P of the protocol game is defined as $P = \{p_1, p_2, p_3, net\}$, where p_1 and p_2 represent the two main parties of the protocol, p_3 stands for the trusted third party, and net denotes the network. If the protocol does not use a trusted third party, then p_3 is left out from P. We denote the set $P \setminus \{net\}$ by P'.

It might seem that it is useless to model the trusted third party explicitly as a player,

because it always behaves correctly, and thus, its actions are fully predictable. However, usually, the payoffs for the main parties depend on the state of the trusted third party, and it is easier to handle the state transitions of the trusted third party if we explicitly model it as a player. In addition, modeling the trusted third party in the same way as we model the other protocol participants leads to a more uniform model. After all, the trusted third party *is* a protocol participant. We will make the distinction between the trusted third party and the potentially misbehaving main parties of the protocol in another way: We restrict the player that represents the trusted third party to follow a particular strategy (the one that represents the correct behavior), whereas we allow the players that represent the potentially misbehaving main parties to choose among several strategies.

As we mentioned before, we assume that the protocol participants interact in synchronous rounds, where every message sent in the first phase of a round is delivered in the second phase of the same round. It might again seem that it is useless to model the network explicitly as a player, because the only action it can perform is the delivery of the messages that were sent in the current round, and therefore, it does not have choices. Nevertheless, we represent the network explicitly as a player. The reason is that it seems to be easier to present the model if we explicitly include the message delivery actions, because they clearly identify the second phases of the rounds, and thus, the points where the states of the players change as the result of obtaining (partial) information about the actions performed by the other players. In addition, modeling the network explicitly as a player makes it easier to extend our model with unreliable networks, because such networks can be modeled as real players that can choose between delivering a message or further delaying it.

4.3.4 Information sets

Each player $i \in P$ has a local state $\Sigma_i(q)$ that represents all the information that i has obtained in the action sequence q. If for two action sequences q and q', $\Sigma_i(q) = \Sigma_i(q')$, then q and q' are indistinguishable to i. Therefore, two action sequences q and q' belong to the same information set of i iff it is i's turn to move after both q and q', and $\Sigma_i(q) = \Sigma_i(q')$.

Before describing what constitutes the local states of the players, we need to introduce the concept of events. We define two types of events: send and receive events. The send event $snd(m, j)$ is generated for player $i \in P'$ when she submits a message $m \in M_\pi(L)$ with intended destination $j \in P'$ to the network, and the receive event $rcv(m)$ is generated for player $i \in P'$ when the network delivers a message $m \in M_\pi(L)$ to i. We denote the set of all events by E (i.e., $E = \{snd(m, j) : m \in M_\pi(L),\ j \in P'\} \cup \{rcv(m) : m \in M_\pi(L)\}$).

The local state $\Sigma_i(q)$ of player $i \in P'$ after action sequence q is defined as a tuple $\langle \alpha_i(q), H_i(q), r_i(q) \rangle$, where

- $\alpha_i(q) \in \{\mathsf{true}, \mathsf{false}\}$ is a boolean, which is true iff player i is still active after action sequence q (i.e., she did not quit the protocol);
- $H_i(q) \subseteq E \times N$ is player i's local history after action sequence q, which contains the events that were generated for i together with the round number of their generation;
- $r_i(q) \in N$ is a non-negative integer that represents the round number for player i after action sequence q.

Initially, $\alpha_i(\epsilon) = \mathsf{true}$, $H_i(\epsilon) = \emptyset$, and $r_i(\epsilon) = 1$ for every player $i \in P'$.

The local state $\Sigma_{net}(q)$ of the network consists of a set $M_{net}(q) \subseteq M_\pi(L) \times P' \times P'$ which contains those messages together with their source and intended destination that were submitted to the network and have not been delivered yet. We call $M_{net}(q)$ the network buffer. Initially, $M_{net}(\epsilon) = \emptyset$.

4.3.5 Available actions

In order to determine the set of actions available for a player $i \in P'$ after an action sequence q, we first tag each message $m \in M_\pi(L)$ with a vector $(\phi_i^m(\Sigma_i(q)))_{i \in P'}$ of conditions. Each $\phi_i^m(\Sigma_i(q))$ is a logical formula that describes the condition that must be satisfied by the local state $\Sigma_i(q)$ of player i in order for i to be able to send message m after action sequence q. Our intention is to use these conditions to capture the assumptions about cryptographic primitives at an abstract level. For instance, it is often assumed that a valid digital signature $\sigma_i(m)$ of player i on message m can only be generated by i. This means that a message $m' \in M_\pi(L)$ that contains $\sigma_i(m)$ can be sent by a player $j \neq i$ only if j received a message that contained $\sigma_i(m)$ earlier. This condition can be expressed by an appropriate logical formula for every $j \neq i$.

Now, let us consider an action sequence q, after which player $i \in P'$ has to move. There are two special actions, called idle_i and quit_i, which are always available for i after q. In addition to these special actions, player i can choose a send action of the form $\mathsf{send}_i(M)$, where M is a subset of the set $M_i(\Sigma_i(q))$ of messages that i is able to send in her current local state.

Formally, we define $M_i(\Sigma_i(q))$ as:

$$M_i(\Sigma_i(q)) = \{(m, j) : m \in M_\pi(L),\ \phi_i^m(\Sigma_i(q)) = \mathsf{true},\ j \in P' \setminus \{i\}\}$$

The set $A_i(\Sigma_i(q))$ of available actions of player $i \in P'$ after action sequence q is then defined as

$$A_i(\Sigma_i(q)) = \{\mathsf{idle}_i,\ \mathsf{quit}_i\} \cup \{\mathsf{send}_i(M) : M \subseteq M_i(\Sigma_i(q))\}$$

The interpretation of the above formula is the following: every player other than the network can do nothing, quit the protocol at any time, or send any subset of the messages that she is able to send in her current local state to any other player in P'. Note that $\mathsf{send}_i(\emptyset) \in A_i(\Sigma_i(q))$. By convention, $\mathsf{send}_i(\emptyset) = \mathsf{idle}_i$.

Let us consider now an action sequence q, after which the network has to move. Since the network is assumed to be reliable, it should deliver every message that was submitted to it in the current round. This means that there is only one action, called $\mathsf{deliver}_{net}$, that is available for the network after q, which will mean the delivery of all messages in the network buffer. Thus,

$$A_{net}(\Sigma_{net}(q)) = \{\mathsf{deliver}_{net}\}$$

The above defined actions change the local states of the players as follows:

- If a player $i \in P'$ performs the action idle_i, then the state of every player $j \in P$ remains the same as before.

  Formally: For any action sequence q, after which player $i \in P'$ has to move, we have that

$$\Sigma_j(q.\mathsf{idle}_i) = \Sigma_j(q)$$

for every $j \in P$.

- If a player $i \in P'$ performs the action quit_i, then the activity flag of i is set to false. The state of every other player $j \in P \setminus \{i\}$ remains the same as before.

  Formally: For any action sequence q, after which player $i \in P'$ has to move, we have that

  $$\begin{aligned}&\alpha_i(q.\mathsf{quit}_i) = \mathsf{false}\\ &H_i(q.\mathsf{quit}_i) = H_i(q)\\ &r_i(q.\mathsf{quit}_i) = r_i(q)\end{aligned}$$

  and for every $j \in P \setminus \{i\}$,

  $$\Sigma_j(q.\mathsf{quit}_i) = \Sigma_j(q)$$

- If a player $i \in P'$ performs an action $\mathsf{send}_i(M)$ such that $M \neq \emptyset$, then the messages in M are inserted in the network buffer, and the corresponding send events are generated for i. The state of every other player $j \in P \setminus \{i, net\}$ remains the same as before.

  Formally: For any action sequence q, after which player $i \in P'$ has to move, and for any available send action $\mathsf{send}_i(M) \in A_i(\Sigma_i(q))$ such that $M \neq \emptyset$, we have that

  $$\begin{aligned}&\alpha_i(q.\mathsf{send}_i(M)) = \alpha_i(q)\\ &H_i(q.\mathsf{send}_i(M)) = H_i(q) \cup \{(snd(m,j), r_i(q)) : (m,j) \in M\}\\ &r_i(q.\mathsf{send}_i(M)) = r_i(q)\end{aligned}$$

  $$M_{net}(q.\mathsf{send}_i(M)) = M_{net}(q) \cup \{(m,i,j) : (m,j) \in M\}$$

  and for every $j \in P \setminus \{i, net\}$,

  $$\Sigma_j(q.\mathsf{send}_i(M)) = \Sigma_j(q)$$

- If the network performs the action $\mathsf{deliver}_{net}$, then for every message in the network buffer, the appropriate receive event is generated for the intended destination of the message if it is still active. Then, every message is removed from the network buffer, and the round number of every active player is increased by one.

  Formally: For any action sequence q, after which the network has to move, we have that

  $$M_{net}(q.\mathsf{deliver}_{net}) = \emptyset$$

  and for every $i \in P'$,

  - if $\alpha_i(q) = \mathsf{true}$, then

    $$\begin{aligned}&\alpha_i(q.\mathsf{deliver}_{net}) = \alpha_i(q)\\ &H_i(q.\mathsf{deliver}_{net}) = H_i(q) \cup \{(rcv(m), r_i(q)) : \exists j \in P' : (m,j,i) \in M_{net}(q)\}\\ &r_i(q.\mathsf{deliver}_{net}) = r_i(q) + 1\end{aligned}$$

  - otherwise

    $$\Sigma_i(q.\mathsf{deliver}_{net}) = \Sigma_i(q)$$

	γ_{p_1}	γ_{p_2}
p_1	$u^-_{p_1}$	$u^+_{p_1}$
p_2	$u^+_{p_2}$	$u^-_{p_2}$

Table 4.1: The values that the items to be exchanged are worth to the protocol parties

4.3.6 Action sequences and player function

The game is played in repeated rounds, where each round consists of the following two phases:

1. each active player in P' moves, one after the other, in order;
2. when each active player in P' moved, the network moves.

The game is finished when every player in P' becomes inactive.

In order to make this formal, let us denote the set of players that are still active after action sequence q and have an index larger than v by $P'(q,v)$ (i.e., $P'(q,v) = \{p_k : p_k \in P',\ \alpha_{p_k}(q) = \mathsf{true},\ k > v\}$). Furthermore, let us denote the smallest index in $P'(q,v)$ by $k_{min}(q,v)$ (i.e.,
$k_{min}(q,v) = \min_{\{k:p_k \in P'(q,v)\}}\ k$).

We define the set Q of action sequences and the player function p of the protocol game together in an inductive manner. By definition, $\epsilon \in Q$. Moreover, $p(\epsilon) = p_1$. In addition,

- if an action sequence q is in Q and $p(q) = p_v$, then
  1. $q.a \in Q$ for every $a \in A_{p_v}(\Sigma_{p_v}(q))$;
  2. if $P'(q.a, v) \neq \emptyset$, then $p(q.a) = p_{k_{min}(q.a,v)}$, otherwise $p(q.a) = net$;
- if an action sequence q is in Q and $p(q) = net$, then
  1. $q.a \in Q$ for the single action $a = \mathsf{deliver}_{net} \in A_{net}(\Sigma_{net}(q))$;
  2. if $P'(q.a, 0) \neq \emptyset$, then $p(q.a) = p_{k_{min}(q.a,0)}$, otherwise $q.a$ is a terminal action sequence, and thus, p is not defined in $q.a$.

4.3.7 Payoffs

Now, we describe a framework for the determination of the payoffs. Let us consider the two main parties p_1 and p_2 of the protocol, and the items γ_{p_1} and γ_{p_2} that they want to exchange[1]. We denote the values that γ_{p_1} is worth to p_1 and p_2 by $u^-_{p_1}$ and $u^+_{p_2}$, respectively. Similarly, the values that γ_{p_2} is worth to p_1 and p_2 are denoted by $u^+_{p_1}$ and $u^-_{p_2}$, respectively (see also Table 4.1).

Intuitively, u^+_i and u^-_i can be thought of as a potential gain and a potential loss of player $i \in \{p_1, p_2\}$ in the game. In practice, it may be difficult to quantify u^+_i and u^-_i. However, our approach does not depend on the exact values; we require only that $u^+_i > u^-_i$ for both $i \in \{p_1, p_2\}$, which we consider to be a necessary condition for the exchange to take place at all. In addition, we will assume that $u^-_i > 0$.

[1] Typically, γ_{p_1} and γ_{p_2} are members of the parameter set L of the protocol.

The payoff $y_i(q)$ for player $i \in \{p_1, p_2\}$ assigned to the terminal action sequence q is defined as $y_i(q) = y_i^+(q) - y_i^-(q)$. We call $y_i^+(q)$ the *gain* and $y_i^-(q)$ the *loss* of player i, and define them as follows:

$$y_i^+(q) = \begin{cases} u_i^+ & \text{if } \phi_i^+(q) = \mathsf{true} \\ 0 & \text{otherwise} \end{cases}$$

and

$$y_i^-(q) = \begin{cases} u_i^- & \text{if } \phi_i^-(q) = \mathsf{true} \\ 0 & \text{otherwise} \end{cases}$$

where $\phi_i^+(q)$ and $\phi_i^-(q)$ are logical formulae. The exact form of $\phi_i^+(q)$ and $\phi_i^-(q)$ depends on the particular exchange protocol being modeled, but the idea is that $\phi_i^+(q) = \mathsf{true}$ iff i gains access to γ_j $(j \neq i)$, and $\phi_i^-(q) = \mathsf{true}$ iff i loses control over γ_i in q. A typical example would be $\phi_i^+(q) = (\exists r : (rcv(m), r) \in H_i(q))$, where we assume that m is the only message in $M_\pi(L)$ that contains γ_j.

Note that according to our model, the payoff $y_i(q)$ of player i can take only four possible values: u_i^+, $u_i^+ - u_i^-$, 0, and $-u_i^-$ for every terminal action sequence q of the protocol game.

Since we are only interested in the payoffs of p_1 and p_2 (i.e., the players that represent the main parties), we define the payoff of every other player in $P \setminus \{p_1, p_2\}$ to be 0 for every terminal action sequence of the protocol game.

4.4 Formal definition of rational exchange and some related properties

In this section, we give a formal definition for rational exchange. We do this by defining the requirements of termination, effectiveness, and rationality within the formal model described in the previous section. We also define the fairness requirement, and two other properties called *gain closed property* and *safe back out property* that we will use later in this thesis. The gained closed property requires that if a party A gains access to the item of the other party B, then B loses control over the same item. The safe back out property requires that if a party abandons the exchange right at the beginning without doing anything else, then it will not lose control over its item (i.e., it is safe to back out of the exchange). All the protocols that we are aware of satisfy the gain closed and the safe back out properties; nevertheless, we need to define them for technical reasons.

Before starting with the formal definitions, recall that an exchange protocol $\pi(L)$ is a set $\{\pi_1(L_1), \pi_2(L_2), \ \ldots\}$ of programs. As such, each $\pi_k(L_k)$ must specify for the protocol participant that executes it what to do in any conceivable situation. This is what a strategy does in a game. Therefore, we relate each $\pi_k(L_k)$ to a strategy s_k^* in the protocol game $G_\pi(L)$ of $\pi(L)$.

Now, we are ready to present the formal definitions:

Definition 4.1 (Properties of Exchange Protocols) *Let us consider a two-party exchange protocol $\pi(L) = \{\pi_1(L_1), \pi_2(L_2), \pi_3(L_3)\}$, where $\pi_1(L_1)$ and $\pi_2(L_2)$ are the programs for the main parties, and $\pi_3(L_3)$ is the program for the trusted third party (if there is any). Furthermore, let us consider the protocol game $G_\pi(L)$ of $\pi(L)$ constructed according to the framework described in Section 4.3. Let us denote the strategy of player p_k that represents $\pi_k(L_k)$ within*

$G_\pi(L)$ by $s^*_{p_k}$ ($k \in \{1,2,3\}$), the single strategy of the network by s^*_{net}, and the strategy vector $(s^*_{p_3}, s^*_{net})$ by $\bar{s}$.

- ***Termination:*** *$\pi(L)$ is said to be* terminating *iff*
  - *for every strategy $s_{p_1|\bar{s}}$ of p_1, there exists a finite prefix q' of q, such that $\alpha_{p_2}(q') =$ false, where $q = o_{|\bar{s}}(s_{p_1|\bar{s}}, s^*_{p_2|\bar{s}})$; and*
  - *for every strategy $s_{p_2|\bar{s}}$ of p_2, there exists a finite prefix q' of q, such that $\alpha_{p_1}(q') =$ false, where $q = o_{|\bar{s}}(s^*_{p_1|\bar{s}}, s_{p_2|\bar{s}})$.*

- ***Effectiveness:*** *$\pi(L)$ is said to be* effective *iff $y^+_{p_1}(q^*) = u^+_{p_1}$ and $y^+_{p_2}(q^*) = u^+_{p_2}$, where $q^* = o_{|\bar{s}}(s^*_{p_1|\bar{s}}, s^*_{p_2|\bar{s}})$.*

- ***Rationality:*** *$\pi(L)$ is said to be* rational *iff*
  - *$(s^*_{p_1|\bar{s}}, s^*_{p_2|\bar{s}})$ is a Nash equilibrium in the restricted protocol game $G_{\pi|\bar{s}}(L)$; and*
  - *both p_1 and p_2 prefer the outcome of $(s^*_{p_1|\bar{s}}, s^*_{p_2|\bar{s}})$ to the outcome of any other Nash equilibrium $(s'_{p_1|\bar{s}}, s'_{p_2|\bar{s}})$ in $G_{\pi|\bar{s}}(L)$.*

- ***Fairness:*** *$\pi(L)$ is said to be* fair *iff*
  - *for every strategy $s_{p_1|\bar{s}}$ of p_1, $y^+_{p_1}(q) = u^+_{p_1}$ implies $y^+_{p_2}(q) = u^+_{p_2}$, where $q = o_{|\bar{s}}(s_{p_1|\bar{s}}, s^*_{p_2|\bar{s}})$; and*
  - *for every strategy $s_{p_2|\bar{s}}$ of p_2, $y^+_{p_2}(q) = u^+_{p_2}$ implies $y^+_{p_1}(q) = u^+_{p_1}$, where $q = o_{|\bar{s}}(s^*_{p_1|\bar{s}}, s_{p_2|\bar{s}})$.*

- ***Gain closed property:*** *$\pi(L)$ is said to be* gain closed *iff for every terminal action sequence q of $G_{\pi|\bar{s}}(L)$ we have that $y^+_{p_1}(q) > 0$ implies $y^-_{p_2}(q) > 0$ and $y^+_{p_2}(q) > 0$ implies $y^-_{p_1}(q) > 0$.*

- ***Safe back out property:*** *Let $Q'_{|\bar{s}} = \{(a_k)_{k=1}^w \in Q_{|\bar{s}} : p_{|\bar{s}}((a_k)_{k=1}^w) = p_1, \nexists v < w : p_{|\bar{s}}((a_k)_{k=1}^v) = p_1\}$, and let $s^0_{p_1|\bar{s}}$ be the strategy of p_1 that assigns quit_{p_1} to every action sequence in $Q'_{|\bar{s}}$. Similarly, let $Q''_{|\bar{s}} = \{(a_k)_{k=1}^w \in Q_{|\bar{s}} : p_{|\bar{s}}((a_k)_{k=1}^w) = p_2, \nexists v < w : p_{|\bar{s}}((a_k)_{k=1}^v) = p_2\}$, and let $s^0_{p_2|\bar{s}}$ be the strategy of p_2 that assigns quit_{p_2} to every action sequence in $Q''_{|\bar{s}}$. $\pi(L)$ satisfies the* safe back out *property iff*
  - *for every strategy $s_{p_1|\bar{s}}$ of p_1, $y^-_{p_2}(q) = 0$, where $q = o_{|\bar{s}}(s_{p_1|\bar{s}}, s^0_{p_2|\bar{s}})$; and*
  - *for every strategy $s_{p_2|\bar{s}}$ of p_2, $y^-_{p_1}(q) = 0$, where $q = o_{|\bar{s}}(s^0_{p_1|\bar{s}}, s_{p_2|\bar{s}})$.*

All the properties above are defined in the restricted game, where the trusted third party is restricted to follow its program faithfully (i.e., to behave correctly).

Termination means that if a player follows the strategy that corresponds to the faithful execution of her program (i.e., she behaves correctly), then no matter what strategy is played by the other player, the well behaving player will terminate computation and reach an inactive state (i.e., she will perform the quit action) in a finite number of rounds.

Effectiveness means that if both players follow the strategy that corresponds to the faithful execution of their programs, then the outcome will be an action sequence in which the gain

of both players is positive (this represents a state, where both players have access to the expected items).

Rationality is defined in terms of a Nash equilibrium. More precisely, we require that the strategies that correspond to the faithful execution of the programs of the main parties form a Nash equilibrium in the restricted protocol game. In addition, we require that no other Nash equilibrium be strongly preferable for any of the main parties. This ensures that both main parties are indeed interested in behaving correctly and executing their programs faithfully, or at least they are not interested in misbehaving and deviating from their programs.

Fairness means that if a player follows the strategy that corresponds to the faithful execution of her program, then the other player can have a positive gain only if the well behaving player also has a positive gain. Recall that having a positive gain represents a state where the player has access to the expected item. So our formal definition corresponds to the informal characterization of fairness given at the beginning of this chapter.

A protocol is then said to be a rational exchange protocol, iff it satisfies the termination, effectiveness, and rationality properties. Similarly, fair exchange protocols should satisfy the termination, effectiveness, and fairness properties.

The gain closed property means that no gain comes from outside of the system. In other words, if a player has a positive gain, then the other player must have a positive loss, no matter what strategies are followed by the players. All the protocols that we are aware of have this property. Note, however, that protocols are not necessarily closed for losses, which means that if a player has a positive loss, then the other player may not necessarily have a positive gain.

In the definition of the safe back out property, the action sequences in $Q'_{|\bar{s}}$ represent those states of the game in which p_1 has to move the first time. The strategy $s^0_{p_1|\bar{s}}$ assigns the quit_{p_1} action to all these action sequences. Thus, if p_1 follows $s^0_{p_1|\bar{s}}$, then she quits the game the first time when she has to move. Similarly, the action sequences in $Q''_{|\bar{s}}$ represents those states in which p_2 has to move the first time, and $s^0_{p_2|\bar{s}}$ prescribes the quit_{p_2} action for p_2 after each of these action sequences. The safe back out property then means that if a player $i \in \{p_1, p_2\}$ follows $s^0_{i|\bar{s}}$, then she cannot have a positive loss (i.e., she does not lose control over her item), no matter what strategy is followed by the other player.

4.5 The relationship between rational exchange and fair exchange

Proposition 4.1 *If the protocol satisfies the effectiveness, gain closed, and safe back out properties, then fairness implies rationality.*

Proof: First, we have to prove that $(s^*_{p_1|\bar{s}}, s^*_{p_2|\bar{s}})$ is a Nash equilibrium in $G_{\pi|\bar{s}}(L)$ where $\bar{s} = (s^*_{p_3}, s^*_{net})$. Let us suppose that it is not. This means that either $s^*_{p_1|\bar{s}}$ is not the best response to $s^*_{p_2|\bar{s}}$, or $s^*_{p_2|\bar{s}}$ is not the best response to $s^*_{p_1|\bar{s}}$. Without loss of generality, we can assume that the first is the case. This means that p_1 has a strategy $s'_{p_1|\bar{s}}$ such that playing $s'_{p_1|\bar{s}}$ against $s^*_{p_2|\bar{s}}$ yields a higher payoff for p_1 than the payoff that she gets if she plays $s^*_{p_1|\bar{s}}$. In other words, $y_{p_1}(q^*) < y_{p_1}(q')$, where $q^* = o_{|\bar{s}}(s^*_{p_1|\bar{s}}, s^*_{p_2|\bar{s}})$, and $q' = o_{|\bar{s}}(s'_{p_1|\bar{s}}, s^*_{p_2|\bar{s}})$. Since q^* is the outcome when both parties behave correctly and, by assumption, the protocol is effective, we have that $y^+_{p_1}(q^*) = u^+_{p_1}$ and $y^+_{p_2}(q^*) = u^+_{p_2}$. In addition, since the protocol is also

gain closed, we get that $y^-_{p_1}(q^*) = u^-_{p_1}$ and $y^-_{p_2}(q^*) = u^-_{p_2}$. This means that $y_{p_1}(q^*) < y_{p_1}(q')$ is possible only if $y^+_{p_1}(q') = u^+_{p_1}$ and $y^-_{p_1}(q') = 0$ hold. However, this is impossible, because, from the fairness property, $y^+_{p_1}(q') = u^+_{p_1}$ implies $y^+_{p_2}(q') = u^+_{p_2}$, and from the gain closed property, $y^+_{p_2}(q') = u^+_{p_2} > 0$ implies $y^-_{p_1}(q') > 0$.

Next, we have to prove that no other Nash equilibrium is strongly preferable for any of the players. Let us suppose the contrary, and assume that there exists a Nash equilibrium $(s'_{p_1|\bar{s}}, s'_{p_2|\bar{s}})$ in $G_{\pi|\bar{s}}(L)$ such that one of the players, say p_1, has a higher payoff if $(s'_{p_1|\bar{s}}, s'_{p_2|\bar{s}})$ is played than if $(s^*_{p_1|\bar{s}}, s^*_{p_2|\bar{s}})$ is played. This means that $y_{p_1}(q^*) < y_{p_1}(q')$, where $q^* = o_{|\bar{s}}(s^*_{p_1|\bar{s}}, s^*_{p_2|\bar{s}})$, and $q' = o_{|\bar{s}}(s'_{p_1|\bar{s}}, s'_{p_2|\bar{s}})$. For similar reasons as before, $y_{p_1}(q^*) < y_{p_1}(q')$ is possible only if $y^+_{p_1}(q') = u^+_{p_1}$ and $y^-_{p_1}(q') = 0$ hold. Now, from the gain closed property, we get that $y^+_{p_1}(q') = u^+_{p_1} > 0$ implies $y^-_{p_2}(q') > 0$, and $y^-_{p_1}(q') = 0$ implies $y^+_{p_2}(q') = 0$. Therefore, the payoff $y_{p_2}(q')$ of p_2 in q' is negative. However, since the protocol has the safe back out property, p_2 can always do better, and achieve a non-negative payoff by not participating in the exchange at all (i.e., quitting at the beginning of the protocol without doing anything). This means that $s'_{p_2|\bar{s}}$ is not the best response to $s'_{p_1|\bar{s}}$, and thus, $(s'_{p_1|\bar{s}}, s'_{p_2|\bar{s}})$ cannot be a Nash equilibrium. □

We have just proved that fairness implies rationality. However, the reverse is not true in general; the payment protocol presented in Section 3.2, for instance, can be proven to be rational in our model (we will prove this in Chapter 5), but it is not fair.

The simple game of Figure 4.5 also illustrates that there may be protocol games that satisfy the effectiveness, gain closed, safe back out, and rationality properties, but fail to satisfy the fairness property. In this game, there are two players 1 and 2. The payoff of each player i has the form of $y^+_i(q) - y^-_i(q)$, where $y^+_i(q)$ can be 3 or 0, and $y^-_i(q)$ can be 2 or 0, depending on the terminal action sequence q. Let s^*_1 be the strategy of player 1 that assigns the action R to both action sequences after which player 1 has to move, and let s^*_2 be the strategy of player 2 that assigns the action R to the single action sequence after which player 2 has to move.

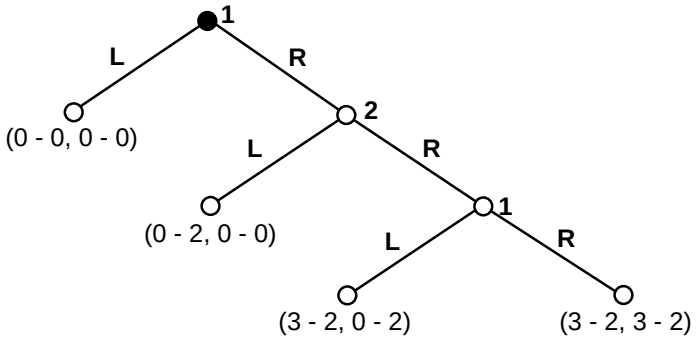

Figure 4.5: A restricted protocol game that satisfies the effectiveness, gain closed, safe back out, and rationality properties, but does not satisfy the fairness property

This game can be interpreted as a restricted protocol game, where the players represent the main protocol parties, the strategy vector (s^*_1, s^*_2) represents the protocol, $y^+_i(q)$ represents the gain and $y^-_i(q)$ represents the loss of player i in q, and the action L represents the action of quitting the protocol. The network and the trusted third party (if any) are not represented,

since these become pseudo players in the restricted protocol game that do not have choices.

It is easy to verify the following:

- *Effectiveness*: $y_1^+(q^*) = y_2^+(q^*) = 3$, where $q^* = \mathsf{R.R.R}$ is the outcome of the game when player 1 and player 2 follow s_1^* and s_2^*, respectively.
- *Gain closed property*: $y_1^+(q) > 0$ is satisfied if $q = \mathsf{R.R.L}$ or $q = \mathsf{R.R.R}$, and $y_2^-(\mathsf{R.R.L}) = y_2^-(\mathsf{R.R.R}) > 0$. $y_2^+(q) > 0$ is satisfied if $q = \mathsf{R.R.R}$, and $y_1^-(\mathsf{R.R.R}) > 0$.
- *Safe back out property*: Let s_1^0 be the strategy of player 1 that prescribes the action L at the beginning of the game (i.e., s_1^0 assigns L to the empty sequence ϵ), and let s_2^0 be the strategy of player 2 that prescribes the action L after the first move of player 1 (i.e., s_2^0 assigns L to the action sequence R).
  - No matter what strategy is followed by player 2, if player 1 follows s_1^0, then the outcome is the action sequence L, and $y_1^-(\mathsf{L}) = 0$.
  - If player 1 follows a strategy different from s_1^0, and player 2 follows the strategy s_2^0, then the outcome is the action sequence $\mathsf{R.L}$, and $y_2^-(\mathsf{R.L}) = 0$. If player 1 follows s_1^0, then the outcome is the action sequence L, and $y_2^-(\mathsf{L}) = 0$.
- *Rationality*:
  - (s_1^*, s_2^*) is a Nash equilibrium.
  - There is only one Nash equilibrium in the game that is different from (s_1^*, s_2^*), which is (s_1^0, s_2^0), and the payoff of each player is greater in (s_1^*, s_2^*), then in (s_1^0, s_2^0).

Note, however, that if player 2 follows s_2^*, and player 1 follows a strategy that assigns the action L to the action sequence $\mathsf{R.R}$, then the outcome of the game is $\mathsf{R.R.L}$, and $y_1^+(\mathsf{R.R.L}) = 3$, while $y_2^+(\mathsf{R.R.L}) = 0$. This means that the protocol game does not satisfy the fairness property.

The direct consequence of Proposition 4.1 and the above discussion is that every fair exchange protocol is a rational exchange protocol (assuming that the protocol is also gain closed and satisfies the safe back out property, which is usually the case), but the reverse is not true in general. This means that a fair exchange protocol achieves stronger guarantees than a rational exchange protocol. Therefore, one expects that rational exchange protocols are less complex than fair exchange protocols. This means that rational exchange protocols can be viewed as a trade-off between complexity and true fairness. They can be used in applications where misbehavior should be prevented, but using a fair exchange protocol would be too "expensive". Micropayments are examples for such applications, and we have already showed in Section 3.3, how the concept of rational exchange can be used to improve them with respect to fairness by making misbehavior uninteresting.

4.6 Towards an asynchronous model

So far, we have assumed that the network that is used by the protocol participants to communicate with each other is reliable. Now we sketch how this assumption can be relaxed, and how unreliable communication links can be included in our model.

A simple extension of our model would be to give choices to the network. More precisely, instead of defining the set of available actions for the network as a singleton $\{\mathsf{deliver}_{net}\}$, which means that at the end of each round the network delivers *every* message that is in the network buffer, we could define the set of available actions for the network as

$$A_{net}(\Sigma_{net}(q)) = \{\mathsf{deliver}_{net}(M) : M \subseteq M_{net}(q)\}$$

which would mean that the network can deliver any subset of the messages that are currently in the network buffer. Thus, depending on the strategy followed by the network, some messages would not be delivered immediately, but they could stay in the network buffer for some time, even forever.

Another possible extension would be to allow reliable and unreliable (as defined above) networks to coexist in the model. For instance, some exchange protocols may require a reliable channel between the trusted third party and a main party, while allowing an unreliable channel between the main parties. Describing such a protocol would require both reliable and unreliable networks in the model. Our model can easily be extended in this direction: we simply need to include more players that represent different types of networks between the protocol participants[2].

Giving choices to the network to delay the delivery of some messages as described above leads to a more general but still synchronous model, since each player's local state still contains the same current round number. However, if all the networks between the protocol participants are unreliable, then a synchronous system model is not realistic. Now, we sketch how asynchronous models can be defined.

The first approach would be to keep the rounds in the protocol game, but to remove any references to round numbers from the local states of the players that model the protocol participants. This means that one must remove the variable that represents the round number, and slightly modify the notion of event history (since so far events have been stored together with the round number of their generation in the history). Removing the round number is easy. The event history can be redefined as a sequence of event sets, where the order of the event sets in the sequence represents a temporal order. Note that we need a sequence of event sets, and not simply a sequence of events, because the protocol participants can still send and receive several messages in a single round; these actions and the corresponding events are considered to be contemporaneous.

Removing the round number from the local states of the protocol participants has a subtle consequence: In the synchronous model, the protocol participants can use their knowledge of the current round number to distinguish between local states that otherwise have the same event histories. This feature allows us to avoid the explicit modeling of timers and timeouts. This is not the case in an asynchronous model. Let us consider, for instance, a protocol participant p with local state Σ. If p's strategy prescribes that it should wait in Σ (i.e., it should perform the idle_p action), then it must wait until it receives a message, and thus, its local state changes[3]. This does not reflect the reality, where protocol participants can always decide to wait or to do some other action without any external intervention. Typically, a

[2]Maybe, in this case, it would be more appropriate to call them channels instead of networks, however, we prefer the term network here, because we want to avoid any confusion with the concept of channels described in the first part of the thesis.

[3]Recall that a strategy assigns the same action to every action sequence that belongs to the same information set, and two action sequences belong to the same information set of a player if the local state of the player is the same after both action sequences.

protocol participant sets a local timer, and when this timer signals a timeout, the participant decides what action to perform next. Modeling this feature requires the explicit modeling of timers and timeouts.

Timers can be modeled as players. For instance, each protocol participant could have a timer associated with it. A timer could perform idle and timeout actions, where the latter would change the local state of the corresponding protocol participant. There are many ways to realize this; for instance, we could introduce a new type of event that represents timeouts and could be placed in the local history of the protocol participants. Like other players, a timer can have many strategies too. However, in order to avoid that a player infinitely waits for a timeout, one may want to require each timer to follow a strategy that does not prescribes an infinite sequence of idle actions for the timer.

The second approach to define an asynchronous model would be to construct a game that is not played in rounds, but the players can perform their actions in a more liberal order. The main problem that arises in this approach is how to schedule the turns of the players. This problem can be solved by introducing a special player, which we can call *scheduler*, that would move in every second step, and decide the player that has to move next. In order to avoid starvation, the scheduler could be restricted to play a strategy that eventually allows every active player (protocol participant, timer, network) to perform an action.

In this model, the protocol participants would no longer be allowed to send several messages within a single send action, and the network would not deliver several messages within a single deliver action either. This leads to a fully asynchronous model, in which the protocol game contains all the possible orders of concurrent actions of different players. In this model, the local event history of the players would be a sequence of events, where the order of the events in the sequence would represent a temporal order.

Note that all the properties defined in Definition 4.1 are defined in terms of strategies, action sequences, and payoffs, and do not assume that all the players are aware of the same current round number, or even that the game is played in rounds. This allows us to easily generalize the definitions in the asynchronous model. Indeed, the only thing that we must take care of is that now the network has several strategies, and there are some additional players, the timers and the scheduler, which also have several strategies. In order to handle this, we can extend the strategy vector $\bar{s}$, with which the protocol game is restricted in the definitions, to include the strategies of the timers and the scheduler as well, and we can require for every property that the conditions of the property be satisfied in every possible restricted protocol game $G_{\pi|\bar{s}}(L)$. For instance, rationality would mean that (1) $(s^*_{p_1|\bar{s}}, s^*_{p_2|\bar{s}})$ is a Nash equilibrium, and (2) no other Nash equilibrium is strongly preferable to any of p_1 and p_2, in *every* possible restricted protocol game $G_{\pi|\bar{s}}(L)$, where $\bar{s} = (s^*_{p_3}, s_{t_1}, s_{t_2}, s_{t_3}, s_{net}, s_{sch})$, and s_{t_1}, s_{t_2}, and s_{t_3} range over the allowed strategies of the timers t_1, t_2, and t_3 of the protocol participants, respectively, s_{net} ranges over the strategies of the network, and s_{sch} ranges over the allowed strategies of the scheduler. (Recall that for the timers, only those strategies are allowed that do not prescribe an infinite sequence of idle actions, while for the scheduler only those strategies are allowed that eventually let every active player move.)

Finally, since the properties require that the appropriate conditions are satisfied in every possible restricted game, the arguments of the proof of Proposition 4.1 remain true in every possible restricted protocol game, and thus the statement that fairness implies rationality (assuming that the protocol satisfies the effectiveness, gain closed, and safe back out properties) carries over to the asynchronous case. However, it is not clear if there is any asynchronous

fair exchange protocol that satisfies the effectiveness property as we defined it.

4.7 Related work

Formal definitions for fair exchange are given by Gaertner *et al.* in [GPV99, PG99]. They adopt the formalism of concurrency theory and define fairness based on safety and liveness properties. Although their proposal certainly has a strong potential, it is somewhat limited to fair exchange, and in particular to the concept of strong and weak fairness[4] as it was defined by Asokan in [Aso98]. They do not attempt to formalize the concept of rational exchange, nor to investigate the relationship between rational exchange and fair exchange.

Kremer and Raskin describes a formal approach to the analysis of non-repudiation protocols (which are strongly related to fair exchange protocols) in [KR00]. They model non-repudiation protocols as games in a similar way as we do. However, they use neither payoffs nor the concept of equilibrium to specify properties of the protocol. Instead, they introduce a game based alternating temporal logic for this purpose. We feel that by doing so they do not fully exploit the power of game theory, and they essentially fall back to the well-known model checking approach for protocol analysis. This, however, has some advantages: they can use an automated model-checker. They do not try to formalize the concept of rational exchange and to relate it to fair exchange.

In [San97], Sandholm proposes a method for managing an exchange between two agents – a supplier and a demander – so that the gains from completing the exchange at any point are larger for both agents than the gains from terminating it. The method consists in splitting the exchange into small chunks in a way that the agents can avoid situations that motivate either of them to defect. Sandholm calls this type of exchange *unenforced exchange* (since it does not rely on enforcement from an external trusted party), and relates it to Nash equilibrium. However, he does not formalize the concept of rational exchange in general (the proposed method can be viewed as a particular rational exchange protocol), nor does he relate his results to fair exchange.

In [ASW00], Asokan *et al.* define a formal security model for fair signature exchange. The model is described in terms of a "game", in which a correctly behaving party A and the trusted third party act in a purely reactive fashion, while the actions of the misbehaving party $B*$ are restricted only by a few rules. $B*$ wins the game if it can obtain the digital signature of A on some message m without A obtaining the digital signature of $B*$ on another message m'. They define fairness to mean that the probability that $B*$ wins the game is negligible (with respect to some security parameter). Although, at first sight, the formal model of Asokan *et al.* might seem to be similar to our approach, it is indeed completely different. First of all, apart from using the terms *game* and *player*, their approach has nothing to do with game theory. Most importantly, they do not use the notion of equilibrium. Their model is much more similar to the standard models that are used in the cryptographic literature to prove the security of cryptographic algorithms, where one explicitly states the assumptions made about the power of the adversary and tries to prove that the system cannot be broken without invalidating those assumptions. As opposed to this, we completely abstract away cryptography in our model. While the formal model of Asokan *et al.* is probably the most rigorous model that can be found in the literature regarding fairness, it is somewhat restricted to signature exchange protocols. In addition, it does not seem to be appropriate to capture

[4]Strong and weak fairness have nothing to do with the distinction between fairness and rationality.

the notion of rationality, which is not a limitation itself, since it was not the goal of the authors to formalize the concept of rational exchange.

4.8 Summary

In this chapter, we introduced a formal model based on game theory, in which exchange protocols can be modeled and their properties can be studied. We described in detail, how exchange protocols can be represented as games in this model. We used the model to give formal definitions for various properties of exchange protocols, including rationality and fairness. One of our main results is to relate the rationality property to the concept of Nash equilibrium in games. In addition, we used the model to study the relationship between rational exchange and fair exchange. We proved that fairness implies rationality (assuming that the protocol has some further, usual properties), but the reverse is not true in general.

All the above results were obtained by assuming that the network that is used by the protocol participants is reliable (i.e., it delivers messages to their intended destinations within a constant time interval). At the end of the chapter, we sketched how this assumption could be relaxed and how asynchronous systems could be modeled as games. We briefly described how the definitions of rationality and fairness (and other properties of exchange protocols as well) can be generalized, and why the statement that fairness implies rationality remains true in the asynchronous model.

The formal model introduced in this chapter helped us to better understand what rational exchange is and how it is related to fair exchange. This understanding is indispensable for the design of rational exchange protocols. The formal model also serves as a basis for rigorous verification of rational exchange protocols. We will illustrate this through the analysis of two protocols in Chapter 5.

Publication: [BH01a]

Chapter 5

Proving protocols to be rational

5.1 Introduction

In this chapter, we study two exchange protocols and formally prove that they satisfy our definition of rational exchange. Our goal is to demonstrate how the model introduced in Chapter 4 can be used in practice. The first protocol that we prove to be rational is the example payment protocol of Section 3.2. The second one will be Syverson's rational exchange protocol [Syv98].

5.2 Proof of the example rational payment protocol

In Section 3.2, we presented the following payment protocol:

The payment protocol of Section 3.2
$U \to V:\ m_1 = (U,\ V,\ tid,\ val,\ h(rnd),\ sig(k_U^{-1}, (U, V, tid, val, h(rnd))))$
$V \to U:\ m_2 = srv$
$U \to V:\ m_3 = rnd$
if V received m_1 and m_3:
$\quad V \to B:\ m_4 = (m_1,\ m_3,\ sig(k_V^{-1}, (m_1, m_3)))$
if V received only m_1:
$\quad V \to B:\ m_4' = (m_1,\ sig(k_V^{-1}, (m_1)))$

where U, V, and B stand for the identifiers of the user, the vendor, and the bank, respectively; k_U^{-1} and k_V^{-1} denote the private keys of U and V, respectively; *tid* is a fresh transaction identifier for U and V; *val* denotes the value of the payment that U is supposed to pay to V; *srv* denotes the service that V is supposed to provide to U in exchange for the payment; *rnd* is a freshly generated random number; h is a publicly known cryptographic hash function; and *sig* is a signature generation function that takes a private key k^{-1} and a message m, and returns a digital signature on m generated with k^{-1}. For the detailed description of the protocol, see Section 3.2.

We will now prove formally that this payment protocol is a rational exchange protocol. In order to do so, we construct the protocol game of the protocol using the framework that we in-

troduced in Section 4.3, and we prove that the protocol satisfies the definitions of termination, effectiveness, and rationality given in Section 4.4.

Before we start, we introduce some further notation that we will use in the proof:

- the public keys of U and V are denoted by k_U and k_V, respectively;
- *vfy* is a signature verification function that takes a public key k, a message m, and a signature σ, and returns true if σ is a valid signature on m that can be verified with k, otherwise it returns false;
- the description of the service *srv* is denoted by *dsc*;
- *fit* is a function that takes a service s and a service description d and returns true if d matches s, otherwise it returns false.

We assume that the service *srv* can be represented with a bit string (e.g., a music or a video file). In this case, one can imagine its description *dsc* as a hash value computed from the bit string using a publicly known cryptographic hash function. The implementation of *fit* is then straightforward.

5.2.1 The set of compatible messages

In order to determine the set of messages compatible with the protocol, we first re-construct the programs of the protocol participants from the informal protocol description:

$$\begin{aligned}&\pi(U, k_U, k_U^{-1}, V, k_V, k_V^{-1}, tid, val, rnd, srv, dsc) = \{\\ &\quad \pi_U(U, k_U^{-1}, V, tid, val, rnd, dsc),\\ &\quad \pi_V(U, k_U, V, k_V^{-1}, tid, val, srv),\\ &\quad \pi_B(U, k_U, V, k_V, tid)\ \ \}\end{aligned}$$

where

$\pi_U(U, k_U^{-1}, V, tid, val, rnd, dsc) =$

1. compute $\eta = h(rnd)$
2. compute $\sigma = sig(k_U^{-1}, (U, V, tid, val, \eta))$
3. send $(U, V, tid, val, \eta, \sigma)$ to V
4. wait until timeout or
   a message m arrives such that $fit(m, dsc) =$ true
5. if timeout then go to step 7
6. send rnd to V
7. exit

$\pi_V(U, k_U, V, k_V^{-1}, tid, val, srv) =$

1. wait until timeout or
   a message $m = (\iota_1, \iota_2, \tau, \nu, \eta, \sigma)$ arrives such that
   - $\iota_1 = U$
   - $\iota_2 = V$
   - $\tau = tid$
   - $\nu = val$

- $vfy(k_U,\ (\iota_1, \iota_2, \tau, \nu, \eta),\ \sigma) = \mathsf{true}$

2. if timeout then go to step 11
3. send srv to U
4. wait until timeout or
   a message m' arrives such that $h(m') = \eta$
5. if timeout then go to step 9
6. compute $\sigma' = sig(k_V^{-1}, (m, m'))$
7. send (m, m', σ') to B
8. go to step 11
9. compute $\sigma' = sig(k_V^{-1}, m)$
10. send (m, σ') to B
11. exit

$\pi_B(U, k_U, V, k_V, tid) =$

1. wait until either a message $(\iota_1, \iota_2, \tau, \nu, \eta, \sigma, \rho, \sigma')$ arrives such that
   - $\iota_1 = U$
   - $\iota_2 = V$
   - $\tau = tid$
   - $h(\rho) = \eta$
   - $vfy(k_U,\ (\iota_1, \iota_2, \tau, \nu, \eta),\ \sigma) = \mathsf{true}$
   - $vfy(k_V,\ (\iota_1, \iota_2, \tau, \nu, \eta, \sigma, \rho),\ \sigma') = \mathsf{true}$

   or a message $(\iota_1, \iota_2, \tau, \nu, \eta, \sigma, \sigma')$ arrives such that
   - $\iota_1 = U$
   - $\iota_2 = V$
   - $\tau = tid$
   - $vfy(k_U,\ (\iota_1, \iota_2, \tau, \nu, \eta),\ \sigma) = \mathsf{true}$
   - $vfy(k_V,\ (\iota_1, \iota_2, \tau, \nu, \eta, \sigma),\ \sigma') = \mathsf{true}$
2. exit

In the above description, π_U is the program of the user, π_V is the program of the vendor, and π_B is the program of the bank. The programs are quite straightforward, however, three issues need to be discussed before going on. First, note that the random number rnd is not generated by the program π_U of the user, but it is given to it as an input. We need to consider the random number as an input, because we want to represent the program of the user with a single strategy in the protocol game. If the program generated the random number, then it should be represented by a set of strategies (where each element of the set belongs to a possible value of the random number) and a probability distribution on this set. We prefer the single strategy model, because it is simpler, and at the same time, it is not a real limitation, because virtually there is no difference between generating a random number on-the-fly, or having it pre-generated and providing it as an input to the computation.

Second, note that the identifiers U and V of the user and the vendor, respectively, as well as the transaction identifier tid are given to the program of the bank, although the bank has no way to know in advance who will execute the protocol and which transaction identifier they will agree on. This seems to be contradictory, but in fact it is not, because actually π_B is not the full program of the bank, but only a part of it. Conceptually, one can think of the full program of the bank as a set $\{\pi_B(U', k_{U'}^{-1}, V', k_{V'}^{-1}, tid') : U' \in \mathcal{U},\ V' \in \mathcal{V}, tid' \in \mathcal{T}\}$

of subprograms, where $\mathcal{U}$ and $\mathcal{V}$ are the sets of all user and vendor identifiers, respectively, and $\mathcal{T}$ is the set of all possible transaction identifiers. When the bank begins its operation, it starts all these subprograms concurrently. If a message m arrives, then the bank gives m to every running subprogram. If m is a valid message that the bank should accept, then exactly one of the subprograms will accept it, and then it exits. This ensures that the bank never accepts two or more messages with the same user, vendor, and transaction identifiers. Since we are interested in the transaction *tid* between user U and vendor V, here we consider only the subprogram $\pi_B(U, k_U, V, k_V, tid)$.

Third, note that $\pi_B(U, k_U, V, k_V, tid)$ does nothing else but waits for a message with a certain structure and content and then stops. In particular, it does not describe what the bank does when it receives a valid message from the vendor. It is sufficient for our purposes to model only the reception of a valid message, because we will assume that the bank behaves correctly and performs the appropriate operation (i.e., updates the accounts of the user and the vendor) when the message is received.

Given the programs of the protocol participants, we can easily determine the set of messages compatible with the protocol:

$$M_\pi(L) = M_\pi(U, k_U, k_U^{-1}, V, k_V, k_V^{-1}, tid, val, rnd, srv, dsc) = M_1 \cup M_2 \cup M_3 \cup M_4 \cup M_4'$$

where

$$\begin{aligned}
M_1 = \{(\iota_1, \iota_2, \tau, \nu, \eta, \sigma) : \\
&\iota_1 = U, \\
&\iota_2 = V, \\
&\tau = tid, \\
&\nu = val, \\
&vfy(k_U,\ (\iota_1, \iota_2, \tau, \nu, \eta),\ \sigma) = \mathsf{true}\} \\
M_2 = \{\varsigma : fit(\varsigma, dsc) = \mathsf{true}\} \\
M_3 = \{\rho : \rho \text{ is a random number (of a given size)}\} \\
M_4 = \{(\iota_1, \iota_2, \tau, \nu, \eta, \sigma, \rho, \sigma') : \\
&\iota_1 = U, \\
&\iota_2 = V, \\
&\tau = tid, \\
&h(\rho) = \eta, \\
&vfy(k_U,\ (\iota_1, \iota_2, \tau, \nu, \eta),\ \sigma) = \mathsf{true}, \\
&vfy(k_V,\ (\iota_1, \iota_2, \tau, \nu, \eta, \sigma, \rho),\ \sigma') = \mathsf{true}\} \\
M_4' = \{(\iota_1, \iota_2, \tau, \nu, \eta, \sigma, \sigma') : \\
&\iota_1 = U, \\
&\iota_2 = V, \\
&\tau = tid, \\
&vfy(k_U,\ (\iota_1, \iota_2, \tau, \nu, \eta),\ \sigma) = \mathsf{true}, \\
&vfy(k_V,\ (\iota_1, \iota_2, \tau, \nu, \eta, \sigma),\ \sigma') = \mathsf{true}\}
\end{aligned}$$

5.2.2 The protocol game

Once the set $M_\pi(L)$ of compatible messages is determined, we can construct the protocol game $G_\pi(L)$ of the protocol by applying the framework of Section 4.3. The player set of the

protocol game is $P = \{U, V, B, net\}$, where U represents the user, V represents the vendor, B represents the bank, and net represents the network, via which the protocol participants communicate with each other. We assume that the network is reliable. The information partition of each player $i \in P$ is determined by i's local state $\Sigma_i(q)$. In order to determine the available actions of the players in $P' = P \setminus \{net\}$, we must tag each message $m \in M_\pi(L)$ with a vector $(\phi_i^m(\Sigma_i(q)))_{i \in P'}$ of logical formulae, where each formula $\phi_i^m(\Sigma_i(q))$ describes the condition that must be satisfied in order for i to be able to send message m in the information set represented by the local state $\Sigma_i(q)$. For the above payment protocol, these vectors of logical formulae are as follows:

- The user can always send any message $m \in M_1$. The vendor and the bank cannot generate messages in M_1, because they cannot generate valid digital signatures of the user. Therefore, the vendor and the bank can send a message $m \in M_1$ only if they received m, or a message that contained m earlier.

  Formally, for any $m \in M_1$:

$$\begin{aligned}
\phi_U^m(\Sigma_U(q)) &= (\alpha_U(q) = \mathsf{true}) \\
\phi_V^m(\Sigma_V(q)) &= (\alpha_V(q) = \mathsf{true}) \wedge \\
&\quad ((\exists r < r_V(q) : (rcv(m), r) \in H_V(q)) \vee \\
&\quad\ (\exists r < r_V(q), m' = (m, \rho, \sigma) \in M_4 : (rcv(m'), r) \in H_V(q)) \vee \\
&\quad\ (\exists r < r_V(q), m' = (m, \sigma) \in M_4' : (rcv(m'), r) \in H_V(q))) \\
\phi_B^m(\Sigma_B(q)) &= (\alpha_B(q) = \mathsf{true}) \wedge \\
&\quad ((\exists r < r_B(q) : (rcv(m), r) \in H_B(q)) \vee \\
&\quad\ (\exists r < r_B(q), m' = (m, \rho, \sigma) \in M_4 : (rcv(m'), r) \in H_B(q)) \vee \\
&\quad\ (\exists r < r_B(q), m' = (m, \sigma) \in M_4' : (rcv(m'), r) \in H_B(q)))
\end{aligned}$$

- The vendor can always send srv, but we assume that it cannot compute any other $\varsigma \in M_2$. We also assume that the user and the bank cannot compute any $\varsigma \in M_2$. These assumptions are reasonable, since we assumed that the description of the service is a hash value computed using a cryptographic hash function, and cryptographic hash functions are preimage resistant and collision resistant [MvOV97].

  Formally, for any $m \in M_2$:

  - if $m = srv$, then

$$\begin{aligned}
\phi_U^m(\Sigma_U(q)) &= (\alpha_U(q) = \mathsf{true}) \wedge \\
&\quad (\exists r < r_U(q) : (rcv(m), r) \in H_U(q)) \\
\phi_V^m(\Sigma_V(q)) &= (\alpha_V(q) = \mathsf{true}) \\
\phi_B^m(\Sigma_B(q)) &= (\alpha_B(q) = \mathsf{true}) \wedge \\
&\quad (\exists r < r_B(q) : (rcv(m), r) \in H_B(q))
\end{aligned}$$

  - if $m \neq srv$, then

$$\begin{aligned}
\phi_U^m(\Sigma_U(q)) &= (\alpha_U(q) = \mathsf{true}) \wedge \\
&\quad (\exists r < r_U(q) : (rcv(m), r) \in H_U(q)) \\
\phi_V^m(\Sigma_V(q)) &= (\alpha_V(q) = \mathsf{true}) \wedge \\
&\quad (\exists r < r_V(q) : (rcv(m), r) \in H_V(q)) \\
\phi_B^m(\Sigma_B(q)) &= (\alpha_B(q) = \mathsf{true}) \wedge \\
&\quad (\exists r < r_B(q) : (rcv(m), r) \in H_B(q))
\end{aligned}$$

- The user can send any random number in M_3, but we assume that the vendor and the bank cannot generate a random number that hashes into $h(rnd)$.

  Formally, for any $m \in M_3$:

  - if $h(m) = h(rnd)$, then

$$\begin{aligned}
\phi_U^m(\Sigma_U(q)) &= (\alpha_U(q) = \mathsf{true}) \\
\phi_V^m(\Sigma_V(q)) &= (\alpha_V(q) = \mathsf{true}) \wedge \\
&\quad ((\exists r < r_V(q) : (rcv(m), r) \in H_V(q)) \vee \\
&\quad (\exists r < r_V(q), m' = (\mu, m, \sigma) \in M_4 : (rcv(m'), r) \in H_V(q))) \\
\phi_B^m(\Sigma_B(q)) &= (\alpha_B(q) = \mathsf{true}) \wedge \\
&\quad ((\exists r < r_B(q) : (rcv(m), r) \in H_B(q)) \vee \\
&\quad (\exists r < r_B(q), m' = (\mu, m, \sigma) \in M_4 : (rcv(m'), r) \in H_B(q)))
\end{aligned}$$

  - if $h(m) \neq h(rnd)$, then

$$\begin{aligned}
\phi_U^m(\Sigma_U(q)) &= (\alpha_U(q) = \mathsf{true}) \\
\phi_V^m(\Sigma_V(q)) &= (\alpha_V(q) = \mathsf{true}) \\
\phi_B^m(\Sigma_B(q)) &= (\alpha_B(q) = \mathsf{true})
\end{aligned}$$

- The user and the bank can send a message $m \in M_4$ only if they received m before, since they cannot generate valid digital signatures of the vendor. The vendor can send a message $m = (\mu, \rho, \sigma) \in M_4$ only if it received μ before, since it cannot generate valid digital signatures of the user. In addition, if $h(\rho) = h(rnd)$, then the vendor must also receive ρ before, since as we assumed earlier, the vendor cannot generate a random number that hashes into $h(rnd)$.

  Formally, for any $m = (\mu, \rho, \sigma) \in M_4$:

  - If $h(\rho) \neq h(rnd)$, then

$$\begin{aligned}
\phi_U^m(\Sigma_U(q)) &= (\alpha_U(q) = \mathsf{true}) \wedge \\
&\quad (\exists r < r_U(q) : (rcv(m), r) \in H_U(q)) \\
\phi_V^m(\Sigma_V(q)) &= (\alpha_V(q) = \mathsf{true}) \ \wedge \ \varphi_1 \\
\phi_B^m(\Sigma_B(q)) &= (\alpha_B(q) = \mathsf{true}) \wedge \\
&\quad (\exists r < r_B(q) : (rcv(m), r) \in H_B(q))
\end{aligned}$$

  - if $h(\rho) = h(rnd)$, then

$$\begin{aligned}
\phi_U^m(\Sigma_U(q)) &= (\alpha_U(q) = \mathsf{true}) \wedge \\
&\quad (\exists r < r_U(q) : (rcv(m), r) \in H_U(q)) \\
\phi_V^m(\Sigma_V(q)) &= (\alpha_V(q) = \mathsf{true}) \ \wedge \ \varphi_1 \ \wedge \ \varphi_2 \\
\phi_B^m(\Sigma_B(q)) &= (\alpha_B(q) = \mathsf{true}) \wedge \\
&\quad (\exists r < r_B(q) : (rcv(m), r) \in H_B(q))
\end{aligned}$$

    where

$$\begin{aligned}
\varphi_1 = \ & ((\exists r < r_V(q) : (rcv(\mu), r) \in H_V(q)) \vee \\
& (\exists r < r_V(q), m' = (\mu, \rho', \sigma') \in M_4 : (rcv(m'), r) \in H_V(q)) \vee \\
& (\exists r < r_V(q), m' = (\mu, \sigma') \in M_4' : (rcv(m'), r) \in H_V(q))) \\
\varphi_2 = \ & ((\exists r < r_V(q) : (rcv(\rho), r) \in H_V(q)) \vee \\
& (\exists r < r_V(q), m' = (\mu', \rho, \sigma') \in M_4 : (rcv(m'), r) \in H_V(q)))
\end{aligned}$$

- The user and the bank can send a message $m \in M_4'$ only if they received m before, since they cannot generate valid digital signatures of the vendor. The vendor can send a message $m = (\mu, \sigma) \in M_4'$ only if it received μ before, since it cannot generate valid digital signatures of the user.

  Formally, for any $m = (\mu, \sigma) \in M_4'$:

$$\begin{aligned}
\phi_U^m(\Sigma_U(q)) &= (\alpha_U(q) = \mathsf{true}) \wedge \\
&\quad (\exists r < r_U(q) : (rcv(m), r) \in H_U(q)) \\
\phi_V^m(\Sigma_V(q)) &= (\alpha_V(q) = \mathsf{true}) \wedge \\
&\quad ((\exists r < r_V(q) : (rcv(\mu), r) \in H_V(q)) \vee \\
&\quad\ (\exists r < r_V(q), m' = (\mu, \rho, \sigma') \in M_4 : (rcv(m'), r) \in H_V(q)) \vee \\
&\quad\ (\exists r < r_V(q) : (rcv(m), r) \in H_V(q))) \\
\phi_B^m(\Sigma_B(q)) &= (\alpha_B(q) = \mathsf{true}) \wedge \\
&\quad (\exists r < r_B(q) : (rcv(m), r) \in H_B(q))
\end{aligned}$$

The above logical formulae determine the sets of available actions of the players. The sets of available actions, in turn, allow us to determine the set Q of action sequences, and to complete the construction of the protocol game $G_\pi(L)$ using the framework of Section 4.3. What remains is to define the payoffs of the players, but we defer this until Subsection 5.2.4. The reason is that we want to define the payoffs only in the restricted protocol game that we obtain from the protocol game by restricting B to follow the strategy that corresponds to the program π_B of the bank. Therefore, in the next subsection, we describe the strategies of the players that correspond to the programs of the protocol participants.

5.2.3 Strategies

Now, we define the strategies s_U^*, s_V^*, and s_B^* that belong to the programs π_U, π_V, and π_B of the protocol participants, respectively. For technical reasons, we must introduce an ordering relation on the messages in $M_\pi(L)$ that we will denote by $\leq$. Actually, any kind of total ordering is appropriate for our purposes. One can take, for instance, the bit string representation of the messages in $M_\pi(L)$, and define $\leq$ as a lexical ordering on bit strings. We need this relation to define the strategy that corresponds to the program of V unambiguously. Note that V must send a message to B that includes a message in M_1 previously received from U. However, if U misbehaves, then she can send several different messages in M_1 to V (even in a single round). We will require V to use the $\leq$ relation to select the "smallest" (according to $\leq$) of those messages.

Strategy s_U^*

- If $\alpha_U(q) = \mathsf{true}$ and $r_U(q) = 1$, then perform the action $\mathsf{send}_U(\{(m_1,\ V)\})$, where $m_1 = (U,\ V,\ tid,\ val,\ h(rnd),\ sig(k_U^{-1}, (U, V, tid, val, h(rnd))))$.

- If $\alpha_U(q) = \mathsf{true}$ and $r_U(q) = 2$, then perform the action idle_U.

- If $\alpha_U(q) = \mathsf{true}$, $r_U(q) = 3$, and there exist a round number $r < 3$ and a message $m \in M_2$ such that $(rcv(m), r) \in H_U(q)$, then perform the action $\mathsf{send}_U(\{(rnd,\ V)\})$.

- If $\alpha_U(q) = \mathsf{true}$, $r_U(q) = 3$, and there exist no round number $r < 3$ and message $m \in M_2$ such that $(rcv(m), r) \in H_U(q)$, then perform the action quit_U.

- If $\alpha_U(q) = \mathsf{true}$ and $r_U(q) = 4$ then perform the action quit_U.

Strategy s_V^*

- If $\alpha_V(q) = \mathsf{true}$ and $r_V(q) = 1$, then perform the action idle_V.
- If $\alpha_V(q) = \mathsf{true}$, $r_V(q) = 2$, and there exist a round number $r < 2$ and a message $m \in M_1$ such that $(rcv(m), r) \in H_V(q)$, then perform the action $\mathsf{send}_V(\{(srv,\ U)\})$.
- If $\alpha_V(q) = \mathsf{true}$, $r_V(q) = 2$, and there exist no round number $r < 2$ and message $m \in M_1$ such that $(rcv(m), r) \in H_V(q)$, then perform the action quit_V.
- If $\alpha_V(q) = \mathsf{true}$ and $r_V(q) = 3$, then perform the action idle_V.
- If $\alpha_V(q) = \mathsf{true}$ and $r_V(q) = 4$, then let M be the set of those message pairs (m, m') for which $m = (\iota_1, \iota_2, \tau, \nu, \eta, \sigma) \in M_1$, $m' = \rho \in M_3$, $h(\rho) = \eta$, and there exist round numbers $r, r' < 4$ such that $(rcv(m), r) \in H_V(q)$ and $(rcv(m'), r') \in H_V(q)$. In addition, let M' be the set of messages $m'' \in M_1$ for which there exist a round number $r'' < 4$ such that $(rcv(m''), r'') \in H_V(q)$. Note that M' cannot be empty, since otherwise V would have already quitted in round 2, and thus $\alpha_V(q)$ could not be true in round 4.
  - If $M \neq \emptyset$, then choose the message pair (m, m') from M for which $(m, m', sig(k_V^{-1}, (m, m')))$ is the smallest (according to the ordering $\leq$), and perform the action $\mathsf{send}_V\{((m, m', sig(k_V^{-1}, (m, m'))),\ B)\}$.
  - If $M = \emptyset$, then choose the smallest message m'' from M' (according to the ordering $\leq$), and perform the action $\mathsf{send}_V\{((m'', sig(k_V^{-1}, m'')),\ B)\}$.
- If $\alpha_V(q) = \mathsf{true}$ and $r_V(q) = 5$, then perform the action quit_V.

Strategy s_B^*

- If $\alpha_B = \mathsf{true}$ and there exist a round number r and a message $m \in M_4 \cup M_4'$ such that $(rcv(m), r) \in H_B(q))$, then perform the action quit_B.
- If $\alpha_B = \mathsf{true}$ and there exist no round number r and message $m \in M_4 \cup M_4'$ such that $(rcv(m), r) \in H_B(q))$, then perform the action idle_B.

From this point on, we will be concerned with the restricted protocol game $G_{\pi|\bar{s}}(L)$, where $\bar{s} = (s_B^*, s_{net}^*)$ and s_{net}^* is the single strategy of the network. Without defining any payoffs, we can already make the following simple statements about this restricted protocol game:

Lemma 5.1 *Let us consider the restricted protocol game $G_{\pi|\bar{s}}(L)$, where $\bar{s} = (s_B^*, s_{net}^*)$. If $(rcv(m), r) \in H_B(q)$ for some message $m = (\mu, \rho, \sigma) \in M_4$, round number $r \in N$, and action sequence $q \in Q_{|\bar{s}}$, then $(rcv(\mu), r') \in H_V(q)$ for some $r' < r$.*

Lemma 5.2 *Let us consider the restricted protocol game $G_{\pi|\bar{s}}(L)$, where $\bar{s} = (s_B^*, s_{net}^*)$. If $(rcv(m), r) \in H_B(q)$ for some message $m = (\mu, \rho, \sigma) \in M_4$, round number $r \in N$, and action sequence $q \in Q_{|\bar{s}}$, and $h(\rho) = h(rnd)$, then $(rcv(\rho), r') \in H_V(q)$ for some $r' < r$.*

Lemma 5.3 *Let us consider the restricted protocol game $G_{\pi|\bar{s}}(L)$, where $\bar{s} = (s^*_B, s^*_{net})$. If $(rcv(m), r) \in H_B(q)$ for some message $m = (\mu, \sigma) \in M'_4$, round number $r \in N$, and action sequence $q \in Q_{|\bar{s}}$, then $(rcv(\mu), r') \in H_V(q)$ for some $r' < r$.*

Lemma 5.4 *Let m be a message in M_2 such that $m \neq srv$. There is no player $i \in P'$, round number $r \in N$, and action sequence $q \in Q$ such that $(rcv(m), r) \in H_i(q)$.*

Lemma 5.1 states that if B receives a message $m = (\mu, \rho, \sigma) \in M_4$ in some round r, then V must receive μ before round r. Lemma 5.2 states that if in addition $h(\rho) = h(rnd)$, then V must also receive ρ before round r. Lemma 5.3 states that if B receives a message $m = (\mu, \sigma) \in M'_4$ in some round r, then V must receive μ before round r. Finally, Lemma 5.4 states that no player can ever receive a message $m \in M_2$ such that $m \neq srv$. Although these statements are intuitively clear from the logical formulae with which we tagged the messages in $M_\pi(L)$, we present a formal proof of each of them in Appendix B.

5.2.4 Payoffs

Since the definitions of termination, effectiveness, and rationality involve only the restricted protocol game $G_{\pi|\bar{s}}(L)$, where $\bar{s} = (s^*_B, s^*_{net})$, we define the payoffs for U and V only in this restricted game.

Let us consider a terminal action sequence q in $G_{\pi|\bar{s}}(L)$. The payoff of $i \in \{U, V\}$ in q is $y_i(q) = y_i^+(q) - y_i^-(q)$, where $y_i^+(q)$ is the gain of i, and $y_i^-(q)$ represents the loss of i in q. Let us denote the value that the agreed service srv is worth for U and V by u_U^{srv} and u_V^{srv}, respectively. We assume that $u_U^{srv} > val > u_V^{srv} > 0$.

The gain of U is u_U^{srv} if U received srv in q, otherwise it is 0. The loss of U depends on the message that B received in q: if B received a message $(\iota_1, \iota_2, \tau, \nu, \eta, \sigma, \rho, \sigma') \in M_4$ or a message $(\iota_1, \iota_2, \tau, \nu, \eta, \sigma, \sigma') \in M'_4$ in q, then the loss of U is ν. If B did not receive any message $m \in M_4 \cup M'_4$, then the loss of U is 0. Similarly, the gain of V in q is ν if B received a message $(\iota_1, \iota_2, \tau, \nu, \eta, \sigma, \rho, \sigma') \in M_4$ in q, otherwise it is 0, and the loss of V is u_V^{srv} if V sent srv in q, and 0 otherwise.

Note that, in principle, the bank has no information about the agreed value val of the service, and therefore, it accepts any message $m \in M_4 \cup M'_4$ regardless of the value ν of the payment inside the message. This means that, seemingly, the loss of U and the gain of V could take other values than 0 and the agreed value val of the payment. However, we prove that this is never the case:

Lemma 5.5 *For every terminal action sequence q of the restricted protocol game $G_{\pi|\bar{s}}(L)$, where $\bar{s} = (s^*_B, s^*_{net})$, if B received a message $(\iota_1, \iota_2, \tau, \nu, \eta, \sigma, \rho, \sigma') \in M_4$ or a message $(\iota_1, \iota_2, \tau, \nu, \eta, \sigma, \sigma') \in M'_4$ in q, then $\nu = val$.*

Proof: Let us consider a message $m = (\iota_1, \iota_2, \tau, \nu, \eta, \sigma, \rho, \sigma') \in M_4$, where $\nu \neq val$, and let us assume that B received m in q. According to Lemma 5.1, this means that V must have received $m' = (\iota_1, \iota_2, \tau, \nu, \eta, \sigma)$ in q. However, this is impossible, because $m' \notin M_\pi(L)$.

In the same way, using Lemma 5.3, we can prove the case of a message $m = (\iota_1, \iota_2, \tau, \nu, \eta, \sigma, \sigma') \in M'_4$, where $\nu \neq val$. □

Taking into account Lemma 5.5, we can define the payoffs for U and V in $G_{\pi|\bar{s}}(L)$ formally as follows:

$$
\begin{aligned}
y_U^+(q) &= \begin{cases} u_U^{srv} & \text{if } \phi_U^+(q) = \mathsf{true} \\ 0 & \text{otherwise} \end{cases} \\
y_U^-(q) &= \begin{cases} val & \text{if } \phi_U^-(q) = \mathsf{true} \\ 0 & \text{otherwise} \end{cases} \\
y_V^+(q) &= \begin{cases} val & \text{if } \phi_V^+(q) = \mathsf{true} \\ 0 & \text{otherwise} \end{cases} \\
y_V^-(q) &= \begin{cases} u_V^{srv} & \text{if } \phi_V^-(q) = \mathsf{true} \\ 0 & \text{otherwise} \end{cases}
\end{aligned}
$$

where

$$
\begin{aligned}
\phi_U^+(q) &= (\exists r \in N : (rcv(srv), r) \in H_U(q)) \\
\phi_U^-(q) &= ((\exists r \in N, m \in M_4 : (rcv(m), r) \in H_B(q)) \ \vee \\
&\quad\ (\exists r' \in N, m' \in M_4' : (rcv(m'), r') \in H_B(q))) \\
\phi_V^+(q) &= (\exists r \in N, m \in M_4 : (rcv(m), r) \in H_B(q)) \\
\phi_V^-(q) &= (\exists r \in N, i \in \{U, B\} : (snd(srv, i), r) \in H_V(q))
\end{aligned}
$$

5.2.5 The proof

We will now prove that the payment protocol of Section 3.2 is rational.

Lemma 5.6 (Gain closed property) *The payment protocol of Section 3.2 is gain closed.*

Proof: It is enough to show that (i) $\phi_U^+(q)$ implies $\phi_V^-(q)$ and (ii) $\phi_V^+(q)$ implies $\phi_U^-(q)$ for every $q \in Q_{|\bar{s}}$. (i) follows from the fact that in the restricted game only two players U and V send messages, and therefore, if U receives a message m, then V must send m. (ii) is trivial. □

Lemma 5.7 (Safe back out property) *The payment protocol of Section 3.2 satisfies the safe back out property.*

Proof: If U does not send any messages, then according to Lemma 5.1 and Lemma 5.3, V cannot send any message in $M_4 \cup M_4'$ to B, which means that the loss of U is 0. If V quits at the beginning then it does not send *srv* to anybody, and therefore, its loss is 0. □

Lemma 5.8 *The strategy profile $(s^*_{U|\bar{s}}, s^*_{V|\bar{s}})$ is a Nash equilibrium in the restricted protocol game $G_{\pi|\bar{s}}(L)$, where $\bar{s} = (s^*_B, s^*_{net})$.*

Proof: We have to prove that (i) if U follows $s^*_{U|\bar{s}}$, then the best response of V to this is to follow $s^*_{V|\bar{s}}$, and (ii) if V follows $s^*_{V|\bar{s}}$, then the best response of U to this is to follow $s^*_{U|\bar{s}}$.

(i) Let q^* be the outcome of the game $G_{\pi|\bar{s}}(L)$ when U and V follow $s^*_{U|\bar{s}}$ and $s^*_{V|\bar{s}}$, respectively (i.e., $q^* = o_{|\bar{s}}(s^*_{U|\bar{s}}, s^*_{V|\bar{s}})$). It is easy to verify that $y_U(q^*) = y_U^+(q^*) - y_U^-(q^*) = u_U^{srv} - val > 0$, and $y_V(q^*) = y_V^+(q^*) - y_V^-(q^*) = val - u_V^{srv} > 0$.

Now, let us suppose that $s^*_{V|\bar{s}}$ is not the best response to $s^*_{U|\bar{s}}$. This means that V has a strategy $s'_{V|\bar{s}}$ such that if U follows $s^*_{U|\bar{s}}$ and V follows $s'_{V|\bar{s}}$, then V achieves a higher payoff than $val - u^{srv}_V$. In other words, if $q' = o_{|\bar{s}}(s^*_{U|\bar{s}}, s'_{V|\bar{s}})$, then $y_V(q') = y^+_V(q') - y^-_V(q') > val - u^{srv}_V$. This is possible only if $y^+_V(q') = val$ and $y^-_V(q') = 0$.

From $y^-_V(q') = 0$, we get that $\phi^-_V(q')$ must be false, which means that V does not send srv in q'. This means that U does not receive srv in q'. In addition, because of Lemma 5.4, U cannot receive any other message in M_2 either. Thus, U does not receive any message in M_2 in q'. Since U follows $s^*_{U|\bar{s}}$, if she does not receive any message in M_2, then she quits the protocol in round 3. This means that the only message that U sends in q' is m_1. Since B does not send any messages in the restricted game, this also means that the only message that V receives in q' is m_1.

From $y^+_V(q') = val$, we get that $\phi^+_V(q')$ must be true, which means that B receives a message $m' = (\mu, \rho, \sigma) \in M_4$ in q'. According to Lemma 5.1, this is possible only if V receives μ in q'. Since the only message that V receives in q is m_1, $\mu = m_1$ must hold. But in this case $h(\rho) = h(rnd)$, and from Lemma 5.2, we get that V must also receive ρ in q', which is a contradiction.

(ii) Now, let us suppose that $s^*_{U|\bar{s}}$ is not the best response to $s^*_{V|\bar{s}}$. This means that U has a strategy $s'_{U|\bar{s}}$ such that if $q' = o_{|\bar{s}}(s'_{U|\bar{s}}, s^*_{V|\bar{s}})$, then $y_U(q') = y^+_U(q') - y^-_U(q') > u^{srv}_U - val$. This is possible only if $y^+_U(q') = u^{srv}_U$ and $y^-_U(q') = 0$.

From $y^+_U(q') = u^{srv}_U$, we get that $\phi^+_U(q')$ must be true, which means that U receives srv in q'. This means that V sends srv in q'. Since V follows $s^*_{V|\bar{s}}$, she sends srv only if she receives a message $m \in M_1$ before round 2.

From $y^-_U(q') = 0$ we get that $\phi^-_U(q')$ must be false, which means that B does not receive any message $m' \in M_4$, neither it receives any message $m'' \in M'_4$ in q'. This is a contradiction, because V follows $s^*_{V|\bar{s}}$, and in round 4, she can send $(m, sig(k^{-1}_V, m)) \in M'_4$ to B. □

Lemma 5.9 *Both U and V prefer $(s^*_{U|\bar{s}}, s^*_{V|\bar{s}})$ to any other Nash equilibrium $(s'_{U|\bar{s}}, s'_{V|\bar{s}})$ in $G_{\pi|\bar{s}}(L)$.*

Proof: Let us suppose that there exists a Nash equilibrium $(s'_{U|\bar{s}}, s'_{V|\bar{s}})$ in $G_{\pi|\bar{s}}(L)$ such that $y_U(q') > y_U(q^*) = u^{srv}_U - val$, where $q' = o_{|\bar{s}}(s'_{U|\bar{s}}, s'_{V|\bar{s}})$ and $q^* = o_{|\bar{s}}(s^*_{U|\bar{s}}, s^*_{V|\bar{s}})$. This is possible only if $y^+_U(q') = u^{srv}_U$ and $y^-_U(q') = 0$. Since $G_{\pi|\bar{s}}(L)$ is gain closed, $y^+_U(q') = u^{srv}_U > 0$ implies $y^-_V(q') > 0$, and $y^-_U(q') = 0$ implies $y^+_V(q') = 0$. Therefore, if U follows $s'_{U|\bar{s}}$ and V follows $s'_{V|\bar{s}}$, then V's payoff is $y_V(q') = y^+_V(q') - y^-_V(q') < 0$. Note, however, that the protocol satisfies the safe back out property, which means that if V quits at the beginning of the game without doing anything else, then her payoff cannot be negative, whatever strategy is followed by U. This means that $s'_{V|\bar{s}}$ is not the best response to $s'_{U|\bar{s}}$, and thus, $(s'_{U|\bar{s}}, s'_{V|\bar{s}})$ cannot be a Nash equilibrium.

In the same way, we can prove that V prefers $(s^*_{U|\bar{s}}, s^*_{V|\bar{s}})$ to any other Nash equilibrium in $G_{\pi|\bar{s}}(L)$. □

From Lemma 5.8 and Lemma 5.9, we obtain the main result of this section:

Proposition 5.1 (Rationality) *The payment protocol described in Section 3.2 is rational.*

In addition, it is easy to prove that the protocol is also terminating and effective:

Lemma 5.10 (Termination) *The payment protocol of Section 3.2 is terminating.*

Proof: It is easy to see that no matter which strategy is followed by U, if V follows $s^*_{V|\bar{s}}$, then it will quit the game in round 5 at latest. Similarly, no matter which strategy is followed by V, if U follows $s^*_{U|\bar{s}}$, then she will quit the game in round 4 at latest. □

Lemma 5.11 (Effectiveness) *The payment protocol of Section 3.2 is effective.*

Proof: It is easy to see that if U follows $s^*_{U|\bar{s}}$ and V follows $s^*_{V|\bar{s}}$, then U receives srv and B receives $m_4 = (m_1, rnd, sig(k_V^{-1},\ (m_1, rnd))) \in M_4$, where
$m_1 = (U,\ V,\ tid,\ val,\ h(rnd),\ sig(k_U^{-1}, (U, V, tid, val, h(rnd))))$. □

5.3 Some limitations of the model

The analysis in the previous section also reveals some limitations of our model. First of all, there is no feature of the model that deals with the limited communicating capabilities of the players. A player can send as many messages as she wishes to send (from the set of messages that she is able to send, of course, but this can be a very large set) in a single round. Note, however, that if we can prove that the best response of A to the correct behavior of B is correct behavior even if A is given more communicating power than she really has, then correct behavior must also be A's best response when her communicating capabilities are limited.

Similarly, there is no feature of the model that deals with the limited computing power of the players. Therefore, we must explicitly deny the players to perform actions that are based on computationally infeasible operations. For instance, a player cannot send a message that contains a digital signature of another player unless the first player received the signature earlier. Similarly, a player cannot send a message that contains a previously unknown preimage of a given cryptographic hash value unless the player received such a preimage earlier (that is why the vendor cannot send a message that matches the description dsc but does not equal srv, unless it received such a message earlier).

A similar issue is the treatment of random secret values (e.g., the random number rnd) in our model. No feature allows us to model that a player can generate random values (e.g., numbers, keys, etc.) but she is not lucky enough to guess a specific secret value (given that the space from which this secret value has been chosen is large enough). Therefore, we must explicitly deny a player to perform an action that is based on guessing a value that is supposed to be a secret to the player. We admit that this may lead to somewhat counterintuitive models. In the previous section, for instance, we allow the vendor to send any random number except for random numbers that hash into $h(rnd)$ (unless the vendor received such a number earlier).

Despite the limitations listed above, our model is still useful, since it allows us to draw interesting conclusions about exchange protocols with reasonable effort. Although it is possible to define a model that addresses the above issues, it would certainly be more complex, and thus, more difficult to use than our model. A similar trade-off is made in the Abadi-Tuttle logic [AT91], where the analysis of a key transport protocol is restricted to "good" runs, in which initially held beliefs about the secrecy of cryptographic keys remain true.

5.4 Proof of Syverson's rational exchange protocol

In [Syv98], Syverson describes the following rational exchange protocol:

The Syverson protocol	
$A \to B:$	$m_1 = (dsc_A,\ enc(k, itm_A),\ w(k),\ sig(k_A^{-1}, (dsc_A, enc(k, itm_A), w(k))))$
$B \to A:$	$m_2 = (itm_B,\ m_1,\ sig(k_B^{-1}, (itm_B, m_1)))$
$A \to B:$	$m_3 = (k,\ m_2,\ sig(k_A^{-1}, (k, m_2)))$

where A and B denote the two protocol participants; itm_A and itm_B denote the items that they want to exchange; dsc_A denotes the descriptions of itm_A; and k denotes a randomly chosen secret key. In addition, *enc* is a symmetric-key encryption function that takes as input a key κ and a message μ, and outputs the encryption of μ with κ; *sig* is a signature generation function defined in the same way as in Section 5.2; and w is a *temporarily secret (bit) commitment* function. Before going into the details of the protocol, we comment on the concept of temporarily secret (bit) commitment, which was introduced in [Syv98].

5.4.1 Temporarily secret (bit) commitment

Bit commitment is a way to commit to a value without revealing that value. Actually, most of the schemes proposed in the literature can be used to commit not only to a single bit, but to a whole bit string; so "bit" commitment is a slight misnomer. The idea of temporarily secret (bit) commitment is similar to that of (bit) commitment. The difference is that the secrecy of the commitment is breakable within acceptable bounds on time (computation). More precisely, if w is a temporarily secret (bit) commitment function, then given $w(x)$, one can determine the bit string x in time t, where t lies between acceptable lower and upper bounds.

At first sight, encryption with a reduced length key seems to be an appropriate candidate for a temporarily secret (bit) commitment function, since such an encryption can be broken with reasonable effort. If the goal were simply to have some computational load involved in breaking the commitment, then this would be fine. The problem is that it is easy to break up the key space, and execute a search for the key in each part of the key space concurrently. This means that no lower bound can be guaranteed on the time needed to break the commitment: the more processors one has, the faster one finds the key. This observation suggests that a temporarily secret (bit) commitment function should be based on an "inherently sequential" computation.

One approach to implement temporarily secret (bit) commitment is based on the time-lock puzzle of Rivest, Shamir, and Wagner [RSW96]. The idea is the following: We randomly choose two large primes p and q and a number a, and we compute $a^{2^t} (\mathrm{mod} n)$, where $n = pq$ and t is a parameter that determines the time bounds within which the commitment can be broken. Knowing the factors of n allows us to compute $a^{2^t} (\mathrm{mod} n)$ efficiently. Then, the temporarily secret (bit) commitment to $x < n$ is defined as $w(x) = (n,\ a,\ t,\ x + a^{2^t} (\mathrm{mod}\, n))$. If $x \geq n$, then we can first encrypt x with a randomly chosen secret key $k < n$, and compute $w(k)$ as described above.

Computing $a^{2^t} (\mathrm{mod}\ n)$ from n, a, and t does not seem to be parallelizable in any way. Thus, someone who knows only $w(x)$ (and does not know p and q) can only determine x

by performing t squaring operation sequentially. The only advantage could be gained by computing $a^{2^t} (\mathrm{mod} n)$ on a faster processor, but in practice, the range of available processor speeds can be readily determined, and t can be adjusted appropriately.

5.4.2 Detailed protocol description

Now, we continue with the description of the Syverson protocol. In the first step, A generates a random secret key k; encrypts item itm_A with k; computes the temporarily secret (bit) commitment $w(k)$; generates a digital signature on the description dsc_A of itm_A, the encryption of itm_A, and the commitment $w(k)$; and sends message m_1 to B.

When B receives m_1, she verifies the digital signature and the description dsc_A of the expected item. If B is satisfied, then she sends message m_2 to A. m_2 contains item itm_B, the received message m_1, and a digital signature of B on these elements.

When A receives m_2, she verifies the digital signature, checks if the received message contains m_1, and checks if the received item matches the expectations. If she is satisfied, then she sends the key k to B in message m_3, which also contains the received message m_2 and the digital signature of A on the message content.

When B receives m_3, she verifies the digital signature, and checks if the received message contains m_2. Then, B decrypts the encrypted item in m_1 (also received as part of m_3) with the key received in m_3.

5.4.3 Brief informal analysis

When B receives m_1, she has something that either turns out to be what she wants or evidence that A cheated, which can be used against A in a dispute. At this point, B might try to break the commitment $w(k)$ in order to obtain k and then itm_A. However, this requires time. If itm_A does not lose its value in time, and the inconvenience of the delay (and the computation) is not an issue for B, then breaking the commitment is indeed the best strategy for B. The Syverson protocol should not be used in this case. So it is assumed that itm_A has a diminishing value in time (e.g., it could be a short term investment advice), and that it is practically worth nothing by the time at which B can break the commitment [Syv98]. Therefore, B is interested in continuing the protocol by sending m_2 to A.

When A receives m_2, she might not send m_3 at all or for a long time. If A does not lose anything until B gets access to itm_A, then this is indeed a good strategy for A. If this is the case, then the Syverson protocol should not be used. So it is assumed that A loses control over itm_A by sending it to B in m_1, even if she sends it only in an encrypted form. In this case, A does not gain anything by not sending m_3 to B promptly.

Note, however, that A may send some garbage instead of the encrypted item in m_1. A deterrent against this is that the commitment can be broken anyhow, which means that the misbehavior of A can be discovered by B. In addition, since m_1 is signed by A, it can be used against A in a dispute. If some punishment (the value of which greatly exceeds the value of the exchanged items) for the misbehavior can be enforced, then it is not in the interest of A to cheat. Note that this punishment could be enforced externally (e.g., by law enforcement).

In the formal analysis of the Syverson protocol, we will take these observations into consideration. The analysis will essentially follow the same steps as that in Section 5.2.

5.4.4 The set of compatible messages

In order to define the set of messages that are compatible with the protocol, we must first introduce some further notation:

- dsc_B will denote the description of itm_B; and
- dec will denote the decryption function that belongs to enc, which takes a key κ and a ciphertext ε, and returns the decryption of ε with κ.

Next, we reconstruct the programs of the protocol participants:

$\pi(A, k_A, k_A^{-1}, B, k_B, k_B^{-1}, itm_A, dsc_A, itm_B, dsc_B, k) = \{$
$\pi_A(A, k_A^{-1}, B, k_B, itm_A, dsc_A, dsc_B, k),$
$\pi_B(B, k_B^{-1}, A, k_A, itm_B, dsc_A)\}$

where

$\pi_A(A, k_A^{-1}, B, k_B, itm_A, dsc_A, dsc_B, k) =$
1. compute $\varepsilon = enc(k, itm_A)$
2. compute $\omega = w(k)$
3. compute $\sigma = sig(k_A^{-1}, (dsc_A, \varepsilon, \omega))$
4. send $(dsc_A, \varepsilon, \omega, \sigma)$ to B
5. wait until timeout or
   a message $m = (\gamma, \mu, \sigma')$ arrives such that
   - $\mu = (dsc_A, \varepsilon, \omega, \sigma)$
   - $fit(\gamma, dsc_B) = \mathsf{true}$
   - $vfy(k_B,\ (\gamma, \mu),\ \sigma') = \mathsf{true}$
6. if timeout then go to step 9
7. compute $\sigma'' = sig(k_A^{-1}, (k, m))$
8. send (k, m, σ'') to B
9. exit

$\pi_B(B, k_B^{-1}, A, k_A, itm_B, dsc_A) =$
1. wait until timeout or
   a message $m = (\delta, \varepsilon, \omega, \sigma)$ arrives such that
   - $\delta = dsc_A$
   - $vfy(k_A,\ (\delta, \varepsilon, \omega),\ \sigma) = \mathsf{true}$
2. if timeout then go to step 6
3. compute $\sigma' = sig(k_B^{-1}, (itm_B, m))$
4. send (itm_B, m, σ') to A
5. wait until timeout or
   a message $m' = (\kappa, \mu, \sigma'')$ arrives such that
   - $\mu = (itm_B, m, \sigma')$
   - $fit(dec(\kappa, \varepsilon), dsc_A) = \mathsf{true}$
   - $vfy(k_A,\ (\kappa, \mu),\ \sigma'') = \mathsf{true}$
6. exit

Once the programs of the protocol participants are given, we can easily determine the set of compatible messages:

$$M_\pi(L) = M_\pi(A, k_A, k_A^{-1}, B, k_B, k_B^{-1}, itm_A, dsc_A, itm_B, dsc_B, k) = M_1 \cup M_2 \cup M_3$$

where

$$\begin{aligned} M_1 = \{(\delta, \varepsilon, \omega, \sigma) : \ & \delta = dsc_A, \\ & vfy(k_A,\ (\delta, \varepsilon, \omega),\ \sigma) = \mathsf{true}\} \end{aligned}$$

$$\begin{aligned} M_2 = \{(\gamma, \mu, \sigma) : \ & \mu \in M_1, \\ & fit(\gamma, dsc_B) = \mathsf{true}, \\ & vfy(k_B,\ (\gamma, \mu),\ \sigma) = \mathsf{true}\} \end{aligned}$$

$$\begin{aligned} M_3 = \{(\kappa, \gamma, \delta, \varepsilon, \omega, \sigma, \sigma', \sigma'') : \ & \\ & (\gamma, \delta, \varepsilon, \omega, \sigma, \sigma') \in M_2, \\ & fit(dec(\kappa, \varepsilon), dsc_A) = \mathsf{true}, \\ & vfy(k_A,\ (\kappa, \gamma, \delta, \varepsilon, \omega, \sigma, \sigma'),\ \sigma'') = \mathsf{true}\} \end{aligned}$$

5.4.5 The protocol game

As we saw in Section 5.2, the next step in the construction of the protocol game is to tag each message $m \in M_\pi(L)$ with a vector $(\phi_i^m(\Sigma_i(q)))_{i \in P'}$ of logical formulae, where each $\phi_i^m(\Sigma_i(q))$ describes the condition that must be satisfied in order for player i to be able to send message m in the information set represented by the local state $\Sigma_i(q)$. For the Syverson protocol, these vectors of logical formulae are the following:

- Since B cannot generate valid digital signatures of A, B can send a message $m \in M_1$ only if she received m or a message that contained m earlier. In addition, we assume that A cannot generate a fake item, different from itm_A, that matches the description dsc_A of itm_A. Similarly, we assume that A cannot randomly generate a ciphertext ε, and a key κ or a commitment $\omega = w(\kappa)$ such that $dec(\kappa, \varepsilon)$ matches dsc_A. In other words, if for some message $m = (\delta, \varepsilon, \omega, \sigma) \in M_1$, $fit(dec(w^{-1}(\omega), \varepsilon), dsc_A) = \mathsf{true}$ and $dec(w^{-1}(\omega), \varepsilon) \neq itm_A$, then A can send m only if she received m or a message that contains m earlier.

  Formally, for any $m = (\delta, \varepsilon, \omega, \sigma) \in M_1$:

  - if $fit(dec(w^{-1}(\omega), \varepsilon), dsc_A) = \mathsf{false}$ or $dec(w^{-1}(\omega), \varepsilon) = itm_A$:

    $$\begin{aligned} \phi_A^m(\Sigma_A(q)) &= (\alpha_A(q) = \mathsf{true}) \\ \phi_B^m(\Sigma_B(q)) &= (\alpha_B(q) = \mathsf{true}) \wedge \\ &\quad ((\exists r < r_B(q) : (\mathrm{rcv}(m), r) \in H_B(q)) \vee \\ &\quad (\exists r < r_B(q), m' = (\gamma', m, \sigma') \in M_2 : (\mathrm{rcv}(m'), r) \in H_B(q)) \vee \\ &\quad (\exists r < r_B(q), m' = (\kappa', \gamma', m, \sigma', \sigma'') \in M_3 : (\mathrm{rcv}(m'), r) \in H_B(q))) \end{aligned}$$

  - otherwise (i.e., if $fit(dec(w^{-1}(\omega), \varepsilon), dsc_A) = \mathsf{true}$ and $dec(w^{-1}(\omega), \varepsilon) \neq itm_A$):

$$\begin{aligned}
\phi_A^m(\Sigma_A(q)) = {} & (\alpha_A(q) = \mathsf{true}) \\
& ((\exists r < r_A(q) : (rcv(m), r) \in H_A(q)) \vee \\
& \ (\exists r < r_A(q), m' = (\gamma', m, \sigma') \in M_2 : (rcv(m'), r) \in H_A(q)) \vee \\
& \ (\exists r < r_A(q), m' = (\kappa', \gamma', m, \sigma', \sigma'') \in M_3 : (rcv(m'), r) \in H_A(q))) \\
\phi_B^m(\Sigma_B(q)) = {} & (\alpha_B(q) = \mathsf{true}) \wedge \\
& ((\exists r < r_B(q) : (rcv(m), r) \in H_B(q)) \vee \\
& \ (\exists r < r_B(q), m' = (\gamma', m, \sigma') \in M_2 : (rcv(m'), r) \in H_B(q)) \vee \\
& \ (\exists r < r_B(q), m' = (\kappa', \gamma', m, \sigma', \sigma'') \in M_3 : (rcv(m'), r) \in H_B(q)))
\end{aligned}$$

- Since A cannot generate valid digital signatures of B, A can send a message $m \in M_2$ only if she received m or a message that contains m earlier. For similar reasons, B can send a message $m = (\gamma, \mu, \sigma) \in M_2$ only if she received $\mu \in M_1$ or a message that contains μ earlier. In addition, we assume that B cannot generate a fake item, different from itm_B, that matches the description dsc_B of itm_B. This means that if $\gamma \neq itm_B$, then B can send m only if she received γ or a message that contains γ earlier.

  Formally, for any $m = (\gamma, \mu, \sigma) \in M_2$:

  - if $\gamma = itm_B$:

$$\begin{aligned}
\phi_A^m(\Sigma_A(q)) = {} & (\alpha_A(q) = \mathsf{true}) \wedge \\
& ((\exists r < r_A(q) : (rcv(m), r) \in H_A(q)) \vee \\
& \ (\exists r < r_A(q), m' = (\kappa', m, \sigma') \in M_3 : (rcv(m'), r) \in H_A(q))) \\
\phi_B^m(\Sigma_B(q)) = {} & (\alpha_B(q) = \mathsf{true}) \wedge \\
& ((\exists r < r_B(q) : (rcv(\mu), r) \in H_B(q)) \vee \\
& \ (\exists r < r_B(q), m' = (\gamma', \mu, \sigma') \in M_2 : (rcv(m'), r) \in H_B(q)) \vee \\
& \ (\exists r < r_B(q), m' = (\kappa', \gamma', \mu, \sigma', \sigma'') \in M_3 : (rcv(m'), r) \in H_B(q)))
\end{aligned}$$

  - if $\gamma \neq item_B$:

$$\begin{aligned}
\phi_A^m(\Sigma_A(q)) = {} & (\alpha_A(q) = \mathsf{true}) \wedge \\
& ((\exists r < r_A(q) : (rcv(m), r) \in H_A(q)) \vee \\
& \ (\exists r < r_A(q), m' = (\kappa', m, \sigma') \in M_3 : (rcv(m'), r) \in H_A(q))) \\
\phi_B^m(\Sigma_B(q)) = {} & (\alpha_B(q) = \mathsf{true}) \ \wedge \ \varphi_1 \ \wedge \ \varphi_2
\end{aligned}$$

    where

$$\begin{aligned}
\varphi_1 = {} & (\exists r < r_B(q) : (rcv(\mu), r) \in H_B(q)) \vee \\
& (\exists r < r_B(q), m' = (\gamma', \mu, \sigma') \in M_2 : (rcv(m'), r) \in H_B(q)) \vee \\
& (\exists r < r_B(q), m' = (\kappa', \gamma', \mu, \sigma', \sigma'') \in M_3 : (rcv(m'), r) \in H_B(q)) \\
\varphi_2 = {} & (\exists r < r_B(q), m' = (\gamma, \mu', \sigma') \in M_2 : (rcv(m'), r) \in H_B(q)) \vee \\
& (\exists r < r_B(q), m' = (\kappa', \gamma, \mu', \sigma', \sigma'') \in M_3 : (rcv(m'), r) \in H_B(q))
\end{aligned}$$

- Since B cannot generate valid digital signatures of A, B can send a message $m \in M_3$ only if she received m earlier (there cannot be another message that contains m in this case). For similar reasons, A can send a message $m = (\kappa, \mu, \sigma) \in M_3$ only if she received $\mu \in M_2$ or a message that contains μ earlier. Note, however, that in general, receiving μ is not sufficient for A to be able to send $m = (\kappa, \mu, \sigma)$, because if the ciphertext ε within μ was not computed by A using the key κ (e.g., if A generated ε randomly), then

A may not be able to guess κ. Nevertheless, since our proofs will rely only on the fact that A must receive μ before sending $m = (\kappa, \mu, \sigma)$, we generously give A the power to guess κ, and we consider that receiving μ is also sufficient for A to send $m = (\kappa, \mu, \sigma)$.

Formally, for any $m = (\kappa, \mu, \sigma) \in M_3$:

$$\begin{aligned}
\phi_A^m(\Sigma_A(q)) = {} & (\alpha_A(q) = \mathsf{true}) \wedge \\
& ((\exists r < r_A(q) : (rcv(\mu), r) \in H_A(q)) \vee \\
& \ (\exists r < r_A(q), m' = (\kappa', \mu, \sigma') \in M_3 : (rcv(m'), r) \in H_A(q))) \\
\phi_B^m(\Sigma_B(q)) = {} & (\alpha_B(q) = \mathsf{true}) \wedge \\
& (\exists r < r_B(q) : (rcv(m), r) \in H_B(q))
\end{aligned}$$

The above logical formulae allow us to complete the construction of the protocol game. Before determining the payoffs and describing the strategies that correspond to the programs of the protocol participants, we can already make a few simple statements about the protocol game:

Lemma 5.12 *If $(snd(m, B), r) \in H_A(q)$ for some message $m = (\kappa, \mu, \sigma) \in M_3$, round number $r \in N$, and action sequence $q \in Q$, then there exists $r' < r$ such that $(rcv(\mu), r') \in H_A(q)$.*

Lemma 5.13 *If $(snd(m, A), r) \in H_B(q)$ for some message $m = (\gamma, \mu, \sigma) \in M_2$, round number $r \in N$, and action sequence $q \in Q$, then there exists $r' < r$ such that $(rcv(\mu), r') \in H_B(q)$.*

Lemma 5.14 *Let m be a message in M_3. There is no round number $r < 3$ and action sequence $q \in Q$ such that $(rcv(m), r) \in H_B(q)$.*

Lemma 5.15 *Let $m = (\delta, \varepsilon, \omega, \sigma)$ be a message in M_1 such that $fit(dec(w^{-1}(\omega), \varepsilon), dsc_A) =$ true and $dec(w^{-1}(\omega), \varepsilon) \neq itm_A$. There is no player $i \in P'$, round number $r \in N$, and action sequence $q \in Q$ such that $(rcv(m), r) \in H_i(q)$.*

Lemma 5.16 *Let $m = (\gamma, \mu, \sigma)$ be a message in M_2 such that $\gamma \neq itm_B$. There is no player $i \in P'$, round number $r \in N$, and action sequence $q \in Q$ such that $(rcv(m), r) \in H_i(q)$.*

Lemma 5.12 states that if A sends a message $m = (\kappa, \mu, \sigma) \in M_3$ in round r in q, then she must receive μ in an earlier round $r' < r$ in q. Similarly, Lemma 5.13 states that if B sends a message $m = (\gamma, \mu, \sigma) \in M_2$ in round r in q, then she must receive μ in an earlier round $r' < r$ in q. Lemma 5.14 is a corollary of the first two lemmas that states that B cannot receive a message $m \in M_3$ before round 3. Finally, Lemma 5.15 states that no player can ever receive a message $m = (\delta, \varepsilon, \omega, \sigma) \in M_1$ such that $fit(dec(w^{-1}(\omega), \varepsilon), dsc_A) = \mathsf{true}$ and $dec(w^{-1}(\omega), \varepsilon) \neq itm_A$, and Lemma 5.16 states that no player can ever receive a message $m = (\gamma, \mu, \sigma) \in M_2$ such that $\gamma \neq itm_B$. The proofs of these lemmas are given in Appendix C.

5.4.6 Strategies

Like in Section 5.2, we introduce an ordering relation on the messages in $M_\pi(L)$ that we denote by $\leq$. We need this ordering in order to be able to define the strategies of the players unambiguously. The strategies that correspond to the programs π_A and π_B of the protocol participants are then defined as follows:

Strategy s_A^*

- If $\alpha_A(q) = \mathsf{true}$ and $r_A(q) = 1$, then perform the action $\mathsf{send}_A(\{(m_1,\ B)\})$, where $m_1 = (dsc_A,\ enc(k, itm_A),\ w(k),\ sig(k_A^{-1}, (dsc_A, enc(k, itm_A), w(k))))$.
- If $\alpha_A(q) = \mathsf{true}$ and $r_A(q) = 2$, then perform the action idle_A.
- If $\alpha_A(q) = \mathsf{true}$ and $r_A(q) = 3$, then let M be the set of those messages $m = (\gamma, \mu, \sigma) \in M_2$ for which $\mu = m_1$ and there exists a round number $r < 3$ such that $(rcv(m), r) \in H_A(q)$.
  - If $M = \emptyset$, then perform the action quit_A.
  - If $M \neq \emptyset$, then choose the smallest message m from M (according to $\leq$), and perform the action $\mathsf{send}_A(\{((k, m, sig(k_A^{-1}, (k, m))),\ B)\})$.
- If $\alpha_A(q) = \mathsf{true}$ and $r_A(q) = 4$, then perform the action quit_A.

Strategy s_B^*

- If $\alpha_B(q) = \mathsf{true}$ and $r_B(q) = 1$, then perform the action idle_B.
- If $\alpha_B(q) = \mathsf{true}$ and $r_B(q) = 2$, then let M be the set of those messages $m \in M_1$ for which there exists a round number $r < 2$ such that $(rcv(m), r) \in H_B(q)$.
  - If $M = \emptyset$, then perform the action quit_B.
  - If $M \neq \emptyset$, then choose the smallest message m from M (according to $\leq$), and perform the action $\mathsf{send}_B(\{((itm_B, m, sig(k_B^{-1}, (itm_B, m))),\ A)\})$
- If $\alpha_B(q) = \mathsf{true}$ and $r_B(q) = 3$, then perform the action idle_B.
- If $\alpha_B(q) = \mathsf{true}$ and $r_B(q) = 4$, then perform the action quit_B.

5.4.7 Payoffs

In this subsection we define the payoffs for the players. We must slightly modify the payoff framework introduced in Subsection 4.3.7, in order to take into account that the value of itm_A diminishes in time. We also have to consider the potential punishment for A if she sends garbage in the first message of the protocol. Taking these into consideration, we define the payoffs of the players as follows.

Let us consider a terminal action sequence q in the protocol game. The payoff of A in q is $y_A(q) = y_A^+(q) - y_A^-(q)$, where $y_A^+(q)$ is the gain and $y_A^-(q)$ is the loss of A in q. Furthermore, the loss of A is defined as $y_A^-(q) = y_A^*(q) + y_A^{**}(q)$, where $y_A^*(q)$ is the loss that stems from losing control over itm_A, and $y_A^{**}(q)$ is the loss that stems from the punishment. The payoff of B in q is $y_B(q) = y_B^+(q) - y_B^-(q)$, where $y_B^+(q)$ is the gain and $y_B^-(q)$ is the loss of B in q.

We denote the values that itm_A and itm_B are worth to A by u_A^- and u_A^+, respectively. Similarly, we denote the value that itm_B is worth to B by u_B^-. The diminishing value of itm_A for B is modeled as a function $u_B^+(r)$, which decreases as the round number r increases (see part (a) of Figure 5.1). We assume that there exists a round number R such that $u_B^+(r) = 0$ for every $r \geq R$, and that breaking a commitment requires more than R rounds. Finally,

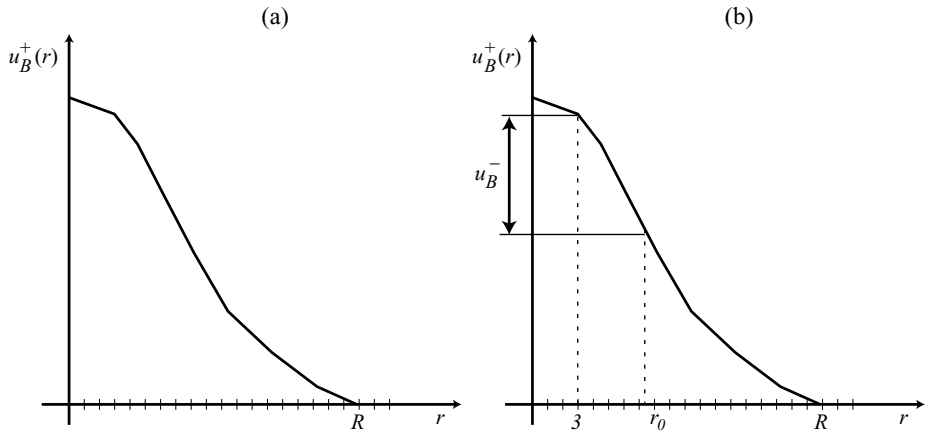

Figure 5.1: The diminishing value of itm_A for B is represented by a decreasing function $u_B^+(r)$. We assume that there exists a round number R such that $u_B^+(r) = 0$ for every $r \geq R$, and that breaking a commitment requires more than R rounds. We also assume that $u_B^+(3) > u_B^- > 0$. Finally, we define r_0 as the smallest round number such that $u_B^+(r_0) \leq u_B^+(3) - u_B^-$.

the value of the punishment is denoted by F. We assume that F is much greater than u_A^+, $u_A^+ > u_A^- > 0$, and $u_B^+(3) > u_B^- > 0$ (see also part (b) of Figure 5.1).

The gain of A is u_A^+ if A receives a message in M_2 that contains itm_B, otherwise it is 0. The value of $y_A^*(q)$ is u_A^- if A sends a message in M_1 that contains itm_A (in an encrypted form), or if A sends a message in M_3 that contains itm_A (in an encrypted form), otherwise it is 0. In addition, the punishment $y_A^{**}(q)$ of A is F if she sends an incorrect message in M_1 that, after breaking the commitment and decrypting the ciphertext in the message, yields an item that does not match the description dsc_A; otherwise the punishment is 0.

The gain of B is $u_B^+(r)$ if B receives a message in M_3 in round r that contains itm_A and no such message is received before round r. Note that receiving only a message in M_1 yields no gain for B, because we assume that by the time at which the commitment can be broken, itm_A loses its value for B. The loss of B is u_B^- if B sends a message in M_2 that contains itm_B, otherwise it is 0.

The formal definitions are given below:

$$y_A^+(q) = \begin{cases} u_A^+ & \text{if } \phi_A^+(q) = \mathsf{true} \\ 0 & \text{otherwise} \end{cases}$$

$$y_A^*(q) = \begin{cases} u_A^- & \text{if } \phi_A^*(q) = \mathsf{true} \\ 0 & \text{otherwise} \end{cases}$$

$$y_A^{**}(q) = \begin{cases} F & \text{if } \phi_A^{**}(q) = \mathsf{true} \\ 0 & \text{otherwise} \end{cases}$$

$$y_B^+(q) = \begin{cases} u_B^+(1) & \text{if } \phi_B^+(q,1) = \mathsf{true} \\ u_B^+(2) & \text{if } \phi_B^+(q,2) = \mathsf{true} \\ \dots & \\ u_B^+(R-1) & \text{if } \phi_B^+(q,R-1) = \mathsf{true} \\ 0 & \text{otherwise} \end{cases}$$

$$y_B^-(q) \quad = \quad \begin{cases} u_B^- & \text{if } \phi_B^-(q) = \mathsf{true} \\ 0 & \text{otherwise} \end{cases}$$

where

$$\begin{aligned}
\phi_A^+(q) &= (\exists r \in N, m = (\gamma, \mu, \sigma) \in M_2 : \\
&\qquad (\gamma = itm_B) \ \wedge \ ((rcv(m), r) \in H_A(q))) \\
\phi_A^*(q) &= (\exists r \in N, m = (\delta, \varepsilon, \omega, \sigma) \in M_1 : \\
&\qquad (dec(w^{-1}(\omega), \varepsilon) = itm_A) \ \wedge \ ((snd(m, B), r) \in H_A(q))) \ \vee \\
&\quad (\exists r \in N, m = (\kappa, \gamma, \delta, \varepsilon, \omega, \sigma, \sigma', \sigma'') \in M_3 : \\
&\qquad (dec(\kappa, \varepsilon) = itm_A) \ \wedge \ ((snd(m, B), r) \in H_A(q))) \\
\phi_A^{**}(q) &= (\exists r \in N, m = (\delta, \varepsilon, \omega, \sigma) \in M_1 : \\
&\qquad (fit(dec(w^{-1}(\omega), \varepsilon), dsc_A) = \mathsf{false}) \ \wedge \ ((snd(m, B), r) \in H_A(q))) \\
\phi_B^+(q, r) &= ((\exists m = (\kappa, \gamma, \delta, \varepsilon, \omega, \sigma, \sigma', \sigma'') \in M_3 : \\
&\qquad (dec(\kappa, \varepsilon) = itm_A) \ \wedge \ ((rcv(m), r) \in H_B(q))) \ \wedge \\
&\quad (\nexists r' < r, m = (\kappa, \gamma, \delta, \varepsilon, \omega, \sigma, \sigma', \sigma'') \in M_3 : \\
&\qquad (dec(\kappa, \varepsilon) = itm_A) \ \wedge \ ((rcv(m), r') \in H_B(q)))) \\
\phi_B^-(q) &= (\exists r \in N, m = (\gamma, \mu, \sigma) \in M_2 : \\
&\qquad (\gamma = itm_B) \ \wedge \ ((snd(m, A), r) \in H_B(q)))
\end{aligned}$$

Note that, by definition, $\phi_B^+(q, r) = \mathsf{true}$ holds for exactly one r, so $y_B^+(q)$ is well defined.

5.4.8 The proof

We will now prove that the Syverson protocol is rational.

Lemma 5.17 (Gain closed property) *The Syverson protocol is closed for gains.*

Proof: It is enough to prove that for every terminal action sequence q in $G_{\pi|\bar{s}}(L)$, (i) $\phi_A^+(q)$ implies $\phi_B^-(q)$ and (ii) for every r, $\phi_B^+(q, r)$ implies $\phi_A^*(q)$. Both (i) and (ii) follow from the fact that there are only two players A and B who send messages, which means that if player $i \in \{A, B\}$ receives a message m, then the other player $j \in \{A, B\}$, $j \neq i$ must send m. □

Lemma 5.18 (Safe back out property) *The Syverson protocol satisfies the safe back out property.*

Proof: If A does not send any messages in an action sequence q, then $\phi_A^*(q) = \mathsf{false}$ and $\phi_A^{**}(q) = \mathsf{false}$, and thus $y_A^-(q) = 0$. If B does not send any messages in an action sequence q, then $\phi_B^-(q) = \mathsf{false}$, and thus $y_B^-(q) = 0$. □

Lemma 5.19 *The strategy profile $(s^*_{A|\bar{s}}, s^*_{B|\bar{s}})$ is a Nash equilibrium in the restricted protocol game $G_{\pi|\bar{s}}(L)$, where $\bar{s} = (s^*_{net})$.*

Proof: We have to prove that (i) $s^*_{A|\bar{s}}$ is the best response to $s^*_{B|\bar{s}}$, and (ii) $s^*_{B|\bar{s}}$ is the best response to $s^*_{A|\bar{s}}$.

(i) Suppose that there is a strategy $s'_{A|\bar{s}}$ for A such that the payoff of A is higher if she plays $s'_{A|\bar{s}}$ than if she plays $s^*_{A|\bar{s}}$ against $s^*_{B|\bar{s}}$. This means that $y_A(q') > y_A(q^*)$, where $q^* = o_{|\bar{s}}(s^*_{A|\bar{s}}, s^*_{B|\bar{s}})$ and $q' = o_{|\bar{s}}(s'_{A|\bar{s}}, s^*_{B|\bar{s}})$. It is easy to verify that $y_A(q^*) = u^+_A - u^-_A$. Thus, $y_A(q') > y_A(q^*)$ is possible only if $y^+_A(q') = u^+_A$, $y^*_A(q') = 0$, and $y^{**}_A(q') = 0$.

From $y^+_A(q') = u^+_A$, it follows that A received a message $m = (\gamma, \delta, \varepsilon, \omega, \sigma, \sigma') \in M_2$ in q' such that $\gamma = itm_B$. This means that B sent m in q'. It follows from Lemma 5.13 that B can send m only if it received $(\delta, \varepsilon, \omega, \sigma) \in M_1$ from A earlier. Thus, A sent $(\delta, \varepsilon, \omega, \sigma) \in M_1$. Since $y^{**}_A(q') = 0$, $fit(dec(w^{-1}(\omega), \varepsilon), dsc_A)$ must be true. Furthermore, from Lemma 5.15, we get that $dec(w^{-1}(\omega), \varepsilon) = itm_A$. This means that $y^*_A(q')$ cannot be 0.

(ii) Suppose that there is a strategy $s'_{B|\bar{s}}$ for B such that the payoff of B is higher if she plays $s'_{B|\bar{s}}$ than if she plays $s^*_{B|\bar{s}}$ against $s^*_{A|\bar{s}}$. This means that $y_B(q') > y_B(q^*)$, where $q^* = o_{|\bar{s}}(s^*_{A|\bar{s}}, s^*_{B|\bar{s}})$ and $q' = o_{|\bar{s}}(s^*_{A|\bar{s}}, s'_{B|\bar{s}})$. It is easy to verify that $y_B(q^*) = u^+_B(3) - u^-_B$. Let r_0 be the smallest round number such that $u^+_B(r_0) \leq u^+_B(3) - u^-_B$ (see part (b) of Figure 5.1). Then, $y_B(q') > y_B(q^*)$ is possible only in two cases: (a) $y^+_B(q') = u^+_B(r)$, where $r < r_0$, and $y^-_B(q') = 0$, or (b) $y^+_B(q') = u^+_B(r)$, where $r < 3$. However, case (b) can never occur, because of Lemma 5.14. Therefore, we have to consider only case (a).

From $y^+_B(q') = u^+_B(r)$, it follows that B received a message $m = (\kappa, \gamma, \delta, \varepsilon, \omega, \sigma, \sigma', \sigma'') \in M_3$ such that $dec(\kappa, \varepsilon) = itm_A$ in round r in q'. This means that A sent m in q'. It follows from Lemma 5.12 that A can send m only if it received $(\gamma, \delta, \varepsilon, \omega, \sigma, \sigma') \in M_2$ from B earlier. Thus, B sent $(\gamma, \delta, \varepsilon, \omega, \sigma, \sigma') \in M_2$. From Lemma 5.16, we get that $\gamma = itm_B$. This means that $y^-_B(q')$ cannot be 0. □

Lemma 5.20 *Both A and B prefer $(s^*_{A|\bar{s}}, s^*_{B|\bar{s}})$ to any other Nash equilibrium $(s'_{A|\bar{s}}, s'_{B|\bar{s}})$ in $G_{\pi|\bar{s}}(L)$, where $\bar{s} = (s^*_{net})$.*

Proof: Let us suppose that there exists a Nash equilibrium $(s'_{A|\bar{s}}, s'_{B|\bar{s}})$ in $G_{\pi|\bar{s}}(L)$ such that $y_A(q') > y_A(q^*) = u^+_A - u^-_A$, where $q' = o_{|\bar{s}}(s'_{A|\bar{s}}, s'_{B|\bar{s}})$ and $q^* = o_{|\bar{s}}(s^*_{A|\bar{s}}, s^*_{B|\bar{s}})$. This is possible only if $y^+_A(q') = u^+_A$ and $y^*_A(q') = y^{**}_A(q') = 0$. Since the protocol is closed for gains, $y^+_A(q') = u^+_A > 0$ implies $y^-_B(q') > 0$, and $y^-_A(q') = 0$ implies $y^+_B(q') = 0$. Therefore, if A follows $s'_{A|\bar{s}}$ and B follows $s'_{B|\bar{s}}$, then B's payoff is $y_B(q') = y^+_B(q') - y^-_B(q') < 0$. Note, however, that because of the safe back out property, if B quits at the beginning of the game without doing anything else, then her payoff cannot be negative, whatever strategy is followed by A. This means that $s'_{B|\bar{s}}$ is not the best response to $s'_{A|\bar{s}}$, and thus, $(s'_{A|\bar{s}}, s'_{B|\bar{s}})$ cannot be a Nash equilibrium.

Now let us suppose that there exists a Nash equilibrium $(s'_{A|\bar{s}}, s'_{B|\bar{s}})$ in $G_{\pi|\bar{s}}(L)$ such that $y_B(q') > y_B(q^*) = u^+_B(3) - u^-_B$. This is possible only in two cases: (a) if $y^+_{p_2}(q') = u^+_B(r)$, where $r < r_0$ (see part (b) of Figure 5.1), and $y^-_B(q') = 0$, or (b) if $y^+_B(q') = u^+_B(r)$, where $r < 3$. However, case (b) can never occur, because of Lemma 5.14. Case (a) can be proven to be impossible using the same technique as in the first part of this proof. □

From Lemma 5.19 and Lemma 5.20, we obtain the main result of this section:

Proposition 5.2 (Rationality) *The Syverson protocol is rational.*

In addition, it is easy to prove the following:

Lemma 5.21 (Termination) *The Syverson protocol is terminating.*

Proof: According to the strategies s_A^* and s_B^*, both A and B quit in round 4 at latest. □

Recall that in the protocol game of the Syverson protocol the possible gain u_B^+ of player B is not a constant, but it is a function of r. This means that we have to modify the definition of effectiveness in order to take this into account[1]. Requiring that $y_B^+(q^*)$ (where $q^* = o_{|\bar{s}}(s^*_{A|\bar{s}}, s^*_{B|\bar{s}})$) be $u_B^+(3)$ is somewhat specific to the Syverson protocol and not general enough. Intuitively, what we expect from an effective protocol is that the payoffs of both players are positive if they both follow the protocol faithfully. Therefore, effectiveness in the Syverson protocol means that $y_A(q^*) > 0$ and $y_B(q^*) > 0$. Now, it is easy to prove the following:

Lemma 5.22 (Effectiveness) *The Syverson protocol is effective.*

Proof: It is easy to verify that $y_A(q^*) = u_A^+ - u_A^-$, and $y_B(q^*) = u_B^+(3) - u_B^-$. The statement from the lemma follows from the fact that we have assumed that $u_A^+ > u_A^-$ and $u_B^+(3) > u_B^-$. □

5.5 Replacing the reliable network with an unreliable one

In the previous sections, we proved that the payment protocol of Section 3.2 and Syverson's exchange protocol [Syv98] are rational exchange protocols. However, the proofs have been carried out in a model where the network is assumed to be reliable. In this section, we assume that the network is not reliable (i.e., there are no bounds on message delivery delays), and discuss how this assumption affects the properties of the above mentioned protocols.

Our main observation is that none of the protocols will be effective and rational. Let us consider first the payment protocol of Section 3.2, and let us assume that both players follow the strategy that corresponds to the correct execution of the protocol. Furthermore, let us assume that the players use timers, and when they receive a timeout signal, they continue with the next step of the protocol. In case of V not receiving *rnd* in time, this means that V sends m_4' to B; in all other cases it means that the player quits the protocol. Now it is easy to see that the network may follow a strategy in which *rnd* is delayed, so that V finally timeouts and sends m_4' to B. This means that there exists a a strategy vector $\bar{s}$ and thus a restricted protocol game $G_{\pi|\bar{s}}(L)$ such that $y_V^+(q^*) = 0$ and $y_V^-(q^*) = u_V^-$ (since *srv* has been sent), where $q^* = o_{|\bar{s}}(s^*_{U|\bar{s}}, s^*_{V|\bar{s}})$. $y_V^+(q^*) = 0$ means that the protocol is not effective. In addition, it can be seen that the total payoff of V in q^* is negative, so V would be better off if she did not participate in the exchange at all. In other words, $s^*_{V|\bar{s}}$ is not the best response to $s^*_{U|\bar{s}}$ in $G_{\pi|\bar{s}}(L)$, and so $(s^*_{U|\bar{s}}, s^*_{V|\bar{s}})$ cannot be a Nash equilibrium in $G_{\pi|\bar{s}}(L)$. This implies that the protocol is not rational.

Let us consider now the Syverson protocol. Let us assume again that both players follow the strategy that corresponds to the correct execution of the protocol, and that the players use timers in a similar way as in the previous case. Now the network may follow a strategy in

[1] If we were interested in fairness, then we would need to modify the definition of fairness as well, since it also involves u_B^+.

which m_3 is delayed, so that B finally timeouts and quits the protocol. This means that there exists a strategy vector $\bar{s}$ and thus a restricted protocol game $G_{\pi|\bar{s}}(L)$ such that $y_B^+(q^*) = 0$ and $y_B^-(q^*) = u_B^-$ (since m_2 has been sent), where $q^* = o_{|\bar{s}}(s^*_{A|\bar{s}}, s^*_{B|\bar{s}})$. Similarly to the previous case, this leads to the conclusion that the protocol is not effective and not rational.

If we assume that the network does not lose messages (i.e., eventually every submitted message is delivered), then we can retain effectiveness in the payment protocol of Section 3.2 by not using timers[2]: If the players follow the correct strategies and wait long enough for messages, then eventually they receive what they want to receive. However, not using timers is impractical. In addition, it does not help us to retain rationality. The reason is that if V behaves correctly and ignores timeouts then the best response of U to this is to quit after receiving *srv* without sending *rnd*, since in this case V will wait for *rnd* forever and will never contact B. With this strategy U's payoff will be u_U^{srv}, which is higher than the payoff $u_U^{srv} - val$ that she obtains if she follows the correct strategy. This means that the strategies that correspond to the correct behavior cannot form a Nash equilibrium in every restricted protocol game. In addition, the termination property is not satisfied either for obvious reasons.

[2]Note that this is not the case in the Syverson protocol, because even if the network eventually delivers m_3, it can still delay it for long enough a time so that itm_A loses its value for B.

Part III

Incentives to cooperate in ad hoc networks

Chapter 6

Stimulation for packet forwarding in ad hoc networks

6.1 Introduction

In this chapter, we describe how ideas similar to the concept of rational exchange could be used in a different context, namely, to stimulate cooperation in ad hoc networks. First, we motivate the problem by showing why cooperation is essential for the network as a whole, and why it is not in the interest of the nodes as individuals. Then, we propose a mechanism to stimulate cooperation. Our design shows strong similarities to rational exchange protocols in the sense that we do not prevent a node from denying cooperation or deviating from the proposed protocols, but we do ensure that the node cannot gain any advantages by doing so. Therefore, we expect that rational (self-interested) nodes will cooperate and deviations will happen only rarely. Finally, we study the behavior of the proposed mechanism by means of simulation.

An ad hoc network is a wireless multi-hop network formed by a set of potentially mobile nodes in a self-organizing way without relying on any established infrastructure. Due to the absence of infrastructure, in an ad hoc network, all networking functions must be performed by the nodes themselves. For instance, packets sent between two distant nodes are expected to be forwarded by intermediate nodes [CFS99, Per01].

The above mentioned operating principle renders cooperation among nodes an essential requirement in ad hoc networks. By cooperation, we mean that the nodes perform networking functions for the benefit of other nodes. Lack of cooperation may have fatal effects on the performance of the network. As an example, let us consider Figure 6.1, which illustrates that the cumulative throughput[1] of the network decreases dramatically as the fraction of the nodes that deny packet forwarding increases. The different curves belong to networks of different sizes (100, 200, 300, and 400 nodes) but with the same node density. The figure also shows that larger networks are more sensitive to this kind of non-cooperative behavior of the nodes. These results are based on our own simulations described in detail in Section 6.4, but similar results have been presented in [MGLB00] as well.

So far, applications of ad hoc networks have been envisioned mainly for crisis situations (e.g., in the battlefield or in rescue operations). In these applications, all the nodes of the

[1] Cumulative throughput is defined as the ratio of the total number of packets delivered to the total number of packets sent in the network in a given time.

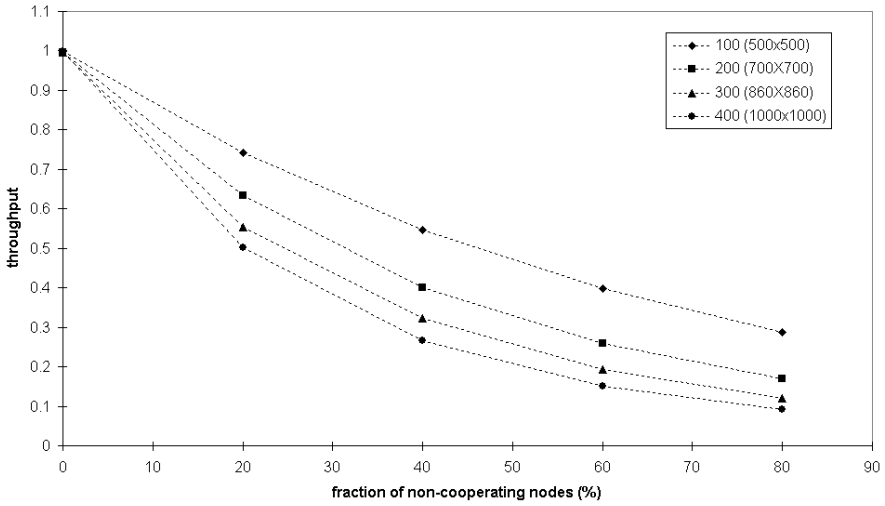

Figure 6.1: The effect of non-cooperating nodes on the throughput of the network

network belong to a single authority (e.g., a single military unit or rescue team) and have a common goal. For this reason, the nodes are naturally motivated to cooperate.

However, with the progress of technology, it will soon be possible to deploy ad hoc networks for civilian applications as well. Examples include networks of cars and provision of communication facilities in remote areas. In these networks, the nodes typically do not belong to a single authority and they do not pursue a common goal. In addition, these networks could be larger and could have a longer lifetime, and they could be completely *self-organizing*, meaning that the network would run solely by the operation of the end-users. In such networks, there is no good reason to assume that the nodes cooperate. Instead, in order to save resources (e.g., battery power, CPU, memory), the nodes tend to be "selfish".

As a motivating example, let us consider packet forwarding again: Even in a small ad hoc network, most of the energy of a given node is likely to be devoted to forwarding packets for the benefit of other nodes. For instance, if the average number of hops from source to destination is around 5, then approximately 80% of the energy devoted to sending packets will be consumed by packet forwarding. Hence, turning the forwarding function off would very noticeably extend the battery life of a node, and increase its overall availability for its user.

In the rest of this chapter, we assume that each node belongs to a different authority, its user, who may try to tamper with the software and the hardware of the node, and modify its behavior in order to better adapt it to her own goals (e.g., to save battery power). We understand that most of the users do not have the required level of knowledge and skills to modify their nodes. Nevertheless, our assumption is still reasonable, because criminal organizations can have enough interest and resources to reverse engineer a node and sell tampered nodes with modified behavior on a large scale. The experience of cellular networks shows that as soon as the nodes are under the control of the end-users, there is a strong temptation to alter their behavior in one way or another.

One approach to solve this problem would be to make the nodes tamper resistant, so that

their behavior cannot be modified. However, this approach does not seem to be very realistic, since ensuring that the whole node is tamper resistant may be very difficult, if not impossible. Therefore, we propose another approach which requires only a tamper resistant hardware module, called *security module*, in each node, and cryptographic protection of packets. One can think of the security module as a smart card (similar to the SIM card in GSM phones) or as a tamper resistant security co-processor [IBM97]. Under the assumption that the user can possibly modify the behavior of the node, but never that of the security module, our design ensures that tampering with the node is not advantageous for the user, and therefore, it should happen only rarely.

We focus on the stimulation of packet forwarding, which is a fundamental networking function that the nodes should perform in ad hoc networks. In a nutshell, we propose a protocol that requires the node to pass each packet (generated as well as received for forwarding) to its security module. The security module maintains a counter, which is decreased when the node wants to send a packet as originator, and increased when the node forwards a packet. The value of the counter must remain positive, which means that if the node wants to send its own packets, then it must forward packets for the benefit of other nodes. The counter is protected from illegitimate manipulation by the tamper resistance of the security module.

Besides stimulating packet forwarding, our mechanism encourages the users to keep their nodes turned on and to refrain from sending a large amount of packets to distant destinations. The latter property is particularly desirable, because, as mentioned in [GK00], the available bandwidth per node declines as the number of nodes increases (assuming that the traffic does not exhibit locality properties).

The present proposal was developed in the framework of the Terminodes Project [BBC+01, HGLV01]. However, it is generic; in particular, it could work in conjunction with any routing algorithm. We should also note that our scheme works only for unicast packets; the multicast case is left for future study.

The outline of this chapter is the following: In Section 6.2, we describe the proposed mechanism to stimulate packet forwarding, and study its behavior through the analysis of a simple model. In Section 6.3, we detail the ways in which the proposed mechanism could be protected against misuse. In Section 6.4, we describe our simulation settings, and the results that we obtained. Finally, in Section 6.5, we report on some related work and in Section 6.6, we sketch some possible directions for further research on this topic.

6.2 Stimulation mechanism

We assume that each node has a counter, called *credit counter*, and the following rules are enforced:

1. When the node wants to send one of its own packets, the number n of intermediate nodes that are needed to reach the destination is estimated. If the credit counter of the node is greater than or equal to n, then the node can send its packet, and the credit counter is decreased by n. Otherwise, the node cannot send its packet, and the credit counter is not modified.

2. When the node forwards a packet for the benefit of other nodes, the credit counter is increased by one.

Let us consider now the following model, the analysis of which will give an insight into the operation of the above mechanism. A node has two incoming and two outgoing flows of packets (Figure 6.2). The incoming flow IN_o represents the packets that are generated by the node itself. We call these packets *own packets*. The other incoming flow IN_f represents the packets that are received for forwarding. We call these packets *forwarding packets*. The packets that the node receives as destination are not represented in the model. Each incoming packet (own as well as forwarding) is either sent or dropped. The outgoing flow OUT represents the packets that are sent by the node. This flow consists of two components OUT_o and OUT_f, where OUT_o represents the own packets that are sent and OUT_f stands for the forwarded packets. The other outgoing flow DRP represents the packets that are dropped. Similarly to OUT, this flow consists of two components too: DRP_o and DRP_f, representing dropped own and forwarding packets, respectively.

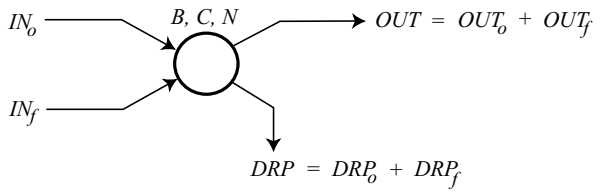

Figure 6.2: Model of a single node

The current state of the node is described by two variables b and c, where b is the remaining battery of the node and c stands for the value of its credit counter. More precisely, we interpret b as the number of packets that the node can send using its remaining energy. The initial values of b and c are denoted by B and C, respectively. To keep the model simple, we assume that when the node sends an own packet, c is decreased by an integer constant $N > 1$, which represents the estimated number of intermediate nodes that are needed to reach the destination. Since c must remain positive, the node can send its own packet only if $c \geq N$ holds. When the node sends a packet that was received for forwarding, c is increased by one. In addition, each time the node sends a packet (own as well as forwarding), b is decreased by one. When b reaches 0 (i.e., when the battery is drained out), the node stops its operation. We assume that the initial number C of credits is not enough to drain the battery out by sending only own packets (i.e., $C/N < B$).

Let us denote the number of own and forwarding packets sent during the whole lifetime of the node by out_o and out_f, respectively. Selfishness of the node could be represented by the goal of maximizing out_o subject to the following conditions:

$$out_o,\ out_f \geq 0 \tag{6.1}$$

$$N\,out_o - out_f \leq C \tag{6.2}$$

$$out_o + out_f = B \tag{6.3}$$

Condition (6.1) is trivial. Condition (6.2) describes the requirement that the number $N\,out_o$ of credits spent by the node cannot exceed the number out_f of credits earned plus the initial value C of the credit counter. Finally, Condition (6.3) represents the fact that the initial energy of the node must be shared between sending own packets and sending forwarding packets.

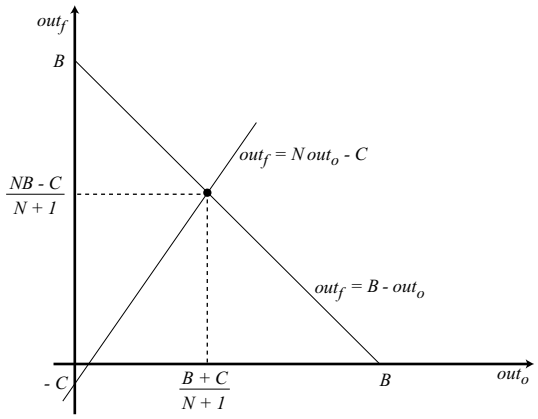

Figure 6.3: Maximizing out_o

Figure 6.3 illustrates the conditions graphically. It is easy to see that the maximum of out_o is $\frac{B+C}{N+1}$. It can also be seen that in order to reach this maximum out_f must be $\frac{NB-C}{N+1}$. Thus, the node must forward this number of packets for the benefit of other nodes if it wants to maximize its own benefit. If there was no credit counter and an enforcing mechanism that does not allow the node to send an own packet when it does not have enough credits, then Condition (6.2) would be missing, and the maximum of out_o would be B. This means that the node would maximize its own benefit by dropping all packets received for forwarding.

In principle, the node can always reach $out_o = \frac{B+C}{N+1}$: When it runs out of credits, it can simply buffer its own packets until it forwards enough packets and earns enough credits to send them. However, this works only if the buffer is large enough and no delay constraint is imposed on the packets. In real-time applications, sending a packet that has spent too much time in the buffer may be useless, which means that the node must drop some of its own packets. It can still reach $out_o = \frac{B+C}{N+1}$, but it is now important how many own packets it must drop meanwhile.

In order to study this situation, we extend our model in the following way: We assume that the node generates own packets with a constant average rate r_o, and receives packets for forwarding with a constant average rate r_f. We denote the time when the battery is drained out by t_{end}. Note that t_{end} is not a constant, since the time when the battery is drained out depends on the behavior of the node. Furthermore, we assume that there is no buffering of own packets, which means that an own packet that cannot be sent immediately (due to the low value of the credit counter) must be dropped.

Selfishness of the node could now be represented by the goal of maximizing out_o and, at the same time, maximizing $z_o = \frac{out_o}{r_o t_{end}}$ (which is equivalent to minimizing the number of own packets dropped) subject to the following conditions:

$$out_o,\ out_f \geq 0 \tag{6.4}$$

$$out_o \leq r_o t_{end} \tag{6.5}$$

$$out_f \leq r_f t_{end} \tag{6.6}$$

$$N\,out_o - out_f \leq C \tag{6.7}$$

$$out_o + out_f = B \tag{6.8}$$

Using $out_f = B - out_o$ from Condition (6.8), we can reduce the number of unknowns and obtain the following set of conditions:

$$out_o \geq 0 \tag{6.9}$$
$$out_o \leq B \tag{6.10}$$
$$t_{end} \geq \frac{out_o}{r_o} \tag{6.11}$$
$$t_{end} \geq -\frac{out_o}{r_f} + \frac{B}{r_f} \tag{6.12}$$
$$out_o \leq \frac{B+C}{N+1} \tag{6.13}$$

Conditions (6.9-6.13) determine the feasible region, on which we have to maximize out_o and z_o. This is illustrated in Figure 6.4. As we have already seen, the maximum of out_o is $\frac{B+C}{N+1}$. Note that $\frac{B+C}{N+1}$ is always less than B, because we assumed that $C/N < B$. Different values of z_o are represented by lines with different slopes all going through the (0,0) point. In order to find the maximum of z_o, we have to find the line with the smallest slope that still intersects the feasible region.

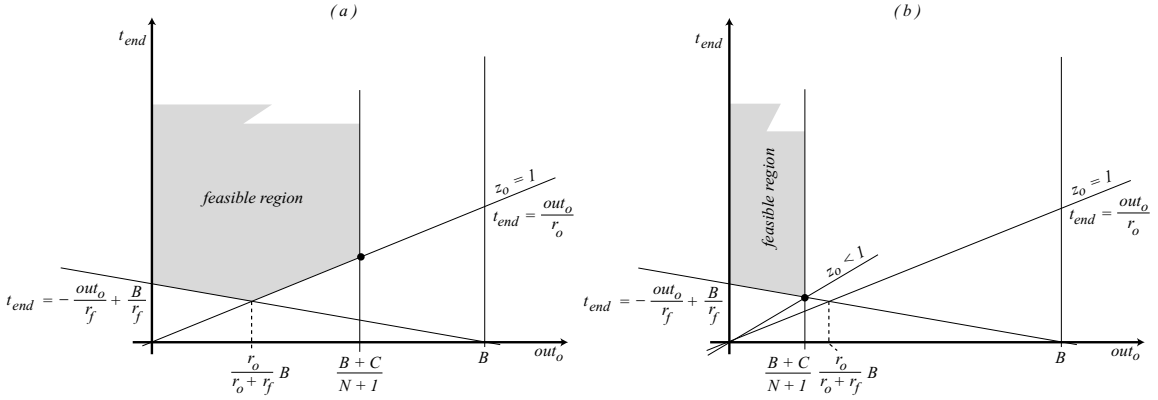

Figure 6.4: Maximizing out_o and $z_o = \frac{out_o}{r_o t_{end}}$

Depending on the ratio r_f/r_o of the rates, we can distinguish the following two cases (Figure 6.4 parts (a) and (b)):

1. If $\frac{r_f}{r_o} \geq \frac{NB-C}{B+C}$ (i.e., $\frac{B+C}{N+1} \geq \frac{r_o}{r_o+r_f}B$) then the maximum of z_o is 1. Because of Condition (6.11), this is the best that can be achieved. This means that in this case, the node does not have to drop any of its own packets.

2. If $\frac{r_f}{r_o} < \frac{NB-C}{B+C}$ (i.e., $\frac{B+C}{N+1} < \frac{r_o}{r_o+r_f}B$), then the maximum of z_o is $\frac{r_f}{r_o}\frac{B+C}{NB-C} < 1$. This means that in this case, the node must drop some of its own packets.

Intuitively, the difference between the two cases above can be explained as follows: In case 1, packets for forwarding arrive with high enough a rate to cover the expenses of sending own packets. On the other hand, in case 2, the arrival rate of forwarding packets is too low,

and the node cannot earn enough credits to send all of its own packets even if it forwards all packets received for forwarding.

The above analysis shows *what* the node can achieve in terms of maximizing its own benefit. However, it does not shed light on *how* the node should actually behave in order to reach this theoretical optimum. It seems reasonable that the node should always send its own packets whenever this is possible (i.e., whenever it has enough credits to do so). But how should the node decide whether to forward or to drop a packet received for forwarding?

In order to get an insight into this question, let us consider the following four forwarding rules, where f denotes the number of forwarding packets sent so far:

Rule 1: **if** $f < \frac{NB-C}{N+1}$ **then** forward
else drop

Rule 2: **if** $f < \frac{NB-C}{N+1}$ **then**
if $c \leq C$ **then** forward
else forward with probability C/c or drop with probability $1 - C/c$
else drop

Rule 3: **if** $f < \frac{NB-C}{N+1}$ **then**
if $c \leq C$ **then** forward
else drop
else drop

Rule 4: **if** $f < \frac{NB-C}{N+1}$ **then**
if $c \leq C$ **then** forward with probability $1 - c/C$ or drop with probability c/C
else drop
else drop

In all four rules, packets are dropped after the threshold $f = \frac{NB-C}{N+1}$ has been reached. The reason is that in this case, it is not necessary to forward more packets, because the node has enough credits to drain its battery out by sending only its own packets. The four rules differ in what happens before this threshold is reached. In Rule 1, packets are always forwarded. In the other rules, the forwarding decision depends on the current value c of the credit counter. In Rule 2, packets are forwarded for sure if $c \leq C$, and with decreasing probability as c increases if $c > C$. In Rule 3, packets are forwarded for sure if $c \leq C$, and they are always dropped if $c > C$. In Rule 4, packets are forwarded with decreasing probability as c increases if $c \leq C$, and they are always dropped if $c > C$. Clearly, the most cooperative rule is Rule 1. Rules 2, 3, and 4 are less cooperative, in this order.

We studied the performance of the rules by means of simulation. We implemented the above described model of a single node in plain C++ language. In our simulations, we set the values of the parameters as follows: $B = 100000$, $C = 100$, $N = 5$. Both the own packets and the packets for forwarding were generated according to a Poisson process. The average generation rate of the own packets were 0.2 packets per second, and we varied the average generation rate of forwarding packets between 0.6 and 1.6 packets per second with a step size of 0.2 (i.e., we varied r_f/r_o between 3 and 8 with a step size of 1, in order to obtain some results for the $\frac{r_f}{r_o} < \frac{NB-C}{B+C} \approx 5$ case as well as for the $\frac{r_f}{r_o} \geq \frac{NB-C}{B+C}$ case). The simulations lasted until the node drained its battery out (i.e., 100000 packets were sent). We run the simulation 8 times for every configuration and took the average of the results obtained. Each

rule reached $out_f = 16683 = \left\lfloor \frac{B+C}{N+1} \right\rfloor$ in every run of the simulation. The values obtained for z_o are depicted in Figure 6.5.

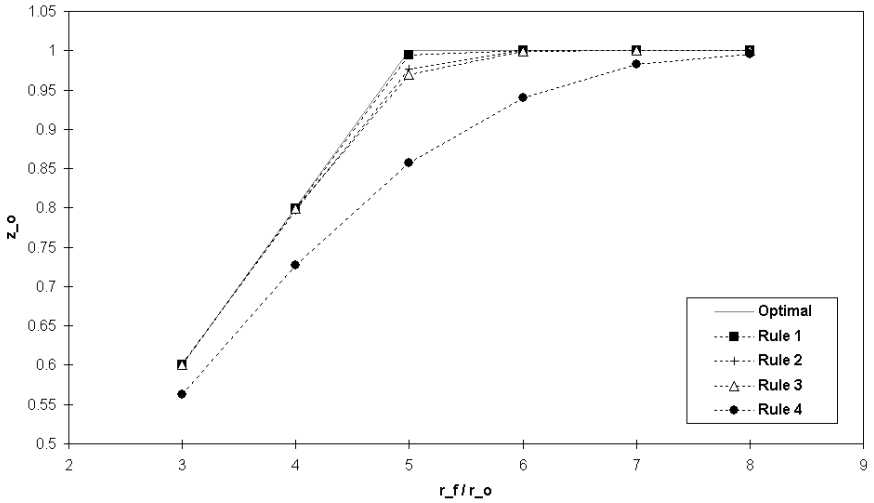

Figure 6.5: Comparison of the forwarding rules

It can be seen that Rule 4 achieves the worst performance as it is the furthest from the theoretical optimum. The first three rules perform almost equally well when $r_f/r_o < 5$ and $r_f/r_o > 5$. However, a remarkable difference appears among the rules when $r_f/r_o = 5 = N$. Interestingly enough, the results show that the most cooperative the rule is the best the performance that it achieves. This means that if the node wants to maximize out_o and z_o at the same time, then the best forwarding rule is Rule 1 (i.e., to always forward).

Figure 6.6 is meant to provide an intuitive explanation for this phenomenon. Parts (a) and (b) of the figure illustrate the operation of Rules 1 and 3, respectively, when $\frac{r_f}{r_o} \geq \frac{NB-C}{B+C}$. The figure should be interpreted in the following way: Let us assume that time is divided into small time slots. Each small grey rectangle in the figure represents the set of possible points that the node can potentially reach in a given time slot assuming that it is in the bottom-left corner of the rectangle at the beginning of that time slot. Therefore, the ratio of the sides of the rectangles is r_f/r_o. The arrows show which points are actually reached by the node when Rules 1 and 3 are used. The dark vertical bars represent the amount of dropped forwarding packets in the time slots.

It can be seen that by using Rule 1, the node tends to get further from the edge of the feasible region that is represented by the $out_f = Nout_o - C$ line. This means that the node has usually more credits in reserve when Rule 1 is used. This property turns out to be advantageous when the ratio r_f/r_o is close to N. The reason is that, due to the random manner in which the packets arrive, there is always a small fluctuation in the ratio of the number of forwarding packets to the number of own packets. On average, this ratio is equal to r_f/r_o, but sometimes it can be less. If this happens and r_f/r_o is close to N, then the node does not receive enough forwarding packets to cover the cost of sending its own packets. In this case, it must use the credits that it has in reserve. By increasing the credit reserve, Rule 1 decreases the probability of temporarily running out of credits and dropping own packets.

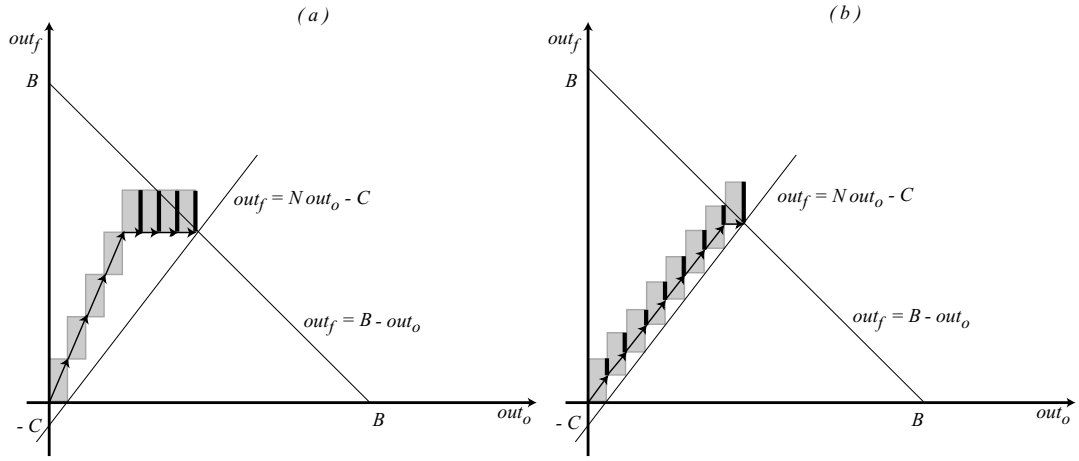

Figure 6.6: Operation of Rule 1 (a) and Rule 3 (b) when $\frac{r_f}{r_o} \geq \frac{NB-C}{B+C}$

6.3 Protection

Clearly, the stimulating mechanism described in the previous section must be secured and protected against various attacks. For instance, one has to prevent the user of the node from manipulating (typically increasing) her credit counter in an illegitimate way. In addition, one has to ensure that the credit counter is increased only if a forwarding packet has indeed been forwarded. We address these and similar issues in this section.

6.3.1 Tamper resistant security module

In order to prevent the user from illegitimately increasing its own credit counter, we require that the credit counter is maintained by a trusted and tamper resistant hardware module in each node. We call this module *security module*. One can imagine a security module as a smart card (similar to the SIM card in GSM phones) or as a tamper resistant security co-processor [IBM97]. For more information on tamper resistant modules, we refer to [PPW97, AK96].

We assume that the security modules are manufactured by a limited number of trusted manufacturers. Furthermore, since the security module is tamper resistant, its behavior cannot be modified. Therefore, security modules are trusted for always functioning correctly.

Our design approach is to put the critical functions in the security module, and the rest of the functions in the node itself. Of course, the functions that are not placed in the security module can be tampered with, and thus, the behavior of the node can still be modified. However, our design ensures that *no advantages can be gained* by tampering with the unprotected functions, and therefore, the user of the node will not be interested in this activity.

6.3.2 Public-key infrastructure

We assume that each security module has a private key and a corresponding public key. The private key is stored in the security module and kept secret. The public key is certified by the manufacturer of the security module and the certificate is stored in the security module

(Figure 6.7). In addition, we assume that the manufacturers cross-certify the public keys of each other, and each security module stores the public-key certificates of all manufacturers issued by the manufacturer of the security module. Finally, we assume that each security module stores an authentic copy of the public key of its manufacturer, which is loaded in the module securely at manufacturing time. Note that storing all these certificates is feasible, because we limit the number of manufacturers.

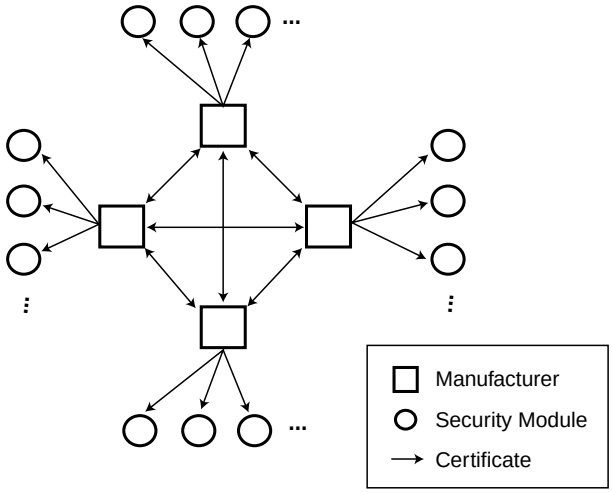

Figure 6.7: Certification structure

In this system, each security module can easily obtain the authentic public key of any other security module in the network. Let us suppose, for instance, that A wants to obtain the public key of B. B can simply send its public-key certificate to A, who can verify it with the public key of the manufacturer of B. A possesses an authentic copy of this public key, since it stores the authentic public-key certificates of all manufacturers issued by its own manufacturer.

Our system is a rather pragmatic solution for the reliable distribution of public keys, and we had to limit the number of manufacturers in order for it to work. The design of a general purpose public-key infrastructure for large, self-organizing ad hoc networks is a challenging problem that is beyond the scope of this thesis. An approach towards the solution of this problem is described in [HBC01].

6.3.3 Security associations

When two nodes become neighbors, their security modules establish a security association. If this fails, the security modules do not consider each other neighbors.

A security association between two neighboring security modules A and B is represented, at A's side, by

- the unique identifier of B;
- the unique identifier of the node that hosts B,
- a symmetric session key k_{AB};

- two sequence numbers $c_{A \to B}$ and $c_{A \leftarrow B}$, which are called *sending* and *receiving sequence numbers*, respectively; and
- a counter $pc_{B@A}$, which is called *pending credit counter*.

At B's side, the same association is represented by

- the unique identifier of A;
- the unique identifier of the node that hosts A;
- the session key k_{AB};
- a sending sequence number $c_{B \to A}$ and a receiving sequence number $c_{B \leftarrow A}$, such that initially, $c_{B \to A} > c_{A \leftarrow B}$ and $c_{B \leftarrow A} < c_{A \to B}$; and
- a pending credit counter $pc_{A@B}$.

The session key k_{AB} is used to compute a message authentication code, which protects the integrity and ensures the authenticity of the packets sent between the nodes of A and B, but k_{AB} can also be used to provide other security functions (e.g., link-by-link encryption of the content of the packets). The sequence numbers are used to detect replayed packets. The pending credit counter $pc_{B@A}$ is used to accumulate credits at A that are due to B. Similarly, $pc_{A@B}$ counts the credits at B that are due to A. The way in which the session key, the sequence numbers, and the pending credit counters are used will be explained in more detail in the next subsection, where we present the envisioned packet forwarding protocol.

The security associations between the security modules are established using some public-key cryptographic protocol, which is executed through the nodes that host the security modules. The security modules obtain each other's public key according to the model of the above described public-key infrastructure.

6.3.4 Packet forwarding protocol

The packet forwarding protocol described in this subsection assumes that the security module runs the routing algorithm used in the network.

If a node P has an own packet to send, it must first pass the packet to its security module A. A estimates the number n of intermediate nodes needed to reach the destination. Precise estimation of this number is not so critical. If the value of the credit counter maintained by A is less than n, then A rejects the packet. Otherwise the credit counter is decreased by n, and the protocol continues.

Using the routing algorithm, A determines the next intermediate node Q toward the destination, and retrieves the security association that corresponds to the security module B of node Q from its internal database. Then, it takes the session key k_{AB} and the sending sequence number $c_{A \to B}$, and generates a *security header* (see also Figure 6.8) for the packet, which contains A, B, $c_{A \to B}$, and the message authentication code $mac_{k_{AB}}(A,\ B,\ c_{A \to B},\ packet)$, where mac is a publicly known MAC function. After this computation, $c_{A \to B}$ is increased by one.

Finally, A outputs the security header and the identifier of the next intermediate node Q, and P can send the packet together with the security header to Q.

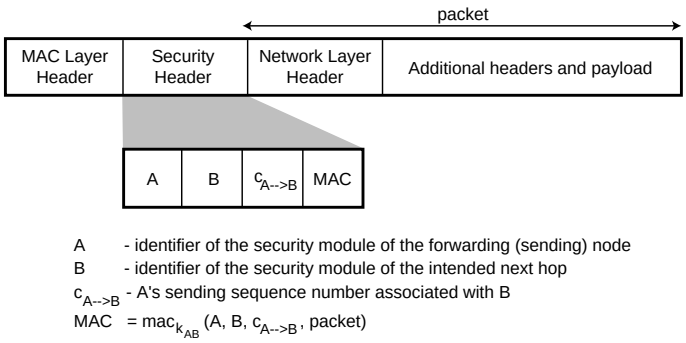

Figure 6.8: Security Header

Now, let us assume that node Q received a packet with a security header for forwarding from node P. If Q wants to forward the packet in order to earn a credit, then Q must pass the packet with the attached security header to its security module B. B takes the identifier of the security module A that generated the security header from the header itself, and retrieves the corresponding security association from its internal database. Then, it verifies if the sending sequence number in the security header is greater than its receiving sequence number $c_{B \leftarrow A}$. If this is the case, then the packet is not a replay. Then, it verifies the received message authentication code. If it is correct, then it accepts the packet, and updates $c_{B \leftarrow A}$ to the value of the sequence number received in the security header.

If the node that hosts A (known to B from the data that represents the security association between B and A) is not the originator of the packet (i.e., if it is an intermediate node), then B increases the pending credit counter $pc_{A@B}$ by one. Finally, B determines the next intermediate security module towards the destination, and generates a new security header for the packet, much in the same way as described earlier, using the security association that corresponds to the next intermediate security module.

6.3.5 Credit synchronization protocol

As it can be seen from the description of the packet forwarding protocol, when an intermediate node forwards a packet, its credit counter is not increased immediately. Instead, the security module of the next node increases the pending credit counter that it maintains for the first node. For clearing, the security modules regularly run a credit synchronization protocol, in which they transfer the pending credits, and reset the pending credit counters to 0. This mechanism ensures that the node is rewarded for the packet forwarding only if it really forwarded the packet.

It may happen, however, that the nodes move out of each other's power range by the next time their security modules want to run the credit synchronization protocol. If this happens, the pending credit counters are reset to 0, and the pending credits are lost. Therefore, the mechanism does not guarantee that the node receives its credit for every forwarded packet. We will study the consequences of this in the next section.

6.3.6 Robustness

The protection mechanism described above is robust and resists against various attacks. The credit counter is protected from illegitimate manipulations by the tamper resistance of the security module. A security header is attached to each packet, which contains a message authentication code that protects the integrity of the packet and the data in the security header. This is important, because the security modules manipulate the credit counters based on the data received in the security header. Replay of packets is prevented by the use of ever increasing sequence numbers. Moreover, the node is rewarded for packet forwarding only if it really forwarded a packet.

We should mention, however, that there is a subtle attack that our scheme may not always prevent in its current form. It is possible to construct a fake node that has two or more security modules. Such a node could bounce a packet back and forth between its security modules, and earn credits without actually transmitting anything. The full understanding of this attack and the design of the proper counter-measure are part of our future work. However, we can already make the following observations: First, this attack would not always work, since routing is performed by the security modules, which means that the next intermediate security module is determined by the security module and not the node. In other words, the security module may output a security header for the packet that will not be accepted by the other security module(s) of the node. To avoid this, the node may falsify routing information that is exchanged between the security modules, but this can be prevented by using appropriate cryptographic techniques. Second, such a fake node would be more expensive than a normal one, since it has two or more legitimate security modules. Whether the benefit obtained by using such a fake node is worth the increased cost is an open question.

6.3.7 Overhead

We must admit that our protection mechanism adds some computational overhead to the system, which is mainly related to the use of cryptographic operations. This issue has two aspects: cryptographic operations need energy and time to be performed. Regarding energy consumption, we note that the energy required to perform computation is negligible when compared to the energy required to perform transmission [PK00]. Therefore, we estimate that the execution of our cryptographic operations have a negligible energy cost when compared to the transmission cost.

Regarding time, we note that the only time critical operations are the generation and the verification of the security header for every packet and for every hop. However, these require only cryptographic hash function computations, which can be done very efficiently. Moreover, the security header is processed by the security module; to some extent, this can be accomplished in parallel with the processing performed by the main processor of the node.

Another issue is the communication overhead, which is due to the establishment of the security associations, the size of the security header, and the periodic execution of the credit synchronization protocol. In order to reduce this overhead, the establishment of the security associations could be integrated with the neighbor discovery protocol that the nodes usually have to run anyhow in mobile ad hoc networks, and the credit synchronization interval should be appropriately chosen. Finally, assuming that the identifiers of the security modules are 8 bytes long, the sequence numbers are 2 bytes long, and the output of the cryptographic hash function used is 16 bytes long (e.g., if MD5 [MvOV97] is used), we get that the security

header is 34 bytes long. This seems to be an acceptable overhead.

6.4 Simulations

In Section 6.2, we studied the proposed stimulation mechanism through the analysis of a simplified model, and showed convincing arguments that it indeed stimulates packet forwarding in that model. In order to study the proposed stimulation mechanism in a more general setting, which is closer to the reality of mobile ad hoc networks, we conducted simulations of a full network written in plain C++ language. In this section, we describe our simulator, and the results that we obtained.

6.4.1 Simulation description

The simulated networks are composed of 100 nodes that are placed randomly (uniformly) on a 500 m $\times$ 500 m rectangle. Each node has the same power range of 120 m. The nodes move according to the *random waypoint* mobility model [BMJ+98]. In this model, the node randomly chooses a destination point in space and moves towards this point with a randomly chosen constant speed. When it reaches the chosen destination, it stops and waits there for a randomly chosen time. Then, it chooses a new destination and speed, and starts to move again. These steps are repeated until the end of the simulation. In our simulations, the nodes choose their speed between 1 m/s and 3 m/s uniformly. The pause time is generated according to an exponential distribution. The average pause time is 60 s.

We do not use any particular MAC layer algorithm. Instead, we model the MAC layer by randomly choosing the packet transmission time between neighbors for each packet and for each hop. The average packet transmission time between neighbors is 10 ms. Packet transmission errors occur with 0.1 probability. If an error occurred, the packet is re-transmitted after a 1 s timeout. When the node is busy with packet transmission, it can still receive packets, which are placed in a buffer, and served when the previous packet transmission is finished.

For routing, we use a *geodesic packet forwarding* algorithm developed within the context of the Terminodes Project, and described in [BGL00]. However, we considerably simplified the original algorithm in order to ease the implementation of its simulator. This does not affect our results, since we are not interested in the performance of the packet forwarding algorithm itself. The simplified geodesic packet forwarding algorithm works in the following way: We assume that each node knows its own geographic position and the geographic positions of its neighbors. Furthermore, the source of a packet knows the current geographic position of the destination. The way in which this information is obtained is not simulated. Before sending the packet, the source puts the coordinates of the destination in the header of the packet. Then, it determines which of its neighbors is the closest to the destination, and sends the packet to this neighbor. When a node receives a packet for forwarding, it first verifies if the destination is its neighbor. If this is the case, then it forwards the packet to the destination. Otherwise, it determines which of its neighbors is the closest to the destination, and sends the packet to this neighbor. This is possible, because the packet header contains the believed coordinates of the destination. If the forwarding node does not have any neighbor that is closer to the destination than the node itself, then the packet is dropped[2]. In our simulations,

[2]This simplification is true only in our simulation setting. The complete geodesic packet forwarding algo-

Parameter	Value
Space	500 m × 500 m
Number of nodes	100
Power range	120 m
Mobility model	random waypoint
Speed	1 m/s – 3 m/s
Average pause time	60 s
Packet generation rate	0.2 (0.5, 0.8) pkt/s
Choice of destination	random
Routing	geodesic packet forwarding
Initial number of credits (C)	100
Credit synchronization interval	5 (10, 15, 20) s
Simulation time	7200 s

Table 6.1: Value of the main simulation parameters

because of the rather high density and the rather low speed mobility of the nodes, packet drops of this kind almost never happened.

Energy consumption of the nodes is not simulated. For this reason, the size of the packets is not important for us. Therefore, we assume that each packet has the same size, and we focus only on the number of packets that are generated, sent, forwarded, and delivered.

Each node generates packets according to a Poisson process. The destination of each packet is chosen randomly (uniformly). In our reference simulation, the average packet generation rate was 0.2 pkt/s, but we also ran simulations with average packet generation rates of 0.5 and 0.8 pkt/s.

The initial value C of the credit counter of each node is 100. When a node i sends an own packet to a node d that is not the neighbor of i, the credit counter of i is decreased by n. Unlike in the simple model of Section 6.2, n is not a constant, but computed according to the following formula:

$$n = \left\lceil \frac{distance(i,d)}{power\ \ range} \right\rceil - 1$$

This gives a lower bound on the number of intermediate nodes needed to reach the destination. When a node forwards a packet, its pending credit counter at the next node is increased by one. In our reference simulation, the credits of each node are synchronized in every 5 s, but we also ran simulations with credit synchronization intervals of 10, 15, and 20 s.

We always ran 8 simulations for a given simulation setting, and considered the average of the obtained values for each observed variable. In each run, 2 hours of network operation were simulated.

We listed the values of the main simulation parameters for an overview in Table 6.1.

rithm described in [BGL00] can cope with such a situation.

6.4.2 Simulation results

Comparison of forwarding rules

In the first set of simulations, our goal was to determine which of Rule 1, Rule 2, or Rule 3 is the most beneficial for the nodes in terms of maximizing z_o. We did not use Rule 4, because it performed much worse than the other three rules in the single node model of Section 6.2. Since battery usage is not taken into consideration in our simulations, we had to modify the rules as follows:

Rule 1': always forward

Rule 2': **if** $c \leq C$ **then** forward
else forward with probability C/c or drop with probability $1 - C/c$

Rule 3': **if** $c \leq C$ **then** forward
else drop

Our approach to determine which of these rules is the best was the following: We set 90% of the nodes to use a given rule (we call this the *majority rule*), and the remaining 10% of the nodes to use first Rule 1', then Rule 2', and finally Rule 3'. We observed the average value of z_o that the 10% of the nodes could achieve in each cases. We repeated the above experiment for packet generation rates of 0.2, 0.5, and 0.8 pkt/s. The results are depicted in Figures 6.9, 6.10, and 6.11.

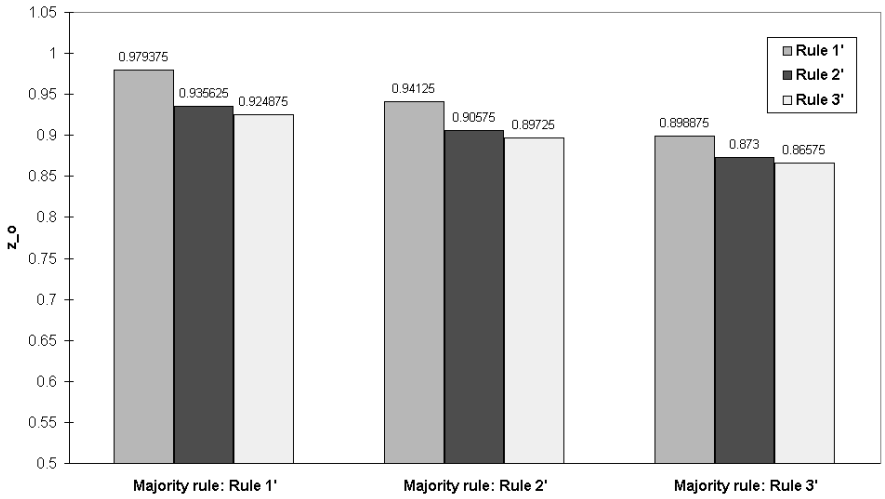

Figure 6.9: Comparison of the forwarding rules when the packet generation rate is 0.2 pkt/s

Remarkably, Rule 1' performed the best in every case. This means that the 10% deviating nodes achieve the highest z_o (i.e., drop the smallest portion of their own packets) when they use Rule 1', no matter whether the 90% of the nodes use Rule 1', Rule 2', or Rule 3'. Furthermore, this is true for every packet generation rate that we have simulated. Therefore, our conclusion is that the proposed stimulation mechanism indeed stimulates packet forwarding, and not only in the simple model of Section 6.2, but in a much more general setting too.

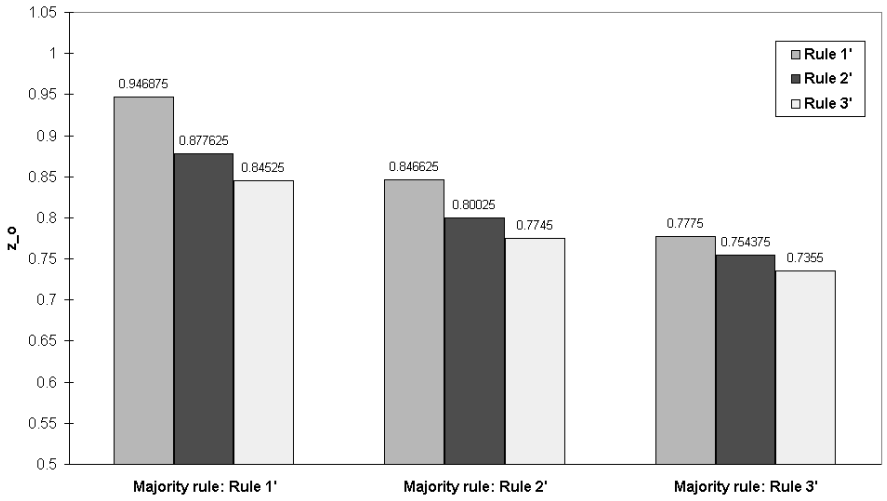

Figure 6.10: Comparison of the forwarding rules when the packet generation rate is 0.5 pkt/s

The effect of less cooperative nodes on the throughput of the network

In the second set of simulations, our goal was to study the effect of less cooperative nodes on the throughput of the network when the proposed stimulation mechanism is used. As opposed to the simulation that yielded Figure 6.1, in which we assumed that some nodes fully deny packet forwarding, here we assumed that some nodes use the least cooperative forwarding rule (i.e., Rule 3'). The rational is that full denial of packet forwarding quickly results in running out of credits, and thus, dropping own packets, and therefore, it is not beneficial at all for the nodes. On the other hand, Rule 3' can be viewed as a trade-off, where the node can send a large portion of its own packets, and it forwards only a small number of packets that is necessary to cover its expenses.

In this experiment, our approach was the following: We first set all the nodes to cooperate (i.e., to use Rule 1'), and then progressively increased the fraction of less cooperative nodes (i.e., the fraction of nodes that use Rule 3'). In order for the results to be comparable with the results shown in Figure 6.1, we ran simulations with networks of 100, 200, 300, and 400 nodes but with the same node density. We observed the cumulative throughput of the network, which is defined as the ratio of the total number of packets delivered to the total number of packets sent. The results are shown in Figure 6.12.

It can be seen that the throughput of the network decreases as the fraction of less cooperative nodes increases, but far less dramatically than in Figure 6.1. Even if all the nodes use Rule 3', the throughput is around 0.9.

The value of this experiment is that it shows that the network can tolerate less cooperative nodes quite well. A node may tend to be less cooperative, when it is about to run out of battery. In this case, it may not be beneficial to use Rule 1', and in this way, increase the credit reserve, because those credits cannot be used if the battery becomes empty. Therefore, the node may decide to use a less cooperative forwarding rule, or even to drop all forwarding packets. However, we note that the battery can usually be reloaded, and the accumulated

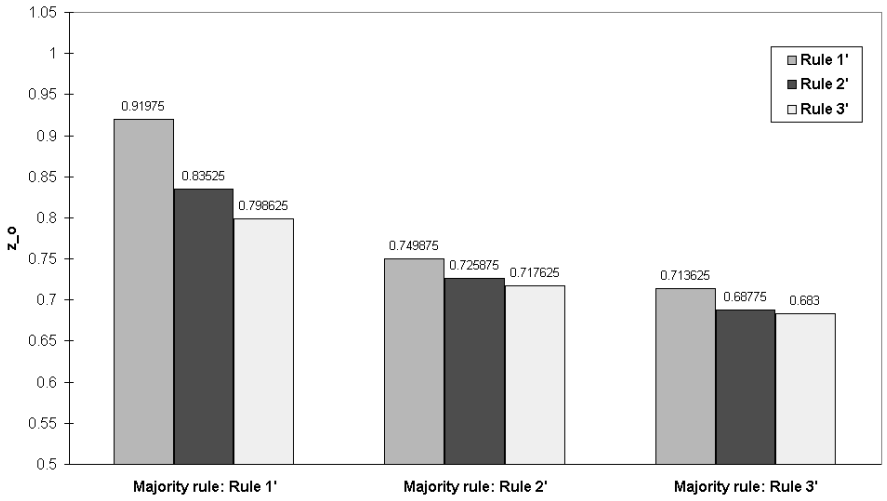

Figure 6.11: Comparison of the forwarding rules when the packet generation rate is 0.8 pkt/s

credits can be used again. For this reason, it is not clear at all whether using a less cooperative rule when running out of battery is a good strategy or not. Nevertheless, the results of the above experiment show that the network would be able to cope with this situation.

Variation of the average credit level in the network

In a third set of simulations, our goal was to study how the average credit level in the network is effected by the number of less cooperative nodes and by the size of the credit synchronization interval. To this end, we observed how the average credit level in the network varies in time as we increase the fraction of less cooperative nodes and as we increase the size of the credit synchronization interval. The results are shown in Figures 6.13 and 6.14.

When most of the nodes are cooperative, the average credit level in the network shows an increasing tendency. This is because the formula that we use to determine the number of intermediate nodes needed to reach a given destination under-estimates the actual number. This means that if a packet is delivered, then the joint credit income of the intermediate nodes is usually higher than the expenses of the source of the packet. Furthermore, when more nodes use Rule 1', packets are delivered with a higher probability, and thus, the average credit level increases more rapidly.

When less cooperative nodes are in majority, the average credit level in the network decreases. However, this decrease slows down, and after some time, it stops, and the average credit level becomes constant. The intuitive explanation is the following: When the nodes use Rule 3', their forwarding decisions depend on the current value of their credit counters. At the beginning, the average credit level is high, and packets are often dropped before they reach their destinations. This results in a decrease of the average credit level in the network. At the same time, the probability of dropping a packet due to the usage of Rule 3' also decreases, since the nodes have less credits in general, and they are more willing to forward. Therefore, more and more packets are delivered, and the decrease of the average credit level

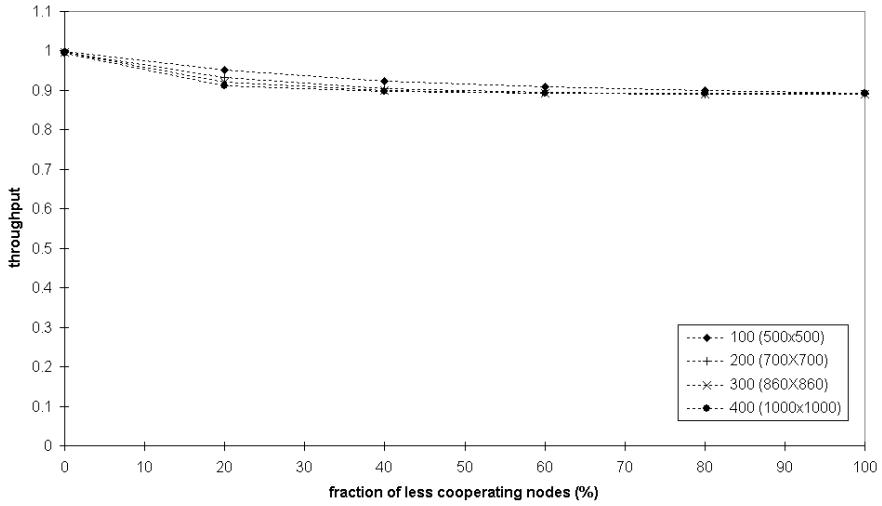

Figure 6.12: Effect of less cooperative nodes on the throughput of the network

slows down. After some time, the decreasing effect of using Rule 3' (i.e., dropping packets) and the increasing effect of under-estimating the actual number of intermediate nodes needed to reach a given destination equalize each other, and the system attains an equilibrium. The fact that this equilibrium is below the initial value $C = 100$ of the credit counters explains why the throughput of the network is around 0.9 even if all the nodes use Rule 3' (see Figure 6.12). The reason is that in the equilibrium, most of the nodes have less than C credits (note that none of the nodes has more than C credits because of Rule 3'), and therefore, most of the nodes are willing to forward.

The effect of the credit synchronization interval on the average credit level in the network is not surprising: The larger the credit synchronization interval is, the slower the increase of the average credit level in the network is. Moreover, when the credit synchronization interval is 20 s, the average credit level continuously decreases in time. The reason is that when the credit synchronization interval is large, the probability that the neighbors of a node move away by the time of the next run of the credit synchronization protocol is high, and thus, the number of credits lost in the system is also high.

If mobility exhibits some locality properties, then this problem can be alleviated by slightly modifying the credit synchronization protocol, and letting the security module keep the accumulated pending credits for a given neighboring node in memory (until this memory is not needed for other purposes) even if that node has moved away and is not a neighbor anymore. In this case, because of the locality of mobility, nodes that were neighbors in the past may become neighbors again with a higher probability, which means that there are good chances that the pending credits can be cleared some time in the near future.

In any case, the size of the credit synchronization interval must be carefully chosen. If it is too small, then the credit synchronization protocol is run too often, which leads to a considerable overhead. However, if it is too large, then the average credit level in the network may become too low. Therefore, one has to find an appropriate trade-off.

An approach to limit the variation of the average credit level in the network would be

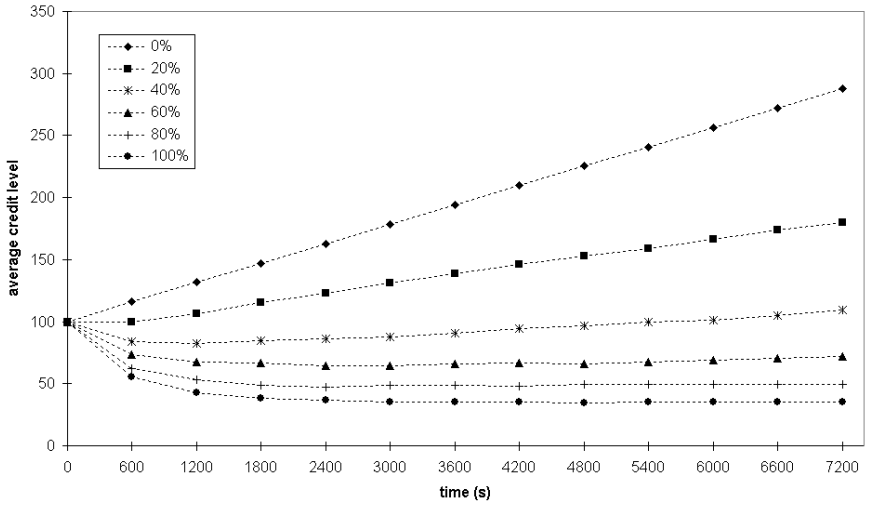

Figure 6.13: The effect of less cooperative nodes on the average credit level in the network

to reset the credit counter to a reference value regularly. For instance, it could be reset each time the battery is reloaded. However, the security module, which maintains the credit counter, may not have reliable information about the battery reload events. On the other hand, since it maintains the credit counter, it can pretty well estimate the number of packets sent by the node by observing the credit incomes and expenses. Thus, it can reset the credit counter after a given number of packets has been sent. This would eliminate the problem of ever increasing or ever decreasing average credit level in the network. However, it is not yet clear to us, what the consequences of this resetting mechanism are on the performance of the different forwarding rules. In particular, it seems, that in this case, the node's goal is not only maximizing z_o, but at the same time, it may want to minimize its credit loss due to the resetting mechanism. It is an open question which forwarding rule would be the best with respect to this new goal.

6.5 Related work

To the best of our knowledge, there are only two papers addressing the problem of non-cooperating nodes in mobile ad hoc networks: [MGLB00] and our previous paper [BH00]. The authors of [MGLB00] consider the case in which some malicious nodes agree to forward packets but fail to do so. In order to cope with this problem, they propose two mechanisms: a *watchdog*, in charge of identifying the misbehaving nodes, and a *pathrater*, in charge of defining the best route avoiding these nodes.

The paper shows that these two mechanisms make it possible to maintain the total throughput of the network at an acceptable level, even in the presence of a high amount of misbehaving nodes. However, the problem is that the selfishness of the nodes does not seem to be castigated; on the contrary, by the combination of the watchdog and the pathrater, the misbehaving nodes will not be bothered by the transit traffic while still enjoying the possibility to send and to receive packets. The proposed mechanisms could be enriched in such a

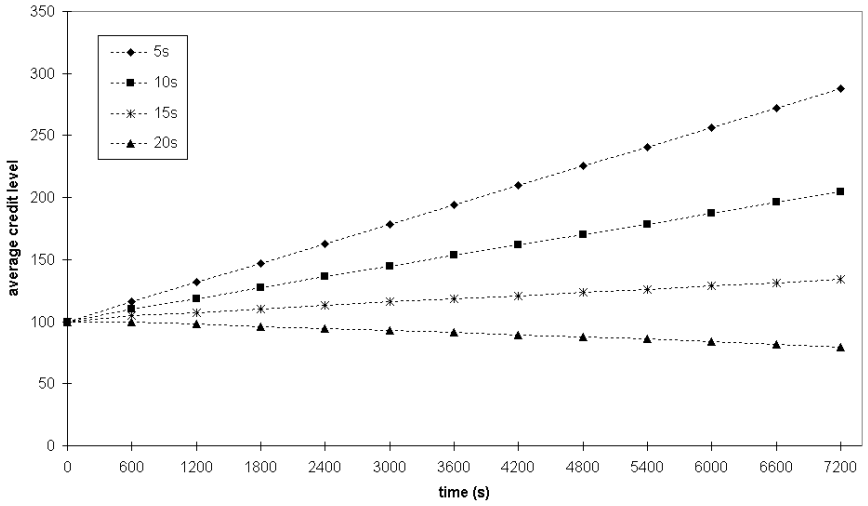

Figure 6.14: The effect of the credit synchronization interval on the average credit level in the network

way that a misbehaving node would be locked out by its neighbors. However, this possibility could be exploited to mount denial of service attacks.

In [BH00], we addressed the same problem as in this chapter, and proposed a stimulation mechanism that is based on a virtual currency, called *nuglets*. Nuglets are used to pay for packet forwarding. We proposed two payment models for this purpose. In the Packet Purse Model, the source of the packet pays by loading some nuglets in the packet before sending it. Intermediate nodes acquire some nuglets from the packet when they forward it. If the packet runs out of nuglets, then it is dropped. In the Packet Trade Model, the packet does not carry nuglets, but it is traded for nuglets by intermediate nodes: Each intermediate node "buys" it from the previous one for some nuglets, and "sells" it to the next one (or to the destination) for more nuglets. In this way, each forwarding node earns some nuglets, and the total cost of forwarding the packet is covered by the destination.

A serious disadvantage of the Packet Trade Model is that it allows overloading of the network, since the sources do not have to pay. For this reason, mainly the Packet Purse Model has been studied. However, the Packet Purse Model has a problem too: it seems to be difficult to estimate the number of nuglets that the source should put in the packet initially. If the source under-estimates this number, then the packet will be discarded with a high probability, and the source loses its investment. The source may over-estimate the number, but this leads to a rapid decrease of the total number of nuglets in the system due to the dropping of packets (for networking reasons) with many nuglets inside.

The mechanism proposed in this chapter overcomes this estimation problem, because the packets do not need to carry credits. At the same time, the property of refraining users from overloading the network is retained. Otherwise, the two mechanisms have a very similar flavor, just like their protection schemes.

6.6 Summary

In this chapter, we addressed the problem of stimulating cooperation in self-organizing ad hoc networks for civilian applications, where the nodes are assumed to be "selfish", meaning that they try to maximize the benefits that they get from the network, while minimizing their contribution to it. We focused on a particular instance of this problem, namely, stimulating packet forwarding. Our approach is based on a counter, called credit counter, in each node. Besides stimulating packet forwarding, the proposed mechanism encourages the users to keep their nodes turned on and to refrain from sending a large amount of packets to distant destinations.

In order to protect the proposed mechanism against misuse, we presented a scheme based on a trusted and tamper resistant hardware module, called security module, in each node, which generates cryptographically protected security headers for packets and maintains the credit counters of the nodes.

The philosophy of our design is similar to that of rational exchange protocols. In particular, the proposed stimulation mechanism and the proposed protection scheme are not intended to make misbehavior of the nodes impossible. What we tried to ensure is that *misbehavior is not beneficial* for the nodes, and therefore, it should happen only rarely. For instance, nodes can still deny packet forwarding, or they may bypass the security module, and send a packet without a valid security header. However, if the node denies packet forwarding, then it runs out of credits, and it cannot send its own packets. Furthermore, if the node sends a packet without a valid security header, then intermediate nodes will be reluctant to forward it. This is because an intermediate node can earn credits with packet forwarding only if it passes the forwarding packet to its security module, but in the absence of a valid security header, the security module will reject the packet.

We studied the behavior of the proposed mechanism analytically and by means of simulations. We showed convincing arguments that it indeed stimulates the nodes for packet forwarding assuming that

- each node of the network generates packets continuously;
- generated packets cannot be buffered, which means that if they cannot be sent, then they must be dropped; and
- selfishness of the nodes is represented by the goal of dropping as few own packets as possible.

A possible extension of this work would be to study the behavior of the proposed mechanism, when these assumptions are weakened. For instance, one could allow buffering of packets, but limit the size of the buffer and the time that the packets can spend in it. Or, one could allow the nodes to generate packets in bursts instead of continuously. Finally, one could study the effect of regularly resetting the credit counters to a reference value (see discussion in Subsection 6.4.2), in which case the assumed goal of the nodes needs to be extended.

Publications: [BH00, BH01b]

Conclusion

Communication systems, such as the Internet and cellular networks, do and will play an important role in society and economy by providing opportunities for new, sophisticated services. However, in order for these services to be successful, they must be secure. In this thesis, we studied two security mechanism, authenticated key transport and rational exchange protocols, which are potential building blocks in the security architecture of a range of different services. We focused on the construction of formal models in which these mechanisms can be represented and their properties can rigorously be studied. We believe that formal models are useful in designing robust security mechanisms, because they help us to better understand the mechanisms and the usual pitfalls of their design, and they can serve as the basis of systematic verification and design tools.

In the first part of the thesis, we proposed a formal model for authenticated key transport protocols, in which protocols are represented as a set of formulae of a simple logic of belief. Contrary to other formal models in the literature, our model is not aimed at formal protocol verification, but direct support of protocol construction. More precisely, based on the model, we proposed a set of synthesis rules that can be used to construct key transport protocols in a systematic way. We illustrated the use of the proposed logic and the synthesis rules through a detailed example.

In our model, the messages of a protocol are idealized, which means that they contain logical formulae that represent their intended meaning. In addition, messages are sent via channels with various access properties, instead of being protected by cryptographic primitives. These features allow us to design abstract protocols, which can be implemented in several different ways by replacing the logical formulae in the messages with bit strings, and the channels with cryptographic primitives. While using idealized messages and channels may be a drawback in protocol verification (see e.g., [BM93]), they proved to be useful in protocol construction, because they provide intuitive abstractions that help the designer to bridge the gap between the (informal) specification and the implementation of the protocol.

In the second part of the thesis, we proposed a formal model for exchange protocols that is based on game theory. In this model, a protocol is represented as a set of strategies in a game that is played by the protocol parties and the network that they use to communicate with each other. We used this model to formally define various properties of exchange protocols, including rationality and fairness. This allowed us to study the relationship between rational exchange and fair exchange within a single model, and to prove that fairness implies rationality in this model, but the reverse is not true in general. In addition, we used the protocol game model to formally verify two rational exchange protocols. We did not use computerized tools in the verification, but our model allows the construction and the usage of such tools.

The formal definitions and the comparison of rational exchange and fair exchange helped us to better understand what rational exchange really is. First, we identified that its essence

can be captured by the well-known notion of Nash equilibrium in game theory. Second, by proving that fairness implies rationality but not vice versa, we suggested that rational exchange protocols provide weaker guarantees, and therefore, one can expect that they are less complex than fair exchange protocols. This means that rational exchange can be viewed as a trade-off between complexity and true fairness. We showed how such a trade-off can provide an elegant solution to the exchange problem in the context of micropayment schemes. More precisely, we applied the concept of rational exchange to improve a family of micropayment protocols with respect to fairness without substantial loss in efficiency in most practical cases. Other possible applications of rational exchange protocols include mobile commerce and pure peer-to-peer systems.

It is, of course, possible to construct formal models for exchange protocols based on principles that are different from those of game theory. While some properties of exchange protocols might be defined more conveniently in a different model, we strongly believe that game theory is the most elegant yet precise way to define rationality.

Finally, in the third part of the thesis, we extended the concept of rational exchange, and we proposed a mechanism based on a similar philosophy to stimulate packet forwarding in ad hoc networks. Our mechanism does not try to guarantee that the nodes of the network will always behave correctly and forward packets. Instead, the mechanism renders misbehavior uninteresting by making packet forwarding the most advantageous strategy. We analyzed the proposed mechanism through a simplified analytical model and by means of simulation. Both analyses indicated that the mechanism indeed stimulates packet forwarding.

Summary of contributions

The following is a list of the original contributions of the thesis:

- a simple logic of channels that provides an abstract model in which one can reason about authenticated key transport protocols;
- an approach to construct authenticated key transport protocols in a systematic way based on a set of synthesis rules derived from the logic of channels;
- analysis of an authenticated key transport protocol proposed in the literature, identification of weaknesses, and discovery of three, previously unknown attacks;
- correction of the flawed key transport protocol mentioned in the previous point by redesigning it using the aforementioned systematic protocol construction approach;
- a formal model of exchange protocols based on game theory;
- a formal definition of rational exchange based on the concept of Nash equilibrium;
- formal definitions of various other properties of exchange protocols, including fairness;
- a proof that fairness implies rationality within the aforementioned game theoretic model, but the reverse is not true in general;
- formal proofs of two exchange protocols to be rational;

- an application of the concept of rational exchange to make misbehavior uninteresting in a family of micropayment schemes without substantial loss in efficiency;
- a mechanism to stimulate the nodes of a mobile ad hoc network for packet forwarding based on an idea similar to rational exchange.

Directions for future research

As we mentioned earlier, there exist only a few rational exchange protocols proposed in the literature. Now that we have obtained more understanding of the concept of rational exchange, it would be interesting to construct new rational exchange protocols. It would be particularly interesting to design an exchange protocol that satisfies the definition of rationality in a meaningful asynchronous model too. This, of course, requires the full development of an asynchronous model, which could be done along the lines that we sketched in Section 4.6. The ultimate goal would be to identify some general design principles for rational exchange protocols, or eventually, to come up with a protocol construction tool similar to the one described in the first part of this thesis.

Regarding the work presented in Chapter 6, a possible future work would be to study the behavior of the proposed mechanism when the main assumptions listed in Section 6.6 are weakened. It would also be interesting to study how the proposed mechanism can be applied in hybrid networks (e.g., in cellular networks in which the nodes have ad hoc networking capabilities), and what consequences of the presence of cellular operators have on the design. Another interesting direction would be to generalize the proposed mechanism so that it can be applied not only to packet forwarding, but to other functions as well (e.g., route discovery and route repair in on-demand protocols). We could even stretch this up to the application layer: In peer-to-peer computing [Ora01], there is a growing concern that some users might parasitically take advantage of resources provided by others (see e.g., [AH00]). Some researchers have made early attempts to introduce a virtual currency to encourage "good citizenship" (e.g., www.mojonation.net). It would be interesting to explore how mechanisms like the one proposed in the last chapter could be used in that context.

Bibliography

[Aba97] M. Abadi. Explicit communication revisited: Two new attacks on authentication protocols. *IEEE Transactions on Software Engineering*, 23(3), March 1997.

[ABKL92] M. Abadi, M. Burrows, C. Kaufman, and B. Lampson. Authentication and delegation with smart-cards. Technical Report SRC RR 67, Digital Equipment Corporation, Systems Research Center, July 1992.

[AFG98] M. Abadi, C. Fournet, and G. Gonthier. Secure implementation of channel abstractions. In *Proceedings of the Thirteenth Annual IEEE Symposium on Logic in Computer Science*, pages 105–116, June 1998.

[AG98] M. Abadi and A. Gordon. A calculus for cryptographic protocols: The Spi calculus. Technical Report SRC RR 149, Digital Equipment Corporation, Systems Research Center, January 1998.

[AH00] E. Adar and B. Huberman. Free riding on Gnutella. *First Monday*, 5(10), October 2000.

[AK96] R. Anderson and M. Kuhn. Tamper resistance – a cautionary note. In *Proceedings of the Second Usenix Workshop on Electronic Commerce*, November 1996.

[AMS95] R. Anderson, C. Manifavas, and C. Sutherland. NetCard – a practical electronic cash system. http://www.cl.cam.ac.uk/users/rja14/, 1995.

[AN95] R. Anderson and R. Needham. Robustness principles for public key protocols. In *Advances in Cryptology – CRYPTO'95*, pages 236–247. Springer-Verlag, 1995.

[AN96] M. Abadi and R. Needham. Prudent engineering practice for cryptographic protocols. *IEEE Transactions on Software Engineering*, 22(1):6–15, January 1996.

[AS97] J. Alves-Foss and T. Soule. A weakest precondition calculus for analysis of cryptographic protocols. In *Proceedings of the DIMACS Workshop on Design and Formal Verification of Security Protocols*, 1997.

[Aso98] N. Asokan. *Fairness in Electronic Commerce*. PhD thesis, University of Waterloo, Ontario, Canada, May 1998.

[ASW97] N. Asokan, M. Schunter, and M. Waidner. Optimistic protocols for fair exchange. In *Proceedings of the ACM Conference on Computer and Communications Security*, April 1997.

[ASW00] N. Asokan, V. Shoup, and M. Waidner. Optimistic fair exchange of digital signatures. *IEEE Journal on Selected Areas in Communications*, 18(4), April 2000.

[AT91] M. Abadi and M. Tuttle. A semantics for a logic of authentication. In *Proceedings of the Tenth Annual ACM Symposium on Principles of Distributed Computing*, pages 201–216, 1991.

[BAN90a] M. Burrows, M. Abadi, and R. Needham. A logic of authentication. Technical Report SRC RR 39, Digital Equipment Corporation, Systems Research Center, February 1990.

[BAN90b] M. Burrows, M. Abadi, and R. Needham. A logic of authentication. *ACM Transactions on Computer Systems*, 8(1):18–36, February 1990.

[BB01] L. Buttyán and N. Ben Salem. A payment scheme for broadcast multimedia streams. In *Proceedings of the 6th IEEE Symposium on Computers and Communications*, July 2001.

[BBC+01] L. Blažević, L. Buttyán, S. Čapkun, S. Giordano, J.-P. Hubaux, and J.-Y. Le Boudec. Self-organization in mobile ad hoc networks: The approach of Terminodes. *IEEE Communications Magazine*, June 2001.

[BDM98] F. Bao, R. Deng, and W. Mao. Efficient and practical fair exchange protocols with off-line TTP. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, pages 77–85, 1998.

[BGH+93] R. Bird, I. Gopal, A. Heizberg, P. Janson, S. Kutten, R. Molva, and M. Yung. Systematic design of a family of attack-resistant authentication protocols. *IEEE Journal on Selected Areas in Communications*, 11(5):679–693, June 1993.

[BGL00] L. Blažević, S. Giordano, and J.-Y. Le Boudec. Self-organizing wide-area routing. In *Proceedings of SCI 2000/ISAS 2000*, July 2000.

[BGSW00] L. Buttyán, C. Gbaguidi, S. Staamann, and U. Wilhelm. Extensions to an authentication technique proposed for the global mobility network. *IEEE Transactions on Communications*, 48(3), March 2000.

[BH00] L. Buttyán and J.-P. Hubaux. Enforcing service availability in mobile ad-hoc WANs. In *Proceedings of the 1st IEEE/ACM Workshop on Mobile Ad Hoc Networking and Computing (MobiHOC)*, August 2000.

[BH01a] L. Buttyán and J.-P. Hubaux. Rational exchange – a formal model based on game theory. In *Proceedings of the 2nd International Workshop on Electronic Commerce (WELCOM)*, November 2001.

[BH01b] L. Buttyán and J.-P. Hubaux. Stimulating cooperation in self-organizing mobile ad hoc networks. Technical Report SSC/2001/046, Swiss Federal Institute of Technology, August 2001.

[Bie90] P. Bieber. A logic of communication in a hostile environment. In *Proceedings of the IEEE Computer Security Foundations Workshop*, pages 14–22, June 1990.

[BM93] C. Boyd and W. Mao. On a limitation of BAN logic. In *Advances in Cryptology – EUROCRYPT'93*, pages 240–247, 1993.

[BM94a] C. Boyd and W. Mao. Design and analysis of key exchange protocols via secure channel identification. In *Advances in Cryptology – ASIACRYPT'94*, 1994.

[BM94b] C. Boyd and W. Mao. Designing secure key exchange protocols. In *Proceedings of the European Symposium on Research in Computer Security*, 1994.

[BMJ+98] J. Broch, D. Maltz, D. Johnson, Y. C. Hu, and J. Jetcheva. A performance comparison of multi-hop wireless ad hoc network routing protocols. In *Proceedings of the ACM/IEEE International Conference on Mobile Computing and Networking (Mobicom)*, 1998.

[BP97] G. Bella and L. Paulson. Using Isabelle to prove properties of the Kerberos authentication system. In *Proceedings of the DIMACS Workshop on Design and Formal Verification of Security Protocols*, 1997.

[BR93] M. Bellare and P. Rogaway. Entity authentication and key distribution. In *Advances in Cryptology – CRYPTO'93*, 1993.

[BR95] M. Bellare and P. Rogaway. Provably secure session key distribution – the three party case. In *Proceedings of the 27th Annual ACM Symposium on the Theory of Computing*, 1995.

[BSW98] L. Buttyán, S. Staamann, and U. Wilhelm. A simple logic for authentication protocol design. In *Proceedings of the IEEE Computer Security Foundations Workshop*, 1998.

[But00] L. Buttyán. Removing the financial incentive to cheat in micropayment schemes. *IEE Electronics Letters*, 36(2), January 2000.

[CFS99] S. Corson, J. Freebersyser, and A. Sastry, editors. *Mobile Networks and Applications (MONET), Special Issue on Mobile Ad Hoc Networking*, October 1999.

[CJ97] J. Clark and J. Jacob. A survey of authentication protocol literature. htt://www.cs.york.ac.uk/ jac/, November 1997.

[Cle90] R. Cleve. Controlled gradual disclosure schemes for random bits and their applications. In *Advances in Cryptology – CRYPTO'89*, pages 573–588, 1990.

[CSNP92] E. Campbell, R. Safavi-Naini, and P. Pleasants. Partial belief and probabilistic reasoning in the analysis of secure protocols. In *Proceedings of the IEEE Computer Security Foundations Workshop*, 1992.

[DGLW96] R. Deng, L. Gong, A. Lazar, and W. Wang. Practical protocols for certified electronic mail. *Journal of Network and Systems Management*, 4(3):279–297, 1996.

[DS81] D. Denning and G. Sacco. Timestamps in key distribution protocols. *Communications of the ACM*, 24(8):198–208, 1981.

[DvOW92] W. Diffie, P. van Oorschot, and M. Wiener. Authentication and authenticated key exchanges. *Designs, Codes, and Cryptography*, 2:107–125, 1992.

[DY83] D. Dolev and A. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, IT-29(2):198–208, March 1983.

[Fed99] H. Federrath. Protection in mobile communications. In G. Mueller and K. Rannenberg, editors, *Multilateral Security in Communications, Volume 3: Technology, Infrastructure, Economy*, pages 349–364. Addison-Wesley, 1999.

[FR97] M. Franklin and M. Reiter. Fair exchange with a semi-trusted third party. In *Proceedings of the ACM Conference on Computer and Communications Security*, pages 1–6, April 1997.

[FT98] M. Franklin and G. Tsudik. Secure group barter: Multi-party fair exchange with semi-trusted neutral parties. In *Proceedings of Financial Cryptography'98*, 1998.

[GK00] P. Gupta and P. R. Kumar. The capacity of wireless networks. *IEEE Transactions on Information Theory*, 46(2):388–404, March 2000.

[GMA+95] S. Glassman, M. Manasse, M. Abadi, P. Gauthier, and P. Sobalvarro. The Millicent protocol for inexpensive electronic commerce. In *Proceedings of the 4th International World Wide Web Conference*, pages 603–618. O'Reilly and Associates, Inc., December 1995.

[GNG97] S. Gritzalis, N. Nikitakos, and P. Georgiadis. Formal methods for the analysis and design of cryptographic protocols: A state-of-the-art review. In *Proceedings of the IFIP Working Conference on Communications and Multimedia Security, Vol. 3.*, pages 119–132, 1997.

[GNY90] L. Gong, R. Needham, and R. Yahalom. Reasoning about belief in cryptographic protocols. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, pages 234–248, 1990.

[GPV99] F. Gaertner, H.-H. Pagnia, and H. Vogt. Approaching a formal definition of fairness in electronic commerce. In *Proceedings of the 18th IEEE Symposium on Reliable Distributed Systems (Workshop on Electronic Commerce)*, pages 354–359, October 1999.

[GS91] K. Gaarder and E. Snekkenes. Applying a formal analysis technique to the CCITT X.509 strong two-way authentication protocol. *Journal of Cryptology*, 3(?):81–98, 1991.

[GS95] L. Gong and P. Syverson. Fail-stop protocols: A new approach to designing secure protocols. In *Proceedings of the 5th International Working Conference on Dependable Computing for Critical Applications*, pages 44–55, 1995.

[HBC01] J.-P. Hubaux, L. Buttyán, and S. Čapkun. The quest for security in mobile ad hoc networks. In *Proceedings of the ACM Symposium on Mobile Ad Hoc Networking and Computing (MobiHOC)*, October 2001.

[HC96] G. E. Hughes and M. J. Cresswell. *A New Introduction to Modal Logic*. Routledge, 1996.

[HGLV01] J.-P. Hubaux, T. Gross, J.-Y. Le Boudec, and M. Vetterli. Towards self-organized mobile ad hoc networks: The Terminodes Project. *IEEE Communications Magazine*, January 2001.

[HM90] J. Halpern and Y. Moses. Knowledge and common knowledge in a distributed environment. *Journal of the ACM*, 37(3):549–587, 1990.

[Hoa85] C. Hoare. *Communicating Sequential Processes*. Prentice-Hall, 1985.

[HSW96] R. Hauser, M. Steiner, and M. Waidner. Micropayments based on iKP. Technical Report RZ 2791, IBM Zurich Research Lab, February 1996.

[HT96] N. Heintze and J. D. Tygar. A model for secure protocols and their compositions. *IEEE Transactions on Software Engineering*, 22(1):16–30, January 1996.

[IBM97] IBM. IBM 4758 PCI cryptographic coprocessor. Secure Way Cryptographic Products, June 1997.

[Jak95] M. Jakobsson. Ripping coins for a fair exchange. In *Advances in Cryptology – EUROCRYPT'95*, pages 220–230, 1995.

[Kem89] R. Kemmerer. Analyzing encryption protocols using formal verification techniques. *IEEE Journal on Selected Areas in Communications*, 7(4):448–457, October 1989.

[Ket95] S. Ketchpel. Transaction protection for information buyers and sellers. In *Proceedings of DAGS'95: Electronic Publishing and the Information Superhighway*, June 1995.

[KMM94] R. Kemmerer, C. Meadows, and J. Millan. Three systems for cryptographic protocol analysis. *Journal of Cryptology*, 7(2):79–130, 1994.

[KR00] S. Kremer and J.-F. Raskin. Formal verification of non-repudiation protocols – a game approach. In *Proceedings of Formal Methods for Computer Security (FMCS 2000)*, July 2000.

[KS96] A. Keromytis and J. Smith. Creating efficient fail-stop cryptographic protocols. Technical Report MS-CIS-96-32, University of Pennsylvania, December 1996.

[LABW92] B. Lampson, M. Abadi, M. Burrows, and E. Wobber. Authentication in distributed systems: Theory and practice. *ACM Transactions on Computer Systems*, 10(4):265–310, November 1992.

[LHB96] R. Lichota, G. Hammonds, and S. Brackin. Verifying the correctness of cryptographic protcols using Convince. In *Proceedings of the 12th IEEE Computer Security Applications Conference*, pages 117–128, 1996.

[Low95] G. Lowe. An attack on the Needham-Schroeder public-key authentication protocol. *Information Processing Letters*, 56(3):131–133, 1995.

[Low96] G. Lowe. Breaking and fixing the Needham-Schroeder public key protocol using FDR. In *Proceedings of the TACAS*, pages 147–166, 1996.

[LR92] D. Longley and S. Rigby. An automatic search for security flaws in key management schemes. *Computers and Security*, 11(1):75–89, January 1992.

[Lyn96] N. Lynch. *Distributed Algorithms*. Morgan Kaufmann, 1996.

[MB93] W. Mao and C. Boyd. Towards formal analysis of security protocols. In *Proceedings of the IEEE Computer Security Foundations Workshop*, pages 147–158, 1993.

[MCF87] J. Millen, S. Clark, and S. Freedman. The Interrogator: Protocol security analysis. *IEEE Transactions on Software Engineering*, SE-13(2):274–288, February 1987.

[Mea91] C. Meadows. A system for the specification and analysis of key management protocols. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, pages 182–195, 1991.

[Mea92] C. Meadows. Applying formal methods to the analysis of a key management protocol. *Journal of Computer Security*, 1(1):5–35, 1992.

[Mea95] C. Meadows. Formal verification of cryptographic protocols: A survey. In *Advances in Cryptology - ASIACRYPT'94*, pages 135–150. Springer-Verlag, 1995.

[Mer83] M. Merritt. *Cryptographic Protocols*. PhD thesis, Georgia Institute of Technology, 1983.

[MG98] A. Mehrotra and L. Golding. Mobility and security management in the GSM system and some proposed future improvements. *Proceedings of the IEEE*, 86(7):1480–1496, 1998.

[MGLB00] S. Marti, T. Giuli, K. Lai, and M. Baker. Mitigating routing misbehavior in mobile ad hoc networks. In *Proceedings of the ACM/IEEE International Conference on Mobile Computing and Networking (Mobicom)*, August 2000.

[Mil99] R. Milner. *Communicating and Mobile Systems: the π-Calculus*. Cambridge University Press, 1999.

[Mos89] L. Moser. A logic of knowledge and belief for reasoning about computer security. In *Proceedings of the IEEE Computer Security Foundations Workshop*, pages 57–63, 1989.

[MPM+98] K. Martin, B. Preneel, C. Mitchell, H. Hitz, G. Horn, A. Poliakova, and P. Howard. Secure billing for mobile information services in UMTS. In *Proceedings of IS&N'98*, 1998.

[MS94] U. Maurer and P. Schmid. A calculus for secure channel establishment in open networks. In *Proceedings of the European Symposium on Research in Computer Security*, 1994.

[MvOV97] A. Menezes, P. van Oorschot, and S. Vanstone. *Handbook of Applied Cryptography*. Discrete Mathematics and Its Applications. CRC Press, Inc., 1997.

[Nes90] D. Nessett. A critique of the burrows, abadi and needham logic. *Operating Systems Review*, 24(2):35–38, April 1990.

[NS78] R. Needham and M. Schroeder. Using encryption for authentication in large networks of computers. *Communications of the ACM*, 21(12):993–999, 1978.

[NT92] B. Nieh and S. Tavares. Modelling and analysing cryptographic protocols using Petri nets. In *Advances in Cryprology – AUSCRYPT'92*, 1992.

[OR94] M. Osborne and A. Rubinstein, editors. *A Course in Game Theory*. MIT Press, 1994.

[Ora01] A. Oram, editor. *Peer-to-Peer: Harnessing the Benefits of a Disruptive Technology*. O'Reilly and Associates, Inc., 2001.

[Pau98] L. Paulson. The inductive approach to verifying cryptographic protocols. *Journal of Computer Security*, 6(1–2):85–128, 1998.

[Pau99] L. Paulson. Inductive analysis of the internet protocol TLS. *ACM Transactions on Information and System Security*, 2(3):332–351, 1999.

[Ped95] T. Pedersen. Electronic payment of small amounts. Technical Report DAIMI-PB-495, Aarhus University, Computer Science Department, 1995.

[Per01] C. Perkins, editor. *Ad Hoc Networking*. Addison-Wesley, 2001.

[PG99] H.-H. Pagnia and F. Gaertner. On the impossibility of fair exchange without a trusted third party. Technical Report TUD-BS-1999-02, Darmstadt University of Technology, Department of Computer Science, March 1999.

[PK00] G. Pottie and W. Kaiser. Wireless integrated sensor networks. *Communications of the ACM*, May 2000.

[PPW97] A. Pfitzmann, B. Pfitzmann, and M. Waidner. Trusting mobile user devices and security modules. *IEEE Computer*, February 1997.

[PS00] A. Perrig and D. Song. Looking for diamonds in the desert – extending automatic protocol generation to three-party authentication and key agreement protocols. In *Proceedings of the IEEE Computer Security Foundations Workshop*, 2000.

[PVG01] H.-H. Pagnia, H. Vogt, and F. Gaertner. Fair exchange. manuscript, August 2001.

[Ran88] P. V. Rangan. An axiomatic basis of trust in distributed systems. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, pages 204–211, April 1988.

[RH93] A. Rubin and P. Honeyman. Formal methods for the analysis of authentication protocols. Technical Report CITI TR 93-7, October 1993.

[Ros95] A. W. Roscoe. Modelling and verifying key exchange protocols using CSP and FDR. In *Proceedings of the IEEE Computer Security Foundations Workshop*, pages 98–107, 1995.

[RS96] R. Rivest and A. Shamir. PayWord and MicroMint: Two simple micropayment schemes. Technical report, MIT Laboratory for Computer Science, 1996.

[RSW96] R. Rivest, A. Shamir, and D. Wagner. Time-lock puzzles and time-release crypto. Technical Report MIT/LCS/TR-684, MIT Laboratory for Computer Science, March 1996.

[Rue91] R. Rueppel. A formal approach to security architectures. In *Advances in Cryptology – EUROCRYPT'91*, pages 387–398, 1991.

[San97] T. Sandholm. Unenforced e-commerce transactions. *IEEE Internet Computing*, 1(6):47–54, November-December 1997.

[Sch98] S. Schneider. Formal analysis of a non-repudiation protocol. In *Proceedings of the IEEE Computer Security Foundations Workshop*, pages 54–65, 1998.

[SM93] P. Syverson and C. Meadows. A logical language for specifying cryptographic protocol requirements. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, pages 165–177, 1993.

[SM95] P. Syverson and C. Meadows. Formal requirements for key distribution protocols. In *Advances in Cryptology - EUROCRYPT'94*, pages 320–331. Springer-Verlag, 1995.

[SN97] S. Suzuki and K. Nakada. An authentication technique based on distributed security management for the Global Mobility Network. *IEEE Journal on Selected Areas in Communications*, 15(8):1608–1617, 1997.

[Sne95] E. Snekkenes. *Formal Specification and Analysis of Cryptographic Protocols*. PhD thesis, University of Oslo, Norway, 1995.

[Son99] D. Song. Athena: An automatic checker for security protocol analysis. In *Proceedings of the IEEE Computer Security Foundations Workshop*, 1999.

[SS98] V. Shmatikov and U. Stern. Efficient finite-state analysis for large security protocols. In *Proceedings of the IEEE Computer Security Foundations Workshop*, pages 106–115, 1998.

[SvO94] P. Syverson and P. van Oorschot. On unifying some cryptographic protocol logics. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, pages 14–28, 1994.

[Syv90a] P. Syverson. Formal semantics for logics of cryptographic protocols. In *Proceedings of the IEEE Computer Security Foundations Workshop*, pages 32–41, June 1990.

[Syv90b] P. Syverson. A logic for the analysis of cryptographic protocols. Technical Report 9305, Naval Research Lab, December 1990.

[Syv91a] P. Syverson. The use of logic in the analysis of cryptographic protocols. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, pages 156–170, 1991.

[Syv91b] P. Syverson. The value of semantics for the analysis of cryptographic protocols. In *Proceedings of the IEEE Computer Security Foundations Workshop*, pages 228–229, 1991.

[Syv92] P. Syverson. Knowledge, belief, and semantics in the analysis of cryptographic protocols. *Journal of Computer Security*, 1(3):317–334, 1992.

[Syv93] P. Syverson. Adding time to a logic of authentication. In *Proceedings of the ACM Conference on Computer and Communications Security*, pages 97–101, November 1993.

[Syv94] P. Syverson. A taxonomy of replay attacks. In *Proceedings of the IEEE Computer Security Foundations Workshop*, 1994.

[Syv96] P. Syverson. Limitations on design principles for public key protocols. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, pages 62–73, 1996.

[Syv98] P. Syverson. Weakly secret bit commitment: Applications to lotteries and fair exchange. In *Proceedings of the IEEE Computer Security Foundations Workshop*, pages 2–13, 1998.

[TB95] D. Trcek and B. Blazic. Formal language for security services base modelling and analysis. *Computer Communications*, 18(12):921–928, December 1995.

[Tou92] M.-J. Toussaint. Deriving the complete knowledge of participants in cryptographic protocols. In *Advances in Cryptology – CRYPTO'91*, pages 24–43. Springer-Verlag, 1992.

[Tyg96] J. D. Tygar. Atomicity in electronic commerce. In *Proceedings of the 15th ACM Symposium on Principles of Distributed Computing*, pages 8–26. ACM Press, May 1996.

[vO93] P. van Oorschot. Extending cryptographic logics of belief to key agreement protocols. In *Proceedings of the ACM Conference on Computer and Communications Security*, pages 232–243, 1993.

[Var89] V. Varadharajan. Verification of network security protocols. *Computers and Security*, 8(8):693–708, 1989.

[Var90] V. Varadharajan. Use of a formal description technique in the specification of authentication protocols. *Computer Standards and Interfaces*, 9:203–215, 1990.

[WABL94] E. Wobber, M. Abadi, M. Burrows, and B. Lampson. Authentication in the Taos operating system. *ACM Transactions on Computer Systems*, 12(1):3–32, February 1994.

[WL93] T. Woo and S. Lam. A semantic model for authentication protocols. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, pages 178–194, 1993.

[YKB93] R. Yahalom, B. Klein, and T. Beth. Trust relationships in secure systems: A distributed authentication perspective. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, pages 150–164, 1993.

[ZG96] J. Zhou and D. Gollmann. A fair non-repudiation protocol. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, pages 55–61, May 1996.

Appendix A

Synthesis rules

(S1) In order for principal P to believe ϕ, it is sufficient to believe ϕ' and $(\phi' \Rightarrow \phi)$.

$$P \models \phi \ \left\{ \begin{array}{l} P \models \phi' \\ P \models (\phi' \Rightarrow \phi) \end{array} \right.$$

(S2) In order for principal P to believe ϕ, it is sufficient to believe that another principal Q believes ϕ and to believe that Q is competent (at least if ϕ is concerned).

$$P \models \phi \ \left\{ \begin{array}{l} P \models (Q \models \phi) \\ P \models ((Q \models \phi) \Rightarrow \phi) \end{array} \right.$$

(S3) In order for principal P to believe that another principal Q believes ϕ, it is sufficient to believe that Q has recently said ϕ and to believe that Q is honest (at least if ϕ is concerned).

$$P \models (Q \models \phi) \ \left\{ \begin{array}{l} P \models (Q \mid\sim \phi) \\ P \models ((Q \mid\sim \phi) \Rightarrow (Q \models \phi)) \end{array} \right.$$

(S4) In order for principal P to believe that another principal Q has recently said a message X, it is sufficient to believe that Q has recently said a message that contains X. In the rule, we denote this message by $X'; X; X''$, but we note that X' or X'' could be missing.

$$P \models (Q \mid\sim X) \ \left\{ \ P \models (Q \mid\sim X'; X; X'') \right.$$

(S5) In order for principal P to believe that another principal Q has recently said a message X, it is sufficient to believe that Q said a message that contains X, and to believe that this message is fresh. In the rule, we denote this message by $X'; X; X''$, but we note that X' and/or X'' could be missing.

$$P \models (Q \mid\sim X) \ \left\{ \begin{array}{l} P \models (Q \mid\sim X'; X; X'') \\ P \models \sharp(X'; X; X'') \end{array} \right.$$

(S6) In order for principal P to believe that another principal Q has recently said a message X, it is sufficient to believe that X arrived on a channel C, the source set of which is believed to be the singleton $\{Q\}$, and to believe that C is timely.

$$P \models (Q \mid\sim X) \ \left\{ \begin{array}{l} P \models (P \lhd C(X)) \\ P \models (s(C) = \{Q\}) \\ P \models \natural(C) \end{array} \right.$$

(S7) In order for principal P to believe that another principal Q said a message X, it is sufficient to believe that Q said a message that contains X. In the rule, we denote this message by $X'; X; X''$, but we note that X' or X'' could be missing.

$$P \models (Q \mid\sim X) \ \{ \ P \models (Q \mid\sim X'; X; X'')$$

(S8) In order for principal P to believe that another principal Q said a message X, it is sufficient to believe that X arrived on a channel C, and to believe that the source set of C is the singleton $\{Q\}$.

$$P \models (Q \mid\sim X) \ \left\{ \begin{array}{l} P \models (P \lhd C(X)) \\ P \models (s(C) = \{Q\}) \end{array} \right.$$

(S9) In order for principal P to believe that a compound message is fresh, it is sufficient to believe that part of the message is fresh. Again, X' or X'' could be missing from $X'; X; X''$ in the rule below.

$$P \models \sharp(X'; X; X'') \ \{ \ P \models \sharp(X)$$

(S10) In order for principal P to recognize that it received a message X on channel C, it is sufficient to see $C(X)$ and to be able to read from C.

$$P \models (P \lhd C(X)) \ \left\{ \begin{array}{l} P \lhd C(X) \\ P \in r(C) \end{array} \right.$$

(S11) In order for principal P to see message X, it is sufficient to see a message that contains X. In the rule, we denote this message by $X'; X; X''$, but we note that X' or X'' could be missing.

$$P \lhd X \ \{ \ P \lhd X'; X; X''$$

(S12) In order for principal P to see a message X, it is sufficient to see $C(X)$ for some channel C, and to be able to read from C.

$$P \lhd X \ \left\{ \begin{array}{l} P \lhd C(X) \\ P \in r(C) \end{array} \right.$$

(*S13) Whenever $P \models (Q \models \phi)$ appears as a goal during the synthesis, then we must also *add* the goal $Q \models \phi$.

$$P \models (Q \models \phi) \ \left\{ \begin{array}{l} P \models (Q \models \phi) \\ Q \models \phi \end{array} \right.$$

(*S14) Whenever $P \models (Q \mid\!\sim X)$ appears as a goal during the synthesis, we must also *add* the goals $Q \lhd X_1$, $Q \lhd X_2$, ..., where $X_1, X_2, \ldots$ represent those parts of message X that cannot be computed by Q.

$$P \models (Q \mid\!\sim X) \ \left\{ \begin{array}{l} P \models (Q \mid\!\sim X) \\ Q \lhd X_1 \\ Q \lhd X_2 \\ \ldots \end{array} \right.$$

(*S15) Whenever $P \models (Q \mid\sim X)$ appears as a goal during the synthesis, we must also *add* the goals $Q \lhd X_1$, $Q \lhd X_2$, ..., where $X_1, X_2, \ldots$ represent those parts of message X that cannot be computed by Q.

$$P \models (Q \mid\sim X) \left\{ \begin{array}{l} P \models (Q \mid\sim X) \\ Q \lhd X_1 \\ Q \lhd X_2 \\ \ldots \end{array} \right.$$

Appendix B

Proofs of Section 5.2

Lemma 5.1

The lemma states that if B receives a message $m = (\mu, \rho, \sigma) \in M_4$ in round r in q in the restricted protocol game $G_{\pi|\bar{s}}(L)$, then V must receive μ in an earlier round $r' < r$ in q.

Proof: Let us assume that B receives m in round r in q, but V does not receive μ before round r in q. Since B receives m in round r, U or V must send m in round r. According to the formulae with which we tagged the messages in M_4, U can send m only if she receives m in an earlier round. In addition, V can send m only if it receives μ or a message in M_4 or in M_4' that contains μ in an earlier round. Note that, by assumption, V does not receive μ.

Let $M_4(\mu) = \{(\mu', \rho', \sigma') \in M_4 : \mu' = \mu\}$ and $M_4'(\mu) = \{(\mu', \sigma') \in M_4' : \mu' = \mu\}$ (i.e., $M_4(\mu)$ and $M_4'(\mu)$ contain those messages in M_4 and M_4', respectively, that contain μ). If U does not receive any message in $M_4(\mu) \cup M_4'(\mu)$ before round r in q, then let $r_1^* = r$, otherwise, let r_1^* be the earliest round in q in which U receives a message in $M_4(\mu) \cup M_4'(\mu)$. Similarly, if V does not receive any message in $M_4(\mu) \cup M_4'(\mu)$ before round r in q, then let $r_2^* = r$, otherwise, let r_2^* be the earliest round in q in which V receives a message in $M_4(\mu) \cup M_4'(\mu)$.

Now, we can distinguish two cases: (a) $r_1^* \leq r_2^*$ and (b) $r_2^* < r_1^*$.

Case (a): Recall that either U must receive $m = (\mu, \rho, \sigma)$ before round r in q, or V must receive a message in $M_4(\mu) \cup M_4'(\mu)$ before round r in q. However, this is not possible if $r_1^* = r$. Thus, $r_1^* < r$ must hold. This means that U receives, and thus, V sends a message in $M_4(\mu) \cup M_4'(\mu)$ in round r_1^*. According to the formulae with which we tagged the messages in M_4 and in M_4', this is possible only if V receives μ or a message in $M_4(\mu) \cup M_4'(\mu)$ in an earlier round $\hat{r} < r_1^* \leq r_2^*$. By assumption, V does not receive μ before round r. Furthermore, V cannot receive any message in $M_4(\mu) \cup M_4'(\mu)$ before round r_2^*.

Case (b): $r_2^* < r$ must hold, since otherwise r_1^* would be greater than r, which is not possible by definition. This means that V receives, and thus, U sends a message $m' \in M_4(\mu) \cup M_4'(\mu)$ in round r_2^*. According to the formulae with which we tagged the messages in M_4 and in M_4', this is possible only if U receives m' in an earlier round $\hat{r} < r_2^* < r_1^*$. However, U cannot receive any message in $M_4(\mu) \cup M_4'(\mu)$ before round r_1^*. □

Lemma 5.2

The lemma states that if B receives a message $m = (\mu, \rho, \sigma) \in M_4$ in round r in q in the restricted protocol game $G_{\pi|\bar{s}}(L)$, and $h(\rho) = h(rnd)$, then V must receive ρ in an earlier

round $r' < r$ in q.

Proof: Let us assume that B receives m in round r in q, but V does not receive ρ before round r in q. Since B receives m in round r, U or V must send m in round r. According to the formulae with which we tagged the messages in M_4, U can send m only if she receives m in an earlier round. In addition, since $h(\rho) = h(rnd)$, V can send m only if she receives ρ or a message in M_4 that contains ρ in an earlier round. Note that, by assumption, V does not receive ρ.

Let $M_4(\rho) = \{(\mu', \rho', \sigma') \in M_4 : \rho' = \rho\}$ (i.e., $M_4(\rho)$ contains those messages in M_4 that contain ρ). If U does not receive any message in $M_4(\rho)$ before round r in q, then let $r_1^* = r$, otherwise, let r_1^* be the earliest round in q in which U receives a message in $M_4(\rho)$. Similarly, if V does not receive any message in $M_4(\rho)$ before round r in q, then let $r_2^* = r$, otherwise, let r_2^* be the earliest round in q in which V receives a message in $M_4(\rho)$.

Now, we can distinguish two cases: (a) $r_1^* \leq r_2^*$ and (b) $r_2^* < r_1^*$.

Case (a): Recall that either U must receive $m = (\mu, \rho, \sigma)$ before round r in q, or V must receive a message in $M_4(\rho)$ before round r in q. However, this is not possible if $r_1^* = r$. Thus, $r_1^* < r$ must hold. This means that U receives, and thus, V sends a message in $M_4(\rho)$ in round r_1^*. According to the formulae with which we tagged the messages in M_4, this is possible only if V receives ρ or a message in $M_4(\rho)$ in an earlier round $\hat{r} < r_1^* \leq r_2^*$. By assumption, V does not receive ρ before round r. Furthermore, V cannot receive any message in $M_4(\rho)$ before round r_2^*.

Case (b): $r_2^* < r$ must hold, since otherwise r_1^* would be greater than r, which is not possible by definition. This means that V receives, and thus, U sends a message $m' \in M_4(\rho)$ in round r_2^*. According to the formulae with which we tagged the messages in M_4, this is possible only if U receives m' in an earlier round $\hat{r} < r_2^* < r_1^*$. However, U cannot receive any message in $M_4(\rho)$ before round r_1^*. □

Lemma 5.3

The lemma states that if B receives a message $m = (\mu, \sigma) \in M_4'$ in round r in q in the restricted protocol game $G_{\pi|\bar{s}}(L)$, then V must receive μ in an earlier round $r' < r$ in q.

Proof: The proof of this lemma is identical to that of Lemma 5.1. □

Lemma 5.4

The lemma states that no player can ever receive a message $m \in M_2$ such that $m \neq srv$.

Proof: Let us assume that a player received a message $m \in M_2$ such that $m \neq srv$ in round r in q. This means that a player sends m in round r in q. Let us denote the earliest round in q in which m is sent by any of the players by r^*. According to the logical formulae with which we tagged the messages in M_2, any player can send m in round r^* only if she receives it in an earlier round. This contradicts the fact that r^* is the earliest round in q in which m is sent. □

Appendix C

Proofs of Section 5.4

Lemma 5.12

The lemma states that if A sends a message $m = (\kappa, \mu, \sigma) \in M_3$ in round r in q, then she must receive μ in an earlier round $r' < r$ in q.

Proof: Let us suppose that A sends m in round r in q, but she does not receive μ before round r in q. It can be seen from the formulae with which we tagged the messages in M_3 that A can send m in round r only if she receives μ or a message in M_3 that contains μ in an earlier round. By assumption, A does not receive μ before round r, and thus, A must receive a message in M_3 that contains μ before round r.

Let $M_3(\mu) = \{(\kappa', \mu', \sigma') \in M_3 : \mu' = \mu\}$ (i.e., $M_3(\mu)$ contains those messages in M_3 that contain μ). Let r^* be the earliest round in q in which A receives a message in $M_3(\mu)$, and let this message be m^*. Such r^* and m^* exist because (i) we know that A must receive a message in $M_3(\mu)$ before round r in q, and (ii) round numbers are positive integers. In addition, from (i), we get that $r^* < r$ must hold.

Since the network is reliable, if A receives m^* in round r^*, then B sends m^* in round r^*. However, it can be seen from the formulae with which we tagged the messages in M_3 that this is possible only if B receives m^* in an earlier round $\hat{r} < r^*$. This means that A sends m^* in round $\hat{r}$. Again from the formulae with which we tagged the messages in M_3, it can be seen that A can send $m^* \in M_3(\mu)$ in round $\hat{r}$ only if she receives μ or a message in $M_3(\mu)$ before round $\hat{r}$. By assumption, A cannot receive μ before round $\hat{r} < r^* < r$. Thus, she must receive a message in $M_3(\mu)$ before round $\hat{r}$. But this contradicts the fact that the earliest round in which such a message is received by A is r^*. □

Lemma 5.13

The lemma states that if B sends a message $m = (\gamma, \mu, \sigma) \in M_2$ in round r in q, then she must receive μ in an earlier round $r' < r$ in q.

Proof: Let us suppose that B sends m in round r in q, but she does not receive μ before round r in q. It can be seen from the formulae with which we tagged the messages in M_2 that B can send m in round r only if she receives μ or a message in M_2 or in M_3 that contains μ in an earlier round. By assumption, B does not receive μ before round r, and thus, B must receive a message in M_2 or in M_3 that contains μ before round r.

Let $M_2(\mu) = \{(\gamma', \mu', \sigma') \in M_2 : \mu' = \mu\}$ and $M_3(\mu) = \{(\kappa', \gamma', \mu', \sigma', \sigma'') \in M_3 : \mu' = \mu\}$ (i.e., $M_2(\mu)$ and $M_3(\mu)$ contain those messages in M_2 and in M_3, respectively, that contain μ). If B does not receive any message in $M_2(\mu)$ before round r in q, then let $r_2^* = r$, otherwise, let r_2^* be the earliest round in q in which B receives a message in $M_2(\mu)$. Similarly, if B does not receive any message in $M_3(\mu)$ before round r in q, then let $r_3^* = r$, otherwise, let r_3^* be the earliest round in q in which B receives a message in $M_3(\mu)$.

Now, we can distinguish two cases: (a) $r_2^* \leq r_3^*$ and (b) $r_3^* < r_2^*$.

Case (a): Recall that B must receive a message in $M_2(\mu)$ or in $M_3(\mu)$ before round r in q. This is not possible if $r = r_2^*$. Thus, $r_2^* < r$ must hold. This also means that B receives a message in $M_2(\mu)$ in round r_2^*. Let us denote this message by m_2^*.

If B receives m_2^* in round r_2^*, then A sends m_2^* in round r_2^*. However, it can be seen from the formulae with which we tagged the messages in M_2 that A can send m_2^* in round r_2^* only if she receives (i) m_2^* or (ii) a message $m_3' = (\kappa', m_2^*, \sigma') \in M_3(\mu)$ in an earlier round $\hat{r} < r_2^*$. We show that neither (i) nor (ii) is possible.

(i) If A receives $m_2^* \in M_2(\mu)$ in round $\hat{r}$, then B sends m_2^* in round $\hat{r}$. It can be seen from the formulae with which we tagged the messages in M_2 that this is possible only if B receives μ or a message in $M_2(\mu)$ or in $M_3(\mu)$ before round $\hat{r}$. By assumption, B does not receive μ before round r. Thus, B must receive a message in $M_2(\mu)$ or in $M_3(\mu)$ before round $\hat{r} < r_2^* \leq r_3^*$. However, because of the definitions of r_2^* and r_3^*, B cannot receive any message in $M_2(\mu)$ before r_2^* and any message in $M_3(\mu)$ before r_3^*.

(ii) If A receives $m_3' = (\kappa', m_2^*, \sigma') \in M_3(\mu)$ in round $\hat{r}$, then B sends m_3' in round $\hat{r}$. It can be seen from the formulae with which we tagged the messages in M_3 that this is possible only if B receives m_3' before round $\hat{r} < r_2^* \leq r_3^*$. However, because of the definition of r_3^*, B cannot receive any message in $M_3(\mu)$ before round r_3^*.

Case (b): If $r_3^* < r_2^*$, then $r_3^* < r$ must also hold (since otherwise r_2^* would be greater than r, which is not possible by definition). This means that B receives a message in $M_3(\mu)$ in round r_3^*. Let this message be $m_3^* = (\kappa^*, \gamma^*, \mu, \sigma^*, \sigma^{**})$. If B receives m_3^* in round r_3^*, then A sends m_3^* in round r_3^*. However, from Lemma 5.12, we know that A can send m_3^* in round r_3^* only if she receives a message $m_2' = (\gamma^*, \mu, \sigma^*) \in M_2(\mu)$ in an earlier round $\hat{r} < r_3^*$. This means that B sends m_2' in round $\hat{r}$. It can be seen from the formulae with which we tagged the messages in M_2 that this is possible only if B receives μ or a message in $M_2(\mu)$ or in $M_3(\mu)$ before round $\hat{r}$. By assumption, B does not receive μ before round r. Thus, B must receive a message in $M_2(\mu)$ or in $M_3(\mu)$ before round $\hat{r} < r_3^* < r_2^*$. However, because of the definitions of r_2^* and r_3^*, B cannot receive any message in $M_2(\mu)$ before round r_2^* and any message in $M_3(\mu)$ before round r_3^*. □

Lemma 5.14

The lemma states that B cannot receive a message $m \in M_3$ before round 3.

Proof: Let us assume that B receives $m = (\kappa, \gamma, \mu, \sigma, \sigma')$ in round r, where $r < 3$. This means that A sends m in round r. According to Lemma 5.12, this is possible only if A receives $m' = (\gamma, \mu, \sigma)$ in an earlier round $r' < r$. Thus, B sends m' in round r'. According to Lemma 5.13, this is possible only if B receives μ in an earlier round $r'' < r' < r$. But this is impossible, since round numbers are positive integers, and $r < 3$. □

Lemma 5.15

The lemma states that no player can ever receive a message $m = (\delta, \varepsilon, \omega, \sigma) \in M_1$ such that $\mathit{fit}(\mathit{dec}(w^{-1}(\omega), \varepsilon), \mathit{dsc}_V) = \mathsf{true}$ and $\mathit{dec}(w^{-1}(\omega), \varepsilon) \neq \mathit{itm}_V$.

Proof: Let us suppose that there exist a player $i \in P'$, a round number $r \in N$, and an action sequence $q \in Q$ such that $(\mathit{rcv}(m), r) \in H_i(q)$. This means that a player j sends m in round r in q. According to the logical formulae with which we tagged the messages in M_1, this is possible only if j receives m or a message in M_2 or in M_3 that contains m before round r – no matter whether j is A or B.

Let $M_2(m) = \{(\gamma', \mu', \sigma') \in M_2 : \mu' = m\}$ and $M_3(m) = \{(\kappa', \gamma', \mu', \sigma', \sigma'') \in M_3 : \mu = m\}$. If no player receives m before round r in q, then let $r_1^* = r$, otherwise let r_1^* be the earliest round in q in which m is received by any of the players. If no player receives any message in $M_2(m)$ before round r in q, then let $r_2^* = r$, otherwise let r_2^* be the earliest round in q in which a message in $M_2(m)$ is received by any of the players. Finally, if no player receives any message in $M_3(m)$ before round r in q, then let $r_3^* = r$, otherwise let r_3^* be the earliest round in q in which a message in $M_3(m)$ is received by any of the players.

Now, we can distinguish three cases: (a) $r_1^* \leq r_2^*, r_3^*$, (b) $r_2^* \leq r_1^*, r_3^*$, and (c) $r_3^* \leq r_1^*, r_2^*$.
Case (a): Recall that j receives m or a message in $M_2(m)$ or in $M_3(m)$ before round r in q. Note that if $r_1^* = r$, then no player receives m or any message in $M_2(m)$ or in $M_3(m)$ before round r in q. Thus, $r_1^* < r$ must hold. This means that a player receives m in round r_1^*. If a player receives m in round r_1^*, then a player must send m in round r_1^*. According to the logical formulae with which we tagged the messages in M_1, this is possible only if that player receives m or a message in $M_2(m)$ or in $M_3(m)$ in an earlier round $\hat{r} < r_1^* \leq r_2^*, r_3^*$. However, because of the definitions of r_1^*, r_2^*, and r_3^*, no player can receive m before round r_1^*, a message in $M_2(m)$ before round r_2^*, and a message in $M_3(m)$ before round r_3^*.
Case (b): Recall that j receives m or a message in $M_2(m)$ or in $M_3(m)$ before round r in q. Note that if $r_2^* = r$, then no player receives m or any message in $M_2(m)$ or in $M_3(m)$ before round r in q. Thus, $r_2^* < r$ must hold. This means that a player receives a message $m' = (\gamma', m, \sigma') \in M_2(m)$ in round r_2^*. If a player receives m' in round r_2^*, then a player must send m' in round r_2^*. According to the logical formulae with which we tagged the messages in M_2, A can send m' in round r_2^* only if she receives $m' \in M_2(m)$ or a message in $M_3(m)$ that contains m' in an earlier round $\hat{r} < r_2^* \leq r_1^*, r_3^*$. Furthermore, B can send m' in round r_2^* only if it receives m or a message in $M_2(m)$ or in $M_3(m)$ in an earlier round $\tilde{r} < r_2^* \leq r_1^*, r_3^*$. However, because of the definitions of r_1^*, r_2^*, and r_3^*, no player can receive m before round r_1^*, a message in $M_2(m)$ before round r_2^*, and a message in $M_3(m)$ before round r_3^*.
Case (c): Recall that j receives m or a message in $M_2(m)$ or in $M_3(m)$ before round r in q. Note that if $r_3^* = r$, then no player receives m or any message in $M_2(m)$ or in $M_3(m)$ before round r in q. Thus, $r_3^* < r$ must hold. This means that a player receives a message $m' = (\kappa', \gamma', m, \sigma', \sigma'') \in M_3(m)$ in round r_3^*. If a player receives m' in round r_3^*, then a player must send m' in round r_3^*. According to the logical formulae with which we tagged the messages in M_3, A can send m' in round r_3^* only if she receives $(\gamma', m, \sigma') \in M_2(m)$ or a message in $M_3(m)$ that contains (γ', m, σ') in an earlier round $\hat{r} < r_3^* \leq r_2^*$. Furthermore, B can send m' in round r_3^* only if she receives $m' \in M_3(m)$ in an earlier round $\tilde{r} < r_3^*$. However, because of the definitions of r_2^*, and r_3^*, no player can receive a message in $M_2(m)$ before round r_2^*, and a message in $M_3(m)$ before round r_3^*. □

Lemma 5.16

The lemma states that no player can ever receive a message $m = (\gamma, \mu, \sigma) \in M_2$ such that $\gamma \neq itm_U$.

Proof: Let us suppose that there exist a player $i \in P'$, a round number $r \in N$, and an action sequence $q \in Q$ such that $(rcv(m), r) \in H_i(q)$. This means that a player j sends m in round r in q. According to the logical formulae with which we tagged the messages in M_2, this is possible only if j receives a message in M_2 or in M_3 that contains γ before round r – no matter whether j is A or B.

Let $M_2(\gamma) = \{(\gamma', \mu', \sigma') \in M_2 : \gamma' = \gamma\}$ and $M_3(\gamma) = \{(\kappa', \gamma', \mu', \sigma', \sigma'') \in M_3 : \gamma' = \gamma\}$. If no player receives any message in $M_2(\gamma)$ before round r in q, then let $r_2^* = r$, otherwise let r_2^* be the earliest round in q in which a message in $M_2(\gamma)$ is received by any of the players. If no player receives any message in $M_3(\gamma)$ before round r in q, then let $r_3^* = r$, otherwise let r_3^* be the earliest round in q in which a message in $M_3(\gamma)$ is received by any of the players.

Now, we can distinguish two cases: (a) $r_2^* \leq r_3^*$, and (b) $r_3^* < r_2^*$.

Case (a): Recall that j receives a message in $M_2(\gamma)$ or in $M_3(\gamma)$ before round r in q. Note that if $r_2^* = r$, then no player receives any message in $M_2(\gamma)$ or in $M_3(\gamma)$ before round r in q. Thus, $r_2^* < r$ must hold. This means that a player receives a message $m' = (\gamma, \mu', \sigma') \in M_2(\gamma)$ in round r_2^*. If a player receives m' in round r_2^*, then a player must send m' in round r_2^*. According to the logical formulae with which we tagged the messages in M_2, A can send m' in round r_2^* only if she receives $m' \in M_2(\gamma)$ or a message in $M_3(\gamma)$ that contains m' in an earlier round $\hat{r} < r_2^* \leq r_3^*$. Furthermore, B can send m' in round r_2^* only if it receives a message in $M_2(\gamma)$ or in $M_3(\gamma)$ in an earlier round $\tilde{r} < r_2^* \leq r_3^*$. However, because of the definitions of r_2^* and r_3^*, no player can receive a message in $M_2(\gamma)$ before round r_2^*, and a message in $M_3(\gamma)$ before round r_3^*.

Case (b): Note that $r_3^* < r$ must hold (since otherwise r_2^* would be greater than r, which is not possible by definition). This means that a player receives a message $m' = (\kappa', \gamma, \mu', \sigma', \sigma'') \in M_3(\gamma)$ in round r_3^*. If a player receives m' in round r_3^*, then a player must send m' in round r_3^*. According to the logical formulae with which we tagged the messages in M_3, A can send m' in round r_3^* only if she receives $(\gamma, \mu', \sigma') \in M_2(\gamma)$ or a message in $M_3(\gamma)$ that contains (γ, μ', σ') in an earlier round $\hat{r} < r_3^* < r_2^*$. Furthermore, B can send m' in round r_3^* only if she receives $m' \in M_3(\gamma)$ in an earlier round $\tilde{r} < r_3^*$. However, because of the definitions of r_2^* and r_3^*, no player can receive a message in $M_2(\gamma)$ before round r_2^*, and a message in $M_3(\gamma)$ before round r_3^*. □

Appendix D

Summary of notations

Logic of channels

C	channel
$C(X)$	message X in channel C
$r(C)$	reader set of channel C
$s(C)$	source set of channel C
$w(C)$	writer set of channel C
K	session key
P, Q	principals
X, Y	messages
ϕ, ψ	logical formulae
;	concatenation ($X;Y$ is a message composed of sub-messages X and Y)
$\sim$	'session key' operator ($K \sim \{P, Q\}$ means that K is a session key for P and Q)
$\models$	'belief' operator ($P \models \phi$ means that P believes that formula ϕ is true)
$\lhd$	'sees' operator ($P \lhd X$ means that P sees message X)
$\mid\!\sim$	'said' operator ($P \mid\!\sim X$ means that P said message X)
$\mid\!\mid\!\sim$	'recently said' operator ($P \mid\!\mid\!\sim X$ means that P has recently said message X)
$\sharp$	'freshness' operator ($\sharp(X)$ means that message X is fresh)
$\natural$	'timeliness' operator ($\natural(C)$ means that channel C is timely)
$\neg$	negation
$\wedge$	logical 'and' operator
$\vee$	logical 'or' operator
$\Rightarrow$	implication
$\Leftrightarrow$	equivalence

Game theory

a	action
$A(q)$	set of available actions after non-terminal action sequence q
ϵ	empty sequence
G	game
$G_{\mid \bar{s}}$	restricted game (the strategies in the strategy vector $\bar{s}$ are fixed)
i	index variable (usually denotes a player)
I_i	information set of player i
$\mathcal{I}_i$	information partition of player i
j	index variable (usually denotes a player)
k	index variable
o	outcome function
p	player function
P	player set
q	action sequence
$q.a$	action sequence q followed by action a
Q	set of action sequences
s_i	strategy of player i
S_i	set of strategies of player i
v	positive integer
w	positive integer
$y_i(q)$	payoff for player i after terminal action sequence q
Z	set of terminal action sequences
$\preceq_i$	preference relation of player i

Protocol games

$A_i(\Sigma_i(q))$	available actions for player i in local state $\Sigma_i(q)$
$\alpha_i(q)$	activity flag of player i after action sequence q (part of i's local state)
E	set of all events
$\phi_i^m(\Sigma_i(q))$	logical formula that describes the condition that must be satisfied by the local state $\Sigma_i(q)$ of player i in order for i to be able to send message m after action sequence q
$G_\pi(L)$	protocol game of protocol $\pi(L)$
γ_i	player i's item to be exchanged
$H_i(q)$	local event history of player i after action sequence q (part of i's local state)
L	set of parameters of a protocol or a program
m	message
M	set of messages
$M_{net}(q)$	network buffer after action sequence q (part of the local state of net)
$M_\pi(L)$	set of messages compatible with protocol $\pi(L)$
N	set of positive integers
net	network
p_1, p_2	main parties of the protocol
p_3	trusted third party (TTP)
P'	equals $P \setminus \{net\}$
$\pi(L)$	protocol, defined as a description of a distributed computation on a set L of parameters, where L usually contains the identifiers of the executing parties, the items to be exchanged, the description of the items, and cryptographic parameters, such as keys, random numbers, etc.
$\pi_i(L_i)$	program of protocol party i, defined as a description of a local computation on a set L_i of parameters, where L_i contains those parameters that are known to i
$r_i(q)$	round number for player i after action sequence q (part of i's local state)
rcv	receive event
snd	send event
$\Sigma_i(q)$	local state of player i after action sequence q
u_i^+	potential gain of player i (the value that the item of the other player is worth to i)
u_i^-	potential loss of player i (the value that i's own item is worth to i)
$y_i^+(q)$	gain of player i after terminal action sequence q
$y_i^-(q)$	loss of player i after terminal action sequence q

Index

Curriculum Vitae

Name:	Levente Buttyán
Date of birth:	December 3, 1970
Place of birth:	Salgótarján, Hungary
Languages:	English, French, Hungarian

1998 – 2001	Ph.D. Student and Research Assistant Institute for Computer Communications and Applications, Swiss Federal Institute of Technology, Lausanne
1997 – 1998	Research Assistant Telecommunications Laboratory, Swiss Federal Institute of Technology, Lausanne
1995 – 1997	Postgraduate Student Department of Telecommunications, Budapest University of Technology and Economics
1990 – 1995	M.Sc. in Computer Science Faculty of Electrical Engineering and Informatics, Budapest University of Technology and Economics (Diploma with Honor)
1985 – 1989	Certificate of Final Examination Földes Ferenc Gimnázium, Miskolc (Mathematics Section)

Publications

- L. Buttyán and J.-P. Hubaux. Rational Exchange – A Formal Model Based on Game Theory. In *Proceedings of the 2nd International Workshop on Electronic Commerce (WELCOM 2001)*, November 2001.
- L. Buttyán and N. Ben Salem. A Payment Scheme for Broadcast Multimedia Streams. In *Proceedings of the 6th IEEE Symposium on Computers and Communications*, July 2001.
- L. Buttyán and J.-P. Hubaux. Enforcing Service Availability in Mobile Ad Hoc WANs. In *Proceedings of the IEEE/ACM Workshop on Mobile Ad Hoc Networking and Computing (MobiHOC 2000)*, August 2000.
- L. Buttyán, C. Gbaguidi, S. Staamann, and U. Wilhelm. Extensions to an Authentication Technique Proposed for the Global Mobility Network. *IEEE Transactions on Communications*, 48(3), March 2000.
- L. Buttyán. Removing the Financial Incentive to Cheat in Micropayment Schemes. *IEE Electronics Letters*, 36(2), January 2000.
- L. Buttyán and J.-P. Hubaux. Accountable Anonymous Access to Services in Mobile Communication Systems. In *Proceedings of the 18th IEEE Symposium on Reliable Distributed Systems (SRDS'99)*, 1st International Workshop on Electronic Commerce (WELCOM'99), October 1999.
- L. Buttyán, S. Staamann, and U. Wilhelm. Multilateral Security in Middleware Based Telecommunication Architectures. In G. Mueller and K. Rannenberg (eds), *Multilateral Security in Communications*, Volume 3: Technology, Infrastructure, Economy, Addison-Wesley, 1999.

- L. Buttyán, S. Staamann, and U. Wilhelm. A Simple Logic for Authentication Protocol Design. In *Proceedings of the 11th IEEE Computer Security Foundations Workshop*, June 1998.
- L. Buttyán and I. Vajda. Searching for the Best Linear Approximation of DES-like Cryptosystems. *IEE Electronics Letters*, 31(11), May 1995.

Co-authored publications

- J.-P. Hubaux, L. Buttyán, and S. Čapkun. The Quest for Security in Mobile Ad Hoc Networks. In *Proceedings of the ACM Symposium on Mobile Ad Hoc Networking and Computing (MobiHOC 2001)*, October 2001.
- L. Blažević, L. Buttyán, S. Čapkun, S. Giordano, J.-P. Hubaux, and J.-Y. Le Boudec. Self-Organization in Mobile Ad Hoc Networks: the Approach of Terminodes. *IEEE Communications Magazine*, 39(6), June 2001.
- U. Wilhelm, S. Staamann, and L. Buttyán. A Pessimistic Approach to Trust in Mobile Agent Platforms. *IEEE Internet Computing*, September-October 2000.
- J.-P. Hubaux, J.-Y. Le Boudec, S. Giordano, M. Hamdi, L. Blažević, L. Buttyán, and M. Vojnović. Towards Mobile Ad-hoc WANs: Terminodes. In *Proceedings of the IEEE Wireless Communications and Networking Conference*, September 2000.
- S. Staamann, L. Buttyán, A. Coignet, E. Ruggiano, U. Wilhelm, and M. Zweiacker. Closed User Groups in Internet Service Centers. In *Proceedings of Distributed Applications and Interoperable Systems (DAIS'99)*, June-July 1999.
- U. Wilhelm, S. Staamann, and L. Buttyán. Introducing Trusted Third Parties to the Mobile Agent Paradigm. In J. Vitek and C. Jensen (eds), *Secure Internet Programming: Security Issues for Mobile and Distributed Objects*, Lecture Notes in Computer Science, Springer-Verlag, 1999.
- S. Staamann, L. Buttyán, and U. Wilhelm. Security in TINA. In *Proceedings of IFIP-SEC'98*, August-September 1998.
- U. Wilhelm, S. Staamann, and L. Buttyán. Protecting the Itinerary of Mobile Agents. In *Proceedings of the 12th European Conference on Object-Oriented Programming*, Workshop on Mobile Object Systems: Secure Internet Mobile Computations, July 1998.
- U. Wilhelm, S. Staamann, and L. Buttyán. On the Problem of Trust in Mobile Agent Systems. In *Proceedings of the IEEE Network and Distributed Systems Security Symposium (NDSS'98)*, March 1998.
- S. Staamann, L. Buttyán, J.-P. Hubaux, A. Schiper, and U. Wilhelm. Security in the Telecommunication Information Networking Architecture - the CrySTINA Approach. In *Proceedings of the TINA'97 Conference*, November 1997.
- I. Vajda and L. Buttyán. On Design Criteria of Conventional Block Ciphers. *Communications*, March 1995. (in Hungarian)