Université Libre de Bruxelles
Faculté des Sciences

Formal Analysis of Optimistic Fair Exchange Protocols

Thèse présentée en vue de l'obtention
du grade de Docteur en Sciences

Steve KREMER
Année académique 2003-2004

Acknowledgements

My foremost thanks go to Yves Roggeman, director of this thesis. He welcomed me in his research unit and always provided the needed support. He invited me each year to attend the Eurocrypt conferences and I wish to thank him for all the pleasant moments we had there.

I am very grateful to Jean-François Raskin, co-director of this thesis, for the endless discussions in his office (which generally—although not always—finished with me agreeing that he was right from the beginning), for his precious advice and also for all the not-work-related moments.

I also wish to thank the other members of the jury.

Hubert Comon agreed to serve on my jury, in spite of his crowded time schedule.

Raymond Devillers spent much time carefully reading the manuscript. His comments and ideas significantly improved my thesis.

Andre Scedrov accepted to be a member of my jury. He hosted a two-month visit of mine at the University of Pennsylvania. I am very thankful for his cordial welcome and the interesting discussions we had.

Many thanks go also to my other collaborators.

I thank Olivier Markowitch, for the pleasant atmosphere he created (unless early Monday morning). I spent many hours in his office and the discussions were not always work related ... Alexandre Boisseau came several times to Brussels and invited me to the ENS de Cachan. I wish to thank him for his enthusiasm, for introducing me to the topic of abstract interpretation and for all the joyful moments spent at the "Atelier". I had the pleasure to work with Rohit Chadha during my visit at the University of Pennsylvania. Moreover, he made me discover the delicious Jamaican cuisine. I also want to thank Dieter Gollmann and Jianying Zhou, although I never met him personally, for the collaborations we had.

I am grateful to all the members of the "Département d'Informatique" who spiced up my work days during these last four years. Special thanks go to the group of people who attend our traditional tea break and to Jean Cardinal who shared my office during two years (without complaining). He always encouraged me and significantly improved my juggling skills.

A very special thank you goes to Marie for encouraging me, reading most parts of this thesis and for all the travelling she has endured to spent some time with me.

Finally, I wish to thank my parents and my brother for all the love they gave me.

Contents

List of Figures

List of Protocols

Notations

Entities

- A, B, C: Alice, Bob and Charlie.
- I: the intruder, also referred to as enemy or spy.
- TTP: the trusted third party, sometimes referred to as T.

Message sendings in protocols

- $X \rightarrow Y : m$: entity X sends message m to Y.
- $X \leftrightarrow Y : m$: entity X fetches message m at Y, for instance using an FTP get operation on a public directory containing m.
- $X \Rightarrow \mathcal{P} : m$: entity X broadcasts message m to the set of entities $\mathcal{P}$.

Cryptographic primitives and keys

- m_1, m_2: the concatenation of messages m_1 and m_2.
- $pk(X)$: the public key of entity X.
- $sk(X)$: the secret key of entity X.
- $pk(X)^{-1}{=}sk(X)$; $sk(X)^{-1}{=}pk(X)$; $k^{-1} = k$ (when k is a symmetric key)
- $E_k(m)$: the encryption of message m with key k. Depending on the key, we either use symmetric key or public-key encryption.
- $D_{k'}(c)$: the decryption of ciphertext c using key k'. Depending on the key, we either use symmetric key or public-key encryption. In any case we have that $D_{k'}(E_k(m)) = m$ if and only if $k^{-1} = k'$.
- $\mathcal{E}_{\{k_1,\ldots,k_n\}}(m)$: the public-key group encryption of message m with the set of public keys $\{k_1, \ldots, k_n\}$.
- $\mathcal{D}_k(c)$: the decryption of the group encryption ciphertext c using key k. In any case we have that $\mathcal{D}_k(\mathcal{E}_{\{k_1,\ldots,k_n\}}(m)) = m$ if and only if $k \in \{k_1^{-1}, \ldots, k_n^{-1}\}$.

- $E_X(m)$: a short-cut notation for $E_{pk(X)}(m)$.
- $D_X(c)$: a short-cut notation for $D_{sk(X)}(c)$.
- $\mathcal{E}_{\mathcal{P}}(m)$: a short-cut notation for $E_{\{pk(X_1),\ldots,pk(X_n)\}}(m)$, where $\mathcal{P} = \{X_i \mid i = 1 \ldots n\}$.
- $S_X(m)$: signature of entity X on message m. We suppose that all signatures are signatures with message recovery.
- $h(m)$: the cryptographic hashing of message m; the hashing is assumed to be one-way and collision resistant.
- $PCS_X(m, Y, Z)$: the private contract signature of entity X on message m, destined to entity Y and convertible by entity Z.

Details about these cryptographic primitives can be found in standard textbooks on cryptography, such as [110, 143]. Private contract signatures and group encryption schemes are more esoteric cryptographic primitives and not explained in these references. Private contract signatures are defined in [67] and will also be defined in this thesis. An example of a group encryption scheme is given in [48].

CHAPTER *1*

Introduction

Security of information has become of increasing importance in today's information society. Sensitive information such as credit card numbers is sent over the Internet every day, personal medical data are stored in large databases and e-commerce, either business to business, or business to consumer, has become very popular. Moreover, the European Parliament and Council published a directive [1] in December 1999 which aims to give a legal value to digital signatures and expresses their willingness to promote international electronic commerce and electronic procedures requiring authentication in general. Therefore, we believe that traditional pen and paper procedures, ranging from tax declarations to managing your bank account or even business negotiations such as contract signing, are going to be replaced more and more by their corresponding electronic procedures. These electronic procedures may have several advantages over the corresponding traditional ones. As an example, consider electronic voting over the Internet. The counting of the poll would be automated, faster and less error-prone. Moreover, the voters do not need to be physically present and can vote from anywhere with Internet access.

Unfortunately, the possibilities offered by interconnected computer networks are also subject to new security risks. Credit card numbers can be intercepted, systems can get hacked, private information stolen or modified and electronic votes manipulated if not protected in an adequate way. However, in our eyes the question *"Is this system*[1] *secure?"* does not make sense, because it raises two other ones. The first one is *"What does it mean for the system to be secure, i.e. what is the system supposed to achieve?"* and the second one is *"Secure against whom or what?"*. To answer the first question, one has to specify the *properties* the system has to ensure. The second question emphasises the need for a *model* specifying the context, i.e. how the different components of a system interact with each other and with their environment and what are the threats we consider. Considering all threats is, on one hand, impossible since the model would be too complicated and simply because we are not aware of all of them and, on the other hand, not always desirable since

[1] Here, we use the general term *system*. As we will see the systems we are going to consider are *security protocols*.

the security measures could make the system impractical or too expensive to set up, with respect to the requested service. Therefore, the model must specify what are the potential *attacks*, or kinds of attack. Moreover, having a model with a well defined semantics enables us to formally define the desired properties with respect to the underlying model.

Information security in general is too vast a field and could not be covered in one thesis. Therefore, we limit ourselves to the security of network communications. We start by introducing some classical security properties. Then, we discuss the notion of security protocols which aim at satisfying those properties. However, before we can reason about the protocols and the expected properties, we first need to define a model to show how the expected properties can be defined in that model and how we can prove whether they are respected or not. This is where *formal methods* come into play. We conclude this introduction by stating the aims of this thesis and outlining its structure.

1.1 Security properties

In this section, we give a short overview of some of the properties that could be desirable when communicating over a network, and the potential risks which exist when these properties are not satisfied. Not all of these properties are always requested. Nor do we cover all of the possibly requested properties and corresponding security threats.

Secrecy. Secrecy, also referred to as confidentiality, is probably the most classical security property. Historically, this was the first and most important one, and sometimes people confuse security and confidentiality. Confidentiality makes sure that an unauthorised person is unable to read sensitive information.

If a plain message is sent over an open network, it is easy to read the message using tools such as network *sniffers*. Examples of sensitive data which one would like to keep secret include credit card numbers in electronic transactions, medical information stored in a distributed database, strategic decisions of a corporation sent by e-mail, etc.

Authentication. Authentication can be divided in two more specific properties: *data* authentication and *entity* authentication.

Data authentication should ensure that a given message comes from the person we think it comes from. For instance, when a program is downloaded from the Internet, we would like to be sure that the program comes from a given, trusted person and has not been replaced by a virus.

While data authentication is not time dependent, entity authentication adds the notion of *freshness*. Entity authentication, or identification, ensures not only that the received data has originated from the expected entity, but also that we are currently communicating with the person, i.e. that it is not an old message that has been resent. For instance, when a bank transfer is ordered electronically, data authentication is not sufficient. One has to be sure that it is not a previous transfer which has been sent a second time.

Non-repudiation. Non-repudiation is a slightly stronger notion than data authentication. Data authentication convinces one given person that the data originated from where we expected. Non-repudiation convinces *everyone*—and not only one specific person—that a given entity is at the origin of a message. In particular, this means that the person cannot repudiate, or deny, being the author of the given message. Imagine that someone buys a faulty piece of software via the Internet. If non-repudiation on the message is provided, the author of the software cannot deny the errors or claim that they were added by some other entity. If data authentication was provided alone, the client would be convinced that the software was emitted in its faulty state, but he could not convince a judge about the origin of the data.

Anonymity. Anonymity is in some sense the opposite to non-repudiation or authentication. It guarantees that the recipient of a message, or any other observer cannot identify the author of this message. A direct application of anonymity is electronic voting, i.e. it must not be possible to link a vote to the corresponding voter.

Fairness. Fairness is a property that intervenes in transactions between entities that do not trust each other. As an example, consider two business partners who want to sign a contract over a network. A fundamental problem is that one of the entities must be the first to send out their signature on the contract. Once their signature is received, the other entity could refuse to send their own signature on this contract. This results in an undesirable situation, which is said to be *unfair*. In the remaining of this thesis, the fairness property will be discussed in more detail.

1.2 Cryptography and security protocols

The goal of cryptography is to provide the "tools" to ensure the properties presented above and to offer a solution to the corresponding security threats. We suppose that the reader has at least a conceptual knowledge of cryptography.

Secrecy, data authentication and non-repudiation can be provided directly by basic cryptographic primitives such as encryption, keyed hash functions or digital signatures. However, when these primitives are used to secure a network communication, they require a setup phase, such as a secret key exchange or the distribution of public key certificates[2]. Such a setup, as well as the more elaborate properties, e.g. entity authentication, require an interaction between the entities involved in the communication. The different steps of these interactions are dictated by a *protocol*. According to the Oxford Dictionary [129], a protocol is defined as

> *the accepted code of procedure or behaviour in a particular situation*

or in a technical context as

> *a set of rules governing the exchange or transmission of data between devices.*

When a protocol makes use of cryptographic tools, we refer to it as a *security protocol*, or as a *cryptographic protocol*.

In a security protocol, cryptographic primitives are used as building blocks. The idea of using *protocols* rather than merely primitives was first presented by Needham

[2]A public key certificate ensures the link between an entity and its public key.

and Schroeder [119] in 1978. In [119], they define protocols that aim to achieve entity authentication by means of encryption. Needham and Schroeder were already aware of the difficulty of designing correct security protocols and conclude their paper with the following paragraph.

> *Finally, protocols such as those developed here are prone to extremely subtle errors that are unlikely to be detected in normal operation. The need for techniques to verify the correctness of such protocols is great, and we encourage those interested in such problems to consider this area.*

1.3 Formal methods

While errors in computer software are never desirable, there exist critical systems where failures are unacceptable. A well known example of a software failure with disastrous consequences is the bug in one of the software modules of the Ariane V rocket which led to its explosion in June 1996.

Such an example illustrates that even intensive testing and simulation may not be sufficient, as it is difficult to cover all cases and pitfalls. Developing *formal methods* for exhaustive verification and mathematical proofs is of crucial importance and an ongoing research field. The use of formal methods can be divided into three steps.

Modelling. The first and probably most delicate phase is to convert the system we want to verify into a mathematical model. This generally requires us to abstract away (hopefully) unimportant details, in order to achieve a concise model which is sufficiently simple to reason about.

Specification. The specification involves stating which properties must hold on the given model. Temporal logics are a popular formalism for describing the desired properties.

Validation. Finally, one must prove that a specification holds on a given model. These proofs can sometimes be completely automated using *model checking*. Unfortunately, the theory of decidability puts limitations on this automation. Sometimes, human intervention is necessary. In most of these cases the proofs can however be computer-assisted, using for instance theorem proving techniques.

In recent years, formal methods have become more mature, being able to handle large and complex systems, and starting to be integrated in industrial development processes. As an example, the Intel corporation completely integrated formal methods in their design process, after the famous bug in the Pentium processor in 1994.

It was recognised rather early that security protocols were also error-prone critical systems, or to quote Needham,

> *Security protocols are three-line programs that people still manage to get wrong.*

Hence, the idea of applying formal methods to security protocols was pursued since the early 1980s.

1.4 Aims and structure of the thesis

The goal of this thesis is to formally analyse an esoteric family of security protocols called optimistic fair exchange protocols. As these protocols were presented during the second half of the 1990s, only a few attempts of applying formal analysis techniques exist.

The need for formal methods in the field of security protocols in general has been recognised for a long time and many techniques have been developed. We are going to show that fair exchange protocols in general, and optimistic fair exchange protocols in particular, have some noticeable differences compared to more classical security protocols. These differences are present at the level of the structure of the protocol, the communication media, the threats we have to consider and the properties that must be provided. Therefore, we propose a new technique which we believe is particularly adequate for this kind of protocols. Our analysis is based on the idea that an optimistic fair exchange protocol is best represented as a game between the involved entities. In this framework, we have a natural specification of the properties in terms of strategies. Formalising the properties will allow us to identify relations that exist between several of these properties. We also define a protocol specification language and automate the validation of the properties using model checking techniques. Finally, we apply our methods to several two-party, as well as multi-party, protocols. We demonstrate the ability of our techniques to find known and new, unknown attacks. Moreover, our analysis provides a better understanding of the properties that should be provided and permits us to highlight ambiguities in the definition of a particular property, called abuse-freeness.

More specifically, the thesis is structured as follows.

Chapter 2. In chapter 2, we give a representative sampling of formal techniques that have been applied to security protocols aiming at secrecy or authentication. Our survey is not intended to be complete but to illustrate the kind of approaches that have been used.

Chapter 3. This chapter is divided into four parts.

First, we introduce fair exchange protocols and discuss several instances of fair exchange protocols, such as certified e-mail, non-repudiation or contract signing protocols.

The second part deals with the properties these protocols should provide. We give a critical survey of the properties presented previously in the literature, emphasising that most definitions, while not really wrong, are either too specific or not sufficiently modular, grouping several different properties in the same definition. Finally, we give an informal but general and modular definition of the properties which a fair exchange protocol should provide. This part of the chapter is based on joint work with Dieter Gollmann and Olivier Markowitch [97].

The third part of the chapter provides a historical review of fair exchange protocols. We discuss early solutions, having several drawbacks, and show their evolution. During the last four years, in collaboration with Olivier Markowitch, we have actively contributed to the field through several publications [83, 84, 85, 86, 98, 99]. This part of the chapter is mainly based on the review paper, realised in collaboration with Olivier Markowitch and Jianying Zhou [87].

The last part, concluding the chapter, emphasises the differences between optimistic fair exchange protocols and more classical protocols. These differences motivate the deployment of new techniques for formal analysis.

Chapter 4. In this chapter, we present the underlying formal model we are going to use: *Alternating Transition Systems*, ATS for short. We also present the *Alternating-time Temporal Logic*, ATL, which will serve to formalise the security properties in the following. We also present a guarded command language, allowing us to describe an ATS in a convenient way. Finally, we report on our implementation of the model-checking which needs to be adapted for our purposes.

Chapter 5. In chapter 5, we define how protocols are formalised and how properties can be expressed in our model.

We start by giving the syntax of the protocol description language and then give the semantics in terms of ATS. We define the different properties an optimistic fair exchange protocol should provide in ATL. Formalising the properties allows us to establish the relation between several properties and provide a better understanding of the latter.

Finally, we briefly report on the implementation of the prototype of a tool which automates the translation from the defined specification language into an ATS and model-checks the properties on this ATS.

The definition of the specification language and the implementation of the tool are preliminary results of an ongoing collaboration with Alexandre Boisseau. The specification of the properties is based on joint work with Jean-François Raskin [88, 89, 90].

Chapter 6. In this chapter, we apply the previously defined techniques on several two-party protocols. For each protocol, we give a detailed study of the different properties that should be provided. We illustrate the power of the tool for finding a large variety of flaws, related to different kinds of problems and properties.

We also show how our analysis can increase the understanding of these new properties, by highlighting an ambiguity in the definition of *abuse-freeness* and present a new and stronger definition, which we believe to better fit the intuition of the property and which overcomes the presented ambiguity.

Most of these results were first obtained in collaboration with Jean-François Raskin [88, 89, 90]. The results can now be obtained in a more automated way, using the tool developed in collaboration with Alexandre Boisseau.

Chapter 7. In chapter 7, we analyse three multi-party fair exchange protocols. The first analysed protocol is a multi-party non-repudiation protocol which we designed in collaboration with Olivier Markowitch [86, 98]. We show how a slight deviation from the protocol could lead to a subtle error. The analysis is based on a joint work with Jean-François Raskin [90].

The two other protocols are multi-party contract signing protocols [68, 22]. We report on preliminary results of the analysis carried out in collaboration with Rohit Chadha and

Andre Scedrov. We demonstrate that the first contract signing protocol is neither abuse-free nor fair. While the second protocol seems to be fair, an implementation aiming at efficiency could break abuse-freeness.

The analysis of multi-party protocols requires entering the protocol model "by hand" since the developed tool does not support multi-party protocols yet. Moreover, the protocol model has to be simplified for model-checking to be feasible. However, we show that even with this simplified model, we are able to find several new attacks, on a protocol which was believed to be secure.

Chapter 8. In this chapter we describe several other approaches which have been used for the analysis of fair exchange protocols, including non-optimistic solutions, and compare them with our approach.

Chapter 9. Finally, we conclude our thesis and present directions for future works.

CHAPTER 2

Formal methods and security

2.1 Introduction

The aim of this chapter is to draw a "rough picture" of the state of the art of formal methods, used for the analysis and verification of security protocols, specifically protocols aiming secrecy and authentication. We will address formal methods used to analyse fair exchange and related protocols separately in chapter 8. This chapter is far from being exhaustive, nor does it go into all technical details of the different techniques, sometimes only giving informal statements. Moreover, we restrict ourselves to formal methods related to what is commonly known as the Dolev-Yao approach [58] and do not consider the computational model, although recent work by Abadi and Rogaway has emphasised connections between these two approaches [4]. Other topics which we are not going to discuss in detail are the issues concerning decidability and complexity. A proof of undecidability for deciding the correctness of a security protocol in the general case is given in [59] by Durgin et al. as well as the complexity of several decidable sub-classes. Deciding whether a protocol is correct with a finite number of parallel sessions has been shown to be NP-complete by Turuani and Rusinowitch [163]. The interested reader can find a more detailed discussion of these issues in Comon and Shmatikov's review paper [51].

To illustrate authentication protocols and to show the kind of attacks we consider, we introduce the Needham-Schroeder public key protocol [119] as an example. This protocol could be considered as the "benchmark protocol". It was believed correct during 17 years before Lowe [91] found an attack on the protocol, which gave this protocol its fame. The protocol aims mutual authentication between Alice and Bob. Basically, authentication should assure Alice and Bob that they are talking to each other (and not to an attacker, which is also called intruder, enemy or spy). This is only an informal statement of what authentication tries to achieve. Many different variants of authentication have been discussed in [72] and [94], but defining them here would be beyond the scope of this chapter.

Protocol 2.1 Needham-Schroeder public-key protocol

1. A $\rightarrow$ B: $E_B(N_A, A)$
2. B $\rightarrow$ A: $E_A(N_A, N_B)$
3. A $\rightarrow$ B: $E_B(N_B)$

The protocol[1] is depicted in protocol 2.1. In the first message, Alice sends to Bob a nonce N_A and her identity, ciphered with Bob's public key. To confirm that Bob has received the first message, Bob replies by ciphering Alice's nonce together with a new nonce N_B using Alice's public key. When Alice receives this message, she should be convinced that the recipient of the first message is indeed Bob, as he is the only person who could decrypt the nonce N_A that was freshly chosen by Alice at the beginning of the protocol. In order to convince Bob that Alice is the sender of the first message, Alice sends a copy of the nonce N_B back, ciphered with Bob's public key. Again, the fact that only Alice could have deciphered Bob's freshly chosen nonce should convince Bob that Alice is participating in this protocol. However, in [91], Lowe came up with the following scenario, in which an intruder can launch a *man-in-the-middle* attack. This attack is depicted in protocol 2.2, where we denote by I(A) the fact that the intruder is acting as if he was Alice. The attack can be summarised as follows. Alice starts the protocol with the intruder, unaware of the fact that he is malicious. Then the intruder launches a second protocol run with Bob and reuses Alice's messages to make Bob believe that he is actually talking to Alice. Hence, Bob is flawed in his belief that he is talking to Alice, as he is actually communicating with the intruder. Without reducing the interest of this scenario, the legitimacy of the "attack" is questionable, as Alice must be willing to run a protocol with a malicious user, the intruder. This contradicts the assumptions made by the designers in [119]:

> *We also assume that each principal has a secure environment in which to compute, such as is provided by a personal computer or would be by a secure shared operating system. Our viewpoint throughout is to provide authentication services to principals that choose to communicate securely. We have not considered the extra problems encountered when trying to force all communication to be performed in a secure fashion or when trying to prevent communication between particular principals in order to enforce restrictions on information flow.*

Note that while these assumptions were rightful in 1978, they are unrealistic nowadays (think for instance of communications through the Internet). In the following, we will refer to Lowe's scenario, as Lowe's attack, even if it is not an attack strictly speaking. Lowe also proposes a fix to the protocol by simply adding Bob's identity to the second message, which becomes

$$2.\ \mathsf{B} \rightarrow \mathsf{A} : E_A(N_A, N_B, B).$$

[1] We here describe the generally used short version of the protocol in which all public keys are assumed common knowledge.

Protocol 2.2 Lowe's attack on the Needham-Schroeder public-key protocol

1. A → I: $E_I(N_A, A)$

1'. I(A) → B: $E_B(N_A, A)$

2'. B → I(A): $E_A(N_A, N_B)$

2. I → A: $E_A(N_A, N_B)$

3. A → I: $E_I(N_B)$

3'. I(A) → B: $E_B(N_B)$

The techniques presented below have mainly been chosen because of their historic importance or because they have been adapted to analyse fair exchange protocols and will be rediscussed in chapter 8. There exist several criteria which would allow a classification of the different methods. Examples of such classification criteria include whether the methods are based on general purpose or special purpose techniques, whether they are fully, partially or not at all automated, whether protocols are described using process algebras or not, whether the techniques limit themselves to a bounded number of protocol runs or not, whether the analysis, if automated, is guaranteed to terminate or not. As we think that there is not a single classification criteria which would be particularly adapted to this chapter, we prefer not to give a unique classification.

2.2 Design principles

Although design principles cannot be classified as *formal methods*, we briefly discuss some of them here. Design principles have been proposed in 1995 by Abadi and Needham [3] as well as by Anderson and Needham [9]. The authors show many examples of flaws that could have been prevented if these *thumb rules* had been applied during the protocol design. They are neither necessary nor sufficient, but every protocol designer should care about them as they give an idea of what kind of things "could go wrong". The main principle is *explicitness*. It should always be clear what a message means, what given cryptographic primitives are used for, etc. We will cite some examples here.

> *Every message should say what it means: the interpretation of the message should depend only on its content.[...]* (Principle 1 in [3])

In particular, there should be no ambiguity on the context within which the message is used. This would avoid many attacks where messages are reused out of context.

> *If the identity of a principal is essential to the meaning of a message, it is prudent to mention the principal's name explicitly in the message.* (Principle 3 in [3])

Typical examples where this principle applies are authentication protocols, where the name of the entities that want to authenticate each other should be a crucial information of each message. Respecting this principle would have avoided Lowe's attack on the Needham-Schroeder public-key protocol.

> *Be clear about why encryption is being done.[...]* (Principle 4 in [3])

Indeed, encryption can be used for many purposes. The most straightforward use is confidentiality, but encryption could also be used to provide authentication in a challenge-response scenario, or to bind several informations together.

> *Sign before encrypting. [...]* (Principle 1 in [9], similar to Principle 5 in [3])

The idea is that when we want to combine authentication, or even non-repudiation, with confidentiality, the order of the cryptographic operations matters. If a ciphertext is signed, one cannot assume that the signer is aware of the message that has been encrypted. Hence non-repudiation on the *message* may be lost. However, this is a good example where the principles may be too strict. There exist contexts where we want to have the non-repudiation on the ciphertext. Ironically, this technique is used in many non-repudiation protocols. Another context is anonymity: blind signatures [45] are an example where you have to sign a message you cannot read. In that case signing the ciphertext is the way to proceed, but one must be aware of what the order of signing and encryption implies.

> *Be careful when signing or decrypting data that you never let yourself be used as an oracle by your opponent.* (Principle 3 in [9])

The previously described attack on the Needham-Schroeder public-key protocol provides an example where Alice is used as a decryption oracle for Bob's nonce N_B. We will revisit this principle when studying the strand space model in section 2.10. In strand spaces this principle will reappear under the name *unintended services*.

> *Robust security is about explicitness; one must be explicit about any properties which can be used to attack a public key primitive, such as multiplicative homomorphism, as well as the usual security properties such as naming, typing, freshness, the starting assumptions and what one is trying to achieve.* (Principle 8 in [9])

This is probably one of the most important issues when studying security protocols. It is crucial to emphasise all the assumptions one makes, and leave implicit as less as possible assumptions. This applies to the properties we want to achieve, as well as to the underlying cryptographic primitives or the power of an attacker. Especially when using formal methods, proving a protocol to be correct only means that it is correct in a given model, with a certain number of assumptions.

Although we only emphasise some of the principles stated in [3] and [9], they should all be considered when designing a protocol.

2.3 The Dolev-Yao attacker

In this section, we do not explain a technique but the attacker model that has been defined in [58]. This model has inspired, with sometimes small modifications or extensions, most of the research in the field of formal methods and security protocols. The *Dolev-Yao attacker* combines non-deterministic behaviour with perfect cryptography.

Perfect cryptography means that it is impossible to decipher a ciphertext, if the decryption key is unknown, and rejects all possibilities for cryptanalysis. The same applies to signatures, which are impossible to forge, or one-way hash function that cannot be inverted. The model also assumes that public keys are common knowledge. The perfect cryptography assumption is sometimes relaxed by introducing special properties of given systems. Examples include the multiplicative property of the RSA encryption scheme, i.e. $E_k(m_1) \cdot E_k(m_2) = E_k(m_1 \cdot m_2)$, the fact that Diffie-Hellman key agreement is based on the commutativity of exponentiation, i.e. $g^{ab} = g^{ba}$, or that an encryption based on "exclusive or" cancels itself out, i.e. $a \oplus b \oplus b = a$.

Moreover the attacker completely controls the network, i.e. he can intercept every message and introduce any message he knows, or can construct from his knowledge[2], into the network. The attacker is also considered as a regular user and can engage, either as initiator or responder, protocol sessions with other users.

2.4 Believe logics

Believe logics experienced great popularity at the end of the 1980s and beginning of the 1990s: they succeeded in finding many flaws, emphasising unstated but necessary assumptions, as well as unnecessary operations that do not contribute to the strength of a protocol. These logics generally aim to examine what a principal can infer and believe from initial assumptions and received messages, using a certain number of inference rules.

In the present section, we mainly focus on BAN, which received its name from the initials of its inventors, Burrows, Abadi and Needham. There exist several published versions describing the logic, such as [34, 35, 36, 37, 38, 154]. We used [36] and [154], giving an overview of belief logics, as our main references. We start by introducing the notation, inference rules and general methodology. Finally, we discuss the severe critique BAN received and several of its shortcomings, before briefly mentioning some of the BAN extensions, that tried to overcome at least some of the shortcomings.

2.4.1 Notation

The only connective used by the logic is conjunction, denoted by comma. In addition to conjunction, the following constructions are used, where P and Q are protocol participants and X is either a message or a formula.

- $P \models X$: the principal P believes[3] X and acts as though X is true,
- $P \triangleleft X$: the principal P sees X, i.e. received a transmission containing X, and all it can deduce from X, by using for instance decryption,

[2] In the following sections, which messages can be constructed (and how) will be made clear for each of the presented techniques.

[3] Calling the $\models$ operator "believes" was probably a bad idea, as it is not clear in all situations what it means to believe a message, e.g. believe a nonce.

- $P \mid\!\sim X$: P once said X, i.e. the principal P has sent a message containing X in the past,
- $P \Rrightarrow X$: the principal P has jurisdiction over X, e.g. P could be a key server trusted to issue good keys,
- $\#(X)$: X is fresh, i.e. X has not been sent in any message before the current protocol run,
- $P \overset{K}{\leftrightarrow} Q$: K is a shared secret key that may be used by P and Q for communication,
- $\overset{K}{\longmapsto} P$: P has K as public key,
- $P \overset{X}{\rightleftharpoons} Q$: X is a secret known only to P and Q and possibly to some trusted server,
- $E_K(X)$: the encryption of X under key K,
- $\langle X \rangle_Y$: the combination of X and Y, where Y is a secret that identifies the sender of $\langle X \rangle_Y$.

2.4.2 Inference rules

In this section, we present the inference rules that are going to be used to derive the goals a protocol achieves. The list of rules is not exhaustive and we consciously omit some rules which are symmetric with respect to some of the rules below.

Message-meaning rules

$$\frac{P \models Q \overset{K}{\leftrightarrow} P,\ P \triangleleft E_K(X)}{P \models Q \mid\!\sim X} \qquad \frac{P \models\ \overset{K}{\longmapsto} Q,\ P \triangleleft E_{K^{-1}}(X)}{P \models Q \mid\!\sim X} \qquad \frac{P \models Q \overset{Y}{\rightleftharpoons} P,\ P \triangleleft \langle X \rangle_Y}{P \models Q \mid\!\sim X}$$

When we see X ciphered with a key we believe shared with Q, signed by Q[4] or combined with a secret we believe shared with Q, then we believe that Q once said X. In the first rule one could argue that P, who knows key K, could have said the message X himself, rather than Q. However, the authors of BAN suppose that an entity always recognises his own messages as such.

Nonce verification rule

$$\frac{P \models \#X,\ P \models Q \mid\!\sim X}{P \models Q \models X}$$

The idea of the above rule is that whenever P believes that Q said X, which P believes to be fresh, then P believes that Q said X recently, i.e. in the current session. Hence, P believes that Q believes X (otherwise Q would not have said it).

[4] K^{-1} denotes the private key corresponding to K and encryption with a private key models signature.

Jurisdiction rule

$$\frac{P \models Q \Rightarrow X,\ P \models Q \models X}{P \models X}$$

This rule reflects the fact that P trusts Q regarding X. When P believes that Q believes X, then P also believes X.

Belief and said "conjuncatenation" rules

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

The above rules have straightforward interpretation. However one should be careful: the above rules do not say that when P believes that Q said X and P believes that Q said Y, then P would believe that Q said (X,Y). Q saying (X,Y) would mean that both X and Y were said in the same message and P has no reason to believe that.

Freshness "conjuncatenation" rule

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

The freshness rule says that each message containing a fresh element is fresh. However, a fresh message does not imply that each of its elements are fresh.

Seeing rules

$$\frac{P \lhd (X,Y)}{P \lhd X} \quad \frac{P \lhd \langle X \rangle_Y}{P \lhd X} \quad \frac{P \models Q \overset{K}{\leftrightarrow} P,\ P \lhd E_K(X)}{P \lhd X}$$

$$\frac{P \models\ \overset{K}{\longmapsto} P,\ P \lhd E_K(X)}{P \lhd X} \quad \frac{P \models\ \overset{K}{\longmapsto} Q,\ P \lhd E_{K^{-1}}(X)}{P \lhd X}$$

The above rules state what an entity can extract from a message, when receiving it.

Symmetry Rules

$$\frac{P \models R \overset{K}{\leftrightarrow} R'}{P \models R' \overset{K}{\leftrightarrow} R} \quad \frac{P \models Q \models R \overset{K}{\leftrightarrow} R'}{P \models Q \models R' \overset{K}{\leftrightarrow} R} \quad \frac{P \models R \overset{X}{\rightleftharpoons} R'}{P \models R' \overset{X}{\rightleftharpoons} R} \quad \frac{P \models Q \models R \overset{X}{\rightleftharpoons} R'}{P \models Q \models R' \overset{X}{\rightleftharpoons} R}$$

The symmetry rules express the natural assumption that whenever a key or a secret is shared between P and Q, it is also shared between Q and P.

2.4.3 Analysis methodology

The BAN methodology to analyse a protocol can be summarised in four steps.

1. Idealise the protocol.

2. Write assumptions about the initial state.

3. Annotate the protocol.

4. Use the logic to express the beliefs held by protocol principals.

The idealisation phase is probably the most difficult step to realize. It consists in writing the protocol in BAN notation, and sometimes encoding what a given message was intended to realize. This is generally an error-prone step that is inherently informal and has been criticised a lot. Writing the assumptions is generally straightforward. They include that principals know public keys, believe that the nonces they generate are fresh and trust servers to create fresh keys. Annotating the protocol is trivial: for each message $P \to Q : M$ of the idealised protocol, assert $Q \triangleleft M$. The annotation phase is comparable to program proofs in Hoare logic. Finally the last step consists in applying the inference rules to the protocol in order to derive the desired goals. In BAN the goals of the protocol are expressed in the same notation as the protocol itself. In a protocol where Alice and Bob want to establish a key for secure communication, authentication goals are generally of the form

$$\begin{aligned} &A \models A \overset{K_{AB}}{\leftrightarrow} B \\ &B \models A \overset{K_{AB}}{\leftrightarrow} B \\ &A \models B \models A \overset{K_{AB}}{\leftrightarrow} B \\ &B \models A \models A \overset{K_{AB}}{\leftrightarrow} B \end{aligned}$$

While the first two formulae state that A and B believe that they share a secret key with the other entity, the third and fourth formula moreover state that A, respectively B are aware of the fact that the other entity is also convinced that that K_{AB} is a shared secret key. There exist of course more than one form of authentication, and expressing the desired goal depends on the underlying protocol.

2.4.4 Critique on BAN, shortcomings and successors

2.4.4.1 Nessett's critique

In 1990, Nessett [120] introduced a simple two message protocol to "demonstrate" that the BAN logic is flawed. The protocol is depicted in protocol 2.3.

Protocol 2.3 Nessett protocol

1. A → B: $S_A(N_A, K_{AB})$

2. B → A: $E_{K_{AB}}(N_B)$

Alice starts by sending a signed message containing a nonce[5] N_A and a key K_{AB} that should serve as a new session key. Bob replies by sending the encryption of a new nonce N_B as a handshake. Obviously, K_{AB} is not a good key, as it has only been signed and not encrypted in the first message.

The idealised version of the protocol is then as follows.

[5]This nonce should rather be seen as a timestamp, since Nessett assumes $B \models \#N_A$ when stating the protocol's initial assumptions.

1. $A \rightarrow B : E_{K_A^{-1}}(N_A, A \overset{K_{AB}}{\leftrightarrow} B)$

2. $B \rightarrow A : E_{K_{AB}}(A \overset{K_{AB}}{\leftrightarrow} B)$

Note that this protocol is structurally identical to a protocol where the first message would have been encrypted rather than signed, as signature is seen as encryption using one's own private key.

Nessett then succeeds in derivating from the protocol that

$$A \models B \models A \overset{K_{AB}}{\leftrightarrow} B$$

which means that Alice considers K_{AB} as a good key, which it is not. Therefore, Nessett concludes that BAN is flawed. However, Burrows et al. explicitly state that they do not attempt to detect unauthorised release of secrets:

> *Furthermore, while we allow the possibility for hostile intruder, there is no attempt to deal with the authentication of an untrustworthy principal, nor to detect weaknesses of encryption or unauthorised release of secrets.* [36]

Hence, Nessett's initial assumption $A \models A \overset{K_{AB}}{\leftrightarrow} B$, necessary to derive his results, is absurd, as the protocol dictates Alice to publish the key. Of course no logic protects against absurd assumptions. Nessett's result once more illustrates the necessity of being clearly aware of the underlying assumptions that are made in a given model. On the other hand, Nessett's "flaw" puts forward BAN's shortcoming of being unable to deal with confidentiality.

Another shortcoming of BAN is its impossibility of stating several flavours of authentication goals and detect replay or interleaving attacks inside a same protocol session. This is mainly due to the division of time which is merely *fresh* and *old*. BAN is unable to express the order of two messages in one same session. Due to these shortcomings, the Needham-Schroeder public-key protocol has been "proven correct", although Lowe showed that an interleaving attack exists.

2.4.4.2 BAN successors

It would be difficult to cover all of the BAN successors, so we restrict our discussion to the most famous ones.

The first successor was GNY [73] in 1990, traditionally named after its inventors Gong, Needham and Yahalom. One of its major contributions was that the notion of possession, e.g. possession of a key, can now be expressed directly in the logic. As a consequence, one can prove in the Nessett protocol that an attacker can possess the exchanged key.

In 1991, Abadi and Tuttle presented the AT logic [5]. This logic was much closer to traditional logics and provided a detailed model of computation and soundness result with respect to that model. Moreover, it included also the extended expressiveness being able to reason about possession.

Two years later, in 1993, van Oorschot introduced VO logic [165]. The main motivation of this work was to include the possibility to reason about Diffie-Hellman type agreement protocols, that are the basis of many modern key exchange protocols.

Finally, in 1994, Syverson and van Oorschot [156] unified the additions brought by each of the previous logics in one logic that was sound with respect to a given model of computation. Unsurprisingly, the logic was called SVO. SVO is probably the most mature of the belief logics, but still suffers from the drawback that all these logics only have a very poor expressiveness concerning the authentication goals of a protocol and generally cannot express interleaving attacks, such as Lowe's attack on the Needham-Schroeder public-key protocol.

2.5 NRL protocol analyser

The NRL Protocol analyser, or NPA for short, was first presented in 1991 by Meadows [104]. The NPA is a special-purpose tool written in Prolog, dedicated to the analysis of security protocols. The tool combines an unusual use of finite state exploration with theorem proving, allowing to prune the state space. The tool has experienced many extensions since its first presentation in 1991, allowing further automation and reducing the interaction with the user. Although, the NPA was one of the first tools for protocol analysis, it is still currently used to analyse real life protocols, such as SET [109] or the Internet Key Exchange protocol [107]. In its latest version it is even able to analyse group protocols based on algebraic properties, such as Diffie-Hellman operations [108]. Our description of the NPA in this section is mainly based on [105] and [106].

2.5.1 Protocol description

The NPA is mostly based on a term-rewriting model, such as proposed in [58] by Dolev and Yao, and considers the verification of a security property as a form of word problem. The different steps of the protocol are formalised as conditional rewrite rules resulting in an algebraic system which can be manipulated by the intruder. The *words* produced by this system obey certain reduction rules, e.g. encryption and decryption with corresponding keys cancel out. We now briefly present how the different participants of a protocol are specified. Each entity is modelled as a state machine with a set of transitions. Local states, as well as transitions are indexed by the principal identifier, a session identifier (a same principal may be involved in different protocol sessions), the corresponding protocol step and the principal's local time. The description language of the participants is very technical and we will not go into its details.

2.5.2 NPA analysis techniques

The first thing to do when analysing a protocol is to give a description of the bad states, i.e. the states in which an attack is successful. The analyser then performs a backward search trying to reach the set of initial states, that is implicitly defined by the protocol description. There is no guarantee of termination for the backward search. That is why a number of pruning techniques have been developed. The first one (and probably the most powerful one) is based on formal languages. The user can specify a formal language, specifying an infinite set of states that he knows to be unreachable. The analyser can generate an inductive proof based on the idea that, in order to learn one word of the language, another word of the language must already be known. As an easy example, consider the following situation. We want to know whether the intruder could produce a word, by manipulating the algebraic system corresponding to the protocol, of the form $E_k(X)$, where k is a given key and X is a variable that stands for any word. The system might respond in a first time that in order for the intruder to know $E_k(X)$, he must know a word of the form $E_Y(E_k(X))$, which can only be learned if the intruder can find a word of the form $E_Z(E_Y(E_k(X)))$, ..., resulting into an infinite descent. Therefore it is possible to generate a language F

$$F \to E_k(A)$$
$$F \to E_A(F)$$

where A is the language containing all irreducible words. In its latest form, the analyser not only proves the languages but also generates languages. Moreover, the analyser remembers the unreachability of some states in order to reapply previous results, when states, proved unreachable before, are reencountered.

Another technique for pruning the state space is based on letting the user direct the search. When finding the predecessors of a state in the backward search, the user can decide to stop the automated search and select a subset of the state description. The idea is that the user can feed the analyser with his intuition that a given subset is easy to prove unreachable. Obviously, this treatment is sound with respect to the unreachability proof since, when a subset of the state description is unreachable, the entire state description is unreachable. However, as the subset may be reachable while the entire state description is not, this may lead to false positives, i.e. false attacks. Therefore, attacks found by the analyser must be hand-checked afterwards by the user.

2.5.3 Property specification

As seen above, properties are expressed as bad states. However describing the protocols goals in terms of the protocol specification language as done here is cumbersome and error-prone. Therefore, in 1996, Syverson and Meadows [155] introduced a logic to express the requirements of a protocol, named NRL Protocol Analyser Temporal Requirement Language, or NPATRL for short. The atomic propositions range over the events, such as message sending and receiving, and can be combined using all usual boolean connectors. Moreover the temporal modality *before* has been introduced. The *before* operator enables us to express causality, e.g. whenever a given message is sent, another message was sent before. Having a logic to express the protocol goals simplifies considerably the protocol goal specification.

2.6 CSP

The language of *Communicating Sequential Processes* is a process algebra providing a well defined semantics and a precise way for reasoning about concurrent systems in general. There have been two main approaches based on the idea of modelling security protocols in CSP, reasoning in different ways about properties these protocols are expected to provide. The first approach uses the *Failures Divergences Refinement checker* FDR, a finite model-checker, to verify protocols. This approach was taken by Roscoe [135] and Lowe [92]. The second one was presented by Schneider [140] and establishes proofs, based on *rank* *functions*, which state that if a rank function can be found then the protocol is correct. We now give a brief overview of the usage of CSP for modelling security protocols, discuss FDR and a specialised interface to FDR, which is Casper, as well as the rank function approach. An in depth discussion of all these techniques can be found in Ryan et al.'s book [138].

2.6.1 Protocol modelling

Introduction to CSP. We give here a very restricted overview of CSP, that merely presents the elements needed for the purpose of this chapter. In CSP, processes are defined by the events they can generate. We consider events of the form $c.i.j.m$ with c being a channel, i being the source, j the destination and m a message to be transmitted.

Processes are described by the events that they are willing to engage in. Hereunder are some of the constructs we have on processes.

- The process *stop* is a special process that does not engage in any event.
- The process $c!i!j!v \rightarrow P$ is the process that initially is only allowed to perform the event $c.i.j.v$, the output of v on channel c, with source i and destination j. After this event, the process behaves as P.
- In a similar way, the process $c?i?j?x \rightarrow P(x)$ is the process awaiting an input on channel c with source i and destination j, after which it behaves as $P(x)$, i.e. x is instantiated.
- The process $P \Box Q$ behaves either as P or Q. There exists also an indexed form $\Box_i P_i$ meaning that the process can behave as any process P_i.
- Finally, $P|[E]|Q$ is the parallel composition of P and Q that have to synchronise on any event in the set of events E. $P|||Q$ is a syntactical shortcut for $P|[\emptyset]|Q$ and $|||_i P_i$ is the indexed form.

Moreover, we allow recursion and processes may take parameters.

CSP model for security protocols. Using CSP, a general model for security protocols can easily be defined. In the different papers, different models have been defined that slightly differ in some minor aspects. Here, we present the one proposed by Schneider in [140]. Let $USER$ be the set of entities on the network. We suppose that the intruder completely controls the network and that any communication passes through the intruder, called enemy, by means of the channels $trans$ or rec. We suppose that user $USER_i$ transmits, respectively receives, all messages using $trans.i$, respectively $rec.i$ events. Then, we can describe the resulting network as the following CSP process.

$$NET = \left(|||_{(i \in USER)} USER_i\right) |[trans, rec]| ENEMY$$

We write $|[trans, rec]|$ to denote a restriction on all events performed on channels $trans$ and rec. This restriction models that all communications pass through the intruder, called enemy.

Modelling the enemy. The enemy is a special process, independent of a particular protocol, defined by the recursive process hereunder.

$$\begin{array}{rl} ENEMY(S) = & trans?i?j?m \rightarrow ENEMY(S \cup \{m\}) \\ & \Box\Box_{i,j,S \vdash m} rec!i!j!m \rightarrow ENEMY(S) \end{array}$$

In the above process, S is the set of messages known to the intruder. This set increases when a user sends a message and the enemy can send out every message it can construct from the set S. The "construction" relation $\vdash$ is defined as follows. If m and n are messages, k is an encryption key and k^{-1} the corresponding decryption key, $m.n$ denotes concatenation of messages m and n, then $\vdash$ is the smallest relation which satisfies

$$\begin{array}{lcl} \{m, n\} & \vdash & m.n \\ \{m.n\} & \vdash & m \\ \{m.n\} & \vdash & n \\ \{m, k\} & \vdash & E_k(m) \\ \{E_k(m), k^{-1}\} & \vdash & m \end{array}$$

and also satisfies the closure conditions

$$\begin{array}{lcl} m \in S & \Rightarrow & S \vdash m \\ T \subseteq S \wedge T \vdash m & \Rightarrow & S \vdash m \\ (\forall v \in T \cdot\ S \vdash v) \wedge T \vdash m & \Rightarrow & S \vdash m \end{array}$$

The above rules state that an enemy can construct messages using pairing, or undo pairing, as well as cipher and decipher messages when the corresponding keys are known. At the beginning, we suppose that the enemy has a set of initial knowledge $INIT$, containing public keys, identities, possibly some compromised keys, etc. Then we have that $ENEMY = ENEMY(INIT)$.

Modelling protocol entities. The different participants in a protocol have a straightforward modelling in CSP. Modelling that a participant could engage a session with any other entity can be naturally expressed using the non-deterministic choice operator $\Box$ and making this operator range over the set of all users.

2.6.2 FDR

FDR is a finite model-checker that takes CSP as an input language. As it is generally the case when using finite model-checkers, this approach may be useful for finding flaws in protocols. While efficient for a finite number of instances, generally only one, they suffer from the state explosion problem when going to multiple sessions and do not prove the correctness of a protocol in the general case, i.e. with an arbitrary number of parallel sessions. One way to deal with this problem is to first prove that verifying a *small* system is enough. In [92], Lowe proved, by hand, that for his fix on the Needham-Schroeder public-key protocol, proving the protocol correct for one session implies its correctness for an arbitrary number of sessions. In [95], again Lowe identifies a set of sufficient, but very restrictive, conditions on protocols, for which correctness of one session of the protocol implies correctness in the general case. A slightly different approach was taken by Stoller in [151], where he proves bounds on the number of sessions for a given protocol that must be considered to prove correctness. However, the bounds require a number of sessions that are too high for current finite state model-checking tools.

We have already seen in the previous section how to model protocol entities and the intruder in CSP. FDR takes as an input two CSP processes, the specification and the implementation. FDR checks whether the implementation is a refinement of the specification. When working in the trace model, as it is the case here, refinement is defined as trace inclusion, i.e. each trace of the implementation is a valid trace of the specification. To ease writing the specification, the processes describing the protocol are slightly changed: special events are added that give information about the protocol execution. For instance, we could add an event *running.A.B* at the beginning of the specification of a user A to indicate that A starts a protocol session with the user B, as well as another event *commit.A.B* at the end of the specification of B to reflect that B finished a protocol run with someone he believes to be A. Given these events, we can describe a specification expressing that A is correctly authenticated to B, by stating that the event *running.A.B* should always precede the event *commit.A.B*.

$$\begin{array}{lcl} S_A & = & running.A.B \rightarrow commit.A.B \rightarrow S_A \\ S_E & = & \{running.A.B, commit.A.B\} \\ S & = & S_A ||| RUN(\Sigma \setminus S_E) \end{array}$$

In the above specification, S_A expresses the precedence of the events, S_E is the set of events that we are interested in, and S is the complete specification process. The set Σ is the set of all events and the process RUN allows all events in $\Sigma \setminus S_E$ to occur in arbitrary order. Using FDR, we can automatically check that all traces allowed by a given implementation, i.e. the protocol description, are included in the traces allowed by the specification S.

In [93], Lowe presents Casper, a Compiler for the Analysis of Security protocols. The aim is to overcome the need of expert knowledge required to model security protocols in CSP and correctly define the properties in order to use the FDR model-checker. Using Casper, protocol roles and the system in which these roles interact, are defined using a notation, which is very similar to the notation used when protocols are defined informally. Casper also defines the possibility to easily check two forms of secrecy and eight different flavours of authentication. The price to pay for the ease of use is that we cannot exploit the entire expressive power of CSP, when specifying the protocols and properties. On one hand, as in many formalisms, only linear protocols, i.e. linear sequences of messages can be described. On the other hand, there is no language for defining other properties than the ten predefined security goals. Also note that, in [113], Millen presents a similar language for protocol description: the Common Authentication Specification Language, CAPSL for short. While CAPSL is not directly bound to a specific tool, there exist interfaces to several tools, such as the NPA protocol analyser [32] or Athena [114].

2.6.3 Rank functions

In [140], Schneider introduces the idea of using *rank functions* in order to prove security protocols correct. In CSP we say that a process P satisfies a property φ if and only if φ

holds on all possible traces induced by P. Authentication can be expressed by specifying that an event from a given set of events R precedes another event from another set of events T on a given trace.

Schneider has proved a central theorem to this approach.

Theorem 2.1 [78] If there exists a rank function[6] $\rho : \mathcal{M} \to \{0,1\}$, satisfying

- $\forall m \in INIT \cdot \ \rho(m) = 1$
- $\forall S \subseteq \mathcal{M}, m \in \mathcal{M} \cdot \ ((\forall m' \in S \cdot \ \rho(m') = 1) \wedge S \vdash m) \Rightarrow \rho(m) = 1$
- $\forall t \in T \cdot \ \rho(t) = 0$
- $\forall j \cdot \ USER_j |[R]| stop$ satisfies *maintains* ρ

then NET satisfies R precedes T. □

Intuitively the rank function ρ maps all messages that should remain secret from the enemy to 0, and the remaining messages to 1. The first condition in the theorem expresses, that the intruder cannot initially know a message being mapped to 0. The second condition expresses that, only knowing messages mapped to 1, it is impossible to deduce a message mapped to 0. The third condition states that all messages in T should remain secret from the enemy. As, in order to prove authentication, we have to ensure that there exists a message in T that precedes a message in R, the intruder should not have the possibility to create a message in R. To understand the last condition we have to explain what it means for a trace to *maintain* ρ. We say that a trace *maintains* ρ if the following holds: if any message m appearing in a receive event is such that $\rho(m) = 1$ then any message m' appearing in a send event also has $\rho(m') = 1$. Basically a process can send out a "secret" message only when it previously received this message. The rank function must be such that this condition holds for all users in the network, restricted on the events in T.

The difficulty of this approach is to find a suitable rank function. While this gives a good insight into the protocol, finding a rank function may be a tedious process. When a rank function cannot be found and the protocol is flawed, the reason why the rank function does not exist generally gives a hint towards the attack. In [78], Heather and Schneider present an algorithm that finds a rank function if it exists. However when the rank function does not exist, this does not imply that the protocol is flawed, i.e. the rank function theorem is sound but not complete.

2.7 Spi calculus

The Spi calculus, introduced in 1998 by Abadi and Gordon [2], is another process algebraic approach to the analysis of security protocols. It is an extension of Milner's π-calculus [115] and basically adds cryptographic operations to it. The authors argue that the π-calculus can be used to describe protocols at a more abstract level, while the Spi calculus describes a possible implementation using cryptographic tools. The most notable difference with a process algebra such as CSP, is the fact that channels can be created dynamically. In the

[6] $\mathcal{M}$ denotes the message space.

π-calculus, and hence also in the Spi calculus, one can write to or read from a channel, once the channel name is known. Powerful scoping rules can be used to restrict a channel to a subset of users who can communicate via this channel. Using scope extrusion, users may achieve knowledge of new channel names (through old ones) and thus obtain the right to use these channels for communication.

Given the above tools of restriction and scope extrusion, the π-calculus can represent secret channels in an abstract way. When a channel name is restricted to only two processes, the channel may serve for secret communications between these two principals. In practice, such a channel could be achieved by exchanging a secret session key, sharing a long-term key or using public-key cryptography. Note, that restriction also provides an elegant way for modelling nonces. If the nonce is restricted to a process, it is impossible for anyone else to guess the nonce.

While the π-calculus provides strong tools to model protocols at an abstract level, the Spi calculus adds primitives for both secret and public key encryption, hashing and digital signatures. Using the Spi calculus, a more detailed description of the protocol can be given. Verification of protocols is done using equivalence testing. In a somehow similar way to the FDR's refinement testing, one needs to give a specification process, which should be "more obviously correct". To make correctness of the specification obvious, the specification process generally implements secret channels directly using restricted channels, rather than cryptography. Moreover, when a secret is sent, in the specification, the receiver *magically* knows the secret in advance and will not accept any other message. A system can be composed of m instances of a protocol, where an instance I is the set of parameters that identify the session. To prove the security of a protocol for an arbitrary number m of sessions, one has to prove that the following equation holds.

$$Sys(I_1, \ldots, I_m) \simeq Sys_{spec}(I_1, \ldots, I_m), \text{ for any instances } I_1, \ldots, I_m.$$

In the above equation, Sys_{spec} denotes the specification process and $\simeq$ is the may testing equivalence[7]. A particularity of this approach is that there is no need in specifying explicitly the intruder, as the equivalence must hold in the most general environment. This means that the Spi calculus is used, much as rank functions, to prove correctness of a protocol, rather than finding errors.

This approach has some evident advantages, which are the well defined underlying semantics, the elegance of specification using scoping rules and the fact that the intruder is implicitly defined. However all the proofs are hand generated and require a great deal of expertise. Machine assistance for carrying out these proofs would be welcomed to make their approach accessible to a broader public.

2.8 Murφ

Like FDR, Murφ is a general purpose finite state model-checker, that has first been used in the context of security protocols in 1997 by Mitchell et al. [116]. As Murφ is a finite state tool, verification is limited to a bounded number of sessions.

[7] May testing is a trace based equivalence relation between processes. The interested reader can find the definition in [2].

2.8.1 Protocol modelling

The protocol description is directly expressed in the Murφ description language. The description language is similar to common programming languages. When modelling a protocol, each protocol rule, which is either a transmission or a reception of messages, is given as a more or less ten line program in the Murφ input language. For instance, when modelling the Needham-Schroeder protocol, one has to encode three rules for both Alice and Bob. A major advantage of this language is that it supports *scalable* models. One can easily, by changing a constant, vary the number of initiators or responders in the protocol. While in CSP special primitives for communication exist, in Murφ, the communications are modelled through shared variables.

2.8.2 The attacker model

The attacker is modelled as a typical Dolev-Yao attacker, i.e. he can spy messages, insert messages constructed from its initial knowledge and passed messages, as well as remove any message from the network. The attacker is then manually modelled as any other participant, using rules, and must be adapted to each individual protocol. As only a finite number of sessions is considered, the number of messages generated by the intruder, which could be accepted by the other protocol participants, is finite. Note that modelling the intruder may be a time consuming and error-prone process. As the authors noticed themselves, modelling the attacker often consumes more than half of the time required to prepare the Murφ description of a protocol. The authors conjecture that, determining the message type, and specifying the attacker could, at least partially, be automated. However, to the best of our knowledge, no effort has yet been spent in such an automation. Finally, the authors briefly discuss some efforts to include special properties of certain cryptographic primitives, such as the RSA multiplicative property. These rules can be encoded in the attacker model, and the authors demonstrate a flaw that can be found automatically based on the attacker's increased power.

2.8.3 Property specification and protocol verification

Properties are specified as invariants in Murφ. According to the authors, expressing authentication or secrecy properties has shown to be rather straightforward. The verification time increases exponentially with the number of participants. For instance, verifying the corrected Needham-Schroeder protocol takes as few as 3 seconds with one initiator and one responder, a bit less than a minute when considering 2 initiators, and more than an hour and a half with two initiators and two responders[8]. In 1998, Shmatikov and Stern [149] propose new state reduction techniques specific to security protocols, that optimise the verification process and allow the verification of larger protocols.

2.9 The inductive approach

In 1997, Paulson introduced the idea of using inductive proofs to verify the correctness of security protocols [126]. A more complete description can be found in [127]. Since

[8]Note that these results were given in 1997, on a workstation with moderate computing power.

then, Paulson applied his method to numerous security protocols [127] including "real life" protocols, such as Kerberos [23] or SET [128].

2.9.1 Protocol modelling

Here, protocols are modelled as possible extensions of traces of events, being generally of the type $\mathsf{Says} AB\ X$, corresponding to the event A sending message X to B. Each protocol step is modelled by a set of conditions which must hold on a trace in order to allow this trace to be extended by the given event. Conditions may for instance require that a nonce is fresh on the given trace or that the trace contains a specific event $\mathsf{Says} AB\ X$.

2.9.2 analz and synth operators

One of the most important tools used in Paulson's method are two operators[9], called analz and synth. If H is a set of messages then $\mathsf{analz} H$ is defined to be the least set closed under the following rules.

$$\frac{X \in H}{X \in \mathsf{analz} H} \qquad \frac{E_K(X) \in \mathsf{analz} H \wedge K^{-1} \in \mathsf{analzH}}{X \in \mathsf{analz} H}$$

$$\frac{X, Y \in H}{X \in \mathsf{analz} H} \qquad \frac{X, Y \in H}{Y \in \mathsf{analz} H}$$

In the above rules K^{-1} denotes the decryption key corresponding to K in either symmetric or asymmetric encryption. Intuitively, analz defines the knowledge one can extract from a set of messages H.

In a similar way, $\mathsf{synth} H$ is defined to be the least set closed under the following rules.

$$\frac{X \in H}{X \in \mathsf{synth} H} \qquad \frac{X \in \mathsf{synth} H \wedge K \in \mathsf{H}}{E_K(X) \in \mathsf{synth} H}$$

$$\frac{X \in \mathsf{synth} H \wedge Y \in \mathsf{synth} H}{X, Y \in \mathsf{synth} H} \qquad \frac{X \in \mathsf{synth} H}{h(X) \in \mathsf{synth} H}$$

An important result is that, when only considering atomic keys, given a set H of messages one can compute the set $\mathsf{synth}(\mathsf{analz} H)$, that corresponds to the set of all messages one can construct from a given set H, i.e. it is possible to always first apply *destruction* rules, before applying *construction* rules. However non-atomic, constructed keys are not handled by his method. As an example, consider $H = \{E_{h(k)}(X), k\}$. In order to retrieve X, one would need to first apply a synth rule on H in order to construct the necessary decryption key $h(k)$.

2.9.3 Attacker model

Paulson models the attacker, called *Spy* in his terminology, by adding an extra rule[10] to the rules derived from the protocol. The additional rule says that, given a trace t, a set

[9] We deliberately omit the parts operator in our presentation, as it is not needed for the basic comprehension we want to provide.

[10] Here, again for the sake of simplicity, we do not include the Oops event in our description.

of messages H corresponding to the messages of t and some initial knowledge of the spy, we may extend t by

$$\mathsf{Says} Spy B\ X$$

where $B \neq Spy$ is some agent. This means that the spy can always send any message he can construct, to any given protocol participant.

2.9.4 Induction

Proofs in Paulson's model are based on an easy induction principle. Therefore we suppose that the empty trace is always part of the possible traces induced by a protocol. When we want to prove a property P, we first have to prove, that it holds on the empty trace, which is generally trivial. Then one needs to prove that if the property P holds on a trace t, the property must hold on the trace t', where t' is the trace extended by any permitted event e. Secrecy, for instance, can be expressed by saying that no trace is permitted to contain a particular Says event. Authenticity may be expressed by saying, when some event arises, it must be preceded by another given event.

Proving such theorems on protocols has been partially automated using the theorem prover Isabelle [125]. Moreover, for most protocols, the same proof strategies can be reapplied. Following Paulson, analysing a completely new protocol takes several days. However, this estimation applies to an expert user. In 2001, Cortier, Millen and Rueß [53] presented a completely automated method, similar to Paulson's approach to prove secrecy. However, although sound, their method is incomplete in the sense it may fail to prove secure protocols.

2.10 Strand spaces

The strand space model has first been introduced in 1998 [160] by Thayer, Herzog and Guttman. Their model has known many extensions, such as [162, 75, 159, 161], to cite only a few of them, and strand spaces are still subject to active research. In this section, we only give a very small overview, high-lightening the main ideas of strand spaces. We first study the works by the inventors of the strand space models. These works are based on induction and can be related to Paulson's work using Isabelle and Schneider's work on rank functions. Then we also briefly discuss the tool Athena, which automates the strand spaces ideas. Finally we look at a finite state model-checker based on constraint solving. This model-checker is based on a completely different analysis methodology but uses strands as input language, and is therefore mentioned in this section.

2.10.1 Strands and bundles

We will not give formal definitions of strand spaces. We refer the interested reader to [76] which gives a good overview of the topic, or to the above cited papers for even more details. We illustrate the strand space model by means of the Needham-Schroeder public-key protocol, depicted in figure 2.1. This figure describes the protocol by means of two *strands*: the A strand and the B strand. A strand is a sequence of message transmissions and receptions, where each message is a term generated by a free term algebra. We also refer to each element of the strand as a node. As we see, we have only given A's and B's individual views of this protocol. Unfortunately, we are not aware of all possibilities to make these two strands match together in the presence of an intruder or, using the strand space terminology, a penetrator.

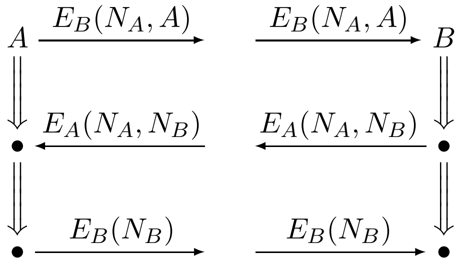


Figure 2.1: Strand specification of the Needham-Schroeder public-key protocol

One way of matching the strands of figure 2.1 together is shown in figure 2.2. This figure describes Lowe's famous attack on the Needham-Schroeder public-key protocol [91]. We see that matching the two strands together results into a directed acyclic graph where the vertices are the nodes of the strands (including the *penetrator strands*) and the edges are either $\Rightarrow$ or $\rightarrow$. Such a graph is called a *bundle*. While a strand corresponds to the protocol description for a participant, the bundle corresponds to a protocol execution.

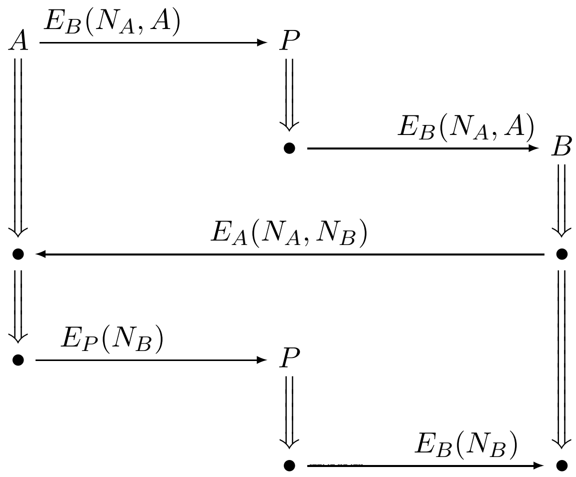


Figure 2.2: Bundle showing Lowe's attack on the Needham-Schroeder public-key protocol

2.10.2 Junk messages and unintended services

A first heuristic to analyse the protocol is to identify all *junk* messages and *unintended services*. A junk message is a message that does not have any authenticating power, i.e. that could obviously be forged by the penetrator. In the Needham-Schroeder public-key protocol, the first message $E_B(A, N_A)$ can be considered to be junk, as anyone can construct a message of this form. Hence, when discarding the junk messages, we remain with only the third message providing any authenticating force to Alice. We can also identify the unintended services provided by the protocol: the Needham-Schroeder protocol contains a *translation* service, offered by A. Giving a message $E_A(N_A, x)$ to A is translated into $E_P(x)$, for any value x, which in this case makes x readable to P and allows the attack.

2.10.3 Penetrator strands

Simplifying the protocol by discarding junk messages and identifying unintended services, gives a good preliminary analysis for authentication and secrecy protocols. However strand spaces do also provide a more systematic way of analysing protocols. We start by looking at the actions a penetrator can perform. These actions can be represented by strands and consist in E-strands, that encrypt giving a key and a plaintext, D-strands, that decrypt when giving a key and a ciphertext, C-strands and S-strands, that concatenate messages and separate concatenations, as well as K-strands and M-strands, that emit known keys, or known atomic texts or guesses, such as nonces.

One can also observe that not any way of composing these different penetrator strands is interesting. For instance, concatenating two messages that have been obtained by separating them just before does not provide any new message to the penetrator. In the same vein, it is possible to always first apply analysis rules, such as separation and decryption, before applying construction rules as encryption and concatenation. This corresponds to Paulson's [127] idea of analysis followed by synthesis.

2.10.4 Authentication and secrecy

We now look at how authentication and secrecy are specified in the strand space model. Authentication mainly expresses that some regular participant in the protocol has done one or more given actions. To express authentication we need to introduce the notions of $\mathcal{C}$-height and originating assumptions. The $\mathcal{C}$-height of a strand R in a bundle $\mathcal{C}$ is the number of nodes of the given strand R that are included in $\mathcal{C}$. Originating assumptions are used to state assumptions that the penetrator cannot guess a nonce (which should be *uniquely originating* in a bundle), or a key (which should be *non-originating* in a bundle). In the strand space model, an authentication goal, where a participant described by strand R wants to authenticate himself to a participant described by role R', is expressed in the following way : for all bundles $\mathcal{C}$, given i and j,

> *if* some originating assumptions hold,
> *and* R has $\mathcal{C}$-height i,
> *then* R' has $\mathcal{C}$-height j.

The height reflects the fact that some actions did take place. Proofs of authentication are mainly carried out using an induction principle based on the partial order defined by the bundle, which is a finite directed acyclic graph. We are not going to give details about incoming and outcoming authentication tests, which are the "tools" used in the proofs.

On the other hand, while an authentication goal says what must have happened, a secrecy goal states what actions must not have happened. A secrecy goal takes the following form: for all bundles $\mathcal{C}$ and strands R, given i,

if some origination assumption holds
and R has $\mathcal{C}$-height i,
then there is no node n such that term(n)=t

where term(n) denotes the term that has been send or received at node n and t is the secret we want to keep.

2.10.5 Athena

The strand space proof method has been implemented in an automated tool Athena [150]. The tool takes as input a slightly modified version of the strand space model, that are parametric strands. This means that the strands may contain variables associated with the roles. In [150], Song proposes a model-checking algorithm that may not terminate if no bound is given on the number of sessions. In practice, the algorithm terminates for most protocols even without bounding the number of sessions and has shown to be very efficient, keeping the state space extremely small. A detailed comparison with finite model-checkers used for protocol analysis, such as Murφ and Brutus, is given in [150].

2.10.6 Constraint solving

Millen and Shmatikov introduced an interesting tool [112] in 2001, that reduces the reachability problem for cryptographic protocols to a constraint solving problem. Although, the protocol roles are described using parametric strands, the tool has several differences with respect to Athena. First it only considers a bounded number of instances of each role, which guarantees termination. A second difference is that Millen and Shmatikov do not reason about penetrator strands, but construct the attacker's capabilities in a similar way to Paulson's analz and synth operators. Another important difference is that contrary to the previous described methods they handle constructed, non-atomic keys. In 2002, Corin and Etalle [52] presented an optimised version of this tool, with several extensions, which we are not going to discuss.

To reduce the verification to a constraint solving problem they proceed as follows. First of all, they add a test strand. For secrecy, a test strand would be defined as a single node receiving s, where s is the secret that has to be guarded by the protocol. A test strand for authentication could take the form of an additional *authenticated* strand, i.e. a strand describing Bob's role, without an additional *authenticating* strand, i.e. a strand describing Alice's role. For a protocol to be vulnerable to an attack, the intruder must be able to match all the given strands into a bundle. Therefore, the tool generates all interleavings of the involved strands, and for each interleaving generates a constraint set. A constraint is of the form $m : T$, where m is a term and T a term set. Such a constraint asserts that the intruder must be able to synthesise a term m from T. Each node sending a message m adds m to the intruder's knowledge T, while each node receiving a message m generates a new constraint $m : T$. Now the problem to be solved is to find a substitution σ of ground terms for all variables such that for each constraint $m : T$, $\sigma(m)$ can be constructed from T.

2.11 Conclusion

In this chapter we have seen that a large number of different approaches, dealing successfully with authentication and secrecy protocols, exists. However, these approaches cannot be directly applied to fair exchange protocols. In the next chapter we will introduce these protocols and motivate the need for techniques adapted to fair exchange protocols.

CHAPTER *3*

Fair exchange protocols

In this chapter we introduce fair exchange protocols. First, we discuss the general problem and look at specific instances of fair exchange. Then we survey different definitions of the properties fair exchange protocols should provide, which were proposed in the literature, and discuss them. We notice that most definitions, while not necessarily wrong, are focussing on a specific instance of the more general fair exchange problem or on the way the properties are achieved. Moreover, different properties are often gathered in a single definition. We conclude by giving a structured and modular, although informal, definition of the different properties. We believe that a good informal understanding is necessary, before trying to express the property in a specific formal setting. Our discussion on the definitions of the properties of fair exchange protocols is based on a joint work with Dieter Gollmann and Olivier Markowitch [97].

We also give a historical overview, showing the evolution of fair exchange protocols. This overview is based on a review paper [87], realized in collaboration with Olivier Markowitch and Jianying Zhou. First attempts to solve the fair exchange problem go back to the beginning of the 1980's. These early protocols were either based on gradual exchange of secrets or a probabilistic notion of fairness. Later, protocols using a trusted third party, TTP, were proposed. The challenge of this approach is to minimise the intervention of the TTP. Most major achievements concerning *optimistic* protocols were realized during the second half of the 1990's. We conclude the chapter, by showing some important differences between classical protocols, aiming authentication or secrecy, and fair exchange protocols. These differences motivate novel methods for the analysis of optimistic fair exchange protocols.

3.1 Instances of fair exchange protocols

Due to the impressive growth of the Internet and of open networks in general, the interest in fair exchange protocols has become more and more important during the last years. Fair exchange protocols are a crucial theoretic basis to make secure electronic commerce and electronic business transactions possible. Fair exchange protocols aim to protect clients against malicious merchants, as well as merchants against dishonest clients, business partners that do not trust each other, etc.

Informally, the fair exchange problem deals with the exchange of two items of a given value between our protagonists Alice and Bob. The aim is to realize this exchange without the risk of either Alice or Bob being disadvantaged, e.g. Alice does obtain Bob's item, but Bob does not obtain Alice's one. Hence the name *fair* exchange. Fair exchange protocols regroup a vast family of protocols, including

- electronic purchase of goods: exchange of an electronic item against an electronic payment,
- digital contract signing: exchange of digital signatures on a given electronic document,
- non-repudiation protocols: exchange of an electronic item and a non-repudiation of origin evidence against the corresponding non-repudiation of receipt evidence,
- certified e-mail: exchange of an electronic message against a proof of receipt,
- barter: an electronic item of a given value is exchanged against another item of a similar value,
- ...

All these problems can be classified as fair exchange protocols, because they solve the same general problem. However, it is often interesting to tailor protocols for each of these specific problems, allowing more efficient solutions or allowing us to include features specific to a given problem. We will discuss some of these more specific details hereunder.

3.1.1 Electronic purchase of goods

Electronic purchase of goods seems to be one of the most natural applications of fair exchange protocols. However, there may be many additional problems, that make its implementation rather complicated. Generally, the fair exchange must be combined with an electronic payment protocol, such as [28] or [103]. Another problem, that is common in the fair exchange of items, is to decide whether the sent item corresponds to the expected one. Generally, a customer possesses a description of the desired item, and one has to decide whether the received item matches this description or not. Often, protocols abstract the decision whether the item matches its description, by assuming the existence of a function able to validate the item. In this chapter, we will not look at this kind of problems, but we will focus on the exchange part of the protocols.

3.1.2 Non-repudiation protocols

The primary goal of a non-repudiation protocol is to produce the following evidences:

- a non-repudiation of origin evidence,

- a non-repudiation of receipt evidence.

The non-repudiation of origin evidence links a message, its originator and its recipient. It must be impossible to reuse the same evidence for different messages or recipients. Otherwise, the recipient Bob could transmit Alice's evidence to Charlie who could falsely prove having received the message from Alice. In the same way, a non-repudiation of receipt evidence links the recipient and the originator to the message, even if the link is different.

It is crucial to distinguish the non-repudiation property provided by digital signatures and non-repudiation evidences. For instance, Alice's digital signature on a message m means that Alice is at the origin of the message. However, a non-repudiation evidence includes the identity of the person intended to receive the message. For example, a non-repudiation of origin evidence proves that Alice sent a given message to Bob, while the mere signature on this message only proves that Alice is at the origin of the message, without specifying any recipient.

With correct non-repudiation evidences, if Alice denies having sent a message to Bob, or if Bob denies having received it, the non-repudiation of origin, respectively receipt evidences can be presented to an adjudicator. The adjudicator can evaluate these evidences in the context they have been generated in, and decide to either accept or reject Alice's and Bob's claims. Hence, a second protocol must be defined, that describes the dispute resolution, and how evidences must be interpreted. Note that maintaining non-repudiation evidences is not easy, as keys may be revocated [118, 137]. The interested reader can find mechanisms to correctly manage these evidences in [87, 169, 173, 181].

An interesting difference with most other examples of fair exchange protocols is that the expected items are not known *a priori*. When Alice sends a message m and the corresponding non-repudiation of origin to Bob, the message m is unknown to Bob, as opposed to a protocol for electronic purchase or barter, where the entities know which item they expect. Although this difference may seem meaningless, it makes it possible to design non-repudiation protocols in a more efficient way, in comparison to more general fair exchange protocols.

3.1.3 Certified e-mail

Electronic mail, or e-mail for short, is probably the best example to illustrate the fact that traditional 'pen and paper' transactions are becoming more and more replaced by equivalent electronic services. Millions of e-mails are sent every day over the Internet. However, the traditional post offers more sophisticated services, such as express mail, compensation for lost mail or certified mail. Hence, the electronic equivalent should offer similar services, even if the mechanisms that implement these services may differ.

Express mail deals with a fast delivery of mails, e.g. delivery before a fixed deadline. On networks, fast delivery is part of the nature of electronic mails and is naturally provided, unless at least one link on each route between the sender and the recipient is broken or denial of service attacks flood servers and disturb the normal operation, but we are not going to study these kinds of issues. In a similar way, compensation for lost mails is almost never necessary when using e-mail: an electronic item does not physically exist and cannot be lost in the same way as a traditional letter. Although e-mails can be misrouted or lost due to network failures, the loss is insignificant as multiple copies of an item can easily be created. If the mail has to be delivered before a given deadline, the loss of a message, or a permanent network failure can be a problem. In that case, compensation could be an expected property, but we do not focus on this aspect.

The third service we mentioned is certified mail. This service provides a proof of delivery to the sender. In opposition to express mail and compensation for lost items, the electronic version of certified delivery is more complicated to provide than the traditional one and is very close to a non-repudiation protocol. In the literature, there have been several papers addressing explicitly certified e-mail [14, 18, 55, 133, 144, 176]. However, with the exception of Zhou and Gollmann in [176], they do not make a difference between a certified e-mail protocol and a non-repudiation protocol. In [176], Zhou and Gollmann argue that non-repudiation of origin is not a required service—in fact, traditional certified mail does not provide it—and define a certified e-mail protocol to be a service that must be offered by an external delivery agent, in the same way it is realized by the post in traditional mail. We do not follow this last point, as the definition of a service provided by a protocol should not depend on *how* it is provided.

In [85], in collaboration with Oliver Markowitch, we defined an additional property, that is specific to certified e-mail protocols and not required in non-repudiation protocols: *author-based selective receipt*. Generally, it is required that once the message is known to the recipient, he cannot prevent the protocol from delivering the receipt. It seems to us that this property is not sufficient: we believe that once the identity of the author is revealed, the receipt has to be delivered to the sender, and of course the message to the recipient. In traditional certified mail, one does not get any information about the mail, neither the content nor the origin, before having signed the receipt. In many cases the origin of a message leaks enough information to guess the mail's content. To better understand the importance of that property, consider the following real life example. If a person does not pay the rent of his flat, he can refuse a certified mail coming from his landlord. He does not need to read the letter to guess that the landlord is claiming the rent and wants a proof for this claim. Knowing the identity of the sender would reveal enough information to guess the content of the letter. Hence, once the recipient knows the identity of the author, he should not be able to refuse reception any more[1]. This is a property that is not required in non-repudiation protocols and shows a difference between these non-repudiation and certified e-mail protocols.

3.1.4 Digital contract signing

Digital contract signing is one of the simplest forms of fair exchange protocols. In a digital contract signing protocol, Alice and Bob want to exchange their corresponding signatures on a given contract text, known to both Alice and Bob. As we will see in this chapter, efficient protocols for fair exchange of digital signatures exist. The fact that one exactly knows what the expected item looks like, makes the protocol design easier.

[1] Of course, *every* sent certified e-mail should provide this property. Otherwise, a recipient always rejects anonymous messages.

3.1.5 Multi-party fair exchange protocols

Above, when illustrating the different instances of fair exchange protocols, we described them as two-party protocols. These instances can also be generalised as an exchange between multiple parties. We are going to discuss multi-party fair exchange protocols in more detail in section 3.6.

In the following, we make a certain number of implicit hypotheses. We suppose that, when applicable, the entities implied in a protocol first agree in a separate, preliminary, protocol on the items to exchange, e.g. the contract to be signed, the price of an electronic good, etc. We also suppose that a public key infrastructure is present, that for each digital signature valid public-key certificates are added and revocation lists are consulted when necessary. These crucial details will not be included in the protocol description to keep these descriptions readable. These problems can be managed separately, and are not directly related to the problem of fair exchange. Of course, correctly maintaining certificates and evidences related to these certificates is not trivial, when certificates can be revocated, but we do not consider these aspects in the following.

3.2 Communication channels

In order to be able to describe the protocols in this chapter, we have to introduce several classes of communication channels. In the framework of fair exchange protocols, we classically distinguish three classes of communication channels: unreliable channels, resilient channels and operational channels. We give here preliminary definitions, which are not formal. We cannot formalise the different channel qualities yet, but we will give precise definitions when describing our model in chapter 5.

Definition 3.1 An *unreliable* channel is a channel on which data may be delayed and even lost. □

Unreliable channels are the most basic ones. They do not provide any guarantee of service.

Definition 3.2 A *resilient* channel is a channel that delivers data correctly after a finite, but unknown amount of time. □

On a resilient channel, data may be delayed, but will eventually arrive. Resilient channels are also referred to as asynchronous channels. Note that resilient channels do not guarantee that successive messages are delivered in the order they were sent.

Definition 3.3 An *operational* channel delivers data correctly in a (known) bounded amount of time. □

Operational channels are sometimes called *synchronous channels* in literature. Operational channels make very strong assumptions on the network services and are rather unrealistic in open, heterogeneous networks. Such channels could be implemented on a dedicated infrastructure, but this would restrict the applicability of protocols relying on such channels.

3.3 Trusted third parties

Most modern protocols rely on the intervention of a TTP. Various types of TTP can be considered, according to their involvement in the protocol.

Definition 3.4 A TTP involved in each message transmission during the protocol, is said to be *inline*. □

Definition 3.5 A TTP involved during each session of the protocol but not during each message transmission, is said to be *online*. □

Definition 3.6 A TTP involved in a protocol only in case of an incorrect behaviour of a dishonest entity, or in case of a network error, is said to be *offline*. □

Protocols with offline TTP are also called *optimistic*.

Definition 3.7 An offline TTP producing evidences indistinguishable from the evidences Alice and Bob should have exchanged in a faultless case, is said to be *transparent*. □

Note that we will use the term trusted *third* party even in the context of multi-party protocols for coherency reasons.

3.4 Properties of fair exchange protocols

The property that is central to fair exchange protocols is *fairness*. The existence of different instances of fair exchange protocols, such as discussed above, is—at least partially—responsible for some exotic definitions of fairness. We also note that two-party fair exchange protocols and multi-party fair exchange protocols have often different fairness definitions (partially due to different topologies used in the multi-party case) [12, 21, 66].

Here, we defend the point of view of a unified definition of fairness, whatever the underlying exchange protocol or formalism may be. We put ourselves in a theoretical context where fairness is always needed and propose a generalised definition of fairness and of the security of an exchange protocol. Our definitions aim to capture *what* a given property provides, and not *how* it is provided. To quote Roscoe [136], we avoid "intensional definitions", which are related to a sequence of actions that must (or must not) happen in a given order, for a property to hold. Intensional specifications are sometimes useful in formal verification as they capture the way the protocol designers have foreseen the protocol execution (and therefore, we do not criticise them) but they are not general enough to give a definition of a central concept such as fairness.

In this vein, when examining the classical statement that "no party has an advantage", we will distinguish between the aspects of an exchange the advantage could apply to, but we will not examine how this advantage could be measured. There are different views of what constitutes an advantage and we have to be able to differentiate these aspects and define them informally but clearly before choosing a particular formalism. The formalism should explain what is meant by gaining information (implying some interesting views, like those proposed by Jakobsson and Syverson in [81, 152]). There can again be different formal definitions of "gaining information". We would like to stress that an informal analysis of security properties is essential before properties are being formalised. The goal is to promote a clear, although informal, understanding of fairness in order to be able to formalise later the needed properties.

As, unfortunately, the functioning of the protocol often influenced the definition of fairness, we already announce, before discussing the protocols themselves in more detail, that historically the first protocols were based on gradual exchange or on a probabilistic feature, before the rising of protocols with a decreasing intervention of a *trusted third party*. The most recent protocols are based on an offline TTP. In these protocols, participants may contact the TTP if a problem arises, via an abort protocol in order to obtain a cancellation, or a recovery protocol to force a successful outcome. Although some definitions make reference to the functioning of the protocols, we start by describing the properties the protocols should achieve, rather than the protocols themselves, as we defend the point of view that the definition of the properties should guide the protocol design and not the other way round.

In this section, we survey and discuss some of the most classic fairness definitions found in the literature, showing the evolution of this definition and why we do consider them as either too specific or too vague. We also discuss some additional properties specific to some particular applications. We then propose a consistent and modular definition of the security of an exchange protocol, where fairness is one of the needed properties. We point out the necessity of other important properties like timeliness, viability etc. in this security definition and define some additional properties that may be required, depending on the context.

3.4.1 Evolution of the fairness definition

We start showing some historical definitions, which are mainly due to the early fair exchange protocols, based on gradual exchange of secrets or probabilistic notions of fairness. Then we show what we call "definitions with ballast", in the sense that the definitions merge several properties into one single definition.

3.4.1.1 Historical definitions

Although the relevance of fairness has been identified since the early 1980s, the first propositions of a fairness definition corresponding to practical solutions were expressed in terms of computing power. The protocols exchanged information piece by piece and it was required that the computational effort, the parties need to spend in order to obtain each others remaining information, should be approximately equal at any stage during the execution of the protocol.

Even et al. [61] proposed a classical definition of fairness in the framework of contract signing, called "concurrency": *if one party X executes the protocol properly, then his counterpart Y cannot obtain X's signature to the contract without yielding his own signature to it.* Unfortunately, they do not propose a solution respecting this definition. In order to solve the exchange problem they introduced a weaker definition, called "approximate-concurrency": *if one party X executes the protocol properly then, with very high probability,* *at each stage during the execution, X can compute his counterpart's signature to the contract using approximately the same amount of work used by Y to compute X's signature to the contract*. This is what we call the computational approach towards fairness.

It was rapidly accepted that gradual release protocols requiring an equal, equivalent or even related[2] computing power between the communicating parties are not reasonable. A large company for instance, generally has more powerful computers, or even clusters of computers, than an individual person. During an exchange, the company would clearly be privileged.

An important evolution was the probabilistic approach, not requiring equivalent computing power, first proposed by Ben Or et al. [25] in the contract signing framework. They defined probabilistic fairness in the following way: *a party is privileged when s/he is capable of causing the judge to rule that the contract is binding on both parties*; *a contract signing protocol is* $(v, \varepsilon)-fair$ *for a party A if the following holds for any contract C when A follows the protocol properly: at any step of the protocol, in which the probability that another party B is privileged is greater than v, the conditional probability that A is not privileged given that B is privileged is at most* ε. ε *denotes an upper bound on the probability that one party is not privileged given that the other is privileged*.

Protocols based on this last definition are traditionally implemented by successive rounds during which, in turn, a party is privileged whereas the other one is not. This yields a situation that could be considered unfair (in the common sense). In our point of view, the fact that the entities are privileged in turn is not unfair. It would be unfair if one party were able to prove that the other party has signed the contract and therefore is linked to it alone.

Putting aside these historical definitions, the actually most widely accepted definition of fairness [13, 14, 17, 61, 70, 153, 167, 175] describes fairness in relation to the end of the exchange protocol run: *at the end of the exchange protocol run, either all involved parties obtain their expected information or none of them receives anything*. We consider this definition to be, almost, the most suitable one[3].

3.4.1.2 Definitions with ballast

There are many recent definitions of fairness that include additional properties in the basic definition, which we therefore call definitions "with ballast". Fairness definitions also often describe the mechanisms necessary to realize fairness in particular cases. The definition is not only based on what is fairness but rather on how to obtain it. We first consider these definitions, given in the context of digital contract signing and then in the more general case of fair exchange protocols.

Digital contract signing. In the context of digital contract signing, Asokan et al. [16] specify by means of a game that an exchange protocol for signatures is not fair if a malicious entity can exchange an invalid signature against a valid one. Although the definition is appropriate and complete, a general definition of fairness should not be based on such

[2] Where the computing power ratio between two communicating parties is known and fixed.

[3] Moreover, this definition can easily be adapted in a probabilistic context: *at the end of the exchange protocol run, there has to be an overwhelming probability that either all involved parties obtain their expected information or none of them receives anything.*

specific concerns (even if the proposed definition is always true in the framework of digital contract signing). We believe that a general definition of fairness can be expressed such that all exchange types are covered. The way followed to obtain fairness depends on the context: for digital contract signing protocols, the security of the signature is an important part but this aspect has to be developed in the security proof of the protocol and not in the fairness definition.

Garay et al. [67] specify that an optimistic contract signing protocol is fair if:

1. *it is impossible for a corrupted participant to obtain a valid contract without allowing the remaining participant to also obtain a valid contract,*
2. *once a correct participant obtains a cancellation message from the* TTP*, it is impossible for any other participant to obtain a valid contract,*
3. *every correct participant is guaranteed to complete the protocol.*

The restriction on the cancellation message (the second rule) seems too restrictive. Indicating that a party, having carried out a cancellation, should not take the risk to continue the protocol is a part of the *protocol's description*: the specification of the behaviour of the parties implied in the protocol belongs to its description and should not belong to the fairness definition. Moreover, as cancellation is specific to optimistic protocols, we would have different fairness definitions depending on the TTP's involvement.

Note that the cancellation (or abort) token is produced during an optimistic protocol to inform the party asking for a cancellation that the TTP will no longer accept recovery requests during this protocol run. This cancellation token, issued during an abort protocol, is necessary to ensure timeliness (which will be clearly defined in section 3.4.6). The timeliness property is respected if the parties always have the ability to reach, in a finite amount of time, a point in the protocol where they can stop the protocol while preserving fairness.

The third rule concerns the ability to complete the protocol. If the guarantee to complete a protocol is related to the fact that a way to securely end the protocol (with or without a completed exchange) must exist, then this corresponds to the timeliness property we just discussed. Otherwise, if completing the protocol is related to the fact that the exchange succeeds, this is the viability property. A protocol is viable if the exchange always succeeds when the involved parties behave honestly (i.e. follow the protocol). Viability differs from fairness and is more difficult to obtain in practice, because the success of the exchange does not depend only on the honesty of the parties but also on the quality of the underlying network.

In [130], Pfitzmann et al. say that *a contract signing scheme is called fair if it fulfils the following requirements:*

1. *correct execution,*
2. *unforgeability of contracts,*
3. *verifiability of valid contracts (a signed contract cannot be invalidated),*
4. *no surprise with invalid contracts (a rejected contract — no party had signed it — cannot be declared signed),*

5. *termination on a synchronous network (the protocol ends after a finite amount of rounds),*

6. *termination on an asynchronous network (after a time-out or a user's manual input, the protocol ends after a fixed time).*

The first rule is related, if no sub-protocol is launched, to the viability property, which is distinct from fairness.

The second, third and fourth rules are specific to digital contract signing. As these rules apply to any contract signing protocol, these definitions could be considered as extensional specifications of a contract signing protocol, in opposition to intensional specifications which apply to one specific protocol. In [130], these statements are defined in terms of precise inputs and outputs of the protocol and in terms of execution of sub-protocols *show* and *sign*[4]. Such definitions refer to the machinery of contract signing.

The fifth and sixth rules are related to the timeliness property, which is also distinct from fairness.

Of course, it may be the case that in certain formalisms, extensional definitions are quite difficult to state.

Fair exchange. Vogt et al. [167] proposed to split the definition of fairness into two aspects, whether the participation of a "faulty" entity is or is not needed when the TTP is requested to help finishing the protocol. Again, this approach described *how* fairness is obtained. We emphasise once again that when defining fairness it is necessary to focus on *what* is fairness and not on *how* to obtain it.

Franklin et al. [65] said that *at the end of the fair exchange the following must be true:*

1. *if A, B, and the* TTP *are honest, A learns B's information and B learns A's information,*

2. *if A and the* TTP *are honest then B does not learn anything about A's information unless if A learns B's information*

3. *if B and the* TTP *are honest A does not learn anything about B's information unless if B learns A's information,*

4. *if A and B are honest then the* TTP *does not learn anything about A's and B's information.*

Again, the first property is the viability property. The fourth property is about confidentiality with regard to the TTP. This is not needed in fairness but is due to the context of key exchanges of their paper.

[4] For example, in the fairness definition [130], the statement "Verifiability of valid contracts" is defined by "If a correct signatory, say A, outputs $(\mathit{signed}, C, \mathit{tid})$ and later executes "show" on input $(\mathit{show}, \mathit{tid})$ then any correct verifier will output $(\mathit{signed}, C, \mathit{tid})$ for any C."

3.4.2 Vague definitions

Zhou et al., in the context of non-repudiation protocols [174, 177, 178, 179], define fairness as follows: *a non-repudiation protocol is fair if it provides the originator and the recipient with valid irrefutable evidence after completion of the protocol, without giving a party an advantage over the other at any stage of the protocol run.*

Boyd et al. [30] propose a similar definition: an exchange protocol is fair if *at no point during the execution of the protocol either of the entities participating in the exchange can gain any (significant) advantage over the other if the protocol is suddenly halted.*

In both definitions the notion of advantage is not defined. These definitions seem practically to exclude any protocol not offering a perfect symmetry (in terms of knowledge and possibility of action). However, such a definition is obviously not formalisable and seems primarily related to the common acceptance of the word fairness.

3.4.3 Weak fairness and transparent TTP

Asokan introduced in [10] the notion of weak fairness in relation to protocols where fairness can be broken under certain circumstances. In a weakly fair protocol, a well behaving despoiled party is able, thanks to the help of the TTP, to prove his honesty to an external adjudicator. More precisely, if Alice does not receive her expected item, she will be able, if the case arises, to prove to an external adjudicator that the other party received the item sent by Alice or is able to retrieve her item without any further intervention from Alice. If the misbehaving party (who has not provided his item) refuses to cooperate, the TTP will transmit to Alice an affidavit in replacement of the missing information.

In practice, this property is interesting in protocols with a low weight TTP, when it is more relevant to obtain affidavits produced by the TTP than the expected low cost items. Note that one also has to define dispute resolution protocols, defining the way an adjudicator has to evaluate these affidavits, as it is the case in non-repudiation protocols. Weak fairness shows an interesting way of linking fairness and non-repudiation. Participants do not get a guarantee that they will obtain the intended item, but at least they get a non-repudiation evidence that the other party was involved in a particular run of the exchange protocol. Weak fairness may be of practical interest, but we restrict the remaining of our work to strong fairness, which we generally just refer to as fairness, without emphasising the "strong" feature.

Recent evolutions [16, 30, 67, 102] in optimistic exchange protocols with transparent TTP, based on verifiable encryption and recoverable signatures, offer solutions where it is possible to maintain strong fairness. In such optimistic protocols the TTP is always able to retrieve the original expected information in case of a problem, without needing the cooperation of the parties, to enforce fairness.

Moreover, with such a transparent TTP, at the end of a protocol where the exchange is realised, it is impossible to decide whether the TTP did intervene in the protocol execution or not. As it is difficult to determine whether the TTP was required during the protocol because of a dishonest party or because of a network problem, a transparent TTP may be particularly relevant, for example, in an electronic commerce environment.

3.4.4 Abuse-free digital contract signing protocols

In 1999, Garay et al. [67] introduced the notion of abuse-free digital contract signing protocols. An optimistic contract signing protocol is abuse-free if it is impossible for a single entity at any point in the protocol to be able to prove to an outside party that he has the power to either terminate (abort) or successfully complete the protocol. This property is motivated by the following kind of example. Alice wants to sell a house to Bob by means of a digital contract signing protocol. If the protocol is not abuse-free, Alice can meanwhile show Bob's offer to Charlie, who could be a second potential buyer. If Alice can convince Charlie that he can decide between accepting or rejecting Bob's offer, Alice can use this argument to make Charlie raise his offer. This is however undesirable for Bob. We will give a detailed analysis of this protocol and the abuse-free property in chapter 6.

3.4.5 Author-based selective receipt in certified e-mail

In [85], we introduce a new property, specific to certified e-mail, called *author-based selective receipt*. Generally, it is required that, once the message is known to the recipient, he cannot prevent the protocol from delivering the receipt corresponding to the given message. It seems to us that this property is not sufficient: we believe that once the identity of the author is revealed, the receipt has to be delivered to the sender, and of course the message to the recipient. In traditional certified mail, one does not get any information about the mail, neither the contents nor the origin, before having signed the receipt.

3.4.6 A general and modular definition of the security of an exchange protocol

In this section, we require the definition of an exchange protocol to explicitly refer to the items the protocol participants want to exchange. The properties below are defined with respect to the items exchanged, as referred to in the exchange protocol definition.

As suggested in [67, 70] we propose to speak about the *security* of exchange protocols. However, we do not consider that non-repudiability or abuse-freeness must imperatively be in the mandatory part of a definition of the security of an exchange protocol.

Definition 3.8 We say that an exchange protocol is *secure* when it respects these three mandatory properties.

- **Viability**: the communication channels quality being fixed, when all participants behave honestly, the exchange succeeds.
- **Fairness**: the communication channels quality being fixed, at the end of the exchange protocol run, either all involved parties obtain their expected items or none (even a part) of the information to be exchanged with respect to the missing items is received.
- **Timeliness**: the communication channels quality being fixed, the parties always have the ability to reach, in a finite amount of time, a point in the protocol where they can stop the protocol while preserving fairness.

Moreover, a secure exchange protocol can respect some optional properties, that may be specific to a given kind of fair exchange protocol. Examples are given hereunder.

- **Non-repudiability**: it is impossible for a single entity, after the execution of the protocol, to deny having participated[5] in a part or the whole of the communication.
- **Abuse-freeness**: it is impossible for a single entity at any point in the protocol to be able to prove to an outside party that he has the power to terminate (abort) or successfully complete the protocol [67].
- **No author-based selective receipt**: it is impossible for a recipient to prevent delivery of a receipt to the sender, once the author's identity is known to the recipient.

□

If we accept the common meaning of "fairness", we can consider that this section deals with three different aspects of fairness:

- fairness, as defined, relating to the items exchanged during the protocol;
- fairness relating to the ability to determine the progress of the protocol, called timeliness;
- fairness relating to the ability to make statements about the possibility to determine the progress of a protocol, called abuse-freeness.

Although these three properties can be bound to the common meaning of the word "fairness", they are related to fairness at different levels and should not be merged in one single definition. To avoid confusion, we prefer not to use the term fairness when talking about timeliness or abuse-freeness and put the emphasis on a modular and general definition.

3.5 Evolution of fair exchange protocols

In this section, we give a survey of existing exchange protocols and classify the protocols by the level of TTP intervention. This classification was chosen as it reflects, modulo some exceptions, the historic evolution of fair exchange protocols. Although we give a rather intensive survey of the literature, we may omit some papers, that we are simply not aware of, or which are similar to other cited papers and do not yield an important contribution to the evolution of fair exchange protocols. Moreover, we are not going to describe in details the different protocols, but rather give the basic idea of the different approaches. We are going to describe in detail some of the protocols, when we attempt to analyse them formally in chapter 6. For more details we refer the reader to [10, 87, 96, 124, 171, 172, 174] which give excellent—sometimes partial—surveys on the topic.

[5]Classical non-repudiation needs are non-repudiation of origin of a message and non-repudiation of receipt of a message.

3.5.1 Impossibility result

In 1980, Even and Yakobi [62] proved a fundamental impossibility result about digital contract signing protocols: no deterministic protocol solving digital contract signing may exist, without the intervention of a trusted third party. Although we do not have access to this technical report, the result has been announced and a brief sketch of the proof has been given by Even when presenting a contract signing protocol [60] in 1981. This result enforces the intuition that it is impossible to "simultaneously" exchange two items, without a "referee".

More recently, in 1999, Pagnina and Gärtner [123], gave a proof of the impossibility of fair exchange without a trusted third party[6]. They show that a fair exchange protocol can be used to implement an algorithm to solve the distributed consensus problem. This means that fair exchange is at least as difficult to solve as the consensus problem. However, it is well known that *there is no asynchronous, deterministic, 1-crash-tolerant consensus protocol* [64]. Hence, this impossibility result also applies to fair exchange protocols. The fact that, in fair exchange protocols, the involved parties do not trust each other reflects that they must be *1-crash-tolerant*, as the simplest attack of a malicious participant would be to just stop the protocol.

The above given result also implies that it is impossible to build a fair exchange protocol with only unreliable channels. Otherwise, all messages to the TTP could be lost, and the protocol would be equivalent to a protocol without TTP.

The fact that we are going to present fair exchange protocols *without* TTP in the next section does not contradict this result. Such protocols are possible, because they do not provide fairness, in the strict sense (the protocols tolerate minor advantages), and are not always deterministic (we are going to present probabilistic solutions).

3.5.2 Protocols without TTP

Although (inefficient) protocols without TTP were the first protocols proposed in the framework of fair exchange of secrets and digital contract signing, non-repudiation protocols without TTP were initially presented at the end of the 1990s [100] (thus, curiously, much later than protocols with TTP, on the contrary to fair exchange protocols).

3.5.2.1 Protocols based on gradual release of secrets

In the middle of the 1980s, fair exchange protocols were developed in order to achieve the exchange of secrets (e.g. secret keys) between two entities. The basic idea was that each entity transmits in turn successive bits of the information to be exchanged. This process continues until the last bit of each information (both informations to exchange are supposed to have the same size) was sent or until one of the two entities stops his participation in the protocol. The amount of computing necessary for each entity to retrieve the missing bits decreases at each step of the protocol. If the protocol is stopped before the information has completely been sent, and if the entities have the same computing power, there could be at most a difference of a factor two in the time needed for each entity

[6] We suppose that the authors were not aware of an existing proof. Anyway, the proof given in [123] differs fundamentally from the one given in [60] and is an interesting contribution to the field.

to retrieve the expected information. In order to reduce this difference of time needed to obtain the information, Tedrick [157, 158] shows how to transmit a fraction of a bit: rather than sending in turn one bit of each information, the entities transmit in turn a binary string which is different from the corresponding string of the information to be transmitted.

Whatever method is used to exchange the bits of information, one has to be able to detect whether an entity attempts to cheat (by sending incorrect bits). Many methods were suggested (e.g. based on the square root problem [157, 26]). A more generic method was proposed by Even et al. [61] using oblivious transfer techniques, such as defined in [143], page 549.

However, these previous methods have some disadvantages: strictly speaking fairness is not achieved, as these protocols allow a slight disadvantage in each round, the number of rounds of a protocol is large, which implies inefficiency, and they require that the communicating entities have the same or an equivalent computing power. This is unrealistic in practice (e.g. individuals versus large organisations). In 2001, Boneh and Naor [29], succeed in relaxing the requirement on the computing power. Their gradual exchange is based on a problem, that is hard to parallelise, which avoids the advantage in the case where high computing power is based on parallel architectures. However, their protocol confirms the "folklore theorem" stating that it is impossible to design fair exchange protocols, based on gradual exchange, having a small number of rounds.

3.5.2.2 Ben Or-Goldreich-Micali-Rivest probabilistic contract signing protocol

In 1990, in the context of digital contract signing protocols, Ben Or et al. [25] proposed to exchange privileges rather than bit information. An entity is said to be more privileged than other ones when it has a greater ability (than the other entities) to convince an external judge that the contract is signed by all the participating entities. The presented two-party protocol is such that the entities are privileged in turn. During the protocol, each entity sends to the other one a message saying that with a probability λ the contract will be valid (signed by both parties) at the (previously agreed) moment D. The probability λ has to increase during each round of the protocol. The protocol ends when $\lambda = 1$ or when the deadline D is reached. After the moment D, each party can present to the judge the last received message. The judge, once, chooses randomly a value between 0 and 1 and compares this value with the probability λ extracted from the message. If λ is greater than or equal to the chosen value, the judge declares the two parties linked to the contract, otherwise he states that λ is too small. The decision is provided to both parties. Although the hypothesis on equivalent computational power is not needed, the protocol requires a large number of messages, making it inefficient.

3.5.2.3 Rational exchange protocols

No other methods without TTP were proposed until the second half of the 1990s. In 1998, Syverson proposed protocols [152] where low value information to be exchanged are ciphered and sent with a commitment to the key used to cipher them. These commitments will, in practice, be breakable if enough time is invested. This amount of time is estimated on the base of a known computing power. Such a commitment is called temporally secret bit commitment and could be implemented via a time-lock puzzle [134]. Such a protocol is not a *fair exchange protocol* as introduced above, and has been called *rational exchange*. The idea is that a party who wants to cheat will not *gain* anything, but can merely harm the other participant.

In 1995, an earlier protocol by Jakobsson [81], that was falsely named fair exchange, already implements this idea. When a customer wants to buy a digital good, he starts by *ripping* the electronic coin, used for payment, in two. He initiates the protocol by sending the first half of the coin to the vendor. Now the coin has been spent and the money is lost, but the vendor cannot yet use this ripped coin. In order to receive the other half of the coin, he must send the desired goods. Once the customer obtains the goods, he sends the second half of the coin. Not sending the second half of the coin will not give any profit to the customer, as the coin has already been spent. Hence, the mere advantage the customer has, is the possibility to harm the vendor by not paying. Obviously, fairness is not maintained.

Rational exchange protocols seem attractive for low value transactions as they provide efficiency without the use of a TTP. However, one must notice that rational exchange offers the possibility for vandalism, where the aim is not to maximise profit, but to harm.

3.5.2.4 Markowitch-Roggeman probabilistic non-repudiation protocol

The first fair non-repudiation protocol without TTP was proposed by Markowitch and Roggeman in 1999 [100]. The goal of this protocol is to avoid the intervention of a TTP at the price of accepting the probabilistic version of fairness. The protocol has to be parameterised on the basis of the most powerful entity's computing power. This iterative protocol is such that, except at the last iteration, no entity is more privileged than another one during the protocol. This is one of the most efficient and secure probabilistic protocols and we give a more detailed description hereunder.

Being freed from a TTP during exchanges not only makes it possible to avoid a bottleneck in the communications but also permits to relax the need of trust in a TTP. The honesty of a TTP is difficult to evaluate. In the protocol here, the risk is known and can be parameterised.

Suppose Alice wants to send a message m and a non-repudiation of origin evidence of this message to Bob in exchange against Bob's non-repudiation of receipt evidence. The protocol is such that Alice will not find it beneficial to stop the protocol before its end. In the same way, if Bob stops the protocol before the last couple of sendings, he will not gain any profit. The only way for Bob to cheat, i.e. obtain Alice's message and the corresponding non-repudiation of origin evidence, without acknowledging receipt (by means of sending a non-repudiation of receipt evidence), is to guess the number of iterations in the protocol. After each iteration Alice launches a $\frac{1}{\varepsilon}-$faced die. Alice stops the protocol when the result gives one, and continues otherwise[7]. At each iteration, the probability that Bob obtains the message and the non-repudiation of origin evidence without sending the non-repudiation of receipt will be smaller or equal to ε.

Alice initiates the protocol by sending to Bob the cipher $c = E_k(m)$, as well as the

[7] In the original paper, the authors suggest to choose a random value, following a geometric distribution. This is equivalent to the die launching described here.

corresponding non-repudiation of origin evidence. Bob acknowledges the reception of this cipher. Then, until reaching the final round, during each round, Alice sends a random number and the corresponding evidences of origin to Bob. These random numbers are potential keys to decipher c. Bob sends to Alice the non-repudiation of receipt evidence for the given random number. During the final round, Alice transmits the deciphering key k (related to the cipher c) and Bob acknowledges the reception of this key which is indistinguishable from the already received random values. After a known delay or after having received a notification from Alice, Bob computes $m = D_k(c)$.

Before the last sending of Alice, Bob did not receive anything usable. Moreover, the only way he can detect whether he received the key k is by deciphering c using the value he received from Alice. But this computation will be supposed too long compared to the time before which Alice stops the protocol, not having received the expected non-repudiation of receipt evidence for the given value. We thus have to choose a cryptosystem whose performances are appropriate with regard to the message size and the time Alice waits before stopping the protocol. For the protocol to work correctly, it is necessary to use a cipher, for which it is impossible to partially decrypt the ciphertext, in order to quickly decide whether the obtained value is the correct key or not. If, for instance, a block cipher is used, Alice must cipher her message, whatever the cryptosystem used, by means of a mode where all the blocks of the ciphertext must be deciphered to be able to obtain any block of the plain text. Such a mechanism is described in [100]. It is possible that an all-or-nothing ciphering mode may not generate sufficient delay for deciphering short messages. Then, a delaying mechanism, like [134] for example, could be used. Note that the protocol needs operational channels to be viable, but provides fairness even on unreliable channels.

3.5.2.5 Mitsianis probabilistic non-repudiation protocol

More recently, Mitsianis [117] proposed a protocol that is similar to Markowitch and Roggeman's protocol [100]. Classically, Alice begins by sending to Bob the cipher c of the message m under a secretly chosen session key K_σ. If Bob acknowledges the reception of this ciphered message, Alice adds a padding ω to the session key K_σ to compose the key K'_σ. The size of the padding is chosen secretly by Alice. Alice ciphers this new padded key with a session key K_λ shared with Bob in order to obtain the ciphered key K_ς. This last ciphering is obtained using a mode similar to the one described in the previous protocol in order to require Bob to obtain all the bits of the ciphered key K_ς to be able to decipher it.

Alice then chooses randomly and secretly, as in the previous protocol, a value n and splits K_ς in n parts κ_i of different random lengths. Then n 2-moves iterations begin. At each iteration, Alice sends to Bob a κ_i (in the order she splits K_ς) and Bob has to acknowledge having received it. When Bob acknowledges the reception of the last part κ_n of K_ς, Alice possesses the non-repudiation of receipt of the message m and Bob could retrieve the session key needed to recover the message m (he composes K_ς by concatenating the different received κ_i, deciphers the ciphered key using K_λ, obtains K'_σ, knowing the size of the session key K_σ he can extract the padding ω and retrieve K_σ; with K_σ he deciphers c and retrieves the message m).

Bob has to obtain all of the κ_i in order to retrieve K_σ, even if he knows the size of K_σ, because of the *all-or-nothing* mode used to cipher K'_σ and his unawareness of the length of ω. Obviously, Alice cannot cheat, since she first needs to send out all the expected information, in order to obtain a valid non-repudiation of receipt evidence.

Exactly as in the previous protocol, if Bob does not send the last acknowledgement (of κ_n), he obtains the message m without providing a complete non-repudiation of receipt evidence with probability $\frac{1}{n}$. Also, Alice waits until a fixed deadline for each acknowledgement of Bob for κ_i before deciding that Bob is trying to cheat and stopping the protocol. A major difference between Mitsianis' protocol and Markowitch and Roggeman's one is that in Mitsianis' one the value n, determining the number of protocol's rounds, is static whereas n is dynamic in Markowitch and Roggeman's one. At the beginning of Mitsianis' protocol, Alice chooses n which will be fixed for the remaining of the protocol. In the Markowitch-Roggeman protocol, the number of rounds is dynamic, as the decision to stop or continue could be taken after each completed round, by launching a $\frac{1}{\varepsilon}-$faced die.

3.5.3 Protocols with inline TTP

An inline TTP was first used in the context of certified e-mail protocols [19, 176]. These protocols are the most trivial ones and make the TTP intervene in each message sent during the protocol. They generally simply send all information to be exchanged to a TTP that acts as an intermediary. Those protocols put a heavy workload onto the TTP which may become a communication, computation and/or storage bottleneck. We therefore only give one example here. In 1996, Coffey and Saidha [49] proposed a non-repudiation protocol which illustrates well the use of an inline TTP. Coffey and Saidha used the TTP as a *non-repudiation server*. The TTP collects the non-repudiation evidences and transmits them to Alice and Bob thereafter. The protocol makes use of time-stamps produced by a time-stamping authority as well as resilient communication channels between each of the entities and the TTP. The time-stamping authority does not consider the content of the received messages (contrarily to what is sometimes awaited from a TTP). It just adds a time-stamp to a message signed by the entity with whom it communicates. To ensure the exchange of the non-repudiation evidences, the protocol uses partial evidences. Such evidences are known as partial because they are only a part of the final non-repudiation evidences. It should also be noticed that the protocol ensures confidentiality of the message transmitted by Alice to Bob.

Alice initiates the protocol by submitting, to the time-stamping authority, the signed partial non-repudiation of origin evidence to be dated and signed by this authority. If Alice's signature is valid, the time-stamping authority replies with the non-repudiation of origin evidence, ciphered for Alice. If this non-repudiation of origin evidence is valid (including the time-stamp), Alice requests the TTP to initialise a non-repudiable communication and sends the "final" non-repudiation of origin evidence as well as the partial non-repudiation of receipt evidence she computed to the TTP (the TTP could compute this partial non-repudiation of receipt evidence, but Alice carries it out in order to decrease the load of the TTP). If both the message sent by Alice and her signature are valid, the TTP sends the partial non-repudiation of receipt evidence to Bob. Bob signs this partial evidence and sends it to the time-stamping authority which, if the signature is correct, responds to Bob with the "final" non-repudiation of receipt evidence. Bob submits this evidence to the TTP. The TTP, after checkings, sends the non-repudiation of origin evidence to Bob and the non-repudiation of receipt evidence to Alice.

To distinguish different protocol sessions using a same TTP, the authors considered the use of a randomly chosen value, a nonce, associated to each session of the protocol. These random values are produced by the TTP and transmitted, by the TTP, to both Alice and Bob. Alice and Bob must present the random value they received during each communication with the TTP.

This protocol ensures fairness, since the TTP collects all information necessary before forwarding them to the concerned entities. It should be noted that Alice and Bob never communicate directly. The TTP is used as a go-between in each transmission, and thus is inline.

The authors did not propose the use of maximum times after which a message is considered as lost. As presented by the authors, the protocol does not respect the timeliness property. However, if a deadline is fixed for each sending, the protocol is timely finite.

In case of disputes, an external judge can be invoked. If Alice affirms to have successfully sent a message to Bob or if Bob affirms to have received a message from Alice, the judge requests to the complaining entity or asks the TTP to provide the non-repudiation of receipt and/or non-repudiation of origin evidence. If these evidences cannot be provided, the judge rejects the complaint. Otherwise, the judge checks the signatures and the time-stamps on the evidences. If all these checks succeed, Alice's or Bob's claim is accepted.

Protocols with inline TTP present several disadvantages. First, they require the management of large databases by the TTP. It must preserve the messages it forwards, as well as the moments of each event. The management of such a database of centralised sensitive information represents a significant security risk. It is advisable to take particularly care of the protection of the information managed by such a TTP. Lastly, the bottleneck produced by the flow of information forwarded by the TTP is maximum. On the other hand, on the contrary to online or offline TTP, an inline TTP can include information about the time a message is sent or received into the evidences. Such evidences can for instance be used for settling disputes about a late submission, by checking the timestamp.

Consequently, an inline TTP requires a particularly significant confidence, as well as the management of a considerable quantity of centralised resources.

3.5.4 Protocols with online TTP

The protocols based on an online TTP are such that the TTP does not act anymore as a delivery authority (as an intermediary for each transmission between the entities, as it is the case with an inline TTP). However, an online TTP still intervenes more or less heavily during each session of the protocol.

Most of the protocols using an online TTP were proposed between 1995 and 1997. As we will see shortly, the first protocol we are going to describe is an exception and was proposed much earlier. In 1995, Cox, Tygar and Sirbu [54] proposed a method for purchasing digital goods and services on the Internet. They use a trusted third party, called the *netbill server* to ensure *atomicity*, the name they gave to what we call fairness. In 1996, Deng et al. [55] proposed a certified e-mail protocol. Their major contribution is that the protocol was designed to be integrated into an existing e-mail protocol. In the following paragraphs, we describe three examples, which are good illustrations of protocols using an online TTP.

To introduce protocols with online TTP, we will exceptionally make a return in time and evoke a protocol suggested by Rabin [132] in 1983. The author proposed an original method, called method by beacons, making it possible to carry out the exchange of an information against an acknowledgement. In this approach, the use of the TTP extremely differs from the traditionally developed methods.

3.5.4.1 Rabin's beacons protocol

This protocol is the first exchange protocol making use of a TTP. The idea is to have a TTP, broadcasting at regular and fixed intervals of time a signed message called the beacon. This beacon is composed of n public keys, plus one deciphering key corresponding to one of the public keys sent during the previous broadcasting and a value j indicating which previously broadcasted public key is associated with the presently broadcasted deciphering key.

In a first time, Alice sends to Bob a message ciphered under a session key k. During the protocol, Alice and Bob communicate directly and have to realize a complete round of the protocol between two broadcastings of the TTP. During a round, Bob randomly chooses an integer i between 1 and n. Bob sends this integer to Alice with a non-repudiation of receipt evidence for the message Alice would like to send to Bob. Then, Alice sends a signed message containing the session key ciphered with the i^{th} public key broadcasted by the TTP in the beacon received just before initiating the current round. If, when the TTP broadcasts the beacon, the value j chosen by the TTP is the same as the value i chosen by Bob, then Bob can retrieve the session key k and Alice's message. Otherwise, we start an additional round. The non-repudiation of origin and receipt evidences are composed of the signed messages sent by Alice and Bob during this round associated with the beacon.

This protocol is probabilistically fair. Alice can decide to stop the protocol when she receives the signed message from Bob and before sending her signed message. In that case, fairness is only broken if Bob and the TTP choose the same value ($i = j$), as Alice received a complete non-repudiation of receipt evidence, without Bob receiving a complete non-repudiation of origin evidence. This situation can happen with a probability equal to $\frac{1}{n}$. If the TTP and Bob choose different values ($i \neq j$), then the message sent to Alice is not a valid non-repudiation of receipt evidence and Alice does not gain anything.

However, this protocol respects a probabilistic version of the timeliness property. We have that the protocol is stopped after the kth broadcasting with probability $\frac{1}{n}(\frac{n-1}{n})^{k-1}$, which corresponds to the geometric distribution. Although the probability that $i \neq j$ during the first r rounds, decreases when the number of rounds r increases, this probability never reaches zero (due to the asymptotic behaviour of the geometrical distribution). The expected number of rounds to end the protocol equals n.

In order to prevent forwarding of the non-repudiation evidences, we suggest that, when Alice and Bob send their signed message to each other, they add the recipient's identity in the messages. Otherwise, Alice could forward her non-repudiation of receipt evidence to Charlie, who would have a valid receipt for a message he never sent.

This protocol, called "confidential disclosure protocol" by the author is rather connected with our definition of a non-repudiation protocol. This probabilistic approach of fairness is comparable to the non-repudiation protocols without TTP presented at the section 3.5.2.

It should also be noted that the need for synchronization between the entities is a heavy constraint to realize. Therefore the deadline between two broadcastings must be parameterised so that the entity having the smallest computing power can provide the necessary message within the interval.

3.5.4.2 Zhang-Shi non-repudiation protocol

In 1996, Zhang and Shi [170] proposed a protocol where Alice transmits to Bob the message ciphered with a session key (and ciphered again by means of Bob's public key) and where the TTP publishes, at the right moment (decided by Bob and agreed by Alice), in a publicly accessible way, e.g. a public board, the session key (provided to the TTP by Alice) needed to retrieve the message that Alice wants to transmit to Bob. Moreover, the TTP manages a database containing the keys used during a protocol run and records the time when these keys are added in that database. In this protocol, in case of dispute, the judge, resolving the given dispute, has to contact the TTP in order to decide which entity is honest. For this reason, the TTP cannot delete any information stored in the database. Such a database grows indefinitely.

The protocol ensures the confidentiality of the transmitted message (even once the session key is revealed, since the ciphered message is ciphered again with Bob's public key). The protocol also ensures the confidentiality of the message with respect to the TTP.

3.5.4.3 Zhou-Gollmann non-repudiation protocol

Also in 1996, Zhou and Gollmann [177] presented a non-repudiation protocol with online TTP. The idea of this protocol is to reduce the work of the TTP to a minimum.

During the protocol, if an incorrect message arrives or if an awaited message does not arrive, the potential recipient stops the protocol.

Alice initiates the protocol by sending the cipher, using session key k, of the message she wants to transmit to Bob, a label identifying the protocol session, a time-out value before which the session key must be submitted to the TTP and after which it can be consulted, as well as the signed non-repudiation of origin evidence for the ciphered message. If Bob accepts the consultation time-out proposed by Alice, he sends to her his signed non-repudiation of receipt evidence for the ciphered message. Alice then sends to the TTP a signed copy of the session key. The TTP accepts only one submission for each protocol session and checks whether Alice's signature is valid and whether the time-out is not exceeded. After the time-out, Bob can get the session key and the non-repudiation of origin evidence for this session key provided by the TTP. This evidence is necessary in order to build a complete non-repudiation of origin evidence for the message that Alice sent to him. In a similar way, Alice consults the TTP to complete her non-repudiation of receipt evidence for the message.

Both Alice and Bob will fetch the session key and the corresponding evidence for this key at the TTP. This evidence serves to Bob as an evidence of origin and to Alice as a proof that the key is accessible to Bob. The entities consult, at the proper time, a read-only public directory managed by the TTP. If one of the entities cannot get the evidence at the TTP, while the other entity does, he will lose a possible future dispute on this subject.

The work of the TTP is thus reduced by rejecting the responsibility for obtaining the information managed by the TTP on the entities. The protocol requires for its good functioning a resilient channel of communication between the TTP and each entities.

If the communication channels between the TTP and respectively Alice and Bob are resilient, the protocol is fair and the protocol respects the timeliness property.

In case of dispute, if Alice claims to have successfully sent a message to Bob, the judge asks her to provide this message and the non-repudiation of receipt evidence for this message. The non-repudiation evidence is composed of the non-repudiation of receipt evidence for the ciphered message, provided by Bob, and the non-repudiation of origin evidence of the session key, provided by the TTP. If all the information provided by Alice to the judge is correct, the judge declares that the assertion of Alice is correct.

If Bob claims to have received a message from Alice, the judge asks him to provide this message as well as the non-repudiation of origin evidence for this message. This evidence is composed of the non-repudiation of origin evidence for the ciphered message, provided by Alice, and the non-repudiation of origin evidence of the session key, provided by the TTP. If all the information provided by Bob to the judge is correct, the judge declares that the assertion of Bob is correct.

The protocol does not propose mechanisms ensuring the confidentiality. The session key and the ciphered message are accessible to any observer.

3.5.4.4 Franklin-Reiter fair exchange protocol

In 1997, Franklin and Reiter proposed a fair exchange protocol with a “semi-trusted” TTP. By “semi-trusted”, they mean that the TTP cannot learn anything about the exchanged items by her own, i.e. without colluding with a dishonest party participating in the protocol.

The protocols aims to exchange two secret keys K_A and K_B between Alice and Bob in a fair way. Moreover, the authors suppose that each entity is in possession of the hash value of the key they want to receive. Alice and Bob start by exchanging two random values x_A, respectively x_B, chosen from a group G. In a second phase, Alice sends

$$h(K_A), h(K_B), K_A x_A^{-1}, h(x_B)$$

to the TTP, and Bob does so for

$$h(K_B), h(K_A), K_B x_B^{-1}, h(x_A).$$

The hash function $h(\ )$ is chosen in such a way that $h(\alpha)h(\beta) = h(\alpha\beta)$. This enables the TTP to check the consistency of the sent messages, by checking that $h(K_A) = h(K_A x_A^{-1})h(x_A)$ and $h(K_B) = h(K_B x_B^{-1})h(x_B)$. If these tests succeed the TTP forwards $K_A x_A^{-1}$ and $K_B x_B^{-1}$ to Bob, respectively Alice. Then Alice and Bob can recover the expected keys K_B and K_A respectively. As the TTP does not know x_A nor x_B, she cannot learn anything about K_A and K_B. However for this to hold we must make the additional hypothesis that the TTP cannot eavesdrop the channel between Alice and Bob. Otherwise the confidentiality of the exchanged keys is not provided.

3.5.5 Protocols with offline TTP

In this section we survey most of the existing protocols with offline TTP. A TTP is said offline if it does not intervene in the protocol while no problem occurs. A problem could be an incorrect behaviour of a dishonest entity or a network error. When such a problem occurs, Alice and/or Bob invoke the TTP to help them to finish the protocol run in a fair way. Such protocols suppose that most of the time no problem will occur. This is the reason why protocols with offline TTP are also called *optimistic*.

The first protocols, a certified e-mail protocol and fair exchange protocols, making use of an offline TTP were presented, independently, in [13] and [111].

For (strong) fairness to be possible, at least one of the exchanged items must either be *generatable* or *revocable* by the TTP. Most of the work in fair exchange is based on generatable items, while revocability was only studied in more detail very recently. These ideas on generatability and revocability were first mentioned in [13], but the protocols presented in that paper, could only guarantee fairness in presence of operational channels. However, as these channels are irrealistic in practice, their protocols are mainly of theoretical interest. Therefore, we are not going to describe them.

3.5.5.1 Generatability by affidavits

In many contexts, an item can be replaced by an affidavit issued by a TTP. For instance, in a non-repudiation protocol, a non-repudiation evidence can be replaced by an evidence signed by the TTP. If one of the parties can gather enough information to prove to the TTP that the other party has obtained or has the possibility to obtain his expected evidence, the TTP can replace an evidence by one issued by himself. The same may also apply to contract signing and certified e-mail protocols. This is the reason, why most of the presented protocols, are either non-repudiation, contract signing or certified e-mail protocols.

Zhou-Gollmann optimistic non-repudiation protocols. In 1997, inspired by Asokan et al.'s optimistic approach [13], Zhou and Gollmann [178] presented a non-repudiation protocol. Their aim was to present an efficient optimistic protocol, that could relax the strong hypothesis on the communication channels. The idea of the protocol can be resumed as follows. During the first two messages, Alice exchanges a ciphertext c and a non-repudiation of origin evidence for this ciphertext against a non-repudiation of receipt evidence for c. Then, in messages 3 and 4, Alice exchanges the decryption key k and a corresponding evidence of origin for k against an evidence of receipt for k. The message, sent by Alice to Bob, and acknowledged by Bob, corresponds to the deciphering of c under the key k. The above described scenario corresponds to the main protocol, in which no TTP intervenes. If a problem occurs, e.g. Bob does not send a receipt for the decryption key, Alice can contact the TTP in order to *recover* the protocol. In that case, the TTP publishes k and provides Alice with an evidence that k is accessible to Bob. This proof replaces Bob's non-repudiation of receipt evidence for k.

However, the protocol as described above does not provide timeliness. Alice can decide to stop the protocol after the first two messages, or recover the protocol at some later time. However Bob has no way of knowing when—and even whether—the key k will be accessible at the TTP. Therefore Zhou and Gollmann have included a time-out t in the first two messages. After the time t, the TTP is not going to accept the recovery anymore. Now, Bob only needs to verify whether the key has been published at time t or not. However, as we high-lightened in [84], this protocol also needs operational channels to be fair. We are going to explain this fact in more detail, when formally analysing this protocol in chapter 6.

Asokan-Shoup-Waidner optimistic fair exchange protocols. In 1998, Asokan, Shoup and Waidner [14] propose protocols that achieve fairness in presence of resilient channels. In order to achieve this, their protocols are composed of a main protocol, for the faultless case, an abort protocol that can be executed by Alice, and two recovery protocols, which can be executed by either Alice or Bob. The main protocol is composed of four messages and similar to the previous protocols. In the first two messages, Alice and Bob exchange a *promise* to exchange the expected items, and in messages 3 and 4, they exchange the actually expected items. If Bob does not obtain the first message, he can stop the protocol. If Alice does not receive the second message, she can contact the TTP to *abort* the protocol. Alice is given the possibility to abort the protocol as she has an inherent disadvantage: once Bob has received the first message, he is able—and could do so if he is dishonest—to launch a *recovery* protocol, while Alice is not. When Bob does not receive the third message, or Alice does not receive the fourth message they can launch recovery protocols. It is of crucial importance that the abort protocol and the recovery protocols are *mutually exclusive*. The abort protocol is a promise of the TTP to Alice, not to accept any recovery protocol for the given protocol instance[8]. In a recovery protocol, Alice, respectively Bob, sends the first two messages of the main protocol to the TTP. With these messages the TTP should be convinced that Alice and Bob started an exchange and the TTP forces the successful outcome by issuing a replacement for the missing item.

This scheme, which involves one main protocol and three sub-protocols, serves as a "template" for most optimistic fair exchange protocols based on generatability. As we will see later, although the general scheme is correct, it is not easy to give a correct instance avoiding flaws. For instance, Zhou, Bao and Deng [175] found several flaws in the certified e-mail protocol given in [14]. These weaknesses and others will be discussed in more detail, when formally analysing the protocol in chapter 6.

Zhou-Bao-Deng and Kremer-Markowitch non-repudiation protocols. In 1999, Zhou, Bao and Deng [174] presented a non-repudiation protocol based on the idea of Asokan, Shoup and Waidner of having three sub-protocols in addition to the main protocol. In 2000, in collaboration with Olivier Markowitch, we [84], not yet aware of Zhou et al.'s work, presented a similar non-repudiation protocol, differing only in some apparently minor technical details. Both protocols are an adaptation of Asokan et al.'s work [14] to the context of non-repudiation. However, in 2000 Boyd and Kearny [31] found a minor design flaw in Zhou et. al.'s presented protocol, which does not appear in ours.

[8] Therefore each protocol run must include a unique protocol identifier allowing the TTP to distinguish different protocol sessions.

3.5.5.2 Generatability and transparent TTP

In the previous protocols with offline TTP, when the TTP intervenes, in case of problems during the communication between Alice and Bob, the TTP digitally signs some pieces of information which will be used as an affidavit. These evidences have the same effect to an adjudicator as those produced by Alice and Bob in a faultless case.

The aim of the protocols described hereunder is to have a *transparent* TTP. At the end of the protocol, by only looking at the produced evidences, it is impossible to decide whether the TTP did intervene in the protocol execution or not. The motivation for transparent TTP is twofold. As the intervention of the TTP can be due to a network failure, rather than a cheating party, transparent TTP can be very useful in the context of electronic commerce, in order to avoid bad publicity, i.e. an external party when seeing evidences issued by the TTP could falsely conclude that a party was trying to cheat, while a network failure was at the origin of the TTP intervention. A second motivation is the *trust* we have to put in the TTP: when the TTP is transparent, the TTP is generally unable to issue one of the expected items outside of the protocol. In the previously proposed protocols, the TTP could issue an affidavit on behalf of Alice, without her being aware of this. When using convertible signatures as it will be the case hereunder, the TTP will be unable to issue a signature without a commitment to a signature by a given party.

The use of a transparent TTP, also called *invisible*, was first proposed by Micali [111] in the framework of certified e-mails. In 1998, Asokan et al. [15] and Bao et al. [20], proposed fair exchange protocols allowing to recover, in case of problem, the original client's signature rather than affidavits produced and signed by the TTP, by means of verifiable convertible signatures. The idea of such signatures is that a protocol party, say Alice, sends in a first step a commitment to a given signature to Bob. This commitment is particular in the sense that a trusted third party can *convert* it into a real signature. Moreover, Bob can verify that the commitment contains the expected signature and that it is convertible by the TTP. Asokan et al.'s protocol is based on verifiable encryption, which is however computationally inefficient. Bao et al. proposed two protocols, from which the first one is inefficient, while the second one, though more efficient, has been broken by Boyd and Foo [30]. In the same paper, Boyd and Foo [30] proposed a fair exchange protocol for electronic payment. Their method allows to recover the original client's signature from the committed one, using designated convertible signatures [46]. They also proposed a concrete protocol based on the RSA signature scheme. However, their scheme requires an additional interactive protocol and hence is rather inefficient. They argue that the commitment must not be universally verifiable, otherwise it could be interpreted as a digital signature and hence the protocol requires an interactive proof of the correctness of the commitment. We do not agree with this, as a commitment, although it may be universally verifiable, does not have the same semantics as a digital signature, and can be defined not to have any legal value.

In 1999, Ateniese [17] presented a scheme based on verifiable encryption, that is more efficient than Asokan et. al's one. However, the most efficient protocol for fair exchange with transparent TTP has recently been proposed by Markowitch and Saeednia [102]. In the following paragraphs, we will discuss this scheme in more detail as well as a more peculiar scheme presented in 2000 by Garay, Jakobsson and MacKenzie.

Markowitch-Saeednia fair exchange protocol. The protocol is based on a specific signature scheme (inspired by the Girault-Poupard-Stern signature scheme [71, 131]). It does not need an additional interactive protocol and is efficient considering both communication and computation. In this protocol, when a fault occurs during the main protocol execution, the TTP produces exactly the same evidences as those produced by Alice and Bob in a faultless case. We give a detailed description of the signature scheme below, as it is very easy to describe, and gives a good illustration of verifiable convertible signatures.

The signature used in the Markowitch-Saeednia protocol is issued in two phases. First the signer produces a committed signature. Then this committed signature is turned into a final signature either by the signer or by the TTP. The recipient of a committed signature is able to check whether the TTP has the ability to transform the committed signature into the signer's final signature.

During an initialisation phase, the TTP chooses an integer $n = pq$, where p and q are large random strong primes (of almost the same size). The TTP also chooses a base α of order $\lambda(n)$, where $\lambda(n) = lcm(p-1, q-1)$, and a small integer c such that $\gcd(\lambda(n), c) = 1$. The TTP computes d such that

$$cd \equiv 1 \pmod{\lambda(n)}$$

and

$$\beta = \alpha^c \mod n$$

Finally, the TTP makes n, β, c and α public, keeps d secret and discards p and q.

A signer u chooses a random integer x_u as a secret key and computes the corresponding public key $y_u = \alpha^{x_u} \mod n$. As usual in public key cryptography, a certificate for the public key has to be obtained and distributed.

To produce the committed signature on a message m the signer u chooses a random r_u and computes

$$t_u = \beta^{r_u} \mod n$$

and

$$z_u = c \cdot r_u + h(t_u, m) \cdot x_u$$

The pair (t_u, z_u) forms the committed signature of signer u and will also be noted $\mathsf{ComSig}_u(m)$.

A verifier v can check the committed signature by comparing $\alpha^{z_u} \mod n$ and $t_u \cdot {y_u}^{h(t_u,m)} \mod n$.

The final signature of signer u can be generated independently by the signer u by computing

$$t'_u = \alpha^{r_u} \mod n$$

or by the TTP by computing

$$t'_u = {t_u}^d \mod n$$

The pair (t'_u, z_u) forms the final signature of signer u and will also be noted $S_u(m)$.

The verifier checks the validity of the final signature by comparing $\alpha^{z_u} \mod n$ to ${t'_u}^c \cdot {y_u}^{h({t'_u}^c \mod n, m)} \mod n$ (in practice, it is sufficient to verify that $t_u = {t'_u}^c \mod n$, as this test implies the previous one).

The protocol itself is based on the classical scheme of one main protocol, and three sub-protocols, one for abort and two for recovery. We also used this signature scheme to design a non-repudiation protocol in collaboration with Olivier Markowitch [99], providing a transparent TTP.

Garay-Jakobsson-MacKenzie contract signing protocol. In 1999, Garay et al. introduced a new primitive, called *private contract signature*, or PCS for short. Note that a similar notion was also introduced by Chaum in patent [47] under the name "private signatures". The aim of PCS is to combine designated verifier signatures [82, 47, 139] and convertible signatures. If Alice uses a PCS to sign a message, meant for Bob, Bob and a given TTP are the only persons who can be convinced by the authenticity of the signature. This is achieved by giving Bob the possibility to *simulate* the signature. Hence, only Bob (and the TTP) can be sure that the signature is not a simulation. Moreover Bob can verify that the TTP can convert the PCS into a conventional signature. In [67], PCS are defined in a probabilistic setting. We will define PCS as *black boxes*, as the "cryptographic" definition is not important for this work.

Definition 3.9 A $PCS_A(m, B, T)$ is A's private contract signature on message m meant for B, with respect to a third party T. A PCS provides

- the designated verifier property, i.e. only B can be sure of the authenticity of the PCS;
- convertibility, i.e. $PCS_A(m, B, T)$ can be converted to a universally verifiable signature $S_A(m)$ by T;
- verifiability, i.e. B can verify that T is able to convert the signature into A's universally verifiable signature.

□

In [67], two versions of this new primitive are presented. The first one provides TTP-accountability, i.e. the universally verifiable signatures produced by A and T are distinguishable. The second one provides TTP-invisibility, i.e. the universally verifiable signatures produced by A and T are undistinguishable. To simplify notations we only consider the second case. This new primitive was motivated by the *abuse-free* property discussed in section 3.4, where it is important that a signature commitment cannot convince a third party Charlie, that a given protocol has been started. Consider for instance the following scenario. Alice sends a PCS to Bob, as a signature commitment. Bob is unable to convince Charlie that the protocol has been started, as Bob could *simulate* the PCS himself. This means that the simulated PCS would be indistinguishable to Charlie from the PCS issued by Alice.

Remark 3.10 Note that PCS give a stronger guarantee than the designated confirmer signatures used by Boyd and Foo in [30]. Boyd et al. used zero-knowledge protocols to realize the verification of the commitment. Although these proofs cannot be transferred, and are not universally verifiable, they are not designated verifier. As for undeniable signatures, one can mount blackmailing [57, 80] and mafia attacks [56] against designated confirmer signatures, i.e. the challenge in the zero-knowledge protocol could be chosen by a group of persons who would be convinced about the origin, or a man-in-the-middle attack can easily be realized. □

Before ending the section, for the sake of completeness we also cite an attempt by Wu and Varadharajan [168] in 2002 to design convertible signatures for fair exchange. However the scheme was fundamentally broken by Markowitch and Saeednia [101].

3.5.5.3 Revocability

The idea of revocable items was given by Asokan, Schunter and Waidner [13] in 1997. However, they only proposed protocols that require a synchronous setting, i.e. operational channels. Only in 2003, Vogt [166] designed a new protocol that works on resilient channels. The new protocol was motivated by the fact that revocability is easier to implement for most popular payment schemes than generatability. Moreover, his protocol is more efficient than the traditional pattern, which requires a 4 message main protocol, an abort protocol and two recovery protocols. Vogt's protocol only needs a three message main protocol, one recovery protocol accessible to Bob, and an abort protocol that can be launched by Alice. Although this protocol seems to be advantageous, one should however note that revocation is less desirable: whenever two parties engage a protocol, this should mean that they prefer a successful ending to a revocation of the exchange. Fortunately, in Vogt's protocol, revocation is only needed if Alice misbehaves. Otherwise the successful ending can be forced by the TTP.

Vogt claims that this is the only protocol exploiting revocability in an asynchronous setting. To the best of our knowledge this is true, in the strict sense of revocability. However, in 2000, Ferrer et al. [63] designed a certified e-mail protocol, that also implemented a main protocol with only three messages. Instead of revocating an item the TTP could issue an abort item given to Bob if Alice does not send the third message. Whenever Alice wants to prove to a judge that Bob received a given message, the judge needs to contact Bob who might be able to present an abort token to reject Alice's claim. The abort token can then be seen as a revocation of the proof Alice received during the protocol. The need of contacting Bob, whenever one wants to check the validity of an evidence, is of course a major drawback of the protocol and considerably limits its usability. It remains an open question, whether "real" revocability can be implemented for digital signatures, in order to revoke non-repudiation evidences or signatures on contracts.

3.6 Multi-party fair exchange

More recently than for the two-party case, in literature, different kinds of *multi-party fair exchange* have been considered. In [66], Franklin and Tsudik propose a classification, depending mainly on the *topology* of the exchange. While two-party non-repudiation, contract signing, or more generally fair exchange protocols are very similar, the different topologies corresponding to particular kinds of fair exchange increase the diversity of the protocols in the multi-party case.

Bao et al. [21] and Franklin and Tsudik [66] concentrated on a ring topology. Each entity e_i $(0 \leq i \leq n-1)$ desires an item (or a set of items) from entity $e_{i \boxminus 1}$ and offers an item (or a set of items) to entity $e_{i \boxplus 1}$, where $\boxplus$ and $\boxminus$ respectively denote addition and subtraction modulo n. The ring topology seems to be the simplest one for protocol design. However, in [74], Gonzalez and Markowitch show that Franklin and Tsudik's protocol [66] is not fair and propose a new protocol. Another topology is the more general matrix topology, where each entity may desire items from a set of entities and offer items to a set of entities. Such protocols have been proposed by Asokan et al. on synchronous networks in [12].

A fundamental difference between non-repudiation and fair exchange protocols is the following. In a fair non-repudiation protocol, the originator sends some data with a non-repudiation of origin evidence to a recipient, who has to respond with a non-repudiation of receipt evidence. The sent data is generally not known to the recipient a priori. In a fair exchange protocol each entity offers an a priori known item (i.e. a kind of specification of the item is known a priori but not its precise content) and receives another item, also known a priori. Therefore, in a multi-party fair exchange protocol one can imagine sending an item to one entity and receiving an item from a different one. In non-repudiation it does not make sense that one entity receives some data and a distinct entity sends the corresponding receipt. Thus a ring topology is not sound. The most natural and here considered generalisation seems to be a one-to-many protocol on a star topology, where one entity sends a message to $n-1$ receiving entities who respond to the sender. In collaboration with Olivier Markowitch, we proposed the first multi-party non-repudiation protocols, with both online and offline TTP, in [83, 86, 98]. The main motivation for these protocols is a significant performance gain with respect to n two-party protocols. Afterwards, Onieva et al. [122] extended our work with online TTP, in order to permit the sending of different messages to each entity.

Another interesting case are multi-party contract signing protocols. In a multi-party contract signing protocol, each entity has to send its signature on a given contract text to all other entities and has to receive all other entities' signatures on this contract. Hence, the topology corresponds to a full graph. Asokan et al. [11] proposed a first solution for a synchronous network. Protocols for asynchronous networks have been proposed independently by Garay and MacKenzie [68] and Baum-Waidner and Waidner [22].

In chapter 6, we are going to analyse our optimistic multi-party non-repudiation protocol, as well as both multi-party contract signing protocols presented in [68] and [22].

3.7 Differences with classical security protocols

In this section, we briefly highlight several major differences between optimistic fair exchange protocols and "classical" security protocols, such as protocols aiming secrecy and/or authentication.

3.7.1 The attacker model and properties

Classically, when analysing a security protocol, one supposes that an attacker is trying to fool a honest entity. Moreover, the properties are such that the entities participating in the protocol want to achieve them together, i.e. the entities are *collaborating* to achieve them. As an example, consider a protocol aiming secrecy. Both Alice and Bob want to keep some information secret. In a key exchange protocol as well, they are both interested in obtaining a same, generally secret, key. In an authentication protocol, Bob wants to be sure that he is talking to Alice, and Alice wants to prove this fact to Bob. Hence they *both* want to achieve this property. In such protocols, an intruder aims at cheating one of the honest entities, by observing, altering or replaying parts of existent protocol executions between honest parties, or even by engaging additional protocol runs with honest entities. We refer to such an attacker as an *external* attacker: even if the attacker may engage a protocol execution, he always aims to make fail a security goal between two honest entities.

In a fair exchange protocol, the malicious party is one (or several in the context of multi-party protocols) of the participating entities, and each participant knows that the other entities may be potentially dishonest. This is due to the properties that have an inherent *adversarial* character. When looking at fairness, Alice wants the protocol to be fair to her, but one cannot assume that Bob is going to help her to achieve this goal. In some sense, Alice always engages a protocol with an adversary.

Moreover, it seems to us that properties, such as fairness or abuse-freeness, are more complicated to express and verify than the more classical properties. It is of course difficult to give a formal statement on how hard it is to express a property, but we observe that more sophisticated techniques are often required to verify the correctness of these properties. Moreover, these properties seem less well understood and, as we have seen in section 3.4, their definition is far from being unique.

3.7.2 The communication model

We already introduced the three different kinds of communication channels that we consider in the framework of fair exchange protocols: unreliable, resilient and operational channels. In the more classical protocols, one considers all channels as unreliable. Sometimes the underlying network is merged with the attacker, who completely controls it. In fair exchange protocols, we need to make stronger assumptions about the underlying network. Due to the impossibility result, discussed in section 3.5.1, achieving fairness requires that at least the communication channels with the TTP need to ensure eventual delivery. Some protocols even require operational channels to be correct. Hence, modelling different channel qualities is necessary, but introduces an additional difficulty when specifying and verifying fair exchange protocols.

3.7.3 Linear vs branching protocols

A last difference we want to emphasise is the fact that classical security protocols are generally *ping-pong* protocols, i.e. except the first message, an entity only sends a message as a response to a message that it receives. On the other hand, optimistic fair exchange protocols are divided in several sub-protocols, making branching possible, although they are intended to be executed in a given order by a honest entity. Most of the recent protocols give to Alice and Bob the possibility to *abort* or *recover* a protocol, in order to force a correct ending. Hence, it is not always easy to foresee all possible interleavings of these protocols, or even be sure at which moment in the protocol these sub-protocols can be launched. As we will see in chapters 6 and 7, changing the order or the moment of execution could result in subtle errors. This is one of the reasons why we propose a new method for the specification and the verification of exchange protocols.

The presence of different sub-protocols also implies that fair exchange protocols are much bigger in size. There exist examples of fair exchange protocols that are composed of 13 different messages, while an authentication protocol may only have as few as 3 messages, e.g. the Needham-Schroeder public-key protocol. Of course, when analysing a real life authentication protocol, e.g. Kerberos, the number and size of messages is much bigger than in the academic versions of the protocols. To the best of our knowledge, there exists no real-life implementation of optimistic protocols. However, we expect that optimistic protocols will have a similar increase in size when used in real applications. The important number of messages intervening in fair exchange protocols can generate a huge number of possible interleavings, and make the analysis of these protocols an even more complex task. Therefore, in section 5.2, we briefly propose the idea to integrate formal analysis in a top-down design methodology. Such methodologies are common in general software design. We believe that for complex protocols, such as optimistic fair exchange protocols, a development by successive refinements could improve the design process.

CHAPTER 4

A model for games and its logic

In this chapter we introduce the formal model of *alternating transition systems* and the *alternating time temporal logic* which were presented in [7] by Alur, Henzinger and Kupferman. We are going to use them to respectively model protocols and express properties the protocols have to verify. We also introduce a guarded command language that is used to describe alternating transition systems in a more convenient way. Finally we report on our implementation of the model-checking algorithms handling fairness constraints, using the tool MOCHA and its script language SLANG.

4.1 Alternating transition systems

The formalism that we are going to use to model exchange protocols as games is the model of alternating transition systems, ATS for short. ATS are a game variant of the usual Kripke structures.

Definition 4.1 An *alternating transition system* is a 6-tuple $S = \langle \Pi, \Sigma, Q, Q_0, \pi, \delta \rangle$ with the following components:

- Π is a set of propositions.
- Σ is a set of players.
- Q is a set of states.
- $Q_0 \subseteq Q$ is a set of initial states.
- $\pi : Q \to 2^{\Pi}$ maps a state to a set of propositions.
- $\delta : Q \times \Sigma \to 2^{2^Q} \setminus \{\emptyset\}$ is a transition function that maps a state and a player to a nonempty set of choices, where each choice is a set of possible next states (with some constraints, stated next). Whenever the system is in state q, each player a

chooses a set $Q_a \in \delta(q, a)$. In this way, a player a ensures that the next state of the system will be in its choice Q_a. However, which state in Q_a will be next depends on the choices made by the other players, because the successor of q must lie in the intersection $\bigcap_{a \in \Sigma} Q_a$ of the choices made by all players. It is assumed that the transition function is non-blocking and that the players together choose a unique next state; that is, if $\Sigma = \{a_1, \ldots, a_n\}$, then for every state $q \in Q$ and every set $Q_1, \ldots, Q_n$ of choices $Q_i \in \delta(q, a_i)$, $Q_1 \cap \ldots \cap Q_n$ is a singleton.

$\square$

Note that we do not lose expressive power by considering only deterministic games. Non-determinism can be modelled by an additional player.

Definition 4.2 For two states q and q' and a player a, we say that q' is an *a-successor* of q if there exists a set $Q' \in \delta(q, a)$ such that $q' \in Q'$. $\square$

We denote by $succ(q, a)$ the set of a-successors of q.

Definition 4.3 For two states q and q', we say that q' is a *successor* of q if for all players $a \in \Sigma$, we have $q' \in succ(q, a)$. $\square$

Thus, q' is a successor of q if and only if whenever the system S is in state q, the players in Σ can cooperate so that q' will be the next state.

Definition 4.4 A *computation* of S is an infinite sequence $\lambda = q_0 q_1 q_2 \ldots$ of states such that for all positions $i \geq 0$, the state q_{i+1} is a successor of the state q_i. $\square$

We refer to a computation starting at state q as a *q-computation.* For a computation λ and a position $i \geq 0$, we use $\lambda[i]$, $\lambda[0, i]$, and $\lambda[i, \infty]$ to denote the i-th state in λ, the finite prefix $q_0 q_1 \ldots q_i$ of λ, and the infinite suffix $q_i q_{i+1} \ldots$ of λ, respectively.

Example 4.5 Let us consider a system composed of three components, a, b and c, that control respectively the boolean variables x, y and z. Initially x, y and z are *false*. When $x = \textit{false}$, a can either leave its value unchanged, or set x to *true*. Once $x = \textit{true}$, a cannot change its value anymore. In a similar way, b can leave y unchanged or set y to *true* when $y = \textit{false}$. Player b must not change the value of y any more once it is set to *true*. The variable z is controlled by c, but depends on the value of y. When $y = \textit{false}$ and $z = \textit{false}$, c cannot change the value of z. When $y = \textit{true}$ and $z = \textit{false}$ c can either leave z unchanged or set z to *true*. Once $z = \textit{true}$, c must leave its value unchanged. We can model this system by the following ATS $S = \langle \Pi, \Sigma, Q, Q_0, \pi, \delta \rangle$.

- $\Pi = \{x, y, z\}$
- $\Sigma = \{a, b, c\}$
- $Q = \{q, q_x, q_y, q_z, q_{xy}, q_{xz}, q_{yz}, q_{xyz}\}$
- $Q_0 = \{q\}$
- We have the following labelling function $\pi : Q \to 2^{\Pi}$

  - $\pi(q) = \emptyset$
  - $\pi(q_x) = \{x\}$
  - $\pi(q_y) = \{y\}$
  - $\pi(q_z) = \{z\}$
  - $\pi(q_{xy}) = \{x, y\}$
  - $\pi(q_{xz}) = \{x, z\}$
  - $\pi(q_{yz}) = \{y, z\}$
  - $\pi(q_{xyz}) = \{x, y, z\}$

- We have the following transition function $\delta : Q \times \Sigma \to 2^{2^Q}$
  - $\delta(q, a) = \delta(q_y, a) = \delta(q_z, a) = \delta(q_{yz}, a) = \{\{q, q_y, q_z, q_{yz}\}, \{q_x, q_{xy}, q_{xz}, q_{xyz}\}\}$
  - $\delta(q_x, a) = \delta(q_{xy}, a) = \delta(q_{xz}, a) = \delta(q_{xyz}, a) = \{\{q_x, q_{xy}, q_{xz}, q_{xyz}\}\}$
  - $\delta(q, b) = \delta(q_x, b) = \delta(q_z, b) = \delta(q_{xz}, b) = \{\{q, q_x, q_z, q_{xz}\}, \{q_y, q_{xy}, q_{yz}, q_{xyz}\}\}$
  - $\delta(q_y, b) = \delta(q_{xy}, b) = \delta(q_{yz}, b) = \delta(q_{xyz}, b) = \{\{q_y, q_{xy}, q_{yz}, q_{xyz}\}\}$
  - $\delta(q, c) = \delta(q_x, c) = \{\{q, q_x, q_y, q_{xy}\}\}$
  - $\delta(q_y, c) = \delta(q_{xy}, c) = \{\{q, q_x, q_y, q_{xy}\}, \{q_z, q_{xz}, q_{yz}, q_{xyz}\}\}$
  - $\delta(q_z, c) = \delta(q_{xz}, c) = \delta(q_{yz}, c) = \delta(q_{xyz}, c) = \{\{q_z, q_{xz}, q_{yz}, q_{xyz}\}\}$

  We see that for instance a, in state q, can either choose to leave the value of x unchanged, that corresponds to the choice $\{q, q_y, q_z, q_{yz}\}$, or set x to *true*, which corresponds to the choice $\{q_x, q_{xy}, q_{xz}, q_{xyz}\}$. Also note that, in a given state, whatever choices are made by the players, the intersection yields a singleton.

We can now use our example to illustrate the notions of *successor* and *computation*. We have that q_{xyz} is an a-successor of the state q, because $q_{xyz} \in \{q_x, q_{xy}, q_{xz}, q_{xyz}\} \in \delta(q, a)$. The same state q_{xyz} is also a b-successor of q. However, q_{xyz} is not a successor of q, because q_{xyz} is not a c-successor of q. Hence there exists no computation of the form $\ldots, q, q_{xyz}, \ldots$. The possible successors of q are q, q_x, q_y and q_{xy}. Possible computations of S include the infinite computations $qq_{xy}q_{xy}q_{xyz}^{\omega}$, as well as $qq_xq_xq_{xy}^{\omega}$ and computations of the form $qq_yq_{xy}^{*}q_{xyz}^{\omega}$. $\square$

4.2 Alternating-time temporal logic

The *Alternating-time Temporal Logic* [7] (ATL for short) is defined with respect to a finite set Π of *propositions* and a finite set Σ of *players*.

Definition 4.6 An ATL *formula* is one of the following:

- p, for propositions $p \in \Pi$.
- $\neg\varphi$ or $\varphi_1 \vee \varphi_2$, where φ, φ_1, and φ_2 are ATL formulae.
- $\langle\langle A\rangle\rangle \bigcirc \varphi$, $\langle\langle A\rangle\rangle \Box\varphi$, or $\langle\langle A\rangle\rangle \varphi_1 \mathcal{U} \varphi_2$, where $A \subseteq \Sigma$ is a set of players, and φ, φ_1, and φ_2 are ATL formulae.

□

The operator $\langle\!\langle\ \rangle\!\rangle$ is a *path quantifier*, and $\bigcirc$ ("next"), $\Box$ ("always"), and $\mathcal{U}$ ("until") are *temporal operators*. The logic ATL is similar to the computation tree logic CTL, except that path quantifiers are parameterised by sets of players. The parameterised path quantifier $\langle\!\langle A\rangle\!\rangle$ is a generalisation of the path quantifiers of CTL: the existential path quantifier $\exists$ corresponds to $\langle\!\langle \Sigma\rangle\!\rangle$, and the universal path quantifier $\forall$ corresponds to $\langle\!\langle \emptyset\rangle\!\rangle$. Sometimes we write $\langle\!\langle a_1,\ldots,a_n\rangle\!\rangle$ instead of $\langle\!\langle \{a_1,\ldots,a_n\}\rangle\!\rangle$. Additional boolean connectives are defined from $\neg$ and $\vee$ in the usual manner. As in CTL, we write $\langle\!\langle A\rangle\!\rangle\Diamond\varphi$ for $\langle\!\langle A\rangle\!\rangle \mathit{true}\,\mathcal{U}\varphi$. Extensions of ATL such as ATL*and the alternating μ-calculus are defined in [7]. We also introduce the $[\![\ ]\!]$ operator, which is the dual of the $\langle\!\langle\ \rangle\!\rangle$ operator. Intuitively, while $\langle\!\langle A\rangle\!\rangle\Diamond\varphi$ means that A has a strategy to finally reach a state where φ is true, $[\![A]\!]\Diamond\varphi$ means that A cannot avoid to finally reach a state where φ is satisfied. So, we can write $[\![A]\!]\bigcirc\neg\varphi$ instead of $\neg\langle\!\langle A\rangle\!\rangle\bigcirc\varphi$, $[\![A]\!]\Diamond\neg\varphi$ instead of $\neg\langle\!\langle A\rangle\!\rangle\Box\varphi$, $[\![A]\!]\Box\neg\varphi$ instead of $\neg\langle\!\langle A\rangle\!\rangle\Diamond\varphi$ and $\neg\varphi_2 \wedge [\![A]\!]\neg\varphi_2\,\mathcal{U}(\neg\varphi_1\wedge\neg\varphi_2)$ instead of $\neg\langle\!\langle A\rangle\!\rangle\varphi_1\,\mathcal{U}\varphi_2$.

We interpret ATL formulae over the states of a given ATS S that has the same propositions and players. The labelling of the states of S with propositions is used to evaluate the atomic formulae of ATL. The logical connectives $\neg$ and $\vee$ have the standard interpretation. To give an idea of how to evaluate a formula of the form $\langle\!\langle A\rangle\!\rangle\psi$ at a state q of S, consider the following two-player game between a protagonist and an antagonist. The game proceeds in an infinite sequence of rounds, and after each round, the position of the game is a state of S. The initial position is q. Now consider the game in some position u. To update the position, first the protagonist chooses for every player $a\in A$, a set $Q_a\in\delta(u,a)$. Then, the antagonist chooses a successor (with respect to definition 4.2) v of u such that $v\in\cap_{a\in A}Q_a$, and the position of the game is updated to v. In this way, the game continues forever and produces a computation. The protagonist wins the game if the resulting computation satisfies the subformula ψ, read as a linear temporal formula whose outermost operator is $\bigcirc$, $\Box$, or $\mathcal{U}$. The ATL formula $\langle\!\langle A\rangle\!\rangle\psi$ holds at the state q if the protagonist has a *winning strategy* in this game.

In order to give a formal definition of the semantics of ATL, we first define the notion of strategies.

Definition 4.7 Consider an ATS $S=\langle \Pi,\Sigma,Q,\pi,\delta\rangle$. A *strategy* for a player $a\in\Sigma$ is a mapping $f_a : Q^+ \to 2^Q$ such that for all $\lambda\in Q^*$ and all $q\in Q$, we have $f_a(\lambda\cdot q)\in\delta(q,a)$. □

Thus, the strategy f_a maps each finite prefix $\lambda\cdot q$ of a computation to a set in $\delta(q,a)$. This set contains possible extensions of the computation as suggested to player a by the strategy. Each strategy f_a induces a set of computations that player a can enforce. We can also define a strategy for a set of players $A\subseteq\Sigma$. Given a state q, a set A of players, and a strategy function F_A, which associates a strategy f_a to each player $a\in A$, we define the *outcomes* of F_A from q to be the set $out(q,F_A)$ of all q-computations that the players in A can enforce when they cooperate and follow the strategies given by F_A; that is, a computation $\lambda=q_0,q_1,q_2,\ldots$ is in $out(q,F_A)$ if $q_0=q$ and for all positions $i\geq 0$, the state q_{i+1} is a successor of q_i satisfying $q_{i+1}\in\bigcap_{a\in A} f_a(\lambda[0,i])$.

We can now turn to a formal definition of the semantics of ATL. We write $S, q \models \varphi$ ("state q satisfies formula φ in the structure S") to indicate that the formula φ holds at state q of S. When S is clear from the context we omit it and write $q \models \varphi$.

Definition 4.8 The *semantics of* ATL are defined by means of the relation $\models$, defined for all states q of S, inductively as follows:

- For $p \in \Pi$, we have $q \models p$ iff $p \in \pi(q)$.
- $q \models \neg\varphi$ iff $q \not\models \varphi$.
- $q \models \varphi_1 \vee \varphi_2$ iff $q \models \varphi_1$ or $q \models \varphi_2$.
- $q \models \langle\!\langle A \rangle\!\rangle \bigcirc \varphi$ iff there exists a strategy function F_A, such that for all computations $\lambda \in out(q, F_A)$, we have $\lambda[1] \models \varphi$.
- $q \models \langle\!\langle A \rangle\!\rangle \Box\varphi$ iff there exists a strategy function F_A, such that for all computations $\lambda \in out(q, F_A)$ and all positions $i \geq 0$, we have $\lambda[i] \models \varphi$.
- $q \models \langle\!\langle A \rangle\!\rangle \varphi_1 \mathcal{U} \varphi_2$ iff there exists a a strategy function F_A, such that for all computations $\lambda \in out(q, F_A)$ there exists a position $i \geq 0$ such that $\lambda[i] \models \varphi_2$ and for all positions $0 \leq j < i$, we have $\lambda[j] \models \varphi_1$.

$\Box$

Note that the next operator $\bigcirc$ is local: $q \models \langle\!\langle A \rangle\!\rangle \bigcirc \varphi$ iff for each player $a \in A$ there exists a set $Q_a \in \delta(q, a)$ such that for each state $q' \in \bigcap_{a \in A} Q_a$, if q' is a successor of q, then $q' \models \varphi$.

We say that the ATS S satisfies φ, noted $S \models \varphi$, iff $\forall q \in Q_0 \cdot\ q \models \varphi$.

We now illustrate the expressive power of ATL. Consider the set of players $\Sigma = \{a, b, c\}$ and the following formulae with their verbal (and intuitive) reading:

- $\langle\!\langle a \rangle\!\rangle \Diamond p$, player a has a strategy against players b and c to eventually reach a state where the proposition p is true;
- $[\![b, c]\!] \Box p$, the coalition of players b and c cannot avoid, when playing against a, that the proposition p remains true forever;
- $\langle\!\langle a, b \rangle\!\rangle \bigcirc (p \wedge \neg\langle\!\langle c \rangle\!\rangle \Box p)$, a and b can cooperate so that the next state satisfies p and from there c does not have a strategy to impose p forever.

Those three formulae are a good illustration of the great expressive power of ATL to express cooperative as well as adversarial behaviours between players.

Example 4.9 For a more specific example, consider the ATS S introduced in example 4.5. The example formulae given hereunder focus on the temporal operators $\Diamond$ and $\Box$, as these two operators will be the only ones needed when specifying the properties of fair exchange protocols. One could be interested in verifying the following formulae.

- Can b and c cooperate, against a, such that eventually y and z become true ?

$$\varphi_1 = \langle\!\langle b, c \rangle\!\rangle \Diamond (y \wedge z)$$

- Can a and c cooperate, against b, such that eventually x and z become true ?

$$\varphi_2 = \langle\!\langle a, c \rangle\!\rangle \Diamond (x \wedge z)$$

- Can b assure, against a and c, that z will never be true ?

$$\varphi_3 = \langle\!\langle b \rangle\!\rangle \Box \neg z$$

- Can a, b and c collaborate such that eventually x, y and z become true ?

$$\varphi_4 = \langle\!\langle a, b, c \rangle\!\rangle \Diamond (x \wedge y \wedge z)$$

We have that $S \models \varphi_1$, because in the initial state q, b has a strategy to set y to *true*, and afterwards, once y has become *true*, c can set z to *true*. However, $S \not\models \varphi_2$, because b can choose to leave $y = \mathit{false}$ forever, and hence c cannot modify z. The fact that b can keep $\neg y$ forever is also expressed by φ_3. Here, one may see that φ_3 implies φ_2. More generally, we have that

$$\langle\!\langle A \rangle\!\rangle \neg\varphi \Rightarrow \neg \langle\!\langle \Sigma \setminus A \rangle\!\rangle \varphi.$$

It should be clear that if the players in A can enforce $\neg\varphi$ it is impossible for the players in $\Sigma \setminus A$ to enforce φ. However, the inverse does not hold. It is possible that $S \models \neg\langle\!\langle A \rangle\!\rangle \varphi$ and $S \models \neg\langle\!\langle \Sigma \setminus A \rangle\!\rangle \neg\varphi$. For instance, in our example, we have that $S \models \neg\langle\!\langle a \rangle\!\rangle \bigcirc (x \wedge \neg y) \vee (\neg x \wedge y)$, while at the same time we have that $S \models \neg\langle\!\langle b, c \rangle\!\rangle \bigcirc (\neg x \vee y) \wedge (x \vee \neg y)$. Intuitively, one cannot state that if the players in A do *not* have a strategy to ensure φ, then the players in $\Sigma \setminus A$ do have a strategy to ensure $\neg\varphi$. As the example shows, it is possible that players in A and in $\Sigma \setminus A$ do need to collaborate to ensure either φ or $\neg\varphi$.

Formula φ_4 is satisfied by S, because the state q_{xyz} is reachable. Note that formula φ_4 is a CTL formula, and can be rewritten as $\exists \Diamond (x \wedge y \wedge z)$. The three preceding formulae could however not be expressed in CTL. $\Box$

4.3 Fairness

As in the usual temporal logic setting, fairness can be used to rule out some computations. For example, to evaluate a formula of the form $\langle\!\langle a \rangle\!\rangle \Diamond \varphi$, we may want to only consider computations where the antagonists, that is the agents in $\Sigma \setminus \{a\}$, respect their fairness constraints.

Definition 4.10 A *fairness constraint* is a function $\gamma : Q \times \Sigma \to 2^{2^Q}$ such that for each $q \in Q$ and $a \in \Sigma$, we have $\gamma(q, a) \subseteq \delta(q, a)$. $\Box$

Given a computation $\lambda = q_0 q_1 \ldots q_n \ldots$, we say that γ is *a-enabled* at position $i \geq 0$ if $\gamma(q_i, a) \neq \emptyset$. We say that γ is *a-taken* at position $i \geq 0$ if there exists a set $Q' \in \gamma(q_i, a)$ such that $q_{i+1} \in Q'$.

Definition 4.11 Given a set $A \subseteq \Sigma$, we say that a computation λ is *weakly $\langle \gamma, A \rangle$-fair* if, for each agent $a \in A$, either there are infinitely many positions of λ at which γ is not a-enabled, or there are infinitely many positions of λ at which γ is a-taken. $\Box$

Intuitively, a player a cannot ignore an infinite number of times an possible action. Note that there exists also, as in the usual temporal logic setting, a notion of strong fairness. The interested reader is referred to [7] for the definition. We will not need the notion of strong fairness. A *fairness condition* Γ is a set of fairness constraints. For a given set $A \subseteq \Sigma$, we say that λ is *weakly* $\langle\Gamma, A\rangle$*-fair*, if λ is weakly $\langle\gamma, A\rangle$-fair for all fairness constraints $\gamma \in \Gamma$.

Definition 4.12 Given an ATS S, and a (weak) fairness condition Γ, we define *the semantics of Fair* ATL as follows:

- For $p \in \Pi$, we have $q \models_F p$ iff $p \in \pi(q)$.
- $q \models_F \neg\varphi$ iff $q \not\models_F \varphi$.
- $q \models_F \varphi_1 \vee \varphi_2$ iff $q \models_F \varphi_1$ or $q \models_F \varphi_2$.
- $q \models_F \langle\langle A\rangle\rangle \bigcirc \varphi$ iff there exists a strategy function F_A, such that for all computations $\lambda \in out(q, F_A)$, we have $\lambda[1] \models_F \varphi$.
- $q \models_F \langle\langle A\rangle\rangle \Box\varphi$ iff there exists a strategy function F_A, such that for all $\langle\Gamma, A\rangle$-fair computations $\lambda \in out(q, F_A)$ and all positions $i \geq 0$, we have $\lambda[i] \models_F \varphi$.
- $q \models_F \langle\langle A\rangle\rangle \varphi_1 \mathcal{U} \varphi_2$ iff there exists a a strategy function F_A, such that for all $\langle\Gamma, \Sigma \setminus A\rangle$-fair computations $\lambda \in out(q, F_A)$ there exists a position $i \geq 0$ such that $\lambda[i] \models_F \varphi_2$ and for all positions $0 \leq j < i$, we have $\lambda[j] \models_F \varphi_1$.

$\Box$

As for ATL without fairness constraints, we write $S \models_F \varphi$ iff $\forall q \in Q_0 \cdot q \models_F \varphi$.

The definition of fair ATL we gave slightly differs from the semantics given in [7]. The difference is that in the case of $\langle\langle A\rangle\rangle \Box\varphi$ and $\langle\langle A\rangle\rangle \varphi_1 \mathcal{U} \varphi_2$ only the protagonist, respectively the antagonist, has to respect its fairness constraints. This is due to the fact that fairness constraints only rule out infinite computations, and do not influence formulae that are validated or invalidated using finite computations.

Let us have a more precise look at the formula $\langle\langle A\rangle\rangle \Box\varphi$ and explain why we do not need to consider the fairness constraints of players in $\Sigma \setminus A$. It is clear that, when this formula is satisfied, it would also be satisfied, when restricting, by means of fairness constraints, the choices of the players in $\Sigma \setminus A$. When $\langle\langle A\rangle\rangle \Box\varphi$ fails, this could be due to the fact that the choices of the players in $\Sigma \setminus A$ were not restricted. However, if $\langle\langle A\rangle\rangle \Box\varphi$ is not satisfied, this means that there exists a computation $\lambda = q_0 q_1 q_2 \ldots$, such that $q_i \models \neg\varphi$ for some i. Suppose that the computation λ is not weakly $\langle\Gamma, \Sigma \setminus A\rangle$-fair. Then, there always exists a weakly $\langle\Gamma, \Sigma \setminus A\rangle$-fair computation λ', having the same prefix $q_0, \ldots, q_i$ than λ, as the players in $\Sigma \setminus A$ can respect their fairness constraints on the remaining of the computation λ'.

A similar reasoning can be done for $\langle\langle A\rangle\rangle \varphi_1 \mathcal{U} \varphi_2$ to show that only players in $\Sigma \setminus A$ have to respect fairness constraints. It should be clear that if this formula fails when the players in A are not restricted by means of fairness constraints, the formula also fails when they are restricted. On the other hand, suppose that the players in A are not restricted and that the formula holds. This means that there exists a computation $\lambda = q_0 q_1 q_2 \ldots$, on which exists a position $i \geq 0$ such that $\lambda[i] \models_F \varphi_2$ and for all positions $0 \leq j < i$, we have $\lambda[j] \models_F \varphi_1$. Then, there always exists a weakly $\langle \Gamma, A \rangle$-fair computation λ', having the same prefix $q_0, \ldots, q_i$ than λ, as the players in A can respect their fairness constraints on the remaining of the computation λ'. A similar reasoning can be done for $\langle\langle A \rangle\rangle \bigcirc \varphi$ to show that no fairness constraints at all need to be considered.

Remark 4.13 Note that, in the particular cases of $\langle\langle \Sigma \rangle\rangle \Diamond \varphi$ and $\langle\langle\ \rangle\rangle \Box \varphi$, corresponding to the CTL formulae $\exists \Diamond \varphi$ and $\forall \Box \varphi$, we do not need to consider fairness constraints at all. On the other hand, the CTL formulae $\exists \Box \varphi$ and $\forall \Diamond \varphi$, corresponding to $\langle\langle \Sigma \rangle\rangle \Box \varphi$ and $\langle\langle\ \rangle\rangle \Diamond \varphi$ require consideration of fairness constraints for all players. □

Remark 4.14 Note that we use the name *fairness*, to both denote fairness constraints in the context of temporal logic, as well as to denote the fairness property of fair exchange protocols. It should always be clear from the context to which notion we refer. □

Example 4.15 We now reconsider the ATS S, introduced in example 4.5, and add a fairness constraint. Remember that when $y = \mathit{false}$, player b can choose to either leave b *false* or set y to *true*. We now add a fairness constraint γ, defined as follows.

$$\gamma(q, b) = \gamma(q_x, b) = \gamma(q_z, b) = \gamma(q_{xz}, b) = \{\{q_y, q_{xy}, q_{yz}, q_{xyz}\}\}$$

We suppose that γ maps all other states and players to the empty set. Intuitively, adding the fairness constraint γ restricts b's behaviour and does not allow him to keep y at *false* forever. This means for instance that $q q_x^\omega$, which is a valid computation of S, is not a weakly $\langle \gamma, b \rangle$-fair computation of S.

Consider again some of the ATL formulae given in example 4.9. We have that $S \not\models \langle\langle a, c \rangle\rangle \Diamond x \wedge z$. However, when adding the fairness constraint γ, i.e. $\Gamma = \{\gamma\}$, we have that $S \models_F \langle\langle a, c \rangle\rangle \Diamond x \wedge z$. This is explained by the fact, that b's spoiling strategy, never setting y to *true*, is now discarded by the fairness constraint γ. □

We also introduce a lemma, which will serve in several proofs in the following chapters.

Lemma 4.16 For any ATS S, any fairness condition Γ and any state q of S we have the following proof rules.

$$\frac{S, q \models_F \langle\langle A \rangle\rangle \Diamond \varphi, \ \varphi \rightarrow \psi}{S, q \models_F \langle\langle A \rangle\rangle \Diamond \psi} \qquad (R_1)$$

$$\frac{S, q \models_F \langle\langle A \rangle\rangle \Diamond (\varphi_1 \wedge \varphi_2)}{S, q \models_F \langle\langle A \rangle\rangle \Diamond \varphi_1 \wedge \langle\langle A \rangle\rangle \Diamond \varphi_2} \qquad (R_2)$$

$$\frac{S, q \models_F \langle\langle A \rangle\rangle \Diamond \langle\langle A \rangle\rangle \Diamond \varphi}{S, q \models_F \langle\langle A \rangle\rangle \Diamond \varphi} \qquad (R_3)$$

$$\frac{S, q \models_F \langle\langle A \rangle\rangle \Diamond (p \wedge \langle\langle A \rangle\rangle \Diamond \varphi), \ p \rightarrow \forall \Box p}{S, q \models_F \langle\langle A \rangle\rangle \Diamond (p \wedge \varphi)} \qquad (R_4)$$

□

Proof:
Rule (R_1) is proven as follows. Suppose that for a given ATS S, a fairness condition Γ and a state q, we have that $S, q \models_F \langle\!\langle A \rangle\!\rangle \Diamond \varphi$. This means that there exists a strategy function F_A, such that for all $\langle \Gamma, \Sigma \setminus A \rangle$-fair computations $\lambda \in out(q, F_A)$ there exists a position $i \geq 0$, such that $\lambda[i] \models_F \varphi$. Moreover, as $\varphi \rightarrow \psi$, we have that $\lambda[i] \models_F \psi$. Hence, the same strategy function F_A, is such that for all $\langle \Gamma, \Sigma \setminus A \rangle$-fair computations $\lambda \in out(q, F_A)$ there exists a positions $i \geq 0$, such that $\lambda[i] \models_F \psi$, which implies that $\langle\!\langle A \rangle\!\rangle \Diamond \psi$.

Rule (R_2) is easily derived from rule (R_1). We have that $\langle\!\langle A \rangle\!\rangle \Diamond (\varphi_1 \wedge \varphi_2)$ and we know that $\varphi_1 \wedge \varphi_2 \rightarrow \varphi_1$, as well as $\varphi_1 \wedge \varphi_2 \rightarrow \varphi_2$. Correctness of rule ($R_2$) is now directly inferred from rule (R_1).

To prove rule (R_3) suppose that for a given ATS S, a fairness condition Γ and a state q, we have that $S, q \models_F \langle\!\langle A \rangle\!\rangle \Diamond \langle\!\langle A \rangle\!\rangle \Diamond \varphi$. This means that there exists

- a strategy function F_A, such that for all $\langle \Gamma, \Sigma \setminus A \rangle$-fair computations $\lambda \in out(q, F_A)$ and some position $i \geq 0$, we have $\lambda[i] \models_F \langle\!\langle A \rangle\!\rangle \Diamond \varphi$,

- a strategy function F'_A, such that for all $\langle \Gamma, \Sigma \setminus A \rangle$-fair computations $\lambda' \in out(\lambda[i], F'_A)$ and some position $j \geq 0$, we have $\lambda'[j] \models_F \varphi$.

Hence, we have a strategy function F''_A, such that for all $\langle \Gamma, \Sigma \setminus A \rangle$-fair computations $\lambda'' \in out(q, F''_A)$ there exists some position $i \geq 0$, such that $\lambda[i] \models_F \langle\!\langle A \rangle\!\rangle \Diamond \varphi$ and some position $j \geq 0$, such that $\lambda''[i + j] \models_F \varphi$. The strategy F''_A is constructed by following strategy F_A until position i and then following strategy F'_A. Hence, $q \models_F \langle\!\langle A \rangle\!\rangle \Diamond \varphi$.

To prove rule (R_4), suppose that for a given ATS S, fairness constraint Γ and state q we have that $S, q \models_F \langle\!\langle A \rangle\!\rangle \Diamond (p \wedge \langle\!\langle A \rangle\!\rangle \Diamond \varphi)$. By taking a similar reasoning as in the proof of rule (R3), we know that there exists a strategy function F_A, such that for all $\langle \Gamma, \Sigma \setminus A \rangle$-fair computations $\lambda \in out(q, F_A)$ there exists a position $i \geq 0$, such that $\lambda[i] \models_F p$ and a position $j \geq i$, such that $\lambda[j] \models_F \varphi$. Moreover, as $p \rightarrow \forall \Box p$, we have that $\forall k \geq i \cdot \lambda[k] \models_F p$. Hence, we have that $\lambda[j] \models_F (\varphi \wedge p)$, and the strategy function F_A is such that for all $\langle \Gamma, \Sigma \setminus A \rangle$-fair computations $\lambda \in out(q, F_A)$ there exists a position $j \geq 0$, such that $\lambda[j] \models_F (p \wedge \varphi)$, which implies that $\langle\!\langle A \rangle\!\rangle \Diamond (p \wedge \varphi)$. $\Box$

4.4 Model-checking

4.4.1 Symbolic model-checking of ATL

ATL is a state logic in the sense that formulae are evaluated in a given state, i.e. for any formula φ and state q of an ATS, either φ is true in q, denoted $q \models \varphi$, or φ is false in q, denoted $q \not\models \varphi$. We denote $[\varphi]_S$ the set of states of the ATS S where φ is true, i.e. $[\varphi]_S = \{q \mid q \models \varphi\}$. In particular, $[true] = Q$, where Q is the set of states of S, and $[false] = \emptyset$. Sometimes, when S is clear form the context, we write $[\varphi]$ instead of $[\varphi]_S$. The set $[\varphi]_S$ can be computed using simple fixpoint algorithms. Those fixpoint algorithms are very similar to the algorithms needed for CTL model-checking, but we need transformers which are different from the usual Pre operator. To define these transformers we extend the transition function to set of players, i.e. for any state q and any set of players $A \subseteq \Sigma$, $\delta(q, A) = \{Q_A = \cap_{a \in A} Q_a \mid \forall a \in A \cdot \ Q_a \in \delta(q, a)$.

Definition 4.17 Given an ATS S, a set of players Σ, a state space Q, and $\tau \subseteq Q$, we have for each $A \subseteq \Sigma$:

$$\mathsf{CPre}(A, \tau) = \{q \mid \exists Q_1 \in \delta(A, q) \cdot \forall Q_2 \in \delta(\Sigma \setminus A, q) : Q_1 \cap Q_2 \subseteq \tau\}$$

and

$$\mathsf{UPre}(A, \tau) = \{q \mid \forall Q_1 \in \delta(A, q) \cdot \exists Q_2 \in \delta(\Sigma \setminus A, q) : Q_1 \cap Q_2 \subseteq \tau\}.$$

□

$\mathsf{CPre}(A, \tau)$ contains the set of states for which the players in A do have a "one-step" strategy to reach τ. In fact, in all states of $\mathsf{CPre}(A, \tau)$, the players in A can resolve their choices such that no matter what the choices of the remaining players are, the next state will be within set τ. We also say that if $q \in \mathsf{CPre}(A, \tau)$, then q is a *controllable predecessor* of the set of states τ for players in A. In a similar way, $\mathsf{UPre}(a, \tau)$ contains the set of states for which the players in A do not have a "one-step" strategy to force the game outside of τ. We call the states in $\mathsf{UPre}(A, \tau)$, the *uncontrollable predecessors* of τ for players in A.

We can now present the ATL model-checking algorithm, which was first presented in [7]. As for the symbolic CTL model-checking algorithm, termination of the here presented algorithm is guaranteed. It is given in algorithm 1 and uses the following primitive operations.

- The function *Sub*, when given an ATL formula φ, returns a queue $Sub(\varphi)$ of all subformulae of φ such that if φ_1 is a subformula of φ and φ_2 is a subformula of φ_1, then φ_2 precedes φ_1 in the queue $Sub(\varphi)$ (note, that sometimes several orders may be possible, but in that case the result of the model-checking algorithm does not depend on them).

- The function *Reg*, when given a proposition $p \in \Pi$, returns the state set $[p]$.

- The functions CPre and UPre are as in definition 4.17.

- Union, intersection, difference, and inclusion test for state sets have their expected meanings.

The algorithm returns the set of states $[\varphi]$ in which the ATL formula φ is satisfied. To check whether the formula is satisfied on a given ATS, we verify that it holds for each initial state, i.e. $Q_0 \subseteq [\varphi]$.

Remark 4.18 Note that in the ATL model-checking algorithm, given in algorithm 1, we give the algorithm for explicitly computing formulae of the form $[\![A]\!]\psi$. These formulae can also be computed by first rewriting them using only the $\langle\!\langle\ \rangle\!\rangle$ operator. However, the direct computation is needed, when using techniques such as abstract interpretation [79], where we restrict negation to be in front of propositional formulae only. □

Algorithm 1 Symbolic model-checking algorithm for ATL

foreach φ' **in** $Sub(\varphi)$ **do**
  case $\varphi' = p$: $[\varphi'] := Reg(p)$
  case $\varphi' = \neg\gamma$: $[\varphi'] := [true] \setminus [\gamma]$
  case $\varphi' = \gamma_1 \vee \gamma_2$: $[\varphi'] := [\gamma_1] \cup [\gamma_2]$
  case $\varphi' = \langle\!\langle A \rangle\!\rangle \bigcirc \gamma$: $[\varphi'] := \mathsf{CPre}(A, [\gamma])$
  case $\varphi' = [\![A]\!] \bigcirc \gamma$: $[\varphi'] := \mathsf{UPre}(A, [\gamma])$
  case $\varphi' = \langle\!\langle A \rangle\!\rangle \Box \gamma$:
    $\rho := [true]$; $\tau := [\gamma]$;
    while $\rho \not\subseteq \tau$ **do** $\rho := \rho \cap \tau$; $\tau := \mathsf{CPre}(A, \rho) \cap [\gamma]$ **od**;
    $[\varphi'] := \rho$
  case $\varphi' = [\![A]\!] \Box \gamma$:
    $\rho := [true]$; $\tau := [\gamma]$;
    while $\rho \not\subseteq \tau$ **do** $\rho := \rho \cap \tau$; $\tau := \mathsf{UPre}(A, \rho) \cap [\gamma]$ **od**;
    $[\varphi'] := \rho$
  case $\varphi' = \langle\!\langle A \rangle\!\rangle \gamma_1 \,\mathcal{U} \gamma_2$:
    $\rho := [false]$; $\tau := [\gamma_2]$;
    while $\tau \not\subseteq \rho$ **do** $\rho := \rho \cup \tau$; $\tau := \mathsf{CPre}(A, \rho) \cap [\gamma_1]$ **od**;
    $[\varphi'] := \rho$
  case $\varphi' = [\![A]\!] \gamma_1 \,\mathcal{U} \gamma_2$:
    $\rho := [false]$; $\tau := [\gamma_2]$;
    while $\tau \not\subseteq \rho$ **do** $\rho := \rho \cup \tau$; $\tau := \mathsf{UPre}(A, \rho) \cap [\gamma_1]$ **od**;
    $[\varphi'] := \rho$
  end case
od;
return $[\varphi]$.

4.4.2 Symbolic model-checking of fair ATL

In order to add fairness constraints to the model-checking algorithm, we have to define an extended version of an ATS that adds explicitly when a fairness constraint is enabled or taken.

Definition 4.19 Given an ATS $S = \langle \Pi, \Sigma, Q, Q_0, \pi, \delta \rangle$, and a fairness condition Γ we define the *extended* ATS $S_F = \langle \Pi_F, \Sigma, Q_F, Q_{0F}, \pi_F, \delta_F \rangle$, where

- for every agent $a \in \Sigma$ and every fairness constraint $\gamma \in \Gamma$, there is a new proposition $\langle \gamma, a, \mathit{enabled} \rangle$ and a new proposition $\langle \gamma, a, \mathit{taken} \rangle$: $\Pi_F = \Pi \cup \langle \Gamma \times \Sigma \times \{\mathit{enabled}, \mathit{taken}\} \rangle_F$, respecting the definition of π_F given below;

- the states of S_F correspond to the transitions of S: $Q_F = \{\langle q, q' \rangle \mid q'$ is a successor of q in $S\}$;

- the initial states of S_F correspond to the transitions available in the initial states of S: $Q_{0F} = \{\langle q, q' \rangle \mid \langle q, q' \rangle \in Q_F, q \in Q_0\}$;

- for every state $\langle q, q' \rangle \in Q_F$ and every agent $a \in \Sigma$, the transition $\delta_F(\langle q, q' \rangle, a)$ is obtained from $\delta(q', a)$ by replacing each state q'' appearing in $\delta(q', a)$ by the state $\langle q', q'' \rangle$, i.e.
$$\delta_F(\langle q, q' \rangle, a) = \big\{ \{ \langle q', q'' \rangle \mid q'' \in Q_a \} \mid Q_a \in \delta(q', a) \big\} ;$$

- for every state $\langle q, q' \rangle \in Q_F$, we have

$$\begin{aligned} \pi_F(\langle q, q' \rangle) \;\; = \;\; & \pi(q) \textstyle\bigcup_{\gamma \in \Gamma} \{ \langle \gamma, a, \mathit{enabled} \rangle_F \mid \gamma(q, a) \neq \emptyset \} \cup \\ & \{ \langle \gamma, a, \mathit{taken} \rangle_F \mid \exists Q' \in \gamma(q, a) \cdot q' \in Q' \}. \end{aligned}$$

□

Intuitively, a state of the form $\langle q, q' \rangle$ in S_F corresponds to the ATS S being in state q with the agents deciding that the successor of q will be q'. There is a one-to-one correspondence between computations of S and S_F, and between strategies in S and S_F. This means that for each computation $\lambda = q_0 q_1 q_2 \dots q_i \dots$ over an ATS S, we have a corresponding computation $\lambda_F = \langle q_0, q_1 \rangle \langle q_1, q_2 \rangle \dots \langle q_i, q_{i+1} \rangle \dots$ over the extended ATS S_F. Also, when F_A is a strategy function, we denote by $out(\langle q_0, q_1 \rangle, F_A)$ the set of computations over the extended ATS, such that $\lambda_F = \langle q_0, q_1 \rangle \dots$ is in $out(\langle q_0, q_1 \rangle, F_A)$ if and only if the corresponding execution $\lambda = q_0 q_1 \dots$ over the initial ATS is in $out(q_0, F_A)$. Moreover, for each $\gamma \in \Gamma$ and each player $a \in \Sigma$, γ is a-enabled in state $\lambda[i]$ if and only if $\langle \gamma, a, \mathit{enabled} \rangle_F \in \pi_F(\lambda_F[i])$ and γ is a-taken in state $\lambda[i]$ if and only if $\langle \gamma, a, \mathit{taken} \rangle_F \in \pi_F(\lambda_F[i])$.

Model-checking fair ATL reduces to a special case of ATL* model-checking. ATL* is an extension of ATL allowing to nest temporal operators. An intensive discussion of ATL* would be beyond the scope of this chapter. The interested reader can find more information in [7]. For our purposes, verifying formulae of the form $\langle\langle A \rangle\rangle \Diamond \varphi$ will be sufficient, and we restrict the presentation of fair ATL model-checking to this case.

Proposition 4.20 [7] A state q of an ATS S, fairly satisfies the ATL formula $\langle\langle A\rangle\rangle\Diamond\varphi$ with respect to a fairness condition Γ, denoted $q \models_F \langle\langle A\rangle\rangle\Diamond\varphi$ if and only if for some $Q_A \in \delta(q, A)$ and for every successor q' of q in Q_A, we have on the corresponding extended ATS S_F that

$$S_F, \langle q, q'\rangle \models \langle\langle A\rangle\rangle \left(\Diamond\varphi \vee \bigvee_{\gamma\in\Gamma, a\in\Sigma\setminus A} \Diamond\Box\left(\langle\gamma, a, \text{enabled}\rangle_F \wedge \neg\langle\gamma, a, \text{taken}\rangle_F\right)\right).$$

□

Intuitively, the above formula says that either the players in A have a strategy, such that φ is satisfied or the players in $\Sigma \setminus A$ violate their fairness constraints. Rather than giving a model-checking algorithm for full ATL*, we only discuss the special case of the formula given in proposition 4.20. Moreover, instead of directly giving the fixpoint algorithm, we first rewrite this ATL* formula to get it under a slightly simplified form.

$$S_F, \langle q, q'\rangle \models \langle\langle A\rangle\rangle \left(\Diamond\varphi \vee \bigvee_{\gamma\in\Gamma, a\in\Sigma\setminus A} \Diamond\Box\left(\langle\gamma, a, \text{enabled}\rangle_F \wedge \neg\langle\gamma, a, \text{taken}\rangle_F\right)\right)$$
$$\Leftrightarrow$$
$$S_F, \langle q, q'\rangle \models \neg [\![A]\!] \left(\Box\neg\varphi \wedge \bigwedge_{\gamma\in\Gamma, a\in\Sigma\setminus A} \Box\Diamond\left(\neg\langle\gamma, a, \text{enabled}\rangle_F \vee \langle\gamma, a, \text{taken}\rangle_F\right)\right)$$
$$\Leftrightarrow$$
$$S_F, \langle q, q'\rangle \models \neg [\![A]\!] \Box \left(\neg\varphi \wedge \bigwedge_{\gamma\in\Gamma, a\in\Sigma\setminus A} \Diamond\left(\neg\langle\gamma, a, \text{enabled}\rangle_F \vee \langle\gamma, a, \text{taken}\rangle_F\right)\right)$$

The fixpoint algorithm for this formula is given in algorithm 2 [7]. The algorithm uses the UPre_F operator, which corresponds to the UPre operator adapted to the structure of the extended ATS S_F, i.e. supposing that δ_F is extended to sets of players in a similar way than δ, $\mathsf{UPre}_F(A, \tau) = \{\langle q, q'\rangle \mid \forall Q_1 \in \delta_F(A, \langle q, q'\rangle) \cdot \exists Q_2 \in \delta_F(\Sigma \setminus A, \langle q, q'\rangle) : Q_1 \cap Q_2 \subseteq \tau\}$.

4.5 Game guarded command language

Modelling a system such as a protocol directly with an alternating transition system would be cumbersome. Instead, we use a simple modelling formalism, based on Dijkstra's guarded command language, previously introduced in [79]. This simple modelling language is a subset of the reactive modules [6] used by the verification tool MOCHA. Let X be a set of variables interpreted over some possibly infinite domains. We denote by $X' = \{x' \mid x \in X\}$ the set of variables obtained by priming each variable in X. Those variables are used to refer to the value of the variables in the next state. A valuation v of the variables X is a function which maps each variable in X into a value in its domain. Denote by $\mathcal{V}_X$ the set of all possible valuations of the variables in X. For any valuation v, we write v' for the valuation obtained by priming each domain variable of v. For any predicate φ over the

Algorithm 2 Symbolic model-checking algorithm for $\langle\langle A\rangle\rangle\Diamond\varphi$ with fairness condition Γ

$\rho := [true]$; $\tau := [true] \setminus [\varphi]$;
while $\rho \not\subseteq \tau$ **do**
  $\rho := \rho \cap \tau$;
  foreach $\gamma \in \Gamma$ **do**
    foreach $a \in \Sigma \setminus A$ **do**
      $\rho'' := [\rho] \cap (([true] \setminus Reg(\langle\gamma, a, enabled\rangle_F)) \cup Reg(\langle\gamma, a, taken\rangle_F))$;
      $\rho' := [false]$; $\tau := [\rho] \cap \rho''$;
      while $\tau' \not\subseteq \rho'$ **do** $\rho' := \rho' \cup \tau'$; $\tau' := \mathsf{UPre}_F(\Sigma \setminus A, \rho') \cap [\neg p])$ **od**;
      $\rho := \tau'$;
    od
  od;
  $\tau := \mathsf{UPre}_F(\Sigma \setminus A, \rho) \cap [\neg p]$
od;
return $[true] \setminus \tau$

variables X and a valuation $v \in \mathcal{V}_X$, denote by $\varphi[v]$ the truth value of the predicate with all the variables of the program P interpreted according to the valuation v. For $Y \subseteq X$, we define $v_{|Y}$ as the valuation with $dom(v_{|Y}) = Y$, such that $\forall y \in Y : v_{|Y}(y) = v(y)$.

Definition 4.21 Given a set of program variables X, a Y-*action* ξ (guarded command) has the form $[\!]$ *guard* $\rightarrow$ *update*, where $Y \subseteq X$, the guard *guard* is a boolean predicate over X, and the update relation *update* is a boolean predicate over $X \cup Y'$. □

We also write $guard_\xi$ and $update_\xi$ to represent the guard and update relation of ξ, respectively. Given a valuation $v \in \mathcal{V}_X$, the action ξ is said to be *enabled* in v if the guard $guard_\xi[v]$ evaluates to true. We assume that for every Y-action, the update relation is functional, that is, for all valuation functions $v \in \mathcal{V}_X$ satisfying the guard, there is exactly one valuation function $u' \in \mathcal{V}_{Y'}$ such that $update_\xi[v \cup u']$ holds. A Y-*process* Ξ is a finite set of Y-actions. We say that the set of variables Y are *the controlled variables* of Ξ.

To model multi-player games, we partition the program variables X into disjoint sets $X_1, X_2, \dots, X_n$, i.e., $X = X_1 \uplus X_2 \uplus \cdots \uplus X_n$. Intuitively, X_i contains the variables updated by player i. A *program* $P = (\Xi_1, \Xi_2, \dots, \Xi_n)$ over X is a tuple of processes such that Ξ_i is an X_i-process. We require the following additional constraint on those processes: for any valuation v of the variables X, for any process Ξ_i, there exists

$$[\!]\ guard_\xi(\mathsf{X}) \rightarrow\ update_\xi(\mathsf{X}, \mathsf{X_i}) \in \Xi_i$$

such that $v \models guard_\xi(X)$. So, in any valuation v, each player has at least one guarded command to play. We also associate to any process Ξ_i a predicate over X_i, noted $init_i(X_i)$, that defines the possible initial valuations of the variables in X_i. The conjunction of those predicates defines the initial states of the game.

Definition 4.22 The *semantics* of a program $P = \{\Xi_1, \Xi_2, \dots \Xi_n\}$ over the program variables X is the alternating transition system S_P where

- $\Sigma = \{1, 2, \ldots, n\}$ (the n players of the program P).
- $Q = \mathcal{V}_X$, i.e., a state of S_P is a valuation of the program variables.
- $Q_0 = \{v \in \mathcal{V}_X \mid \bigwedge_{i=1}^{n} init_i[v]\}$
- $\delta : \mathcal{V}_X \times \Sigma \to 2^{2^{\mathcal{V}_X}}$ is the transition function defined as
$\delta(v, i) = \big\{\{r \mid r \in \mathcal{V}_X \wedge r_{|X_i} = w\} \mid \xi \in \Xi_i \wedge guard_\xi[v] \wedge update_\xi[v \cup w']\big\}$
- Π is a set of boolean predicates over the program variables X.
- π maps a state q, which is a valuation, to the set $\pi(q)$ of propositions, i.e. the predicates of Π, that the valuation satisfies.

□

Note that for all $v \in \mathcal{V}_X$, we have that $\cap_{i=1}^{n} \delta(v, i)$ is a singleton, as the update relation is functional and $X = X_1 \uplus X_2 \uplus \cdots \uplus X_n$. Intuitively, for $i \in \Sigma = \{1, 2, \ldots, n\}$, player i controls the actions in Ξ_i.

A program is run in steps. At each step, player 1 chooses an enabled action $\xi_1 \in \Xi_1$, and updates the values of the variables in X_1 according to the predicate $update_{\xi_1}$. Independently[1] and simultaneously, player 2 chooses an enabled action $\xi_2 \in \Xi_2$ and updates the values of the variables in X_2 according to $update_{\xi_2}$ and so on for the $n-2$ other players.

We introduce fairness in our modelling language at the level of actions. Given a set of actions Ξ_i of player i, a fairness constraint for player i is a subset $\Gamma_i \subseteq \Xi_i$ of actions in regard of which player i must be (weakly) fair. Intuitively, an action that belongs to Γ_i cannot be constantly enabled without being taken by player i.

Definition 4.23 A fairness condition for a program $P = \{\Xi_1, \Xi_2, \ldots, \Xi_n\}$ is a set $\{\Gamma_1, \Gamma_2, \ldots, \Gamma_n\}$ where $\Gamma_i \subseteq \Xi_i$ for $i : 1 \leq i \leq n$. This set of subsets of actions of the program P induces the fairness condition Γ on the ATS defining the semantics of P whose transition relation is δ. Γ contains a fairness constraint γ_ξ, for each action $\xi \in \Gamma_i$, defined as follows:

$$\gamma_\xi(v, j) = \begin{cases} \emptyset & \text{if } i \neq j \\ \emptyset & \text{if } i = j \text{ and } \neg guard_\xi[v] \\ \{Q' \mid Q' \in \delta(v, j) \wedge \forall u' \in Q' : update_\xi[v \cup u']\} & \text{if } i = j \text{ and } guard_\xi[v]. \end{cases}$$

□

Intuitively, when an action is restricted by a fairness constraint, this action, if it can be taken, cannot be ignored an infinite amount of time. We use the subscript F in $\underset{\mathsf{F}}{\|}\ guard_\xi(\mathsf{X}) \rightarrow\ update_\xi(\mathsf{X}, \mathsf{X_a})$ to indicate that ξ is subject to fairness.

Example 4.24 We can now rewrite, in our game guarded command language, the ATS given in example 4.5 and the fairness constraint considered in example 4.15. In this example, and in the remaining of the thesis, we use the following syntactic assumptions.

[1] A player cannot base his choice on the choice of another player.

$$
\begin{aligned}
&\Xi_{\mathsf{a}} \equiv \\
&init_{\mathsf{a}} = \neg \mathsf{x} \\
&\quad [\!] \ \neg \mathsf{x} \rightarrow \ \mathsf{x}' := true \\
&\quad [\!] \ true \rightarrow \\
\\
&\Xi_{\mathsf{b}} \equiv \\
&init_{\mathsf{b}} = \neg \mathsf{y} \\
&\quad [\!]_{\mathsf{F}} \ \neg \mathsf{y} \rightarrow \ \mathsf{y}' := true \\
&\quad [\!] \ true \rightarrow \\
\\
&\Xi_{\mathsf{c}} \equiv \\
&init_{\mathsf{c}} = \neg \mathsf{z} \\
&\quad [\!] \ \mathsf{y} \wedge \neg \mathsf{z} \rightarrow \ \mathsf{z}' := true \\
&\quad [\!] \ true \rightarrow
\end{aligned}
$$

Figure 4.1: An example of a program in our guarded command language

First, in the update part of the command, if a variable x does not occur in the right-hand side of the command, it is considered as unchanged by the command; so, implicitely, we have $x' = x$ for each such variable. Second, the command $[\!] \ true \rightarrow$ means that the player owning this command can choose to leave unchanged its controlled variables. This is a kind of "idle" action. Note that as the guard is true, that action can be taken at any step of the game if no fairness constraints are imposed on the player. In the program in figure 4.1, we express the fact that a can leave the value of x unchanged, whether x equals *true* or *false* by giving a the possibility to *idle*. The fairness constraint in Ξ_b is denoted by adding the subscript F, and means intuitively that this command can only be ignored a finite number of times, when enabled. This fairness constraint corresponds to the fairness constraint introduced in example 4.15. Also note that the guards may contain variables controlled by a different player, while the updated variables must be controlled variables. Moreover, rather than $x' = x$, we use the notation $x' := x$ in the update part of the commands, as required by the MOCHA input language. □

4.6 An implementation using MOCHA and SLANG

The formalisms described in the preceding sections of this chapter are supported by the finite state model-checker MOCHA [8], except for fairness constraints. Fairness constraints are important when modelling security protocols. So we had to adapt the model-checking algorithms using the script language SLANG which is provided in an experimental version of MOCHA.

4.6.1 Implementation of the basic model-checking algorithms

SLANG allows us to directly manipulate the binary decision diagrams (BDDs) [33], which are the structures used to represent sets of states as well as the transition functions. The script language provides primitives for operations on sets of states, such as union and intersection, as well as existential quantification of variables.

Using these basic primitives, we have implemented the CPre and UPre set transformers, such as presented in definition 4.17. We also implemented the different fixpoint algorithms which are the building blocks of the model-checking algorithm 1. Finally, the ATS and the set of states corresponding to where a given proposition holds are constructed using the MOCHA parser, which takes the guarded command language we defined above, as input language.

4.6.2 Implementing fairness constraints

We had to re-implement the MOCHA model-checking algorithm in order to deal with fairness constraints. In section 4.4.2, we introduced an extended version of an ATS which defined propositions specifying whether a fairness constraint is a-enabled, or a-taken in a given state. Fortunately, we do not need to construct such an extended ATS and can avoid major modifications of the existing tool MOCHA. In order to avoid this construction, we restrict ourselves to a subclass of fairness constraints, that are sufficient for our purpose and allow us to define the propositions $\langle\Gamma\times\Sigma\times\{enabled, taken\}\rangle$ on the basic ATS, which can be constructed directly by MOCHA.

Definition 4.25 A restricted fairness constraint γ, with respect to an ATS S, is a fairness constraint such that for any computation $\lambda = q_0, \ldots, q_i, \ldots$, for any player $a \in \Sigma$,

1. if there exists a position $i \geq 0$ such that γ is a-taken at position i, then for any $j > i$, we have that γ is not a-enabled at position j;
2. if there is no position i such that γ is a-taken at position i, then for any $j \geq 0$, we have that

$$\forall Q_a \in \delta(\lambda[j], a) \text{ such that } \lambda[j+1] \in Q_a \cdot \ (\nexists q \in Q \cdot Q_a \in \gamma(q, a))$$

□

Intuitively, the first restriction states that once a fairness constraint is taken, it will never be enabled again. The second restriction means that for a player a, if γ is never a-taken on a given computation, then the choices taken on this computation cannot be subject to fairness constraints in whatever state $q \in Q$. In practice, we need to manually verify that the fairness constraints respect these restrictions. However, when analysing security protocols, as we will see in chapter 5, the restrictions are easily verified. We can now define predicates $\langle\Gamma \times \Sigma \times \{enabled, taken\}\rangle$ for an ATS without the need of constructing an extended ATS.

Definition 4.26 Given an ATS and a set of restricted fairness constraints, we define the predicates $\langle\Gamma \times \Sigma \times \{enabled, taken\}\rangle$, such that

$$\langle\gamma, a, enabled\rangle \text{ is true in state } q \text{ iff } \gamma(q, a) \neq \emptyset$$

and

$$\langle\gamma, a, taken\rangle \text{ is true in state } q \text{ iff } q \in Q_a \subseteq Q \text{ and } Q_a \in \cup_{\widetilde{q}\in Q}\gamma(\widetilde{q}, a).$$

□

Note that the definition of $\langle\gamma, a, enabled\rangle$ is equivalent to the definition given on the extended ATS (cf the last point of definition 4.19). However, the definition of $\langle\gamma, a, taken\rangle$ slightly differs. To determine whether a fairness constraint is a-enabled in a given state of a computation, one has to consider its successor state. Therefore the extended ATS considers states of the form $\langle q, q'\rangle$ where q' is a successor of q. Here, we redefine a predicate $\langle\gamma, a, taken\rangle$, which is defined on a single state. The predicate is true in a state if we can reach this state, by means of a choice, subject to a fairness constraint.

We will now show, that when all the fairness constraints are of the restricted form, model-checking fair ATL does not require the construction of an extended ATS.

Theorem 4.27 Given an ATS S, a set of restricted fairness constraints Γ, and a corresponding extended ATS S_F, for each $A \subseteq \Sigma$, $q \in Q$ and ATL formula φ we have that:

$$S_F, \langle q, q'\rangle \models \langle\langle A\rangle\rangle \left(\Diamond\varphi \vee \bigvee_{\gamma\in\Gamma, a\in\Sigma\setminus A} \Diamond\Box\left(\langle\gamma, a, \text{enabled}\rangle_F \wedge \neg\langle\gamma, a, \text{taken}\rangle_F\right)\right)$$

for every successor q' of q, such that $q' \in Q_A$ for some well-chosen $Q_A \in \delta(q, A)$,

if and only if

$$S, q \models \langle\langle A\rangle\rangle \left(\Diamond\varphi \vee \bigvee_{\gamma\in\Gamma, a\in\Sigma\setminus A} \Diamond\Box\left(\langle\gamma, a, \text{enabled}\rangle \wedge \neg\langle\gamma, a, \text{taken}\rangle\right)\right).$$

□

Proof:
We start by showing that for a given ATS S, a set of restricted fairness constraints Γ, and a corresponding extended ATS S_F, for each agent $a \in A$, $q \in Q$ and ATL formula φ, if

$$S_F, \langle q, q'\rangle \models \langle\langle A\rangle\rangle \left(\Diamond\varphi \vee \bigvee_{\gamma\in\Gamma, a\in\Sigma\setminus A} \Diamond\Box\left(\langle\gamma, a, \text{enabled}\rangle_F \wedge \neg\langle\gamma, a, \text{taken}\rangle_F\right)\right)$$

for every successor q' of q, such that $q' \in Q_A$ for some well-chosen $Q_A \in \delta(q, A)$,

then

$$S, q \models \langle\langle A\rangle\rangle \left(\Diamond\varphi \vee \bigvee_{\gamma\in\Gamma, a\in\Sigma\setminus A} \Diamond\Box\left(\langle\gamma, a, \text{enabled}\rangle \wedge \neg\langle\gamma, a, \text{taken}\rangle\right)\right).$$

If the set of players in A have a strategy, such that the ATL* formula

$$\left(\Diamond\varphi \vee \bigvee_{\gamma\in\Gamma, \Sigma\setminus A} \Diamond\Box\left(\langle\gamma, a, \text{enabled}\rangle_F \wedge \neg\langle\gamma, a, \text{taken}\rangle_F\right)\right)$$

holds, then there exists a set of winning strategy function F_A. As there exists a one-to-one correspondance between computations and strategies of S and S_F (cf page 76), we have for each computation $\lambda_F \in out(\langle q, q' \rangle, F_A)$, such that

$$\lambda_F = \langle q_0, q_1 \rangle \langle q_1, q_2 \rangle \ldots \langle q_i, q_{i+1} \rangle \ldots,$$

a corresponding computation

$$\lambda = q_0 q_1 \ldots q_i q_{i+1} \ldots$$

over the ATS S.

We are now proving that λ would also validate the ATL* formula on the ATS S. Note, that φ does not range over the propositions $\langle \gamma, a, \text{enabled} \rangle$ and $\langle \gamma, a, \text{taken} \rangle$. Since φ only ranges over propositions in $\Pi \cap \Pi_F$ and there exists a one-to-one correspondance between computations and strategies over S and S_F, we have to show that for any $\gamma \in \Gamma$ and $a \in \Sigma \setminus A$,

$$\Diamond\Box(\langle \gamma, a, \text{enabled} \rangle_F \wedge \neg \langle \gamma, a, \text{taken} \rangle_F)$$

holds on λ_F if and only if

$$\Diamond\Box(\langle \gamma, a, \text{enabled} \rangle \wedge \neg \langle \gamma, a, \text{taken} \rangle)$$

holds on λ.

First note that, by definition, we have that, for any $\langle q, q' \rangle \in Q_F$,

$$\langle q, q' \rangle \models \langle \gamma, a, \text{enabled} \rangle_F \Leftrightarrow q \models \langle \gamma, a, \text{enabled} \rangle.$$

For the "taken" case, we discuss two cases, whether there exists a position on λ_F where γ is a-taken or not.

1. There exists at least one position on λ where γ is a-taken.

   Let us denote by $i \geq 0$, the first position, such that γ is a-taken at $\lambda[i]$. Due to the first restriction on fairness constraints given in definition 4.25, we have that for any $j > i$ that γ is not $(\Sigma \setminus A)$-enabled at position j. As discussed on page 76, $\lambda_F[j] \models \langle \gamma, a, \text{enabled} \rangle_F$ if and only if γ is a-enabled at position j on λ. Hence,

   $$\lambda_F[j] \not\models \langle \gamma, a, \text{enabled} \rangle_F.$$

   As for any $\langle q, q' \rangle \in Q_F$,

   $$\langle q, q' \rangle \models \langle \gamma, a, \text{enabled} \rangle_F \Leftrightarrow q \models \langle \gamma, a, \text{enabled} \rangle$$

   we have that for any $j > i$ that

   $$\lambda[j] \not\models \langle \gamma, a, \text{enabled} \rangle.$$

   Hence,

   $$\Diamond\Box(\langle \gamma, \Sigma \setminus A, \text{enabled} \rangle_F \wedge \neg \langle \gamma, a, \text{taken} \rangle_F)$$

   neither holds on λ_F nor does

   $$\Diamond\Box(\langle \gamma, a, \text{enabled} \rangle \wedge \neg \langle \gamma, a, \text{taken} \rangle)$$

   hold on λ.

2. There does not exist any position i on λ, such that γ is a-taken in $\lambda[i]$.

   As for any $i \geq 0$, γ is not a-taken on λ at position i, we have that $\lambda_F[i] \not\models \langle\gamma, a, \text{taken}\rangle_F$. We show that $\lambda_F[i] \not\models \langle\gamma, a, \text{taken}\rangle_F$ implies that $\lambda[i] \not\models \langle\gamma, a, \text{taken}\rangle$. The last point of definition 4.19 implies that for all $Q_a \in \delta(\lambda[i], a)$, such that $\lambda[i+1] \in Q_a$
$$Q_a \notin \gamma(\lambda[i], a).$$
   By the second restriction on fairness constraints, we have that
$$\nexists q \in Q \cdot \; Q_a \in \gamma(q, a).$$
   Hence,
$$Q_a \notin \cup_{q \in Q} \gamma(q, a)$$
   and, by definition 4.26, $\langle\gamma, a, \text{taken}\rangle$ is false in $\lambda[i+1]$. Now, for any $i > 0$ we have that
$$\lambda_F[i] \models \neg\langle\gamma, a, \text{taken}\rangle_F,$$
$$\lambda[i] \models \neg\langle\gamma, a, \text{taken}\rangle$$
   and
$$\lambda_F[i] \models \langle\gamma, a, \text{enabled}\rangle_F \Leftrightarrow \lambda[i] \models \langle\gamma, a, \text{enabled}\rangle.$$
   We can conclude that, for $i > 0$,
$$\lambda_F[i] \models \langle\gamma, a, \text{enabled}\rangle_F \wedge \neg\langle\gamma, a, \text{taken}\rangle_F$$
$$\Leftrightarrow$$
$$\lambda[i] \models \langle\gamma, a, \text{enabled}\rangle \wedge \neg\langle\gamma, a, \text{taken}\rangle.$$
   Hence,
$$\Diamond\Box(\langle\gamma, a, \text{enabled}\rangle_F \wedge \neg\langle\gamma, a, \text{taken}\rangle_F)$$
   holds on λ_F if and only if
$$\Diamond\Box(\langle\gamma, a, \text{enabled}\rangle \wedge \neg\langle\gamma, a, \text{taken}\rangle)$$
   holds on λ.

This terminates the first part of the proof.

In the second part of the proof, we have to show that, given an ATS S, a set of restricted fairness constraints Γ, and a corresponding extended ATS S_F, for each agent $a \in A$, $q \in Q$ and ATL formula φ
if
$$S, q \models \langle\langle A \rangle\rangle \left(\Diamond\varphi \vee \bigvee_{\gamma \in \Gamma, a \in \Sigma \setminus A} \Diamond\Box\left(\langle\gamma, a, \text{enabled}\rangle \wedge \neg\langle\gamma, a, \text{taken}\rangle\right)\right)$$
then there exists a set $Q_A \in \delta(q, A)$ such that for every successor q' of q, $q' \in Q_A$, we have that
$$S_F, \langle q, q' \rangle \models \langle\langle A \rangle\rangle \left(\Diamond\varphi \vee \bigvee_{\gamma \in \Gamma, a \in \Sigma \setminus A} \Diamond\Box\left(\langle\gamma, a, \text{enabled}\rangle_F \wedge \neg\langle\gamma, a, \text{taken}\rangle_F\right)\right)$$

We start by showing that, if there exists a winning strategy function F_A on the initial ATS, then there exists a set $Q_A \in \delta(q, A)$ such that for every successor q' of q, $q' \in Q_A$, all the computations in $out(\langle q, q' \rangle, F_A)$ correspond to computations in $out(q, F_A)$. The existence of Q_A is trivial, as we can always choose $Q_A = \cap_{a \in A} f_a(q)$, where f_a is player a's strategy given by F_A.

Since, we have already shown in the first part of the proof that

$$\Diamond\Box(\langle\gamma, a, \text{enabled}\rangle_F \wedge \neg\langle\gamma, a, \text{taken}\rangle_F)$$

holds on λ_F if and only if

$$\Diamond\Box(\langle\gamma, a, \text{enabled}\rangle \wedge \neg\langle\gamma, a, \text{taken}\rangle)$$

holds on λ we can conclude the proof. $\Box$

The above shown result is usefull in practice, when we need to verify fair ATL formulae, since we do not require the construction of an extended ATS.

4.7 Conclusion

In this chapter, we have introduced ATS, the corresponding logic ATL, as well as a guarded command language which provides a convenient way to describe an ATS. We also give the model-checking algorithms, and discuss our implementation of them using the script language of the tool MOCHA. Moreover, we have shown how to extend ATS and ATL to handle fairness constraints. The semantics of fair ATL, which we gave here, slightly differ from the original paper [7]. Moreover, we have proven that for a specific class of fairness constraints, which we call restricted fairness constraints, the model checking can be simplified and avoid the construction of an extended ATS. In the following chapters, we are going to use the here introduced tools to analyse optimistic fair exchange protocols.

CHAPTER 5

Modelling protocols and security goals

In the present chapter, we are going to introduce a specification language for optimistic fair exchange protocols. We are giving the semantics of this language in terms of ATS, using the guarded command language presented in chapter 4. Then, we define the properties a fair exchange protocol should respect using ATL. Finally, we briefly report on the implementation of a tool which automates the translation from our protocol specification language (defined below) into guarded commands, on which the properties can be verified using the tool Mocha. The definition of the specification language and the automation of the translation have been realized in collaboration with Alexandre Boisseau and Jean-François Raskin. The definition of the properties has been carried out in collaboration with Jean-François Raskin [88, 89, 90].

5.1 Protocol specification language

In the literature, protocols are often described using an informal notation, such as $\mathsf{A} \to \mathsf{B}$: m, to denote that A sends a message m to B. Such a notation may be ambiguous, mainly if the protocols are branching, as it is the case in optimistic exchange protocols. It is difficult to express with such a notation where branching is possible and/or allowed by the protocol. Therefore we introduce here a protocol specification language that makes branching explicit and directly includes details, such as the quality of the communication channels, into the specification. We first introduce the running example of this section. Then we present the syntax and the semantics of this protocol specification language.

Example 5.1 To illustrate that specifying optimistic fair exchange protocols may be difficult, consider the informal description of the ASW certified e-mail protocol [14] presented in 1998 by Asokan, Shoup and Waidner. The ASW certified e-mail protocol has been designed for use on asynchronous (resilient) networks. The protocol uses a main protocol, an abort protocol and two recovery protocols. Before describing the protocol, we introduce some additional notations specific to this protocol.

- N_A and N_B: two nonces generated by A and B, respectively.
- $c = E_{TTP}(m, N_A, A, B)$: the encryption with the TTP's public key of the message m, concatenated with N_A and the identities of the participants.

Main protocol. The main protocol (protocol 5.1) consists of four messages. In the first message, Alice sends to Bob a commitment to the message m, as well as to the non-repudiation of origin evidence, by sending the hash of the message m, as well as the encryption for the TTP of both the message and the nonce N_A. Bob replies by sending a commitment to his nonce N_B. Then they go on in messages 3 and 4 by exchanging m and N_A against N_B.

Protocol 5.1 ASW certified e-mail protocol—Main

1. A → B: $m_1 = A, B, \mathsf{TTP}, c, h(m), S_A(A, B, \mathsf{TTP}, c, h(m))$
2. B → A: $m_2 = h(N_B), S_B(m_1, h(N_B))$

   if A times out then abort
3. A → B: $m_3 = m, N_A$

   if B times out then recovery_B
4. B → A: $m_4 = N_B$

   if A times out then recovery_A

Abort protocol. The abort protocol is executed by Alice if Bob does not reply quickly enough to the first message of the main protocol. When Alice wants to abort the protocol (protocol 5.2), she sends a signed abort request to the TTP. If at that moment, Bob has already recovered the protocol, the TTP requests Alice to launch a recovery protocol. Otherwise the TTP sends an abort confirmation to Alice.

Protocol 5.2 ASW certified e-mail protocol—Abort

1. A → TTP: $a_1 = aborted, m_1, S_A(aborted, m_1)$

   if B recovered **then** recovery_A **else**
2. TTP → A: $a_2 = abort\_token = a_1, S_{\mathsf{TTP}}(a_1)$

Recovery protocol. When the third message of the main protocol does not arrive quickly enough to Bob, he can execute a recovery protocol (protocol 5.3). Therefore Bob sends a recovery request, including the cipher c to the TTP. If the protocol has already been aborted, the TTP sends the abort token to Bob. Otherwise the TTP decrypts c and sends the message m, as well as N_A to Bob.

Protocol 5.3 ASW certified e-mail protocol—Recovery_B

1. B → TTP: $r_{b1} = B, m_1, m_2, N_B$

   if aborted **then**

2. TTP → B: $a_2 = abort\_token$

   else

3. TTP → B: $r_{b2} = m, N_A$

In the same way as Bob, Alice can also recover the protocol (protocol 5.4) if the fourth message of the main protocol is not arriving quickly enough. If the protocol has been aborted previously, the TTP sends the abort token. Otherwise an affidavit is issued by the TTP, replacing N_B.

Protocol 5.4 ASW certified e-mail protocol—Recovery_A

1. A → TTP: $r_{a1} = A, m_1, m_2, m, N_A$

   if aborted **then**

2. TTP → A: $a_2 = abort\_token$

   else

3. TTP → A: $r_{a2} affidavit\_token = affidavit, r_{a1}, S_{TTP}(affidavit, r_{a1})$

The non-repudiation of origin evidence NRO corresponds to (m_1, N_A) and the non-repudiation of receipt evidence NRR to (m_1, m_2, N_B) or *affidavit_token*. □

We may see that the above given protocol description has several drawbacks when we want to analyse the protocol. It is not easy to see where are the different possible branchings in the protocol. Moreover, as we only give the description of a honest player, determining whether and where a dishonest protocol can launch subprotocols at unforeseen moments is not trivial. In our specification language, we are going to give the description of each participant in terms of a tree, which has the advantage of visualising the foreseen branching points. It is also possible to explicitly define whether a participant is honest or not and to systematically induce the possible dishonest behaviours, given only the description of the honest behaviours. Moreover, the communication channels and their respective quality are explicitly defined between each ordered pair of participants and are part of the protocol description.

Before giving the syntax and semantics of this specification language, we want to make the following comments. The semantics of the specification language will define the model in which our analysis is carried out. As all mathematical models, our model is only an approximation of the reality and therefore all the claims we can make are specific to the defined model. Let us already make some simplifications of our model explicit. We do not consider any *social attacks*, e.g. use some trick to make a user reveal his passphrase or secret key. Moreover, we do not verify a given implementation and do not give any guarantees with respect to weaknesses such as buffer overflows. Such simplifications may seem natural and are rather standard when using formal methods to analyse protocols, but misunderstandings about protocols "proven correct" do exist [142] and therefore we prefer to emphasise these points.

We now continue by giving some simplifications that are sometimes relaxed nowadays, above all when analysing authentication protocols. We assume perfect cryptography as suggested in the Dolev-Yao model [58], i.e. it is impossible to break, even partially, an encryption or forge a signature on whatever message. Nor do we consider any algebraic properties of a specific cryptographic scheme. We also suppose strong typing, i.e. one can never confuse messages that are not of exactly the same type. Heather et al. have shown in [77] that strong typing can be achieved using explicit flags which indicate the purpose of each part of the message. Moreover, we only consider one protocol run and restrict the messages that could be send on the network, to the messages of the given protocol execution. These simplifications are motivated by the specificity of the considered protocols. Generally, protocol messages are digitally signed which makes it impossible to modify them. Moreover, the protocol messages contain session identifiers, which should prevent interference between different runs of a same protocol. These session identifiers are crucial in optimistic fair exchange protocols, as the TTP must be able to identify each protocol execution. In our running example (the ASW certified e-mail protocol) m_1, which depends on the message M, the nonce N_A, as well as the identities of the participating entities, is used as the session identifier and included in each of the recovery and abort request. Of course these simplifications imply that one could design trivially flawed protocols, e.g. by omitting to specify any identifier, which would not be considered as flawed in our model. Imagine that the TTP cannot identify a protocol session. Then an easy attack for Bob is as follows. In a first session, Bob stops after having received the first message. In a second session, Bob waits for having received the third message of the main protocol, before stopping. As the TTP cannot distinguish the two sessions, the TTP would not accept Alice's recovery request, because she already recorded an abort. Such an attack is not captured in our model. However, as reported in chapter 6, when analysing real protocols, our model has shown to be sufficient to detect subtle flaws and is interesting even with these simplifications. In chapter 8, we will see that some other methods make similar simplifications when studying optimistic fair exchange protocols.

5.1.1 Syntax

5.1.1.1 Messages

We suppose a set of types

$$\{Id, Key, Nonce, Data, SymmetricCipher, AsymmetricCipher, Hash, Signature, PCS\}$$

and a given set C of constants, together with a partition (C_{Id}, C_{Key}, C_{Nonce}, C_{Data}) of C. We can then define the set of messages Msg by the grammar given in figure 5.1.

$$
\begin{array}{rcl}
Msg & ::= & Id \mid Key \mid Nonce \mid \\
 & & Data \mid SymmetricCipher \mid \\
 & & AsymmetricCipher \mid Hash \mid \\
 & & Signature \mid PCS \mid Msg, Msg \\
Id & ::= & C_{Id} \\
Key & ::= & C_{Key} \\
Nonce & ::= & C_{Nonce} \\
Data & ::= & C_{Data} \\
SymmetricCipher & ::= & E_{Key}(Msg) \\
AsymmetricCipher & ::= & E_{Id}(Msg) \\
Hash & ::= & h(Msg) \\
Signature & ::= & S_{Id}(Msg) \\
PCS & ::= & PCS_{Id}(Msg, Id, Id)
\end{array}
$$

Figure 5.1: Grammar for messages.

5.1.1.2 Actions

We define two basic kinds of actions: send actions and receive actions. Let *Act* denote the set of actions, defined by the grammar given in figure 5.2. Note that we do not define who is the sender in a receive action, in order to model that one cannot determine the sender of a message, unless the content of the message can be used to deduce it.

$$
\begin{array}{rcll}
Act & ::= & Act_{\mathsf{send}} \mid Act_{\mathsf{recv}} & \\
Act_{\mathsf{send}} & ::= & send(m)\ to\ u & (m \in Msg, u \in C_{Id}) \\
Act_{\mathsf{recv}} & ::= & rec(m) & (m \in Msg)
\end{array}
$$

Figure 5.2: Grammar for actions

5.1.1.3 Participants

Definition 5.2 We define a *participant* that may interact in a protocol, as a 4-tuple $P = (id, role, ik, behaviour)$, where

- $id \in C_{Id}$ is the unique identity of the participant;
- $role$ gives a description of the honest behaviour of a participant. A role R is a finite tree[1] where edges are labeled by *Act* and any two edges starting at the same node are labelled by different elements in *Act* (equivalently, R is a finite, prefix-closed language in $(Act)^*$). Moreover, we suppose that any word $a_1 \ldots a_n \in R$ is such that if $i \neq j$ then $a_i \neq a_j$[2];

[1] An interesting extension could be to consider graphs, instead of trees, with some restriction, e.g. use acyclic graphs, avoiding infinite loops. We here only consider trees for the sake of simplicity.

[2] In practice this is not restrictive: if the protocol foresees to send equal messages, then they should be differentiated by specific labels.

- $ik \subseteq Msg$ is a finite set of messages representing the initial knowledge of a participant;
- $behaviour \in \{strongly, weakly, honest, ttp\}$ describes whether the participant is following the protocol in a malicious or honest way. More details about the several levels of behaviour are given when defining the semantics.

□

Definition 5.3 The set of *positions* of a role R, denoted $\mathsf{pos}(R)$, is defined as follows. Each position $(a_1 \dots a_n)$ in $\mathsf{pos}(R)$ corresponds to a word $a_1 \dots a_n \in R$ (including the empty word ϵ). □

Definition 5.4 The set of *terminal positions* of a role R, denoted $\mathsf{pos_T}(R)$ is the set of positions $(a_1 \dots a_n)$ corresponding to a word $a_1 \dots a_n \in R$, such that there exists no $a \in Act$ with $a_1 \dots a_n a \in R$. □

Example 5.5 Consider for instance the ASW protocol and the set of participants

$$\mathcal{P}art_{ASW} = \{Alice, Bob, TTP\}.$$

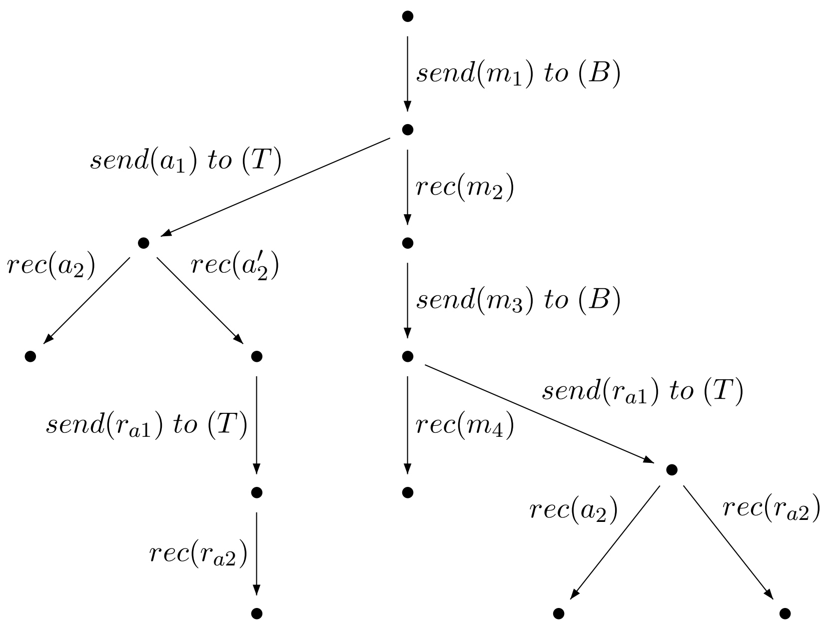
We may define the participants in the following way:

$$\begin{array}{lcl} Alice & = & (A, R_A, \{N_A, M\}, weakly) \\ Bob & = & (B, R_B, \{N_B, M\}, honest) \\ TTP & = & (T, R_T, \emptyset, ttp) \end{array}$$

where the respective roles are depicted in figures 5.3, 5.4 as well as 5.5. There are some minor differences with the original protocol. On one hand, in our specification language, we suppose that the signatures allow message recovery, while in the informal protocol description, as given in [14], the signatures were with appendix (for more details the reader is referred to [110]). On the other hand, the informal protocol description did not foresee an explicit message, when the TTP has to request Alice to launch a recovery protocol. This action is described in the text going with the protocol. Therefore, we had to introduce the additional message a_2'.

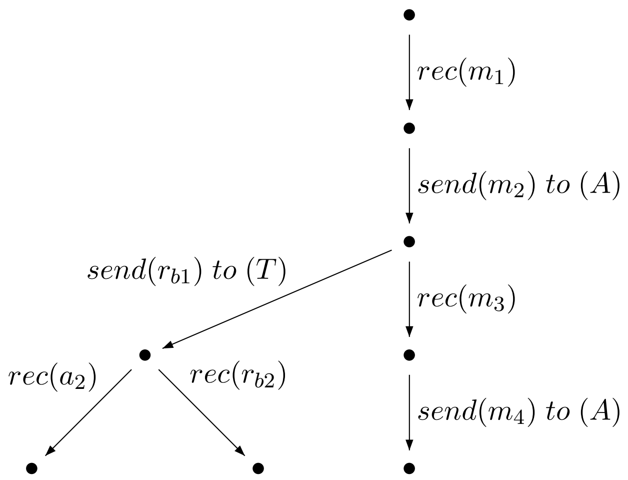
Note, that describing the TTP role as a tree is the most complicated part, and probably the most error-prone. We have to foresee the different possible executions, i.e. for each protocol, we need to foresee the launching of all other protocols afterwards. This step could be improved by having a specific description language for the TTP, which would automate this part even more.

Note that public keys, owned private keys and identities do not need to be specified in the initial knowledge. They are always supposed to be known. In this example Alice is weakly dishonest and Bob is honest. The type declarations of the constants has been omitted, but they should be clear from the protocol description, given before. □



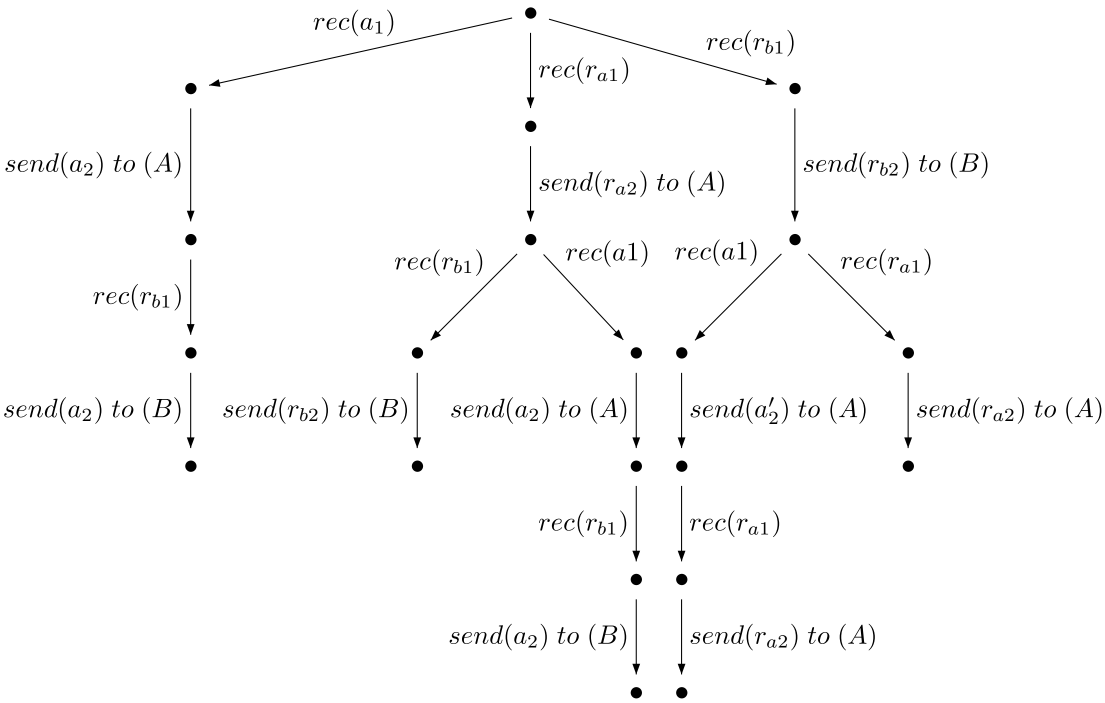
$m_1 = S_A(A, B, T, E_T(M, N_A, A, B), h(M))$
$m_2 = S_B(m_1, h(N_B))$
$m_3 = M, N_A$
$m_4 = N_B$
$a_1 = S_A(\mathsf{abort}, m_1)$
$a_2 = S_T(a_1)$
$a'_2 = \mathsf{recovery}$
$r_{a1} = A, m_1, m_2, M, N_A$
$r_{a2} = S_T(\mathsf{affidavit}, r_{a1});$

Figure 5.3: A-role of the ASW certified e-mail protocol



$m_1 = S_A(A, B, T, E_T(M, N_A, A, B), h(M))$
$m_2 = S_B(m_1, h(N_B))$
$m_3 = M, N_A$
$m_4 = N_B$
$a_2 = S_T(a_1)$
$r_{b1} = B, m_1, m_2, N_B$
$r_{b2} = M, N_A$

Figure 5.4: B-role of the ASW certified e-mail protocol



$m_1 = S_A(A, B, T, C, h(M))$
$m_2 = S_B(m_1, h(N_B))$
$a_1 = S_A(\textsf{abort}, m_1)$
$a_2 = S_T(a_1)$
$a_2' = \textsf{recovery}$
$r_{a1} = A, m_1, m_2, M, N_A$
$r_{a2} = S_T(\textsf{affidavit}, r_1)$
$r_{b1} = B, m_1, m_2, N_B$
$r_{b2} = M, N_A$

Figure 5.5: T-role of the ASW certified e-mail protocol

5.1.1.4 Communication channels

Definition 5.6 We define a *communication channel* as a triple $C = (S, R, Quality)$, where

- S and R in C_{Id} are respectively the sender and the receiver of the information sent on the communication channel;
- $Quality \in \{operational, resilient, unreliable\}$ defines the quality of service provided by the communication channel. Details about the different levels of channel quality will be given, while defining the semantics.

$\square$

Example 5.7 Consider again our running example, the ASW protocol. We have the set of communication channels

$$\mathcal{C}om_{ASW} = \{C_{AB}, C_{BA}, C_{AT}, C_{TA}, C_{BT}, C_{TB}\}$$

where a possible definition for these channels could be:

$$\begin{array}{ll} C_{AB} = (A, B, unreliable) & C_{BA} = (B, A, unreliable) \\ C_{AT} = (A, T, resilient) & C_{TA} = (T, A, resilient) \\ C_{BT} = (B, T, resilient) & C_{TB} = (T, B, resilient) \end{array}$$

In our example we have defined the channels between Alice and Bob as being unreliable, while all the channels to or from the TTP are resilient. This defines the assumptions about the network quality, under which the protocol will be analysed. They are easily modified and stated explicitly in the protocol description. $\square$

5.1.1.5 Protocol instance

Definition 5.8 A *protocol instance* is a finite set of participants $\mathcal{P}art$ and a finite set of communication channels $\mathcal{C}om$ such that for each participant

$$P = (p, R_P, ik_P, behaviour_P) \in \mathcal{P}art$$

and each position $(a_1 \ldots a_n) \in \mathsf{pos}(R_P)$, we have that

- if $a_n = rec(m)$, then there exists a participant

  $$Q = (q, R_Q, ik_Q, behaviour_Q) \in \mathcal{P}art$$

  such that $P \neq Q$ and there exists a position $(a'_1 \ldots a'_{n'}) \in \mathsf{pos}(R_Q)$, such that $a'_{n'} = send(m)\ to\ p$;

- if $a_n = send(m)\ to\ q$, then there exists a participant

  $$Q = (q, R_Q, ik_Q, behaviour_Q) \in \mathcal{P}art$$

  such that $P \neq Q$ and there exists a position $(a'_1 \ldots a'_{n'}) \in \mathsf{pos}(R_Q)$, such that $a'_{n'} = rec(m)$.

Moreover,

$$\mathcal{C}om = \{C_{sr} = (s, r, q) \mid \exists S \neq R \in \mathcal{P}art \cdot\ S.id = s, R.id = r \text{ and } q \text{ is uniquely determined by } S \text{ and } R\}.$$

□

Intuitively, in a protocol instance, we fix a finite set of participants for which the roles match, i.e. each message that can be received, should have a potential sender, and vice-versa, each message that can be sent should have a potential receiver in the protocol instance. Moreover, the communication channels connect each ordered pair of participants. We suppose that the protocol description does satisfy these conditions. The current version of our tool, described in section 5.3, does not check that a protocol description is well formed.

Definition 5.9 Given a protocol instance, we define the set M of *protocol messages* as the following set of messages.

$$M = \{m \in Msg \mid \exists P = (p, R_P, ik_P, behaviour_P) \in \mathcal{P}art \cdot \exists (a_1 \dots a_n) \in \mathsf{pos}(R_P) \cdot\ a_n = rec(m)\}$$

□

5.1.2 Semantics

We suppose a given protocol instance and the corresponding set of protocol messages M. For this instance we can define the variables controlled by each of the players.

- For each honest or TTP participant $P = (p, R_P, ik_P, behaviour_P) \in \mathcal{P}art$, we have the following variables and associated domains:
  - $P.pos$: $\mathsf{pos}(R_P)$ the current position in the role describing the participant's behaviour,
  - $P.rec$: 2^M the set of received messages,
  - $\forall u \neq p \cdot\ P.send\_to\_u$: 2^M the set of messages sent to participant u,
  - $P.stop$: boolean indicating whether P stopped the protocol or not.
- For each (weakly or strongly) dishonest participant $P = (p, R_P, ik_P, behaviour_P) \in \mathcal{P}art$, we have the following variables and associated domains:
  - $P.rec$: 2^M the set of received messages,
  - $\forall u \neq p \cdot\ P.send\_to\_u$: 2^M the set of messages sent to participant u,
  - $P.stop$: boolean indicating whether P stopped the protocol or not.
- For each communication channel $C_{uv} \in \mathcal{C}om$ we have
  - $C_{uv}.fwd$: 2^M the set of messages sent by u and delivered to v.

Let us denote by X this set of typed variables.

Definition 5.10 A *protocol state* is a valuation v that assigns to each variable $x \in X$ a value of the intended type. The set of protocol states Q is the set of all possible valuations. □

One can note that the set of protocol states is always finite, as we only consider the finite set M of protocol messages. To define the transition function of each player we will use the guarded command language defined in section 4.5. The guarded commands will be built from the following predicates given with their semantics. For each participant P and each message $m \in M$ we have

- $v \models P.\mathit{send}(m)\_to\_u$ iff $m \in v(P.\mathit{send}\_to\_u)$,
- $v \models P.\mathit{rec}(m)$ iff $m \in v(P.\mathit{rec})$,
- $v \models P.\mathit{stop}$ iff $v(P.\mathit{stop}) = \mathit{true}$.

Moreover, for each honest and TTP participant P and each position $c \in \mathsf{pos}(R_P)$ we have:

- $v \models P.c$ iff $v(P.\mathit{pos}) = c$.

For each communication channel C_{uv} and each message m we have

- $v \models C_{uv}.\mathit{fwd}(m)$ iff $m \in v(C_{uv}.\mathit{fwd})$.

Moreover, guarded commands respect the following rules:

- $(v_1, v_2) \models p'$ iff $v_2 \models p$,
- $(v_1, v_2) \models p$ iff $v_1 \models p$,
- boolean operators have the usual interpretation.

We introduce one more predicate, that needs to be presented in more details. Following the Dolev-Yao model [58], we formalise the capacity of a participant to construct a given message by a predicate that we call *knows*. The value of this predicate for a participant P depends on P's initial knowledge, as well as the set of received or spied messages, being function of the state v of the ATS where the predicate is evaluated. The definition of *knows* is formalised by the inference rules given in figure 5.6. Note that, in this figure, the type of some of the variables is left undefined for the sake of readability, as the type should be clear from the context. In particular, we suppose that a, b, t and u are in C_{Id} and $k \in C_{Key}$. We also remind the reader that *sk* and *pk* return the secret, respectively public, key of a given identity. These keys are supposed to be in C_{Key}. One can also notice, that we do not make precise how P knows which key, in the set of keys he may know, he must use to decipher a message. We only model the capacity of deciphering the message, not the way he is going to realize this action.

In the sequel, for a set of messages S, we denote by $\mathit{knows}(S, m)$ the fact that the message m can be constructed by a given entity P if the set of messages S is known. This notation is related to the previous one by the following statement. If v and ik_P are such that $S = \{m' \in Msg \mid v \models P.\mathit{rec}(m')\} \cup ik_P$ then $\mathit{knows}(S, m) \Leftrightarrow v \models P.\mathit{knows}(m)$.

Paulson [126] has shown that the relation $\mathit{knows}(S, m)$, in the case we only consider atomic keys, can be decided in two steps:

Analysis rules:

$$\frac{m \in ik_P}{v \models P.knows(m)} \qquad \frac{v \models P.rec(m)}{v \models P.knows(m)}$$

$$\frac{u \in C_{Id}}{v \models P.knows(u)} \qquad \frac{}{v \models P.knows(sk(P.id))} \qquad \frac{u \in C_{Id}}{v \models P.knows(pk(u))}$$

$$\frac{v \models P.knows(m_1, m_2)}{v \models P.knows(m_1)} \qquad \frac{v \models P.knows(m_1, m_2)}{v \models P.knows(m_2)}$$

$$\frac{v \models P.knows(E_k(m)) \quad v \models P.knows(k^{-1})}{v \models P.knows(m)} \qquad \frac{v \models P.knows(S_u(m))}{v \models P.knows(m)}$$

$$\frac{v \models P.knows(PCS_a(m, b, t))}{v \models P.knows(m)}$$

Synthesis rules:

$$\frac{v \models P.knows(m_1) \quad v \models P.knows(m_2)}{v \models P.knows(m_1, m_2)}$$

$$\frac{v \models P.knows(m) \quad v \models P.knows(k)}{v \models P.knows(E_k(m))} \qquad \frac{v \models P.knows(m)}{v \models P.knows(h(m))}$$

$$\frac{v \models P.knows(m) \quad v \models P.knows(sk(u))}{v \models P.knows(S_u(m))}$$

$$\frac{v \models P.knows(sk(a)) \quad v \models P.knows(m)}{v \models P.knows(PCS_a(m, b, t))}$$

$$\frac{v \models P.knows(PCS_a(m, b, t)) \quad v \models P.knows(sk(t))}{v \models P.knows(S_a(m))}$$

Figure 5.6: Inference rules defining the *knows* predicate

1. using a function $\mathsf{analz} : 2^{Msg} \to 2^{Msg}$ we associate to a set of messages its closure under the analysis rules;

2. using a function $\mathsf{synth} : 2^{Msg} \to 2^{Msg}$ we associate to a set of messages its closure under the synthesis rules. Then we have $knows(S, m)$ is true if and only if $m \in \mathsf{synth}(\mathsf{analz}(S))$.

One can notice that:

1. $\mathsf{analz}(S)$ is a subset of the set of all subterms of S, and hence is finite;

2. if S is finite, knowing whether $m \in \mathsf{synth}(S)$ is decidable, since the computation is bounded by the size of m.

Hence, if S is finite, computing $knows(S, m)$ is decidable, and moreover, for a given message m and a finite set of messages M, we can compute the set of minimal subsets $S \subseteq M$, such that $knows(S, m)$ holds. For more information about the algorithm and implementation of the *knows* predicate, we refer the reader to Boisseau's thesis [27].

We are now equipped to give the semantics of the protocol description language previously described in this chapter. The semantics are expressed in terms of an ATS. As the direct description of the ATS would be rather cumbersome, the ATS will be described in terms of guarded commands. For each participant P and each communication channel C_{uv} of the protocol instance, we describe the corresponding player ϕ_P, respectively $\phi_{C_{uv}}$, in terms of guarded commands. We will also give a predicate $init_P$, respectively $init_{C_{uv}}$, for each of the processes. The predicate *init* defining the initial states of the ATS will be the conjunction of the local *init* predicates. Note that each atomic variable x owned by P, respectively by C_{uv}, will be denoted under the form $P.x$, respectively $C.x$.

Remark 5.11 In section 4.5, we gave the constraint that for any process and any valuation, there exists a guarded command that is enabled, i.e. the transition function is non blocking. In the following sections, for this constraint to be met, we assume that an additional guarded command is always added. Denote by $\xi_1, \ldots, \xi_n$ the guarded commands of a given process. The additional guarded command is defined as follows.

$$[\!] \bigwedge_{\mathsf{i}=1}^{\mathsf{n}} \neg guard_{\xi_\mathsf{i}} \to$$

With this additional guarded command the constraint is trivially verified as any process must idle if none of its guarded commands are enabled. There is no need for adding a fairness constraint with respect to this guard, since the guard can only be taken when all other guards cannot, and in that case it must be taken. □

5.1.2.1 Participants

For each participant $P = (p, R_P, ik, behaviour)$ we now describe the corresponding player ϕ_P. For any honest or TTP participant P, we have the following $init_P$ predicate, where

M is the set of protocol messages.

$$\begin{aligned} init_P = \quad & \bigwedge_{m \in M, u \in \mathcal{P}art \setminus \{P\}} \neg P.\, send(m)\ to\ (u.id) \\ & \wedge \bigwedge_{m \in M} \neg P.\, rec(m) \\ & \wedge \bigwedge_{c \in \mathsf{pos}(R_P) \setminus \{\epsilon\}} \neg P.c \\ & \wedge P.\epsilon \wedge \neg P.stop \end{aligned}$$

For any dishonest participant P, we have the following $init_P$ predicate.

$$\begin{aligned} init_P = \quad & \bigwedge_{m \in M, u \in \mathcal{P}art \setminus \{P\}} \neg P.\, send(m)\ to\ (u.id) \\ & \wedge \bigwedge_{m \in M} \neg P.\, rec(m) \\ & \wedge \neg P.stop \end{aligned}$$

The transition function depends on the behaviour of the player. We are now going to discuss the different cases.

Honest behaviour. The transition function of a honest participant is defined in figure 5.7.

The guarded commands (5.1)–(5.2) describe the honest behaviour of P at each non-terminal position $c \in \mathsf{pos}(R_P) \setminus \mathsf{pos}_\mathsf{T}(R_P)$ of the protocol. We can note that when sending out a message, we first verify whether P is able to construct this message at that point of the protocol. Also note that P does not know the originator of the message, unless he can deduce it from its content (in particular, P may not know from which channel the message comes, as logically separate channels, may not be physically separated). When awaiting a message from a participant u, it accepts a valid message from any participant v.

The guarded command (5.3) states that the agent is also able to wait, i.e. to execute no action (*idle*). The possibility to wait models the time-out, which is chosen by the player himself, before he contacts the TTP. Moreover, guarded command (5.4) allows P to stop whenever he is in a terminal position.

We have added fairness constraints in order to avoid that a participant "stops" the protocol by waiting forever. The guarded commands inducing fairness constraints, as presented in definition 4.23, are denoted using the subscript F. A honest participant can only wait for a finite amount of time before continuing the protocol. Also note that the given fairness constraints are restricted according to definition 4.25. On one hand we have that if a fairness constraint is taken, it cannot be enabled any more. This is due to the monotonicity of the propositions representing a given send or receive action, i.e. once they are true they remain true. As for a fairness constraint corresponding to an action a, the proposition $P.a$ must be false in order to be enabled, we have that the constraints can never be enabled after having been taken. On the other hand, when a fairness constraint, related to an action a, is never taken, there exists no choice to reach a state where $P.a$ becomes true, without taking the fairness constraint. Therefore the second restriction is respected too.

TTP behaviour. The guarded commands given in figure 5.8 describing the behaviour of a TTP, are very close to a honestly behaving participant. However, the TTP does not have the choice to help one player or the other. In order for the TTP to be deterministic, we remove the fairness constraints and eliminate the guarded command (5.3), which allows the honest participant to idle.

For each $c = (a_1 \dots a_n) \in \mathsf{pos}(R_P) \setminus \mathsf{pos}_\mathsf{T}(R_P)$, each position $(ca) \in \mathsf{pos}(R_P)$, each identity $u, v \neq p \in C_{Id}$ we have:

- if $a = send(m)\ to\ (u)$:

$$\mathsf{P} \underset{\mathsf{F}}{[\!]} \left(\begin{array}{l} \neg\mathsf{P.stop} \wedge \mathsf{P.c} \wedge \neg\mathsf{P.a} \wedge \\ \mathsf{P.a_1} \wedge \dots \wedge \mathsf{P.a_n} \wedge \\ \mathsf{P.knows(m)} \end{array} \right) \rightarrow \mathsf{P.a}' := \mathsf{true};\ \mathsf{P.c}' := \mathsf{false};\ \mathsf{P.ca}' := \mathsf{true} \quad (5.1)$$

- if $a = rec(m)$:

$$\mathsf{P} \underset{\mathsf{F}}{[\!]} \left(\begin{array}{l} \neg\mathsf{P.stop} \wedge \mathsf{P.c} \wedge \neg\mathsf{P.a} \wedge \\ \mathsf{P.a_1} \wedge \dots \wedge \mathsf{P.a_n} \wedge \\ \mathsf{C_{vp}.fwd(m)} \end{array} \right) \rightarrow \mathsf{P.a}' := \mathsf{true};\ \mathsf{P.c}' := \mathsf{false};\ \mathsf{P.ca}' := \mathsf{true} \quad (5.2)$$

In order to wait, we have:

$$\mathsf{P}\ [\!]\ \mathsf{true} \rightarrow \quad (5.3)$$

For each $c = a_1 \dots a_n \in \mathsf{pos}_\mathsf{T}(R_P)$, in order to stop the protocol, we have:

$$\mathsf{P} \underset{\mathsf{F}}{[\!]} \neg\mathsf{P.stop} \wedge \mathsf{P.a_1} \wedge \dots \wedge \mathsf{P.a_n} \wedge \mathsf{P.c} \rightarrow\ \mathsf{P.stop}' := \mathsf{true} \quad (5.4)$$

Figure 5.7: Honest participant

For each $c = (a_1 \dots a_n) \in \mathsf{pos}(R_P) \setminus \mathsf{pos}_\mathsf{T}(R_P)$, each position $(ca) \in \mathsf{pos}(R_P)$, each identity $u, v \neq p \in C_{Id}$ we have:

- if $a = send(m)\ to\ (u)$:

$$\mathsf{P}\ [\!] \left(\begin{array}{l} \neg\mathsf{P.stop} \wedge \mathsf{P.c} \wedge \neg\mathsf{P.a} \wedge \\ \mathsf{P.a_1} \wedge \dots \wedge \mathsf{P.a_n} \wedge \\ \mathsf{P.knows(m)} \end{array} \right) \rightarrow \mathsf{P.a}' := \mathsf{true};\ \mathsf{P.c}' := \mathsf{false};\ \mathsf{P.ca}' := \mathsf{true} \quad (5.5)$$

- if $a = rec(m)$:

$$\mathsf{P}\ [\!] \left(\begin{array}{l} \neg\mathsf{P.stop} \wedge \mathsf{P.c} \wedge \neg\mathsf{P.a} \wedge \\ \mathsf{P.a_1} \wedge \dots \wedge \mathsf{P.a_n} \wedge \\ \mathsf{C_{vp}.fwd(m)} \end{array} \right) \rightarrow \mathsf{P.a}' := \mathsf{true};\ \mathsf{P.c}' := \mathsf{false};\ \mathsf{P.ca}' := \mathsf{true} \quad (5.6)$$

For each $c = a_1 \dots a_n \in \mathsf{pos}_\mathsf{T}(R_P)$, in order to stop the protocol, we have:

$$\mathsf{P}\ [\!]\ \neg\mathsf{P.stop} \wedge \mathsf{P.a_1} \wedge \dots \wedge \mathsf{P.a_n} \wedge \mathsf{P.c} \rightarrow\ \mathsf{P.stop}' := \mathsf{true} \quad (5.7)$$

Figure 5.8: TTP participant

Weakly dishonest behaviour. A weakly dishonest behaviour of P is modelled by the fact that he is able to send each protocol message that he can build to each participant and record each message that is sent to him. The fact of being able to construct a message is modelled by the predicate $P.knows(\ )$. The capabilities of a weakly dishonest participant are formalised in figure 5.9.

For each $m \in M$ and $u \neq p \in C_{Id}$ we have:

$$\mathsf{P} \parallel \neg\mathsf{P.stop} \wedge \mathsf{P.knows(m)} \wedge \neg\mathsf{P.send(m)\ to\ (u)} \rightarrow\ \mathsf{P.send(m)\ to\ (u)}' := \mathsf{true} \tag{5.8}$$

$$\mathsf{P} \parallel \neg\mathsf{P.stop} \wedge \mathsf{C_{up}.fwd(m)} \wedge \neg\mathsf{P.recv(m)} \rightarrow\ \mathsf{P.rec(m)}' := \mathsf{true} \tag{5.9}$$

In order to wait, we have:

$$\mathsf{P} \parallel \mathsf{true} \rightarrow \tag{5.10}$$

In order to stop the protocol, we have:

$$\mathsf{P} \parallel \neg\mathsf{P.stop} \rightarrow\ \mathsf{P.stop}' := \mathsf{true} \tag{5.11}$$

Figure 5.9: Weakly dishonest participant

Strongly dishonest behaviour. A strongly dishonest behaving participant P is modelled by the fact that he is able to eavesdrop messages on *all* channels. His behaviour is formalised in figure 5.10.

For each $m \in M$ and $u \neq p, v \neq u \in C_{Id}$ we have:

$$\mathsf{P} \parallel \neg\mathsf{P.stop} \wedge \mathsf{P.knows(m)} \wedge \neg\mathsf{P.send(m)\ to\ (u)} \rightarrow\ \mathsf{P.send(m)\ to\ (u)}' := \mathsf{true} \tag{5.12}$$

$$\mathsf{P} \parallel \neg\mathsf{P.stop} \wedge \mathsf{C_{uv}.fwd(m)} \wedge \neg\mathsf{P.rec(m)} \rightarrow\ \mathsf{P.rec(m)}' := \mathsf{true} \tag{5.13}$$

In order to wait, we have:

$$\mathsf{P} \parallel \mathsf{true} \rightarrow \tag{5.14}$$

In order to stop the protocol, we have:

$$\mathsf{P} \parallel \neg\mathsf{P.stop} \rightarrow\ \mathsf{P.stop}' := \mathsf{true} \tag{5.15}$$

Figure 5.10: Strongly dishonest participant

5.1.2.2 Communication channels

For each communication channel $C_{uv} = (u, v, quality)$ we now describe the corresponding player $\phi_{C_{uv}}$. For channel C_{uv} we define the *init* predicate as follows.

$$init_{C_{uv}} = \bigwedge_{m \in M} \neg C_{uv}.fwd(m)$$

Generally, when analysing authentication protocols or protocols aiming secrecy, the communication channels quality is supposed to be very poor, i.e. the communication channels are unreliable and do not offer any service. However, as noted in section 3.5.1, optimistic exchange protocols are impossible if only unreliable channels are available. In order to give us the ability of analysing protocols in a detailed way, we therefore model three levels of quality for communication channels. The transition function of the communication channels depends on the given quality. We now discuss the three possible cases.

Operational channels. An operational communication channel must immediately transmit each message he receives and does not permit any delay. This behaviour is described in figure 5.11.

For each $m \in M$ we have:

$$\mathsf{C_{uv}} \; [\!] \; \mathsf{u.send(m)\ to\ (v)} \wedge \neg \mathsf{C_{uv}.fwd(m)} \rightarrow \; \mathsf{C_{uv}.fwd(m)' := true} \tag{5.16}$$

Figure 5.11: Operational channel

Resilient channel. The difference with an operational channel is that each message can be delayed for an arbitrary, finite amount of time, but must be delivered eventually. Modelling a resilient channel can be achieved using fairness constraints, as shown in figure 5.12. To see that the fairness constraints are of a restricted form, consider a given message m and the fairness constraint induced by the guarded command corresponding to m. The first restriction on the fairness constraints holds, because the fairness constraint is only enabled as long as $C_{uv}.fwd(m)$ is false. For any computation, if there exists a position i such that the fairness constraint is taken, $mathitC_{uv}.fwd(m)$ becomes true in position $i + 1$ and always remains true. Therefore the fairness constraint cannot be enabled anymore. On the other hand, if the fairness constraint is never taken on a computation, there is no choice on the computation where $C_{uv}.fwd(m)$ would have been set to true. However, the set of non-empty choices returned by the fairness constraint are all such that for each state in a given choice, $C_{uv}.fwd(m)$ is true. Therefore, the second restriction also holds.

Unreliable channel. An unreliable channel can delay and even lose messages. To obtain an unreliable channel from a resilient one, we suppress the fairness constraints, as shown in figure 5.13.

For each $m \in M$ we have:

$$\mathsf{C_{uv}} \mathop{[\!]}_{\mathsf{F}} \mathsf{u.send(m)\ to\ (v)} \wedge \neg \mathsf{C_{uv}.fwd(m)} \rightarrow\ \mathsf{C_{uv}.fwd(m)' := true} \tag{5.17}$$

$$\mathsf{C_{uv}} \ [\!]\ \mathsf{true} \rightarrow \tag{5.18}$$

Figure 5.12: Resilient channel

For each $m \in M$ we have:

$$\mathsf{C_{uv}} \ [\!]\ \mathsf{u.send(m)\ to\ (v)} \wedge \neg \mathsf{C_{uv}.fwd(m)} \rightarrow\ \mathsf{C_{uv}.fwd(m)' := true} \tag{5.19}$$

$$\mathsf{C_{uv}} \ [\!]\ \mathsf{true} \rightarrow \tag{5.20}$$

Figure 5.13: Unreliable channel

5.2 Expressing protocol goals formally

In the present section, we show how the requirements that a fair exchange protocol must fulfil, can be *naturally* rephrased as the existence of *strategies* for the participating entities to reach their goal. The logic ATL is used to formalise those requirements. The properties described are general properties that do not directly apply to a specific protocol. As we will see in chapter 6 when analysing protocols, the properties will need to be instantiated for each given protocol. However these instantiations, given the general form, are rather straightforward. When giving the specifications of the properties hereunder, we assume that Alice and Bob are exchanging respectively $\mathsf{item_A}$ and $\mathsf{item_B}$ and that these denote predicates, indicating that Bob, respectively Alice, received the other entity's item.

5.2.1 Fairness

In chapter 3, we have longly discussed informal definitions of the notion of fairness. Informally, we introduced fairness as the property that either both entities receive the desired item or none of them receives the expected item. The items could for instance be a non-repudiation of origin evidence, respectively a non-repudiation of receipt evidence. Note that we do not consider the nature of the evidence in this work. For instance, we do not verify whether a non-repudiation evidence or an affidavit contains all necessary (and no more) informations. This would concern the non-repudiability property, which will be discussed in chapter 8. We can now formulate the fairness requirement in a rather different way, as the existence of strategies. We say that the protocol is fair for Alice if "Bob does not have a strategy to reach a state where Bob has Alice's item and a honest Alice has no strategy to obtain Bob's item". This can be formally expressed by the following ATL formula:

$$\neg\langle\!\langle B, \mathcal{C}om\rangle\!\rangle\Diamond(\mathsf{item_A} \wedge \neg\langle\!\langle A_h\rangle\!\rangle\Diamond\mathsf{item_B}) \tag{5.21}$$

Here, $\mathcal{C}om$ denotes the set of all communication channels, A_h denotes Alice behaving honestly and B denotes Bob behaving dishonestly[3]. The expressive power of the logic ATL is well illustrated by this example. Cooperation and adversarial behaviour are naturally expressed. Note that the TTP is usually modelled in a deterministic way, which means that it cannot *help* or *play against* one of the coalitions. Adding the TTP to one coalition or the other would not change the outcome, as the deterministic modelling of the TTP forces the same behaviour, and we therefore take the convention to omit it in the formulae. If the above fairness formula is not verified, this means that the protocol contains a flaw. On the other hand, if the formula holds, this only means that Alice has a strategy to *defend* herself against a malicious entity, but it does not mean that this strategy is the only strategy that the protocol admits. Note that even if Alice is modelled in a honest way, this usually leaves certain non-deterministic choices, due to the common "underspecification" of the protocols. For instance, it is not always clearly defined, where the protocol can be stopped, what values should be used for the time-outs, etc. So, verifying the formula with honest Alice, only verifies whether Alice can resolve her internal choices, that all belong to the honest behaviour, in order to achieve fairness. To show that each implementation is correct we have to verify stronger formulas. Before discussing these stronger formulas let us consider the following CTL formula, in order to better understand the advantages of using a game logic to formalise this requirement:

$$\neg\exists\Diamond(\mathsf{item}_\mathsf{A} \wedge \neg\exists\Diamond\mathsf{item}_\mathsf{B}) \tag{5.22}$$

This may look as an appropriate CTL candidate for the formalisation of fairness for Alice. Note that we have obtained this formula by replacing the two teams in formula (5.21) by the entire set of players Σ. The formula (5.22) can be rewritten in the equivalent and more readable positive form:

$$\forall\Box(\mathsf{item}_\mathsf{A} \rightarrow \exists\Diamond\mathsf{item}_\mathsf{B}) \tag{5.23}$$

This formula says that "on every state of every run of the protocol, if Bob has received Alice's item, then there should exist a continuation of the protocol on which Alice eventually receives Bob's item". This way, we have lost however the ability of distinguishing between cooperative and adversarial behaviours. Let us show what are some consequences of this, so to say, "loss of precision". For example, it may be the case that the formula (5.23) is verified because, in the course of the protocol, a *resilient* channel helps Alice to obtain her proof by delivering a message within a given bound (which is not generally the case for such a channel). This assumption of cooperative behaviour of the resilient channels is not realistic. In order to be fair the protocol should guarantee to Alice that she obtains her proof even if the resilient channel does not deliver the message within a certain bound. This fact is not ensured by the above given CTL formalisation but is ensured by our ATL formalisation. In fact, in formula (5.21), it is explicitly stated that the communication channels play against Alice. A resilient channel has only the obligation to deliver the message after a finite amount of time and not within a given bound, even if that helps Alice to obtain her proof. This non-cooperative aspect is precisely formalised in ATL. Also formula (5.22) may be verified because there exists some paths, where Bob is honest and allows Alice to obtain her proof. Again, the protocol should not make the hypothesis that

[3]While defining the properties we do not distinguish between weakly and strongly dishonest behaviours.

Bob is honest, and should ensure fairness for Alice even if Bob is trying to cheat her. This adversarial behaviour is formally modelled by our ATL formula and cannot be formalised in CTL. The other way round, we can also have that the given CTL formula is false, even if the ATL formula holds. Such a situation can occur if Alice's specification contains some non-determinism. For instance, if the specification does not contain at which moment Alice can stop the protocol, she may stop it at a "wrong" moment. The CTL formula will then fail, while the ATL formula may hold, if not stopping the protocol at that given moment would have lead to a winning strategy.

We can now show different flavours of the fairness requirement. The weakest one could be expressed as follows.

$$\neg\langle\langle B\rangle\rangle\Diamond(\mathsf{item}_\mathsf{A} \wedge \neg\langle\langle A_h, \mathcal{C}om\rangle\rangle\Diamond\mathsf{item}_\mathsf{B}) \tag{5.24}$$

The above formula expresses a very weak form of fairness. It verifies whether Bob could always break fairness, even if the communication channels are behaving in the most favourable way for Alice. One way of expressing stronger versions of the fairness property is to change the coalitions. Making more players cooperate with the malicious participant, i.e. in this case Bob, results in stronger versions of the property.

$$\neg\langle\langle B, \mathcal{C}om\rangle\rangle\Diamond(\mathsf{item}_\mathsf{A} \wedge \neg\langle\langle A_h\rangle\rangle\Diamond\mathsf{item}_\mathsf{B}) \tag{5.25}$$

Making the communication channels collaborate with the malicious entity results into the same formula as the one presented at the very beginning of the section. However, as noted before, even if this property holds, the protocol may still contain a flaw. Therefore, we can even reinforce our requirements on the protocol using the following ATL formula.

$$\neg\langle\langle B, \mathcal{C}om, A_h\rangle\rangle\Diamond(\mathsf{item}_\mathsf{A} \wedge \neg\langle\langle\emptyset\rangle\rangle\Diamond\mathsf{item}_\mathsf{B}) \tag{5.26}$$

In this formula we ask whether the honest Alice could collaborate with Bob, i.e. resolve all non-deterministic choices in Bob's favour, and still ensure that she will obtain Bob's item, when Bob obtains Alice's item. This formula is equivalent to the following CTL formula, rewritten in a positive form.

$$\forall\Box(\mathsf{item}_\mathsf{A} \rightarrow \forall\Diamond\mathsf{item}_\mathsf{B}) \tag{5.27}$$

Situations where the protocol is not well defined, occur frequently when integrating the use of formal methods as an aid in the protocol designing process. During the first steps while designing the protocol, we usually do not want to deal with all details: the aim of the verification is then to determine whether the given specification contains a winning strategy that leads to a correct protocol. This means that there exists a correct implementation, but it does not yet prove that all implementations are correct. Such a requirement corresponds to the first two versions of the fairness property, i.e. formulae (5.24) and (5.25). For a more precise analysis, we have to set all, or at least most non-deterministic choices. Once this is done, a stronger formula can be checked in order to verify that *all* possible implementations of this further refined implementation are fair. As seen above, one way of doing this, is to let Alice cooperate with Bob and the communication channels. This means that Alice, tries to fool herself, while respecting her restricted behaviour. This results in the formula (5.26), respectively the corresponding CTL formula (5.27), which is obtained by systematically reinforcing the initial ATL formula. It means that at the end of the specification and design level, a last check should be done to assure that all possible remaining non-deterministic choices, e.g. whatever time-out is chosen, the protocol provides fairness. However, we believe that at a specification level, as well as during the design process, the fairness requirement should explicitly express the adversarial as well as the cooperative behaviours. Although such a refinement process is used in general software development, to the best of our knowledge, no such top-down methodology for security protocol design has been proposed. There exist verification methods, such as the spi calculus or the FDR's refinement checking in CSP, that give a more abstract, and hopefully more obviously correct specification, but this specification is used for verifying protocols and not directly during the design process.

Note, that an even alternate way to specify fairness, would be to give a *positive* specification of the property. Above, we expressed fairness by the fact that Bob does *not* have a strategy to fool Alice. Another way to express this, would be to say that "Alice has a strategy such that in each state, when Bob received her item, she has a strategy to receive Bob's item". The different flavours of this requirement can be stated in ATL as follows.

$$\langle\!\langle A_h, \mathcal{C}om \rangle\!\rangle \Box(\mathsf{item_A} \rightarrow \langle\!\langle A_h, \mathcal{C}om \rangle\!\rangle \Diamond \mathsf{item_B}) \tag{5.28}$$

$$\langle\!\langle A_h \rangle\!\rangle \Box(\mathsf{item_A} \rightarrow \langle\!\langle A_h \rangle\!\rangle \Diamond \mathsf{item_B}) \tag{5.29}$$

$$\langle\!\langle \emptyset \rangle\!\rangle \Box(\mathsf{item_A} \rightarrow \langle\!\langle \emptyset \rangle\!\rangle \Diamond \mathsf{item_B}) \tag{5.30}$$

While the last formula is equivalent to the CTL formula (5.27) given before, formula (5.28) and (5.29) are stronger than the previously introduced corresponding versions. We have that formula (5.28) implies formula (5.24) and formula (5.29) implies formula (5.25), whereas the reverse is not true. This is due to the fact that in ATL for each $A \subseteq \Sigma$, $\langle\!\langle A \rangle\!\rangle \psi \rightarrow \neg \langle\!\langle \Sigma \setminus A \rangle\!\rangle \neg \psi$, but the reverse is not true, while in CTL $\exists \psi \equiv \neg \forall \neg \psi$.

To complete the fairness requirement, we express fairness for Bob in a similar way, as fairness for Alice.

$$\neg \langle\!\langle A, \mathcal{C}om \rangle\!\rangle \Diamond(\mathsf{item_B} \wedge \neg \langle\!\langle B_h \rangle\!\rangle \Diamond \mathsf{item_A}) \tag{5.31}$$

We can of course express all the different flavours of fairness for Bob as we have done it for Alice. A complete fairness specification would be expressed by the conjunction of fairness for Alice and fairness for Bob.

5.2.2 Timeliness

In the previous section, we have shown that the expressiveness of ATL allows us to express different flavours of fairness. In this section, we are going to express the timeliness property using the ATL logic. The informal definition of timeliness states that the parties always have the ability to reach, in a finite amount of time, a point in the protocol where they can stop the protocol while preserving fairness.

Timeliness for Alice is formalised by the following formula.

$$\langle\!\langle A_h \rangle\!\rangle \Diamond(\mathsf{stop}_A \wedge (\neg \mathsf{item_B} \rightarrow \neg \langle\!\langle B \rangle\!\rangle \Diamond \mathsf{item_A}) \tag{5.32}$$

The above formula expresses that "Alice has a strategy to finish the protocol and if she does not have Bob's item at that point, Bob will not be able to obtain Alice's item neither". To represent the fact that Alice finishes the protocol, we introduce a special variable stop_A. Once stop_A is set to true, all of Alice's actions are disabled. In this way, we model the fact that Alice stops this protocol session, including all related subprotocols that could be waiting for an additional message (e.g. a recovery or an abort message from the TTP) to arrive. The reader might have noticed that, in the above formula, we did not mention the communication channels, neither the TTP. Changing coalitions results, as for fairness, into different levels of timeliness.

$$\langle\langle A_h, \mathcal{C}om\rangle\rangle\Diamond(\mathsf{stop}_A \wedge (\neg\mathsf{item_B} \rightarrow \neg\langle\langle B\rangle\rangle\Diamond\mathsf{item_A})) \tag{5.33}$$

$$\langle\langle A_h\rangle\rangle\Diamond(\mathsf{stop}_A \wedge (\neg\mathsf{item_B} \rightarrow \neg\langle\langle B, \mathcal{C}om\rangle\rangle\Diamond\mathsf{item_A})) \tag{5.34}$$

$$\langle\langle\emptyset\rangle\rangle\Diamond(\mathsf{stop}_A \wedge (\neg\mathsf{item_B} \rightarrow \neg\langle\langle B, \mathcal{C}om, A_h\rangle\rangle\Diamond\mathsf{item_A})) \tag{5.35}$$

Formulae (5.33)–(5.35) give the different flavours of timeliness from the weakest towards the strongest, where the last formula can be rewritten in CTL as follows.

$$\forall\Diamond(\mathsf{stop}_A \wedge (\neg\mathsf{item_B} \rightarrow \forall\Box\neg\mathsf{item_A})) \tag{5.36}$$

Timeliness for Bob can of course be expressed in a similar way.

We can also use the formalisation of the fairness and the timeliness properties to prove the intuition that an unfair protocol cannot provide timeliness.

Theorem 5.12 An unfair protocol does not provide timeliness, i.e. given any ATS S and state q

$$S, q \models \neg(\forall\Box(\mathsf{item_A} \rightarrow \forall\Diamond\mathsf{item_B}))$$

$$\Rightarrow$$

$$S, q \models \neg(\forall\Diamond(\mathsf{stop}_A \wedge (\neg\mathsf{item_B} \rightarrow \forall\Box\neg\mathsf{item_A})))$$

and

$$S, q \models \neg(\forall\Box(\mathsf{item_B} \rightarrow \forall\Diamond\mathsf{item_A}))$$

$$\Rightarrow$$

$$S, q \models \neg(\forall\Diamond(\mathsf{stop}_B \wedge (\neg\mathsf{item_A} \rightarrow \forall\Box\neg\mathsf{item_B}))).$$

□

Proof:
We start by proving that

$$S, q \models \neg(\forall\Box(\mathsf{item_A} \rightarrow \forall\Diamond\mathsf{item_B}))$$

$$\Rightarrow$$

$$S, q \models \neg(\forall\Diamond(\mathsf{stop}_A \wedge (\neg\mathsf{item_B} \rightarrow \forall\Box\neg\mathsf{item_A}))).$$

By contradiction, suppose that

$$S, q \models \neg(\forall\Box(\mathsf{item_A} \to \forall\Diamond\mathsf{item_B})) \tag{5.37}$$

and

$$S, q \models \forall\Diamond(\mathsf{stop}_A \wedge (\neg\mathsf{item_B} \to \forall\Box\neg\mathsf{item_A})). \tag{5.38}$$

We can rewrite formula 5.37 as follows.

$$\exists\Diamond(\mathsf{item_A} \wedge \exists\Box\neg\mathsf{item_B})$$

Hence there exists a q-computation λ, such that

$$\exists i \geq 0 \cdot \ \lambda[i] \models \mathsf{item_A} \wedge \forall j \geq i \cdot \ \lambda[j] \models \neg\mathsf{item_B}.$$

As $\mathsf{item_A}$ and $\mathsf{item_B}$ are monotonic, i.e. once these predicates are true, they remain true, we have that

$$\exists i \geq 0 \cdot \ \forall j \geq i \cdot \ \lambda[j] \models \mathsf{item_A}$$

and

$$\forall i \geq 0 \cdot \ \lambda[i] \models \neg\mathsf{item_B}.$$

By formula 5.38, we have for every q-computation λ that

$$\exists i \geq 0 \cdot \ \forall j \geq i \cdot \ \lambda[j] \models \neg\mathsf{item_A}$$

or

$$\exists i \geq 0 \cdot \ \lambda[i] \models \mathsf{item_B}$$

which contradicts the previous statement and concludes the first part of the proof.

The second part of the proof is symmetric. □

The above theorem has been stated for the strongest versions of fairness and timeliness, i.e. the CTL formulae. Moreover, if we have that one of the ATL versions of fairness fails, we know that the CTL version of timeliness fails too. However, we do not know whether the ATL version of timeliness does hold or not.

5.2.3 Viability

We are also interested in first verifying a basic (weaker) notion of fairness: "Alice and Bob cooperate to achieve fairness". This would be translated by the following ATL formula.

$$\langle\langle A, B\rangle\rangle\Diamond((\mathsf{item_A} \wedge \mathsf{item_B}) \vee (\neg\mathsf{item_A} \wedge \neg\mathsf{item_B})) \tag{5.39}$$

Viability can be seen as a variation of the property above. Viability states that "Alice and Bob can cooperate, such that the exchange succeeds". This could be expressed as follows.

$$\langle\langle A_h, B_h\rangle\rangle\Diamond(\mathsf{item_A} \wedge \mathsf{item_B}) \tag{5.40}$$

Note that, in formula (5.40), Alice and Bob have to achieve a successful exchange, whatever the behaviour of the communication channels (with respect to their offered quality) is. When expressing viability in CTL, and assuming that the channels also cooperate with Alice and Bob, in contrary to fairness and timeliness, we obtain a weaker form of viability, described as follows.

$$\exists\Diamond(\mathsf{item_A} \wedge \mathsf{item_B}) \tag{5.41}$$

Of course we could state a stronger CTL formula as follows.

$$\forall\Diamond(\mathsf{item_A} \wedge \mathsf{item_B}) \tag{5.42}$$

However, requesting that all paths end with a successful outcome is too restrictive. Optimistic fair exchange protocols do not succeed to enforce a successful exchange in all of their executions. Therefore, this requirement would fail for any real and intuitively correct. protocol.

5.2.4 Balance and abuse-freeness in digital contract signing

Now we will discuss balance and abuse-freeness. These properties are specific to contract signing. Balance is very easy to express in terms of strategies. Informally we say that *"a protocol is balanced for Bob, if Alice does not have at any state of the protocol both the power to complete the contract and the power to abort it"*. Note that the term *abort* is here a bit ambiguous as in most optimistic protocols an abort protocol can always be run after a successful run, where the TTP did not intervene. So by having the power of aborting, we mean that the protocol must be aborted and the other party does not already have and cannot obtain a signed contract. The ATL formulation of this property is as follows.

$$\neg(\langle\langle A\rangle\rangle\Diamond(\mathsf{contract_B}) \wedge \langle\langle A\rangle\rangle\Diamond(\mathsf{aborted} \wedge \neg\langle\langle B_h\rangle\rangle\Diamond\mathsf{contract_A})) \tag{5.43}$$

As the property is specific to contract signing, we replace the notions of $\mathsf{item_A}$ and $\mathsf{item_B}$ by $\mathsf{contract_A}$ and $\mathsf{contract_B}$, that respectively denote Alice's and Bob's signature on the contract. Also note that the two first strategy quantifiers are not nested, but at the same level. Balance is the property that has been studied by Shmatikov et al. in [147, 148] and Chadha et al. in [43], as they did not give a complete definition of abuse-freeness. However it is possible to completely model abuse-freeness. In the original paper by Garay et al. [67], the authors gave the following definition. *"An optimistic contract signing protocol is abuse-free if it is impossible for a single player at any point in the protocol to be able to prove to an outside party that he has the power to terminate (abort) or successfully complete the contract."* The difficult part is to model the fact that an entity can prove to an external party having a given power on the protocol outcome. Therefore we have to discuss what it means to prove having the power to abort and to complete a protocol run to an external party, that we call Charlie. If an entity has a strategy, then he can prove it to Charlie, by showing this strategy. However, this is not enough in the context of contract signing. We must also prove that a protocol run has indeed been started between Alice and Bob. To prove this fact, we have to show a universally verifiable signature to Charlie, which contains information related to the contract. When Alice wants to prove to Charlie that she engaged a protocol run with Bob, she has to show a message signed by Bob, claiming that the protocol run is started. As such a signature cannot be forged by Alice, Charlie is convinced that the protocol is started. Here PCS, defined previously in definition 3.9, play an essential role: Charlie cannot distinguish, whether a given PCS has been created by Bob or by Alice herself and so a PCS cannot be used to prove anything to Charlie. In our model we introduce special propositions, $\mathsf{prove2C_A}$ and $\mathsf{prove2C_B}$, that are used as a signal to notify that a given entity has a signature, that can be used to convince Charlie that the protocol is started between Alice and Bob. This proposition must be instantiated for each particular protocol. We do not formally prove that these propositions are sufficient to convince an outsider. However, when considering the cryptographic primitives as black boxes, the correctness of the propositions is intuitive. In the ASW protocol, Alice can set $\mathsf{prove2C_A}$ to true, once she knows the second or the fourth message of the main protocol. Bob has to know the first or the third message to set his proposition to true. In the GJM protocol, which we are going to analyse in section 6.4, where we use PCS for the first two messages, Alice must receive the fourth message and Bob the third message before setting $\mathsf{prove2C_A}$, respectively $\mathsf{prove2C_B}$ to true. So, we can model abuse-freeness for Bob by the following ATL formula.

$$\neg\langle\langle A\rangle\rangle\Diamond(\mathsf{prove2C_A} \wedge \langle\langle A\rangle\rangle\Diamond\mathsf{contract_B} \wedge \langle\langle A\rangle\rangle\Diamond(\mathsf{aborted} \wedge \neg\langle\langle B_h\rangle\rangle\Diamond\ \mathsf{contract_A})) \quad (5.44)$$

Above, we only gave the definitions of balance and abuse-freeness for Bob. The corresponding properties for Alice are written in a similar way. The complete specification of balance and abuse-freeness is the conjunction of those properties for Alice and Bob.

To the best of our knowledge, we gave the first complete modelling of abuse-freeness for verification purposes. Modelling abuse-freeness and balance in ATL also allows us to formally show the relationship between both properties. Balance has been introduced as a variant of abuse-freeness, but the relationship with abuse-freeness has not clearly been stated. Given the formal definition in ATL of both balance and abuse-freeness allows us to investigate the exact relationship between those properties.

Theorem 5.13 Balance implies abuse-freeness, i.e. for any ATS S and state q

$$S, q \models \neg(\langle\langle A\rangle\rangle\Diamond(\mathsf{contract_B}) \wedge \langle\langle \mathsf{A}\rangle\rangle\Diamond(\mathsf{aborted} \wedge \neg\langle\langle \mathsf{B_h}\rangle\rangle\Diamond\mathsf{contract_A}))$$

$$\Rightarrow$$

$$S, q \models \neg\langle\langle A\rangle\rangle\Diamond(\mathsf{prove2C_A} \wedge \langle\langle A\rangle\rangle\Diamond\mathsf{contract_B} \wedge \langle\langle A\rangle\rangle\Diamond(\mathsf{aborted} \wedge \neg\langle\langle B_h\rangle\rangle\Diamond\ \mathsf{contract_A}))$$

and

$$S, q \models \neg(\langle\langle B\rangle\rangle\Diamond(\mathsf{contract_A}) \wedge \langle\langle \mathsf{B}\rangle\rangle\Diamond(\mathsf{aborted} \wedge \neg\langle\langle \mathsf{A_h}\rangle\rangle\Diamond\mathsf{contract_B}))$$

$$\Rightarrow$$

$$S, q \models \neg\langle\langle B\rangle\rangle\Diamond(\mathsf{prove2C_B} \wedge \langle\langle B\rangle\rangle\Diamond\mathsf{contract_A} \wedge \langle\langle B\rangle\rangle\Diamond(\mathsf{aborted} \wedge \neg\langle\langle A_h\rangle\rangle\Diamond\ \mathsf{contract_B})).$$

□

Proof:
Looking at the logical specification of both the balance property and the abuse-freeness

property, we can prove that balance implies abuse-freeness. To prove this statement, we consider its contraposition, i.e. if a protocol is not abuse-free then the protocol is not balanced.

Using rules (R_1)–(R_3), given in lemma 4.16, we can easily prove that if abuse-freeness is violated for Bob, then balance is violated for Bob too.

$$S,q \models \langle\!\langle A\rangle\!\rangle\Diamond(\mathsf{prove2C_A} \wedge \langle\!\langle A\rangle\!\rangle\Diamond\mathsf{contract_B} \wedge \langle\!\langle A\rangle\!\rangle\Diamond(\mathsf{aborted} \wedge \neg\langle\!\langle B\rangle\!\rangle\Diamond\ \mathsf{contract_A}))$$
$$\Rightarrow$$
$$S,q \models \langle\!\langle A\rangle\!\rangle\Diamond\mathsf{contract_B} \wedge \langle\!\langle A\rangle\!\rangle\Diamond(\mathsf{aborted} \wedge \neg\langle\!\langle B\rangle\!\rangle\Diamond\ \mathsf{contract_A})$$

1. $S,q \models \langle\!\langle A\rangle\!\rangle\Diamond(\mathsf{prove2C_A} \wedge \langle\!\langle A\rangle\!\rangle\Diamond\mathsf{contract_B} \wedge \langle\!\langle A\rangle\!\rangle\Diamond(\mathsf{aborted} \wedge \neg\langle\!\langle B\rangle\!\rangle\Diamond\ \mathsf{contract_A}))$
2. $S,q \models \langle\!\langle A\rangle\!\rangle\Diamond(\langle\!\langle A\rangle\!\rangle\Diamond\mathsf{contract_B} \wedge \langle\!\langle A\rangle\!\rangle\Diamond(\mathsf{aborted} \wedge \neg\langle\!\langle B\rangle\!\rangle\Diamond\ \mathsf{contract_A}))$ by 1, (R_1), since
$$(\mathsf{prove2C_A} \wedge \langle\!\langle A\rangle\!\rangle\Diamond\mathsf{contract_B} \wedge \langle\!\langle A\rangle\!\rangle\Diamond(\mathsf{aborted} \wedge \neg\langle\!\langle B\rangle\!\rangle\Diamond\ \mathsf{contract_A}))$$
$$\rightarrow (\langle\!\langle A\rangle\!\rangle\Diamond\mathsf{contract_B} \wedge \langle\!\langle A\rangle\!\rangle\Diamond(\mathsf{aborted} \wedge \neg\langle\!\langle B\rangle\!\rangle\Diamond\ \mathsf{contract_A}))$$
3. $S,q \models \langle\!\langle A\rangle\!\rangle\Diamond\langle\!\langle A\rangle\!\rangle\Diamond\mathsf{contract_B} \wedge \langle\!\langle A\rangle\!\rangle\Diamond\langle\!\langle A\rangle\!\rangle\Diamond(\mathsf{aborted} \wedge \neg\langle\!\langle B\rangle\!\rangle\Diamond\ \mathsf{contract_A})$ by 2, (R_2)
4. $S,q \models \langle\!\langle A\rangle\!\rangle\Diamond\mathsf{contract_B} \wedge \langle\!\langle A\rangle\!\rangle\Diamond(\mathsf{aborted} \wedge \neg\langle\!\langle B\rangle\!\rangle\Diamond\ \mathsf{contract_A})$ by 3, (R_3)

The proof that violating abuse-freeness for Alice, implies violating balance for Alice is similar. These two facts can be combined to end our proof. □

This also means that to prove that a protocol is abuse-free, it is sufficient to prove that the protocol is balanced. This corresponds to the intuition that when a protocol is for instance balanced for Bob, i.e. Alice does not have the power to choose between aborting or successfully finishing the protocol, she cannot prove having this power to Charlie (since she does actually not have it).

When looking at the ATL formulae we gave above to define those properties, the careful reader may notice that we did not include the communication channels to collaborate either with Alice or Bob. As for fairness, there may exist different flavours of those properties. Note that theorem 5.13 holds for each of the different specifications for balance and the corresponding specification for abuse-freeness. We are going to discuss the meaning of some of these different versions.

As for fairness and timeliness, the strongest version of both balance and abuse-freeness would be expressed by the following formulae.

$$\neg(\exists\Diamond(\mathsf{contract_B}) \wedge \exists\Diamond(\mathsf{aborted} \wedge \neg\forall\Diamond\mathsf{contract_A})) \tag{5.45}$$

$$\neg\exists\Diamond(\mathsf{prove2C_A} \wedge \exists\Diamond\mathsf{contract_B} \wedge \exists\Diamond(\mathsf{aborted} \wedge \neg\forall\Diamond\ \mathsf{contract_A})) \tag{5.46}$$

In terms of logic, these formulae are of course stronger than the versions in ATL, that are not expressible in CTL. However, these formulae do not correspond to the intuitive definition of abuse-freeness and are too strong. They state for instance that balance is broken if there *exists* a path such that the protocol can succeed in signing a contract and there also *exists* a path in which the protocol is aborted. However, such executions do *exist* in all protocols that make sense in the context of optimistic contract signing protocols. Properties, such as balance and abuse-freeness express that the malicious party has the power to achieve successful contract signing as well as abort. Having the power to determine the outcome implicitly means having the power, against all (honest) behaviours of the other party. This is exactly what is expressed when considering the two versions of the properties that can be stated in ATL and *cannot* be stated in CTL. A first weaker ATL version of the two properties is the following.

$$\neg(\langle\langle A\rangle\rangle\Diamond(\mathsf{contract_B}) \wedge \langle\langle A\rangle\rangle\Diamond(\mathsf{aborted} \wedge \neg\langle\langle B_h, \mathcal{C}om\rangle\rangle\Diamond\mathsf{contract_A})) \tag{5.47}$$

$$\neg\langle\langle A\rangle\rangle\Diamond(\mathsf{prove2C_A} \wedge \langle\langle A\rangle\rangle\Diamond\mathsf{contract_B} \wedge \langle\langle A\rangle\rangle\Diamond(\mathsf{aborted} \wedge \neg\langle\langle B_h, \mathcal{C}om\rangle\rangle\Diamond\ \mathsf{contract_A})) \tag{5.48}$$

A slightly stronger version of balance and abuse-freeness is achieved when changing the coalitions.

$$\neg(\langle\langle A, \mathcal{C}om\rangle\rangle\Diamond(\mathsf{contract_B}) \wedge \langle\langle A, \mathcal{C}om\rangle\rangle\Diamond(\mathsf{aborted} \wedge \neg\langle\langle B_h\rangle\rangle\Diamond\mathsf{contract_A})) \tag{5.49}$$

$$\neg\langle\langle A, \mathcal{C}om\rangle\rangle\Diamond(\mathsf{prove2C_A} \wedge \langle\langle A\rangle\rangle\Diamond\mathsf{contract_B} \wedge \langle\langle A, \mathcal{C}om\rangle\rangle\Diamond(\mathsf{aborted} \wedge \neg\langle\langle B_h\rangle\rangle\Diamond\ \mathsf{contract_A})) \tag{5.50}$$

The difference between formulae (5.47) and (5.49), as well as (5.48) and (5.50), is the fact that the communication channels collaborate in the weaker version with the honest party and in the stronger one with the malicious party. Usually, it is supposed that the malicious party controls the communication channels. However, on resilient channels, an attacker can only introduce finite delays, not being able to remove a message or delay it for an infinite amount of time. As it seems rather strange that an attacker can delay messages without removing them, we believe that the collaboration of a dishonest participant with the communication channel models the worst-case behaviour, rather than a complete control. The distinction between worst-case behaviour and controlling the communication channel is of no interest when verifying fairness and timeliness, as these properties should hold in all circumstances, i.e. for any behaviour of the communication channels (although analysing the weaker properties may be interesting). When studying balance and abuse-freeness, the dishonest party must prove to Charlie that he has a strategy to either abort or successfully finish the protocol run. In order to prove this, one may argue that he should have this power in all circumstances, no matter what delays are introduced on the communication channels. Otherwise, he can only prove that he may be able to choose between aborting or successfully signing the contract, depending on the behaviour of the communication channels. Resilient channels are a source of non-determinism, and so two cases are possible. Either they are in coalition with the malicious party and they help him to cheat the honest party, or they are in coalition with the honest party and help this party to avoid being cheated. For fairness and timeliness, it is clear that those properties must hold, even when the communication channels react in favour of the malicious party. For balance and abuse-freeness this is not so clear. Can a dishonest party *prove* its power to Charlie, if this power depends on the behaviour of the communication channels ? Hence, when verifying these properties, it is interesting to consider all the possible cases: where the communication channels collaborate or not with the malicious party. During the analysis of two contract signing protocols in chapter 6, we shall see that analysing different flavours of abuse-freeness gives us new insights into this property. Abuse-freeness, in all of its flavours, cannot be expressed in CTL, which emphasises once more the need for a more powerful logic such as ATL.

5.2.5 Author-based selective receipt in certified e-mail

We now discuss a property, specific to certified e-mail, that we introduced in [85] in collaboration with Olivier Markowitch. Informally, we say that a certified e-mail protocol is secure against *author-based selective receipt* if "it is impossible for a recipient to prevent delivery of a receipt to the sender, once the author's identity is known to the recipient". The tricky part of the specification is the part dealing with the knowledge of the author's identity. If we suppose that a network address, e.g. an IP address, is sufficient to identify a person the protocol needs additional mechanisms in order to provide anonymity. Our specification is not going to consider anonymity at all. We suppose that these mechanisms are ad-hoc and can be analysed using specific methods, e.g. Shmatikov [145]. In our model, the only leaking of identity that we consider is the content of a message: if the message contains the author's identity or is signed by the author, the identity is known by the recipient once the message is delivered. Therefore, we suppose that for each protocol, we can define a predicate $\mathsf{identity}$ that is initialised to false and becomes true, once the author's identity is revealed to the receiver. Using this predicate, we are now ready to write the ATL specification of *author based selective receipt.*

$$\neg\langle\langle B\rangle\rangle\Diamond(\mathsf{identity} \wedge \langle\langle B\rangle\rangle\Diamond(\mathsf{aborted} \wedge \neg\langle\langle A_h\rangle\rangle\Diamond\ \mathsf{item_B})) \tag{5.51}$$

In the above formula $\mathsf{item_B}$ denotes the receipt Alice should receive for the sent message from Bob. In terms of strategies, the property claims that Bob does not have a strategy to reach a state where Alice's identity has been revealed and Bob has afterwards a strategy to abort the protocol. By abort we mean, that Bob can get the abort evidence from the TTP and Alice is not able to get Bob's receipt.

As for fairness, we can write the different flavours of this property.

$$\neg\langle\langle B\rangle\rangle\Diamond(\mathsf{identity} \wedge \langle\langle B\rangle\rangle\Diamond(\mathsf{aborted} \wedge \neg\langle\langle A_h, \mathcal{C}om\rangle\rangle\Diamond\ \mathsf{item_B})) \tag{5.52}$$

$$\neg\langle\langle B, \mathcal{C}om\rangle\rangle\Diamond(\mathsf{identity} \wedge \langle\langle B, \mathcal{C}om\rangle\rangle\Diamond(\mathsf{aborted} \wedge \neg\langle\langle A_h\rangle\rangle\Diamond\ \mathsf{item_B})) \tag{5.53}$$

$$\neg\langle\langle B, \mathcal{C}om, A_h\rangle\rangle\Diamond(\mathsf{identity} \wedge \langle\langle B, \mathcal{C}om, A_h\rangle\rangle\Diamond(\mathsf{aborted} \wedge \neg\langle\langle \emptyset\rangle\rangle\Diamond\ \mathsf{item_B})) \tag{5.54}$$

Formulae (5.52) to (5.54) denote these different versions, from the weakest towards the strongest requirement. As for fairness, the strongest version is equivalent to the following CTL formula.

$$\forall\Box(\mathsf{identity} \rightarrow \forall\Box(\mathsf{aborted} \rightarrow \forall\Diamond\ \mathsf{item_B})) \tag{5.55}$$

Note that this property, in contrary to all other previously presented properties only applies to Alice and not to Bob (the requirement is asymmetrical). It is also possible to relate this property to abuse-freeness, when stating abuse-freeness in a more general context, in terms of $\mathsf{item_A}$ and $\mathsf{item_B}$, rather than $\mathsf{contract_A}$ and $\mathsf{contract_B}$.

Theorem 5.14 Author-based selective receipt implies abuse-freeness for Alice, i.e. for any ATS S and state q

$$S, q \models \neg\langle\langle B\rangle\rangle\Diamond(\mathsf{identity} \wedge \langle\langle B\rangle\rangle\Diamond(\mathsf{aborted} \wedge \neg\langle\langle A_h\rangle\rangle\Diamond\ \mathsf{item_B}))$$

$$\Rightarrow$$
$$S, q \models \neg\langle\langle B\rangle\rangle\Diamond(\mathsf{prove2C_B} \wedge \langle\langle B\rangle\rangle\Diamond\mathsf{contract_A} \wedge \langle\langle B\rangle\rangle\Diamond(\mathsf{aborted} \wedge \neg\langle\langle A_h\rangle\rangle\Diamond\ \mathsf{contract_B})).$$

□

Proof:
We are going to prove the contraposition of the theorem, i.e. we have to prove that

$$S, q \models \langle\langle B\rangle\rangle\Diamond(\mathsf{prove2C_B} \wedge \langle\langle B\rangle\rangle\Diamond\mathsf{contract_A} \wedge \langle\langle B\rangle\rangle\Diamond(\mathsf{aborted} \wedge \neg\langle\langle A_h\rangle\rangle\Diamond\ \mathsf{item_B})$$
$$\Rightarrow$$
$$S, q \models \langle\langle B\rangle\rangle\Diamond(\mathsf{identity} \wedge \langle\langle B\rangle\rangle\Diamond(\mathsf{aborted} \wedge \neg\langle\langle A_h\rangle\rangle\Diamond\ \mathsf{item_B})$$

First we note that $\mathsf{prove2C_B} \Rightarrow \mathsf{identity}$. If Bob can prove to Charlie that he has started a protocol with Alice, Bob obviously knows Alice's identity in order to prove it to Charlie. Now the proof is trivial by applying the (R_1) proof rule, given in lemma 4.16. □

Although, abuse-freeness has no direct meaning in certified e-mail, and neither has author-based selective receipt in contract signing, this means that one can modify the certified e-mail protocol to serve as a contract signing protocol, that provides abuse-freeness for Alice without any non standard cryptographic primitive, such as private contract signatures.

5.3 Automation

We have developed in collaboration with Alexandre Boisseau a prototype of a tool, which nearly completely automates the protocol analysis. The tool takes as an input the description of a protocol instance, as described in section 5.1.1 and a set of ATL formulae, which represent the properties the protocol should verify, as discussed in section 5.2. On one hand, the tool performs a translation of the protocol specification into an ATS, following the semantics defined in section 5.1.2. Note that the tool automates all the computations with regard to the knowledge of a participant, i.e. the *knows* predicate. Therefore, we can automate the decision when a participant can execute a given action, e.g. launch a subprotocol. On the other hand, the tool generates for each ATL formula a SLANG script implementing the model-checking of the given formula. The tool performing this translation has been implemented by Alexandre Boisseau in the OCAML language, while the SLANG scripts are based on our model-checking primitives, which we briefly discussed in section 4.6. Finally, we can use the finite state tool MOCHA to perform the model-checking on the constructed ATS of the scripts corresponding to the given ATL formulae.

In order to have a model closer to the Dolev-Yao model and to avoid some of our additional simplifications, we have adapted the abstract interpretation techniques presented in [79] to the context of fair exchange protocols. We do not report on these works, as they are still in progress. Preliminary results can be found in Alexandre Boisseau's thesis [27].

CHAPTER 6

Analysis of two-party fair exchange protocols

In this chapter we demonstrate how the methodology and tool presented in the previous chapter can be used to analyse several two-party protocols. Most of this work was first performed by manually encoding the protocols using guarded commands and writing the model-checking scripts. These preliminary results were carried out in collaboration with Jean-François Raskin and published in [88, 89, 90]. Generating the Mocha specification and writing the model-checking scripts has now been automated by the prototype of the tool, presented in the previous chapter. Presently, the guarded command processes generated by the tool produce models that contain some redundancy, which simplified the generation, but slow down significantly the model-checking process. Optimising the generated models should not be difficult, but requires significant changes to the tool. This work is left for a future version of the tool.

In this chapter we first analyse the Asokan-Shoup-Waidner certified e-mail protocol, which served as our running example in the previous chapter. We highlight two known attacks against fairness and show one additional weakness, which may also compromise fairness when the hypothesis on the confidentiality of the communication channels is not respected. The second protocol that we analyse is the Zhou-Gollmann non-repudiation protocol, for which we show that the given hypotheses on the communication channels are not sufficient to enforce fairness. We go on analysing the Kremer-Markowitch non-repudiation protocol and present a variant of this protocol, which is fair, but does not respect timeliness. This example should emphasise the importance of the timeliness property. Finally, we analyse two contract signing protocols, the Asokan-Shoup-Waidner and Garay-Jakobsson-MacKenzie contract signing protocols, and give a detailed study of the abuse-freeness property on these two protocols. During the analysis of the different protocols, we often discuss dishonest participants without making precise whether they are weakly or strongly dishonest. If no precision is given, this means that the results are identical for weakly and strongly dishonest participants.

6.1 Asokan-Shoup-Waidner certified e-mail protocol

The ASW certified e-mail protocol has already been described in the previous chapter, where it served as our running example. Therefore we directly present our analysis of the protocol here.

6.1.1 Analysis

Remember that the protocol aims at exchanging a message and its corresponding non-repudiation of origin evidence against the corresponding non-repudiation of receipt evidence. The non-repudiation of origin evidence NRO is defined as

$$B.knows(m_1) \wedge B.knows(N_A)$$

and the non-repudiation of receipt evidence NRR as

$$A.knows(m_2) \wedge (A.knows(N_B) \vee A.knows(\mathit{affidavit\_token}).$$

6.1.1.1 Viability

Viability is generally the first property one should test, as it also provides a kind of "sanity check" regarding the specification which makes sure that it is possible to realize a successful exchange. Therefore we check the following formula.

$$\langle\langle A_h, B_h\rangle\rangle\Diamond(\mathsf{NRO} \wedge \mathsf{NRR})$$

Note that this formula opposes Alice and Bob to the communication channels. Indeed, changing the quality of the channels alters the result when verifying the formula. In a first time, one could suppose that the quality of the channels between Alice and Bob is unreliable, while the quality of the channels to and from the TTP is resilient. In that case, viability is not provided, as the first message from Alice to Bob could be lost by the channels. If we change the channels, such that they are all resilient, viability is provided. We can also verify the weaker CTL formula.

$$\exists\Diamond(\mathsf{NRO} \wedge \mathsf{NRR})$$

This formula is verified whatever channel quality is proposed.

6.1.1.2 Fairness

When verifying the fairness property, we can show several attacks. The first two attacks were found by Zhou et al. [175] without the use of formal methods.

The first problem compromises fairness for Alice. We can show that the following formula, which represents the negation of fairness for Alice, holds.

$$\langle\langle B, \mathcal{C}om\rangle\rangle\Diamond(\mathsf{NRO} \wedge \neg\langle\langle A_h\rangle\rangle\Diamond\mathsf{NRR})$$

Intuitively, we ask whether it is possible for Bob to reach a state where he obtained the expected non-repudiation evidence and it is not possible for Alice any more to obtain

Bob's non-repudiation of receipt evidence. The attack is due to the fact that Bob can execute his recovery protocol immediately after having received the first message. This has not been foreseen by the protocol: at this moment, all Alice can do is launch an abort protocol. The TTP replies to Alice by inviting her to launch her recovery protocol. Unfortunately, at that point in the protocol, Alice has not enough information to launch her recovery protocol as requested by the TTP and she cannot receive the non-repudiation of receipt evidence for her message. The fact that our model requires that the predicate $A.knows(r_{a1})$ is true before we allow Alice to send the recovery message (cf the guarded command (5.1) defining the semantics of a honest player on page 102), makes it possible to capture this kind of problems with our tool. Due to this modelling, the guarded command representing the sending of Alice's recovery request, is not enabled and makes it impossible for Alice to recover.

The second problem compromises fairness for Bob. This attack is shown by the fact that the following formula holds.

$$\langle\langle A, \mathcal{C}om \rangle\rangle \Diamond (\mathsf{NRR} \wedge \neg \langle\langle B_h \rangle\rangle \Diamond \mathsf{NRR})$$

In the second attack, compromising fairness for Bob, Alice launches a recovery protocol and afterwards the abort protocol. The problem is that, in the abort protocol, the TTP only checks whether *Bob* did already recover or not. The TTP does not check whether *Alice* did already recover. Hence, a recovery protocol and an abort protocol are not mutually exclusive if both are executed by Alice in the given order. It was not foreseen by the authors of the protocol that Alice could launch a recovery protocol followed by an abort protocol. Executing the protocols in this order makes it impossible for Bob to launch his recovery protocol and, hence, breaks fairness. Again, this attack is easily detected by our verification method: the guards of the commands launching the abort protocol and the recovery protocol are both enabled. Therefore Alice can choose which command to take, determine the order of execution of the protocols and cheat Bob.

There exists also another scenario found by our tool which might compromise fairness for Bob. However, this attack only applies if Alice can eavesdrop on the channel between Bob and the TTP. This is in contradiction with the hypotheses made by Asokan et al. [14], who assume that channels are confidential. Nevertheless, the scenario is worth mentioning as it highlights the need of the assumption. In our model, giving Alice the possibility of spying other channels is reflected by declaring Alice to be *strongly* dishonest. In that case consider the following scenario. Alice sends the first message, aborts the protocol and stops sending out messages. The only thing Bob can do is launch a recovery request, which contains N_B. As Alice can spy the channel, Alice will learn the value of N_B and hence know NRR. The TTP replies to Bob with an abort token and Bob has no possibility for getting NRO. A similar attack was already discovered by Shmatikov and Mitchell [147] on the Garay-Jakobsson-MacKenzie contract signing protocol. It was unknown that a similar attack could also applies to the ASW certified e-mail protocol if we relax the hypotheses made on the channels. However, we do not claim this to be an attack as the authors explicitly mention that they assume confidential channels.

6.1.1.3 Timeliness

As fairness is not respected, we do not need to study the timeliness property. By theorem 5.12 we know that an unfair protocol cannot provide timeliness.

6.2 Zhou-Gollmann optimistic non-repudiation protocol

Shortly after the introduction of the optimistic approach for fair exchange protocols by Asokan, Schunter and Waidner [13], Zhou and Gollmann [178] presented an optimistic non-repudiation protocol. While Asokan et al.'s protocol was designed for synchronous channels, Zhou and Gollmann's protocol aims to operate on asynchronous channels. However, we will show that the assumption of having resilient channels rather than operational channels may compromise fairness.

6.2.1 Protocol description

We first give an informal description of the protocol using the following notation:

- m: the message sent from A to B
- k: the session key A uses to cipher m
- $c = E_k(m)$: the cipher of m under the key k
- $\ell = h(m, k)$: a label that, in conjunction with the entities (A,B), identifies a protocol run
- f: a flag indicating the purpose of a message
- t: the time-out chosen by A

The protocol generates the following evidences:

- $\mathsf{EOO} = S_A(f_{\mathsf{EOO}}, B, \ell, t, c)$: the evidence of origin of c
- $\mathsf{EOR} = S_B(f_{\mathsf{EOR}}, A, \ell, t, c)$: the evidence of receipt of c
- $\mathsf{EOO}_k = S_A(f_{\mathsf{EOO}_k}, B, \ell, t, k)$: the evidence of origin of k
- $\mathsf{EOR}_k = S_B(f_{\mathsf{EOR}_k}, A, \ell, t, k)$: the evidence of receipt of k
- $\mathsf{Sub}_k = S_A(f_{\mathsf{Sub}_k}, B, \ell, k)$: the evidence of submission of k
- $\mathsf{Con}_k = S_{\mathsf{TTP}}(f_{\mathsf{Con}_k}, A, B, \ell, t, k)$: the evidence of confirmation of k, issued by the TTP

The protocol is divided into two subprotocols: a main protocol and a recovery protocol. As the protocol is based on the optimistic approach, the trusted third party (TTP) does only intervene in the recovery protocol. We start by describing the main protocol, depicted in protocol 6.1.

Protocol 6.1 ZG non-repudiation protocol—Main

1. A → B: $f_{\mathsf{EOO}}, B, \ell, t, c, \mathsf{EOO}$
2. B → A: $f_{\mathsf{EOR}}, A, \ell, \mathsf{EOR}$
3. A → B: $f_{\mathsf{EOO}_k}, B, \ell, k, \mathsf{EOO}_k$

   if A times out then recovery
4. B → A: $f_{\mathsf{EOR}_k}, A, \ell, \mathsf{EOR}_k$

Main protocol. First Alice sends the digitally signed cipher $c = E_k(m)$ to Bob. In the second message Bob responds with the evidence of receipt for this cipher (EOR). If Alice does not receive the second transmission she stops the protocol, otherwise she sends the signed decryption key k to Bob. Bob answers by sending the receipt EOR_k for the key. The label ℓ, present in each transmission, identifies the protocol run. The time-out t specified in message 1 is used in the recovery protocol. Alice may initiate the recovery protocol if Bob does not send the receipt for the key.

Recovery protocol. The steps of the recovery protocol are described in protocol 6.2.

Protocol 6.2 ZG non-repudiation protocol—Recovery

1. A → TTP: $f_{\mathsf{Sub}_k}, B, \ell, t, k, \mathsf{Sub}_k$
2. B ↔ TTP: $f_{\mathsf{Con}_k}, A, B, \ell, t, k, \mathsf{Con}_k$
3. A ↔ TTP: $f_{\mathsf{Con}_k}, A, B, \ell, t, k, \mathsf{Con}_k$

Alice sends the signed key, together with the deadline t to the TTP. If the key arrives after t, the TTP does not accept the recovery. Otherwise the TTP publishes the key together with a confirmation Con_k for the key in a read-only accessible directory, where both Alice and Bob can fetch the key as well as Con_k. Con_k serves to Bob as the evidence of origin of the key, and to Alice as the evidence of receipt of the key as it is accessible to Bob. The deadline t is necessary for Bob to know the moment when either the key is published or will not be published anymore. In this protocol, the non-repudiation of origin evidence NRO is composed of EOO and EOO_k or of EOO and Con_k. The non-repudiation of receipt evidence NRR is composed of EOR and EOR_k or of EOR and Con_k.

Dispute Resolution. The dispute resolution defines the protocol Alice, respectively Bob, has to execute with the judge, whenever Bob, respectively Alice, denies the reception, respectively the origin, of a given message m. This protocol is not directly part of the exchange protocol. Nevertheless, we present it, as it defines the non-repudiation evidences and the way they have to be interpreted.

When Alice denies the origin of the message, Bob has to present to the judge EOO, EOO_k or Con_k, ℓ, c, m and k. The judge verifies that

- $\mathsf{EOO} = S_A(f_{\mathsf{EOO}}, B, \ell, c)$,
- $\mathsf{EOO}_k = S_A(f_{\mathsf{EOO}_k}, B, \ell, k)$ or $\mathsf{Con}_k = S_{\mathsf{TTP}}(f_{\mathsf{Con}_k}, A, B, \ell, k)$,
- $\ell = h(m, k)$,
- $c = E_k(m)$.

If Bob can provide all the required items and all the checks hold, the adjudicator claims that Alice is at the origin of the message.

When Bob denies the receipt of m, Alice can prove his receipt of the message by presenting EOR, EOR_k or Con_k, ℓ, c, m and k to a judge. The judge verifies that

- $\mathsf{EOR} = S_B(f_{EOR}, A, \ell, c)$,
- $EOR_k = S_B(f_{\mathsf{EOR}_k}, A, \ell, k)$ or $\mathsf{Con}_k = S_{\mathsf{TTP}}(f_{\mathsf{Con}_k}, A, B, \ell, k)$,
- $\ell = h(m, k)$,
- $c = E_k(m)$.

If Alice can present all of the items and all the checks hold, the adjudicator concludes that Bob received the message.

6.2.2 Analysis

Zhou et al. suppose that the channels between Alice and Bob are unreliable and the channels between the TTP and both Alice and Bob are resilient. In our framework we do not model the FTP get operation, but assume that the TTP takes the initiative to send messages 2 and 3 of the recovery protocol to Alice, respectively Bob, as soon as these actions are possible. Moreover, we model a special player "timer", who sends at some moment a message to the TTP to model the fact that the time-out has expired. The player "timer" is supposed to act honestly, which means that he may wait for an arbitrary, but *finite* amount of time before sending the message to the TTP. Moreover, the communication channel between the TTP and the timer is assumed to be operational in order to avoid the time-out being delayed, once it is triggered. We define the non-repudiation of origin evidence

$$\mathsf{NRO} = B.knows(\mathsf{EOO}) \wedge (B.knows(\mathsf{EOO}_k) \vee B.knows(\mathsf{Con}_k))$$

and the non-repudiation of receipt evidence

$$\mathsf{NRR} = A.knows(\mathsf{EOR}) \wedge (A.knows(\mathsf{EOR}_k) \vee A.knows(\mathsf{Con}_k)).$$

6.2.2.1 Viability

As for the ASW certified e-mail protocol, we start by checking the viability property with the help of our tool. We have, as in the previous protocol, that the protocol is viable, when the communication channels between Alice and Bob are resilient. If the communication channels are unreliable the first message could get lost. Again, as in the previous protocol, the weaker version of the property, expressed by the CTL formula, holds whatever quality we assume on the communication channels.

6.2.2.2 Fairness

While investigating the protocol, we succeeded in finding a flaw in the protocol. The flaw described below has briefly been discussed in [84], where we presented a non-repudiation protocol overcoming this problem. Remember that the communication channels between the TTP and both Alice and Bob are resilient and that a finite time-out must be chosen by Alice at the beginning of the protocol. The fact that Alice chooses the time-out is reflected by letting Alice cooperate with the player "timer". Even if this is not an accurate model of the reality, as Alice should fix the time-out at the beginning, and should not be able to adapt it, we will see that this modelling gives interesting results on the protocol. When the main protocol has been executed until step 3, Bob has received all of his evidences and may decide to stop the protocol in order to try to cheat Alice. At this point we check whether Alice (in cooperation with the timer) does have a strategy against Bob who is cooperating with the communication channels, to receive her non-repudiation of receipt evidences. As Alice cooperates with the timer, she entirely controls the triggering of the time-out. Intuitively, it means that Alice can choose the time-out to occur as late as she wants. As Alice does not receive her evidence, she has to initiate a recovery. For the recovery protocol to be successful, the request needs to arrive before the time-out. However, as communication channels can help Bob, this message can be delayed until the time-out occurs. The intuition is that a resilient channel has the capacity to delay messages: thus for each possible finite time-out value chosen by Alice, Bob (with the help of the communication channels) can delay long enough the delivery of the message in order for the TTP to reject the recovery request. The flaw is detected, while verifying the following ATL-formula that evaluates to true.

$$\langle\langle \mathsf{B}, \mathsf{Com} \rangle\rangle \Diamond (\mathsf{NRO} \wedge \neg \langle\langle \mathsf{A}, \mathsf{Timer} \rangle\rangle \Diamond (\mathsf{NRR}))$$

The formula verifies whether Bob, collaborating with the communication channels, has a strategy to cheat Alice. However, when we change the communication channel to be operational (messages are transmitted immediately) the protocol becomes fair. Unfortunately, operational channels are rather unrealistic in practice. On the other hand the formula verifying whether Alice has a strategy to cheat Bob fails.

Unfortunately, unlike invariant checking, when an ATL formulae is violated, we cannot provide an error trace. This is due to the fact that we reason about strategies, and the malicious behaviour may change with respect to the honest player's action. However, we can also use our method to find the origin of a flaw. We can change coalitions to detect the player(s) that make the property fail. As an example, take a look at the flaw we described in the previous paragraph. We asked whether Bob, together with all the communication channels, has a strategy to obtain the NRO evidence, such that Alice, in cooperation with the timer, does not have any strategy to receive the NRR evidence. Using MOCHA we showed the existence of a flaw, by showing that this formula is verified on the ZG protocol. On the other hand, when adding the communication channel between Alice and the TTP to Alice's coalition we succeeded in avoiding the flaw. This is reflected by the fact that the following formula does not hold.

$$\langle\langle \mathsf{B} \rangle\rangle \Diamond (\mathsf{NRO} \wedge \neg \langle\langle \mathsf{A}, \mathsf{Timer}, \mathsf{Com} \rangle\rangle \Diamond \mathsf{NRR})$$

Once we know that the communication channel is at the origin of the error, it may be interesting to alter the channel's quality. In that way we see which modifications are needed to recover the fairness property. In this example we can conclude that replacing the resilient channel by an operational channel is enough to prevent protocol failure. Hence, we have a rather easy and quick investigation method to locate and possibly prevent flaws. This methodology may also be useful during the design process of new protocols.

6.2.2.3 Timeliness

In this protocol timeliness is assured by the time-out and rather trivially verified when the communication channels with the TTP are operational, as it is requested for the protocol to be fair. The next protocols give us more interesting case studies for the timeliness property.

6.3 Kremer-Markowitch non-repudiation protocol

In 2000, we presented in collaboration with Olivier Markowitch the KM non-repudiation protocol [84] in order to overcome the shortcomings of the previously described ASW certified e-mail and ZG non-repudiation protocols. In this section, we also introduce a variant of the protocol in [84]. This variant protocol is designed such that fairness is respected, but not timeliness. It should show the importance of the timeliness property and highlight the kind of problems one can face when timeliness is not ensured.

6.3.1 Protocol description

The protocol is composed of a main, a recovery protocol and an abort protocol. The recovery protocol can be launched by both Alice and Bob, while the abort protocol can be executed only by Alice. While a recovery protocol forces the exchange to take place, the abort protocol informs the TTP of Alice's intention to stop the protocol. The recovery protocol and the abort protocol are mutually exclusive. The mutual exclusion is guaranteed by the TTP. The channels between Alice, respectively Bob, and the TTP are supposed to be resilient. The channels between Alice and Bob may be unreliable. We use the same notation, including the label ℓ and the cipher c, as in the ZG non-repudiation protocol. The evidences generated in this protocol are the following.

- the evidence of origin for the cipher: $\mathsf{EOO} = S_A(f_{\mathsf{EOO}}, B, \mathsf{TTP}, \ell, h(c))$
- the evidence of receipt for the cipher: $\mathsf{EOR} = S_B(f_{\mathsf{EOR}}, A, \mathsf{TTP}, \ell, h(c))$
- the submission evidence for key k: $\mathsf{Sub} = S_A(f_{\mathsf{Sub}}, B, \ell, E_{\mathsf{TTP}}(k))$
- the evidence of origin for key k: $\mathsf{EOO}_k = S_A(f_{\mathsf{EOO}_k}, B, \ell, k)$
- the evidence of receipt for key k: $\mathsf{EOR}_k = S_B(f_{\mathsf{EOR}_k}, A, \ell, k)$
- the recovery request: $\mathsf{Rec}_X = S_X(f_{\mathsf{Rec}_X}, Y, \ell)$
- the confirmation evidence for key k: $\mathsf{Con}_k = S_{\mathsf{TTP}}(f_{\mathsf{Con}_k}, A, B, \ell, k)$

- the abort request: $\mathsf{Abort} = S_A(f_{\mathsf{Abort}}, B, \ell)$
- the abort confirmation evidence: $\mathsf{Con}_a = S_{\mathsf{TTP}}(f_{\mathsf{Con}_a}, A, B, \ell)$

Main protocol. The detailed description of the main protocol is given in protocol 6.3. The main protocol can be divided into two parts. The first part is the exchange of the cipher of message m under key k and the evidence of origin for the cipher against an evidence of receipt for this cipher. The second part consists of the exchange of the key k and the corresponding evidence of origin against the evidence of receipt for the key k. If the second message does not arrive to Alice before a reasonable amount of time[1], she executes an abort protocol. If the third or fourth messages do not arrive, Bob and Alice, respectively, can launch a recovery protocol.

Protocol 6.3 KM non-repudiation protocol—Main

1. A → B: $f_{\mathsf{EOO}}, f_{\mathsf{Sub}}, B, \mathsf{TTP}, \ell, c, E_{\mathsf{TTP}}(k), \mathsf{EOO}, \mathsf{Sub}$
2. B → A: $f_{\mathsf{EOR}}, A, \mathsf{TTP}, \ell, \mathsf{EOR}$

   if A times out then abort
3. A → B: $f_{\mathsf{EOO}_k}, B, \ell, k, \mathsf{EOO}_k$

   if B times out then recovery
4. B → A: $f_{\mathsf{EOR}_k}, A, \ell, \mathsf{EOR}_k$

   if A times out then recovery

Abort protocol. The abort protocol, described in protocol 6.4, can be launched at any time by Alice. If a valid abort request arrives, the TTP first verifies if the current protocol run has not yet been recovered or aborted. The protocol run is uniquely identified by the label ℓ and the identities (A, B). If neither a recovery protocol nor an abort protocol has been executed, the TTP informs both Alice and Bob that the protocol has been aborted. It is important to see that an abort evidence does not mean that the exchange did not take place. It is possible to complete a faultless main protocol and execute (dishonestly) the abort protocol later on. An abort evidence only informs Alice and Bob that no recovery will be accepted any more by the TTP, regarding this protocol run.

Recovery protocol. A detailed description of the recovery protocol is given in protocol 6.5. The recovery protocol is intended to be executed by either Alice or Bob. Bob can launch the protocol, as soon as the first message of the main protocol arrives. Alice can launch the protocol once the second message of the main protocol arrived. The aim of the recovery protocol is to provide Alice with the possibly missing evidence of receipt for the cipher (EOR), as well as a substitution (Con_k) for the evidence of receipt of the key k, and

[1] Alice chooses herself how long she decides to wait for a given message before reacting.

Protocol 6.4 KM non-repudiation protocol—Abort

1. A $\rightarrow$ TTP: $f_{\mathsf{Abort}}, \ell, B, \mathsf{Abort}$

   if *aborted* or *recovered* then stop

   else *aborted*:=true
2. TTP $\rightarrow$ A: $f_{\mathsf{Con}_a}, A, B, \ell, \mathsf{Con}_a$
3. TTP $\rightarrow$ B: $f_{\mathsf{Con}_a}, A, B, \ell, \mathsf{Con}_a$

Bob with a substitution (Con_k) of the missing evidence of origin for the key k, as well as the key itself.

Protocol 6.5 KM non-repudiation protocol—Recovery

1. X $\rightarrow$ TTP: $f_{\mathsf{Rec}_X}, f_{\mathsf{Sub}}, Y, \ell, h(c), E_{\mathsf{TTP}}(k), \mathsf{Rec}_X, \mathsf{Sub}, \mathsf{EOR}, \mathsf{EOO}$

   if *aborted* or *recovered* then stop

   else *recovered*:=true
2. TTP $\rightarrow$ A: $f_{\mathsf{Con}_k}, A, B, \ell, k, \mathsf{Con}_k, \mathsf{EOR}$
3. TTP $\rightarrow$ B: $f_{\mathsf{Con}_k}, A, B, \ell, k, \mathsf{Con}_k$

Dispute resolution. When Alice denies the origin of the message, Bob has to present to the judge EOO, EOO_k or Con_k, ℓ, c, m and k. The judge verifies that

- $\mathsf{EOO} = S_A(f_{\mathsf{EOO}}, B, \mathsf{TTP}, \ell, h(c))$,
- $\mathsf{EOR} = S_B(f_{\mathsf{EOR}}, A, \mathsf{TTP}, \ell, h(c))$ or $\mathsf{Con}_k = S_{\mathsf{TTP}}(f_{\mathsf{Con}_k}, A, B, \ell, k)$,
- $\ell = h(m, k)$,
- $c = E_k(m)$.

If Bob can provide all the required items and all the checks hold, the adjudicator claims that Alice is at the origin of the message.

When Bob denies the receipt of m, Alice can prove his receipt of the message by presenting EOR, EOR_k or Con_k, ℓ, c, m and k to a judge. The judge verifies that

- $\mathsf{EOO}_k = S_A(f_{\mathsf{EOO}_k}, B, \ell, k)$,
- $\mathsf{EOR}_k = S_B(f_{\mathsf{EOR}_k}, A, \ell, k)$ or $\mathsf{Con}_k = S_{\mathsf{TTP}}(f_{\mathsf{Con}_k}, A, B, \ell, k)$,
- $\ell = h(m, k)$,
- $c = E_k(m)$.

If Alice can present all of the items and all the checks hold, the adjudicator concludes that Bob received the message.

Note that, in this work, we are not proving the correctness of the dispute resolution protocol, and hence we do not guarantee that the evidences are correct. We focus on the exchange part of the protocol. In chapter 8, we discuss some other approaches which consider the correctness of the evidences and the dispute resolution.

6.3.2 Analysis

To analyse the KM non-repudiation protocol, we first have to define the non-repudiation evidences, in a similar way than in the ZG protocol.

$$\mathsf{NRO} = B.knows(\mathsf{EOO}) \wedge (B.knows(\mathsf{EOO}_k) \vee B.knows(\mathsf{Con}_k))$$

$$\mathsf{NRR} = A.knows(\mathsf{EOR}) \wedge (A.knows(\mathsf{EOR}_k) \vee A.knows(\mathsf{Con}_k))$$

When analysing the protocol with Mocha we did not find any errors. However, we also designed an interesting variant of the protocol where we removed the abort protocol. Removing the abort protocol introduces a new kind of flaw, that does not compromise fairness, as the previous shown errors did, but the timeliness property. The description of the variant protocol is very similar to the original one, but if the second message does not arrive, Alice does not have the possibility to abort the protocol. Note that a protocol which is fair, but does not respect timeliness can be achieved with a three message main protocol. Here, we keep the four message main protocol and only remove the abort protocol, in order to stay as close as possible to the original protocol.

6.3.2.1 Viability

As in the previous protocols, we have that viability holds unless the communication channels between Alice and Bob are unreliable. In that case viability fails as all messages that should be exchanged between the two participants could be lost. Again, the weaker CTL formula holds, whatever quality is defined for the communication channels.

6.3.2.2 Fairness

We have analysed both the original and the variant protocol with different communication channels qualities. When the communication channels between Alice and Bob, as well as between the TTP and respectively Alice and Bob are unreliable, we obtain the following results. When verifying fairness for Alice, we have that the formula

$$\langle\langle B, \mathcal{C}om \rangle\rangle \Diamond (\mathsf{NRO} \wedge \neg \langle\langle A_h \rangle\rangle \Diamond \mathsf{NRR})$$

holds, which means that when a dishonest Bob controls the communication channels, the protocol is not fair for Alice. If for instance Alice executes the main protocol until the third message, Bob can stop the protocol and the communication channels can lose Alice's recovery request to the TTP. On the other hand, when Bob does not control the communication channels, fairness holds, independently of the quality of the channels. For instance we have that the following formula holds.

$$\neg\langle\langle B\rangle\rangle\Diamond(\mathsf{NRO} \wedge \neg\langle\langle A_h, \mathcal{C}om\rangle\rangle\Diamond\mathsf{NRR})$$

We have the same results when verifying fairness for Bob.

In [84], we assumed that the communication channels between the TTP and both Alice and Bob are resilient, while the channels between Alice and Bob may be unreliable. In that case we have that fairness holds, even if the malicious participant collaborates with the channels.

6.3.2.3 Timeliness

While the original and the variant protocol give similar results for the fairness property, they differ when analysing timeliness. We suppose that the communication channels' quality is as specified in [84], i.e. the channels between Alice and Bob are unreliable, while the channels between the TTP and respectively Alice and Bob are resilient. We first study the timeliness property for Bob. We verify the following formula.

$$\langle\langle B_h\rangle\rangle\Diamond(B.\mathit{rec}(m_1) \rightarrow (\mathsf{stop}_B \wedge (\neg\mathsf{NRO} \rightarrow \neg\langle\langle A\rangle\rangle\Diamond\mathsf{NRR})))$$

The formula is slightly modified. We only require the protocol to hold if Bob received the first message. This is expressed by the predicate $B.\mathit{rec}(m_1)$. If the first message is not received, we consider that the protocol is not started. The formula is verified for both the original protocol as well as the variant protocol. The same holds for the stronger CTL formula.

The more interesting part is when verifying timeliness for Alice. We verify the following formula.

$$\langle\langle A_h\rangle\rangle\Diamond(\mathsf{stop}_A \wedge (\neg\mathsf{NRR} \rightarrow \neg\langle\langle B\rangle\rangle\Diamond\mathsf{NRO}))$$

While verifying timeliness for Alice, we found that this property is respected for the original protocol, but does not hold on the variant protocol. Note that the protocol was designed in that way to highlight the importance of the timeliness property, which might not hold even if the protocol is fair. Consider the following scenario. Alice sends the first message. Now Bob is in a state where he knows m_1 (he just received it from Alice) and he knows m_2 (he is supposed to send it to Alice). This means that Bob is able to launch a recovery protocol. If the recovery protocol is launched, and Alice did not stop the protocol, both will receive the expected non-repudiation evidences. However, if Bob does not send the second message, Alice cannot "move", as the variant protocol does not offer a possibility to abort and recovery is not yet possible for Alice. All Alice can do is to wait for Bob to launch a recovery protocol. However, Bob could never launch this recovery protocol, which means that Alice has to wait for an infinite time. On the other hand, Alice must not quit the protocol, because otherwise Bob could recover and receive the non-repudiation of origin evidence, without Alice receiving the non-repudiation of receipt evidence. Note that this scenario does not violate fairness, but only timeliness.

6.4 Analysis of two contract signing protocols

In this section we analyse two contract signing protocols. The first one is due to Asokan, Shoup and Waidner [14], while the second one has been presented by Garay, Jakobsson and MacKenzie [67]. We have regrouped these two protocols, because the analysis of abuse-freeness in these protocols is related. We first describe both protocols, before giving a detailed analysis.

6.4.1 Description of the ASW contract signing protocol

The protocol was initially presented in [14] and consists of a main protocol, an abort protocol and a recovery protocol. We use the following additional notation to describe the protocol.

- t: the contract text
- N_A: a nonce chosen by A
- N_B: a nonce chosen by B

Main protocol. The main protocol (protocol 6.6) consists of four messages. In the first two messages Alice and Bob express their intention to sign a given contract t. Therefore both of them commit to their respective nonces, N_A and N_B, without however revealing them, by sending a hash of the nonces. In the messages 3 and 4, they exchange the previously committed nonces. The signed commitment and the nonce itself, are considered as an evidence that the contract is signed. Now, we show how each entity has to react in case of problem, i.e. if a given message does not arrive, either because it has been lost or never been sent. If the first message does not arrive, the protocol is not considered as started, and so Bob does not have to take any special action. If the second message does not arrive (this can also be due to the fact that the first message has been lost previously), Alice can launch an abort protocol. Note, that at this point Alice cannot launch the recovery protocol, as she does not have any proof that the protocol has been started with Bob. If the third or the fourth message does not arrive, Bob, respectively Alice, can launch a recovery protocol, to force the successful termination of the protocol.

Abort protocol. Alice starts the abort protocol (protocol 6.7) if the second message of the main protocol does not arrive. Therefore she sends a signed abort request to the TTP. Note that the abort protocol and the recovery protocol are mutually exclusive. If a recovery protocol, for the given protocol run, identified by the first message of the main protocol, has been launched before, the TTP will issue an affidavit, replacing the nonce, to witness to a successful contract signing. Otherwise, the TTP sends a signed abort token. Note, that this abort token does not mean that the protocol has not been signed. Alice could run a successful main protocol, and hence get a signed contract with Bob, and run an abort protocol afterwards. The abort token only represents the promise of the TTP not to recover the same protocol run later.

Protocol 6.6 ASW contract signing protocol—Main

1. A → B: $m_1 = S_A(A, B, \mathsf{TTP}, t, h(N_A))$
2. B → A: $m_2 = S_B(h(m_1), h(N_B))$

   if A times out then abort
3. A → B: $m_3 = N_A$

   if B times out then recovery
4. B → A: $m_4 = N_B$

   if A times out then recovery

Protocol 6.7 ASW contract signing protocol—Abort

1. A → TTP: $a_1 = S_A(abort, m_1)$

   if recovered(m_1) then
2. TTP → A: $S_{\mathsf{TTP}}(m_1, m_2)$

   else

   aborted(m_1):=true
3. TTP → A: $S_{\mathsf{TTP}}(aborted, a_1)$

Recovery protocol. The recovery protocol (protocol 6.8) is normally launched if the third or the fourth message of the protocol does not arrive quickly enough. As the protocol is similar whether it is launched by Alice or Bob, we denote by X, the initiator of the protocol. The recovery request, consists in a signed message containing the two first messages of the main protocol. If the given protocol run has already been aborted, the TTP sends a copy of the abort token. Otherwise, the TTP sends an affidavit to confirm that the contract is signed.

Protocol 6.8 ASW contract signing protocol—Recovery

1. X $\rightarrow$ TTP: m_1, m_2

   if aborted(m_1) then

2. TTP $\rightarrow$ X: $S_{\mathsf{TTP}}(aborted, a_1)$

   else

   recovered(m_1):=true

3. TTP $\rightarrow$ X: $S_{\mathsf{TTP}}(m_1, m_2)$

6.4.2 Description of the GJM contract signing protocol

In [67], Garay, Jakobsson and MacKenzie presented a new protocol, as well as the concept of abuse-free contract signing. The structure of the protocol is very similar to the ASW protocol. However, to achieve abuse-freeness, they introduced a new cryptographic primitive, known as *Private Contract Signatures*, which we already recalled in definition 3.9.

Main protocol. In the main protocol (protocol 6.9), Alice and Bob first exchange, in messages 1 and 2, their respective PCS on the contract. Then they go on, in messages 3 and 4, to exchange their universally verifiable signatures on the contract. The general structure of the protocol is similar to the ASW protocol. If the second message does not arrive in due time, Alice launches an abort protocol. If the third or the fourth message does not arrive quickly enough, Bob, respectively Alice, execute the recovery protocol.

Abort protocol. The abort protocol (protocol 6.10) is executed by Alice if the second message of the main protocol does not arrive in due time. Alice therefore sends a signed abort request to the TTP. As in the ASW protocol, the abort and the recovery protocols are mutually exclusive. The protocol run is identified by the identities of Alice and Bob, and the contract text t. If the current protocol run has already been recovered, the TTP sends Bob's universally verifiable signature to Alice.

Recovery protocol. The recovery protocol (protocol 6.11) is foreseen to be executed if either the third or the fourth message does not arrive correctly. As the protocol is symmetric for Alice and Bob, we denote by X the initiator of the protocol and by Y the

Protocol 6.9 GJM contract signing protocol—Main

1. A $\rightarrow$ B: $m_1 = PCS_A(t, B, \mathsf{TTP})$
2. B $\rightarrow$ A: $m_2 = PCS_B(t, A, \mathsf{TTP})$

   if A times out then abort
3. A $\rightarrow$ B: $S_A(t)$

   if B times out then recovery
4. B $\rightarrow$ A: $S_B(t)$

   if A times out then recovery

Protocol 6.10 GJM contract signing protocol—Abort

1. A $\rightarrow$ TTP: $a_1 = S_A(t, A, B, abort)$

   if recovered(A, B, t) then
2. TTP $\rightarrow$ A: $a_2 = S_B(t)$

   else

   aborted(A, B, t):=true
3. TTP $\rightarrow$ A: $a_3 = S_{\mathsf{TTP}}(a_1)$

other entity. To start the protocol, the initiator sends both his signature and the other participant's PCS to the TTP. If the protocol has been aborted before, the TTP sends a copy of the abort token. Otherwise the TTP converts both PCS, saves the universally verifiable signatures, and sends Y's universally verifiable signature to X.

Protocol 6.11 GJM contract signing protocol—Recovery

1. X → TTP: $S_X(t), PCS_Y(t, X, \mathsf{TTP})$

   if aborted(A, B, t) then

2. TTP → X: $S_{\mathsf{TTP}}(a_1)$

   else

   recovered(A, B, t):=true

3. TTP → X: $S_Y(t)$

6.4.3 Analysis

In this section we define two predicates contract$_\mathsf{A}$ and contract$_\mathsf{B}$, which become true when Bob receives Alice's contract, respectively when Alice receives Bob's contract. In the ASW protocol contract$_\mathsf{A}$ is defined as

$$\mathsf{contract_A} = B.knows(m_1) \wedge (B.knows(m_3) \vee B.knows(S_{\mathsf{TTP}}(m_1, m_2))).$$

In a similar way, the predicate contract$_\mathsf{B}$ is defined as

$$\mathsf{contract_B} = A.knows(m_2) \wedge (A.knows(m_4) \vee A.knows(S_{\mathsf{TTP}}(m_1, m_2))).$$

In the GJM protocol, we have that

$$\mathsf{contract_A} = B.knows(S_A(t))$$

and

$$\mathsf{contract_B} = A.knows(S_B(t)).$$

6.4.3.1 Viability

When testing viability for both protocols, we have similar results as for the other protocols. For the ATL formula representing the stronger version of viability to hold, we need that the channels between Alice and Bob are resilient. If the channels collaborate with Alice and Bob or if we consider the CTL version of viability the property holds whatever assumptions we make on the communication channels.

6.4.3.2 Fairness

When studying fairness for the ASW protocol, we come to the conclusion that the protocol requires that the channels between the TTP and Alice, respectively Bob need to be resilient. The channels between Alice and Bob may be unreliable. If the channels' quality is lower, it is easy to see that the protocol cannot provide fairness. Also note that the protocol remains fair when the dishonest participant is strongly dishonest. In [14], Asokan et al. suppose that the channels are confidential. In the case of the contract signing protocol this hypothesis can thus be relaxed, while it was indeed needed for Asokan et al.'s certified e-mail protocol. Shmatikov and Mitchell already noted in [146] that confidentiality is not needed.

In [147], Shmatikov and Mitchell discovered a flaw compromising fairness for Bob in the GJM protocol, if the hypothesis on the confidentiality of the communication channels is relaxed. The scenario is as follows. Alice sends the first message and then aborts the protocol. Bob sends the second message and waits for the third message. However, Alice does not send the third message and waits for Bob to launch a recovery protocol. As the protocol has been aborted before, the TTP replies with an abort confirmation. However, Bob's recovery request contained his signature on the contract text t. Hence, Alice can eavesdrop the signature on the channel and obtain Bob's signature on the contract. The authors of the protocol did specify that the channels need to be confidential, so this scenario, although interesting, is not considered to be an attack. Shmatikov and Mitchell proposed a fix for the case where channels are not confidential, which consists in replacing the recovery request by the following message:

$$PCS_A(t, B, \mathsf{TTP}), PCS_B(t, A, \mathsf{TTP}).$$

With this modification the TTP needs to convert one additional PCS, but the attack is prevented.

Using our tool the above attack on the unmodified protocol can be detected by modelling Alice as a strongly dishonest participant. The attack is discovered by noting that the following formula holds.

$$\langle\!\langle A, \mathcal{C}om \rangle\!\rangle \Diamond (\mathsf{contract_B} \wedge \neg \langle\!\langle B_h \rangle\!\rangle \Diamond \mathsf{contract_B})$$

However, when Alice is modelled as being weakly dishonest, the attack is prevented. In the remaining, we study the fixed version, as proposed by Shmatikov and Mitchell. We did not find any additional errors in the protocol.

6.4.3.3 Timeliness

For both protocols we have that timeliness holds. As for the KM non-repudiation protocol, when verifying timeliness for Bob, we slightly modify the formula in order to require that the first message has been sent.

6.4.3.4 Balance and abuse-freeness

The major contribution of our work considering these two protocols is the analysis of balance and abuse-freeness. Remember that in our discussion on the modelling of abuse-freeness in section 5.2.4, we proposed two possible ways of modelling this property. The difference between the two ways of modelling balance and abuse-freeness lies in the formation of the coalitions, i.e. the control of the communication channels differs. We believe that generally the collaboration of a dishonest participant with the communication channel does not model a complete control of the channels, but is used to model worst-case behaviour. If we want to verify that the protocol respects balance, respectively abuse-freeness in a worst-case behaviour, then we should consider the collaboration between the dishonest party and the communication channels. However, when considering an attack on abuse-freeness, a dishonest participant must prove to Charlie that he can choose the outcome of the protocol. There it might be interesting to ask whether this participant can choose the outcome in all circumstances, i.e. independently of the behaviour of the communication channels. This can be modelled by letting the communication channels collaborate with the honest participant, i.e. the worst case behaviour is differs from properties such as fairness or timeliness. We also discussed the fact that the CTL formulation of the property would result into a too strong requirement. Hence the usage of ATL is crucial to achieve an accurate modelling of this property.

We verify balance and abuse-freeness in both scenarios where the communication channels collaborate with either the dishonest party, or the honest party and show that the results can differ depending on the chosen coalition. To overcome this ambiguity, we propose a new definition of abuse-freeness, and also discuss this case. We start discussing our analysis when the dishonest party completely controls the communication channels.

Dishonest party controlling the communication channels. To verify whether the GJM protocol is balanced for Bob we verify the following formula:

$$\neg(\langle\langle A, \mathcal{C}om\rangle\rangle\Diamond\ \mathsf{contract_B} \wedge \langle\langle A, \mathcal{C}om\rangle\rangle\Diamond(\mathsf{aborted} \wedge\ \langle\langle B_h\rangle\rangle\Diamond\ \mathsf{contract_B}))$$

Under the assumption that the communication channels are resilient and completely controlled by the dishonest party, neither the GJM, nor the ASW protocols are balanced. This is due to the fact, that once Alice has received the second message of the main protocol she has all needed information to either abort the protocol, by sending an abort request to the TTP, or to successfully complete the protocol, by sending to Bob the third message of the main protocol, or sending a recovery request to the TTP. We can also show that neither the ASW nor the GJM protocol is balanced for Alice. When Alice sends the first message, Bob does not respond. At that point, all Alice can do is send an abort. At the same time Bob can send a recovery request, and schedule the communication channels such that either the abort or the recovery request arrives first in order to determine the outcome of the protocol.

Abuse-freeness for Bob is modelled by the following formula.

$$\neg\langle\langle A, \mathcal{C}om\rangle\rangle\Diamond(A.\mathsf{prove2C}\wedge\langle\langle A, \mathcal{C}om\rangle\rangle\Diamond\mathsf{contract_B}\wedge\langle\langle A, \mathcal{C}om\rangle\rangle\Diamond(\mathsf{aborted}\wedge\neg\langle\langle B_h\rangle\rangle\Diamond\ \mathsf{contract_A}))$$

For the specification of abuse-freeness to be complete, we have to define the $A.\mathsf{prove2C}$ predicate. Remember, that the predicate should be true once we have evidence, which is verifiable by Charlie that Bob is involved in the protocol. We can define the predicate for the ASW protocol as follows.

$$A.\mathsf{prove2C} = A.knows(m_2) \vee A.knows(m_4)$$

In the case of the GJM protocol the definition is slightly different and should be as follows.

$$A.\mathsf{prove2C} = A.knows(m_4)$$

The difference is due to the usage of PCS, which cannot be verified by Charlie. We also need to define the predicate $B.\mathsf{prove2C}$ in order to be able to verify abuse-freeness for Alice. In the ASW protocol, we have the following definition.

$$B.\mathsf{prove2C} = B.knows(m_1) \vee B.knows(m_3)$$

In the GJM protocol we have that

$$B.\mathsf{prove2C} = B.knows(m_3).$$

We do not include abort replies sent by the TTP in the definition of prove2C. When Alice receives an abort reply, it does not contain Bob's signature. Moreover, Alice could launch an abort request, and receive an abort reply from the TTP without Bob being aware of the protocol run. On the other hand, when Bob receives an abort reply, the outcome of the protocol is already decided. Otherwise, the protocol would be unfair. Hence, prove2C is set to true whenever we can prove that the other participant has started the protocol and the outcome is still open.

When verifying abuse-freeness we find that the ASW protocol is not abuse-free, while the property holds on the GJM protocol. Abuse-freeness does not hold on the ASW protocol as the protocol is not balanced and Alice can prove to an external party that the protocol has been started, once she received Bob's first message. Here we see why PCS are useful in the GJM protocol. In the GJM protocol, Alice does not get an evidence, that the protocol has been started with Bob, before having either aborted or successfully finished the protocol. These results reflect what was widely believed and claimed in the paper, where the GJM protocol was presented, namely that PCS are needed to achieve abuse-free contract signing.

Honest party controlling the communication channels. Now we are changing the assumptions, so that the dishonest party does not collaborate with the communication channels any more, but has to prove to Charlie, that he has the power to abort or successfully sign a contract with Bob, under all circumstances, independent of the way the communication channels react. Now, to verify whether, for instance, the GJM protocol is balanced for Bob or not we verify the following formula.

$$\neg(\langle\!\langle A\rangle\!\rangle\Diamond\ (Sig_B) \wedge \langle\!\langle A\rangle\!\rangle\Diamond(\mathsf{aborted} \wedge\ \langle\!\langle B_h, \mathcal{C}om\rangle\!\rangle\Diamond\ Sig_A))$$

Under these assumptions both protocols are balanced for Bob. If Alice gets the second message of the main protocol, she does not have a strategy anymore to abort the protocol. Consider the following scenario, where Alice tries to abort the protocol. Alice sends an abort request to the TTP, but the request is delayed. In between, Bob, not having received the third message of the main protocol, times out and sends a recovery request to the TTP. Bob's recovery request may arrive at the TTP before Alice's abort request, and hence Alice's attempt to abort the protocol fails. Note, that fairness is not broken by this scenario, as in case of a recovery both Alice and Bob get signed contracts and in case of an abort none of them gets a signed contract. In a similar way, we can show that the protocols are both balanced for Alice. As the protocols are balanced, by theorem 5.13, we have that they are also abuse-free. This means that under these assumptions, no PCS are needed to achieve abuse-freeness.

A novel definition of abuse-freeness. Although the ASW protocol may be abuse-free with the weaker definition, the fact that Alice can prove to Charlie that a contract signing protocol has been started with Bob, can be considered as a problem. Alice does perhaps not have a strategy to choose between aborting or signing, but Bob does not really control the communication channels neither. We only model a worst case behaviour. Therefore, Alice's success to choose between aborting and signing depends on the way the communication channels react. We think that this ambiguity should be considered as a problem. Therefore we propose a new definition of abuse-freeness, which is slightly stronger, but deals with the same initial problem. We redefine abuse-freeness for Bob by the following ATL formula.

$$\neg\langle\langle A\rangle\rangle\Diamond(A.\mathsf{prove2C} \wedge \neg\langle\langle B_h\rangle\rangle\Diamond\ \mathsf{contract_A})$$

The new definition requires that Alice cannot reach a state in the protocol, where she can prove to Charlie that the protocol has been initiated with Bob, without Bob having a strategy to successfully complete the protocol. Abuse-freeness for Alice is defined in a similar way.

Theorem 6.1 The new definition of abuse-freeness implies the previous definition, i.e. given any ATS S and state q

$$S, q \models \neg\langle\langle A\rangle\rangle\Diamond(A.\mathsf{prove2C} \wedge \neg\langle\langle B_h\rangle\rangle\Diamond\ \mathsf{contract_A})$$

$$\Rightarrow$$

$$S, q \models \neg\left(\langle\langle A\rangle\rangle\Diamond(A.\mathsf{prove2C} \wedge \langle\langle A\rangle\rangle\Diamond\mathsf{contract_B} \wedge \langle\langle A\rangle\rangle\Diamond(\mathsf{aborted} \wedge \neg\langle\langle B_h\rangle\rangle\Diamond\ \mathsf{contract_A}))\right)$$

and

$$S, q \models \neg\langle\langle B\rangle\rangle\Diamond(B.\mathsf{prove2C} \wedge \neg\langle\langle A_h\rangle\rangle\Diamond\ \mathsf{contract_B})$$

$$\Rightarrow$$

$$S, q \models \neg\left(\langle\langle B\rangle\rangle\Diamond(B.\mathsf{prove2C} \wedge \langle\langle B\rangle\rangle\Diamond\mathsf{contract_A} \wedge \langle\langle B\rangle\rangle\Diamond(\mathsf{aborted} \wedge \neg\langle\langle A_h\rangle\rangle\Diamond\ \mathsf{contract_B}))\right).$$

□

Proof:
Using the proof rules, given in lemma 4.16, we can formally show that the new definition of abuse-freeness for Bob implies the former one:

$$S, q \models \neg\langle\langle A\rangle\rangle\Diamond(A.\mathsf{prove2C} \wedge \neg\langle\langle B_h\rangle\rangle\Diamond\ \mathsf{contract_A})$$
$$\Rightarrow$$
$$S, q \models \neg\left(\langle\langle A\rangle\rangle\Diamond(A.\mathsf{prove2C} \wedge \langle\langle A\rangle\rangle\Diamond\mathsf{contract_B} \wedge \langle\langle A\rangle\rangle\Diamond(\mathsf{aborted} \wedge \neg\langle\langle B_h\rangle\rangle\Diamond\ \mathsf{contract_A}))\right)$$

We prove this statement by proving its contraposition.

1. $S, q \models \langle\langle A\rangle\rangle\Diamond(A.\mathsf{prove2C} \wedge \langle\langle A\rangle\rangle\Diamond\mathsf{contract_B} \wedge \langle\langle A\rangle\rangle\Diamond(A.\mathsf{aborted} \wedge \neg\langle\langle B\rangle\rangle\Diamond\ \mathsf{contract_A}))$

2. $S, q \models \langle\langle A\rangle\rangle\Diamond(A.\mathsf{prove2C} \wedge \langle\langle A\rangle\rangle\Diamond(\mathsf{aborted} \wedge \neg\langle\langle B\rangle\rangle\Diamond\ \mathsf{contract_A}))$ by 1, (R_1)

3. $S, q \models \langle\langle A\rangle\rangle\Diamond(A.\mathsf{prove2C} \wedge \langle\langle A\rangle\rangle\Diamond\neg\langle\langle B\rangle\rangle\Diamond\ \mathsf{contract_A})$ by 2, (R_1)

4. By the fact that $A.\mathsf{prove2C}$ is monotonic, i.e. once $A.\mathsf{prove2C}$ is true it remains true, we have that $A.\mathsf{prove2C} \rightarrow \forall\Box A.\mathsf{prove2C}$ and
$S, q \models \langle\langle A\rangle\rangle\Diamond(A.\mathsf{prove2C} \wedge \neg\langle\langle B\rangle\rangle\Diamond\ \mathsf{contract_A})$ by 3, (R_4)

To complete the proof a symmetric reasoning is done for abuse-freeness for Alice. □

With this new definition, we have that the GJM protocol is still abuse-free, but the ASW protocol is not abuse-free anymore, no matter whether the communication channels collaborate with Alice or Bob. Abuse-freeness for the GJM protocol is enforced by the properties of PCS. Bob receives his first proof that the protocol has been started with Alice only after the third message has been sent. So Bob receives this proof at the same moment he gets his signed contract. Alice needs to wait for the fourth message of the protocol to prove to Charlie that the protocol is engaged with Bob. This also corresponds to the state of the protocol where she gets her signed contract. Hence, both Alice and Bob, receive their proofs when they get their signed contract.

Considering the new definition, we can show that the ASW protocol is not abuse-free for Alice by looking at the following scenario. Alice sends the first message to Bob. As Alice has not yet received the second message of the protocol, the only thing she can do is to launch an abort protocol or wait for the second message to arrive. Hence, she has no strategy to successfully complete the protocol. If Bob does not send the second message, he reaches a state where he can prove to Charlie that the protocol has been started, as he received the first message signed by Alice, without Alice having a strategy to successfully complete the protocol. This scenario does not depend on the reaction of the communication channels and violates our new definition of abuse-freeness. As in the ASW protocol, conventional signatures, that can be shown as proofs to Charlie, are used from the beginning, when it is still possible to abort the protocol, this protocol should not be considered as abuse-free. However, considering the former definition of abuse-freeness, under certain assumptions the protocol respects abuse-freeness. Intuitively, this is due to the fact that the power to both complete and abort the protocol depends on the reaction of the communication channels. In our new definition however the result of the verification of the ASW protocol, as well as the GJM protocol is independent of whether the communication channels collaborate with the honest party or the malicious party. Therefore we believe that the above given definition is more concise and better reflects the initial intuition of the property than the classical one.

6.5 Conclusion

In this chapter, we used the prototype of the tool presented in chapter 5 to analyse several optimistic fair exchange protocols. We first analysed a certified e-mail protocol and a non-repudiation protocol which contained errors related to the fairness property. The associated attacks illustrate well the power of our method to find different kinds of attacks, due to the possibility to test all possible interleavings, specify different channel qualities and differentiate between strongly and weakly dishonest attackers. Then, we show that a variant of the KM protocol does respect fairness, but not timeliness. This example should emphasise the difference between these two properties and show the importance of timeliness, which is too often neglected. Finally, we study two contract signing protocols. In these protocols we focussed above all on balance and abuse-freeness. We came to the surprising result that under some hypotheses the ASW contract signing can be abuse-free without making use of PCS. Moreover, we propose an alternate, slightly stronger definition of abuse-freeness, which the ASW contract signing does not respect, while the GJM protocol does, as one would expect it. Therefore, we believe that the new definition better reflects the underlying intuition of abuse-freeness. We conclude that our methodology and tool provide great flexibility in modelling different assumptions, can find a great diversity of attacks as well as perform fine-grained analysis, able to capture subtle differences in complicated properties such as balance and abuse-freeness.

CHAPTER 7

Analysis of multi-party fair exchange protocols

In this chapter, we are going to analyse multi-party fair exchange protocols. To the best of our knowledge, this is the first study of group protocols using formal methods in the context of fair exchange protocols. More precisely, we focus on non-repudiation and contract signing protocols. The different instances of fair exchange protocols, such as non-repudiation protocols or contract signing protocols, are very similar in the two-party case. As explained in section 3.6, when generalising fair exchange protocols to more than two entities, the similarity is not maintained. This is due to the differences in the topology of the exchange. Non-repudiation protocols correspond to a one to many exchange, while contract signing is done on a full graph topology. There exist also ring topologies, but these protocols seem to have much simpler solutions than the protocols studied here. In this chapter we manually encoded the protocol description in terms of guarded commands or, as we will see for the contract signing protocols, we use dedicated C++ programs. The tool presented previously does not easily apply to this kind of protocols in its current state. Group protocols are parametric systems, which cannot be verified by the model-checker Mocha. Therefore, we only analyse instances with a fixed number of participants. During this chapter, we are going to study three protocols: one non-repudiation protocol and two contract signing protocols. The study of the non-repudiation protocol was carried out in collaboration with Jean-François Raskin [90], while the contract signing protocols were studied in collaboration with Rohit Chadha and Andre Scedrov.

In the non-repudiation protocol we show that fairness is compromised if race conditions are not handled correctly. In the first contract signing protocol, we present an attack that compromises abuse-freeness, even for the case of three participants, as well as several attacks on fairness, when the number of participants is at least four. When studying the second contract signing protocol, we did not find any attack on fairness. However, we show how abuse-freeness could be compromised in an implementation aiming at efficiency. In the present chapter, we do only describe the analysis of the properties for which we found interesting attacks or weaknesses. We do not discuss viability or timeliness, which were also studied, but are respected in the protocols considered here, whenever fairness holds. These properties have been intensively discussed for two-party protocols in the previous chapter.

7.1 The Markowitch-Kremer multi-party non-repudiation protocol

We start studying the multi-party non-repudiation protocol, which we presented in collaboration with Olivier Markowitch [86, 98]. In this protocol we have one originator, Alice, who sends a message to several recipients. We studied the protocol with two recipients. In the original papers, we discuss a problem that could arise if the clocks of Alice and the TTP are not synchronised and propose a solution to this synchronization problem. Using Mocha we show that the protocol is indeed unfair for Alice if the synchronization problem is not handled correctly, as proposed in the original paper.

7.1.1 Protocol description

To describe the protocol we have to add the following notations. The remaining notations are similar to the ZG and KM two-party protocols, discussed in chapter 6.

- $\mathcal{B}$: the set of receiving entities.
- $\mathcal{B}'$: the set of receiving entities having sent an evidence of receipt for the cipher to Alice,
- $\ell = h(m,k)$: a label that, in conjunction with the identity A of the sender, uniquely identifies a protocol run.

The generated evidences are the following.

- $\mathsf{EOO} = S_A(f_{\mathsf{EOO}}, \mathcal{B}, \ell, t, h(c))$
- $\mathsf{EOR}_i = S_{B_i}(f_{\mathsf{EOR}}, A, \ell, t, h(c))$
- $\mathsf{Sub} = S_A(f_{\mathsf{Sub}}, \mathcal{B}, \ell, E_{\mathsf{TTP}}(k))$
- $\mathsf{EOO}_k = S_A(f_{\mathsf{EOO}_k}, \mathcal{B}', \ell, h(k))$
- $\mathsf{EOR}_{i,k} = S_{B_i}(f_{\mathsf{EOR}_k}, A, \ell, h(k))$
- $\mathsf{Rec}_X = S_X(f_{\mathsf{Rec}_X}, A, \mathcal{B}, \ell)$
- $\mathsf{Con}_k = S_{\mathsf{TTP}}(f_{\mathsf{Con}_k}, A, \mathcal{B}', \ell, h(k))$
- $\mathsf{Early} = S_{\mathsf{TTP}}(f_{\mathsf{Early}}, \ell)$
- $\mathsf{Set} = S_A(f_{\mathsf{Set}}, \mathcal{B}', \ell)$

Protocol 7.1 MK multi-party non-repudiation protocol—Main

1. $\mathsf{A} \Rightarrow \mathcal{B}$: $f_{\mathsf{EOO}}, f_{\mathsf{Sub}}, \mathcal{B}, \ell, t, c, E_{\mathsf{TTP}}(k), \mathsf{EOO}, \mathsf{Sub}$
2. $\mathsf{B_i} \rightarrow \mathsf{A}$: $f_{\mathsf{EOR}}, A, \ell, \mathsf{EOR}_i$

   where $\mathsf{B_i} \in \mathcal{B}$, $i \in \{1, \ldots, |\mathcal{B}|\}$ and $\mathsf{B_i} \neq \mathsf{B_j}$ if $i \neq j$
3. $\mathsf{A} \Rightarrow \mathcal{B}'$: $f_{\mathsf{EOO}_k}, \mathcal{B}', \ell, \mathcal{E}_{\mathcal{B}'}(k), \mathsf{EOO}_k$
4. $\mathsf{B'_j} \rightarrow \mathsf{A}$: $f_{\mathsf{EOR}_k}, A, \ell, \mathsf{EOR}_{j,k}$

   where $\mathsf{B'_j} \in \mathcal{B}'$, $j \in \{1, \ldots, |\mathcal{B}'|\}$ and $\mathsf{B_j} \neq \mathsf{B_k}$ if $j \neq k$

Main protocol. In the main protocol (protocol 7.1) Alice sends a cipher of the message to all potential receivers $\mathcal{B}$. Several of these receivers $\mathcal{B}' \subseteq \mathcal{B}$ are sending a receipt for this cipher. Alice continues the protocol with the receivers in $\mathcal{B}'$ by sending them the key corresponding to the cipher of message 1.

The first message destined to all receivers in $\mathcal{B}$ includes the label, the cipher, as well as the key k ciphered using the public key of the TTP (this information is used by the TTP in the case of a recovery). Alice also sends a time-out t: a recovery may only be initiated after t. If one of the receivers does not accept the time-out, he may stop the protocol.

After having sent the cipher in message 1, Alice decides of the moment when to continue the protocol. All receipts arriving after this moment are not considered anymore, without any risk for the receivers of losing fairness.

Afterwards, when Alice sends the deciphering key, it is crucial that only the recipients in $\mathcal{B}'$ receive it. Therefore we need to use a cipher (this is not needed in an optimistic two-party protocol). In order to cipher only once and to use multicasting, Alice uses a group encryption scheme, e.g. the one described in [48].

Recovery protocol. At each moment during the main protocol Alice and Bob have the possibility to launch the recovery protocol (protocol 7.2) with the TTP. The recipients in $\mathcal{B}'$ launch a recovery if Alice does not multicast the ciphered key quickly enough. Alice initiates the recovery protocol, if not all recipients in $\mathcal{B}'$ send a receipt for the ciphered key in due time.

Before t (t is specified by Alice during the first message in the main protocol and it is resent together with Alice's signature during the recovery request) the recovery cannot be initiated. If someone tries to execute a recovery before t, which is resent together with Alice's signature during the recovery request, the TTP sends a message to notify that the recovery cannot be initiated yet.

When the recovery protocol is initiated the first time (if any) after t, the TTP uses an FTP get operation to fetch the set $\mathcal{B}'$ at Alice. For this purpose Alice maintains a read-only publicly accessible directory containing the set of users in $\mathcal{B}'$ and her signature on this set. If the directory is not accessible the TTP supposes $\mathcal{B}' = \emptyset$.

Now the TTP sends to Alice the confirmation of receipt of the key that may be used to substitute $\mathsf{EOR}_{i,k}$ for all i such that $B_i \in \mathcal{B}'$. The TTP sends to each receiver $B_i \in \mathcal{B}'$

Protocol 7.2 MK multi-party non-repudiation protocol—Recovery

1. X $\rightarrow$ TTP: $f_{\mathsf{Rec}_X}, f_{\mathsf{Sub}}, A, \mathcal{B}, \ell, t, h(c), E_{\mathsf{TTP}}(k), \mathsf{Rec}_X, \mathsf{Sub}, \mathsf{EOO}$

   if recovered then

2. TTP $\rightarrow$ X: $f_{\mathsf{Con}_k}, A, \mathcal{B}', \ell, \mathcal{E}_{\mathcal{B}'}(k, S_{\mathsf{TTP}}(k)), \mathsf{Con}_k$

   else if before t then

3. TTP $\rightarrow$ X: $f_{\mathsf{Early}}, \mathsf{Early}$

   else

   recovered:=true

4. TTP $\leftrightarrow$ A: $f_{\mathsf{Set}}, \mathcal{B}', \ell, \mathsf{Set}$
5. TTP $\rightarrow$ A: $f_{\mathsf{Con}_k}, A, \mathcal{B}', l, \mathsf{Con}_k$
6. TTP $\Rightarrow$ $\mathcal{B}' \cup \{\mathsf{X}\} \backslash \{\mathsf{A}\}$: $f_{\mathsf{Con}_k}, A, \mathcal{B}', \ell, \mathcal{E}_{\mathcal{B}'}(k, S_{\mathsf{TTP}}(k)), \mathsf{Con}_k$

the confirmation of the key, that substitutes EOO_k, as well as the signed key ciphered for $\mathcal{B}'$ using a group encryption scheme.

If a receiver $B_i \notin \mathcal{B}'$ wants to perform a recovery, after the recovery has already been performed for the first time, the TTP uniquely identifies each protocol run by ℓ and A, the TTP sends to this entity the same message as to each $B_i \in \mathcal{B}'$. This message is however useless as k has been ciphered for the set $\mathcal{B}'$ and only informs the recipient that he does not belong to this set.

Note that, after successfully finishing the protocol, a participant $B \in \mathcal{B}'$ could send the message m to any participant $\widetilde{B} \in \mathcal{B} \setminus \mathcal{B}'$. However, this behaviour does not violate fairness as participant $\widetilde{B}$ could not get a non-repudiation of origin evidence (the recipients are specified in this evidence). Such an "attack" would break confidentiality of the message. However, it is impossible to provide confidentiality when participants finishing correctly the protocol do collude with participants stopping the protocol early.

7.1.2 Protocol analysis

To perform the analysis we manually encoded the guarded commands modelling the protocol as well as the model checking scripts. The participants and communication channels are encoded in a similar way than it is done by our tool. This means that a malicious participant can send messages as soon as he can construct them and he can spy other communication channels. The analysis has been performed while fixing the number of recipients to two.

Using MOCHA we were able to show that the above described version of the protocol contains a subtle flaw, due to a possible shift between the TTP's clock and Alice's clock, associated to a race in the transmissions. In order to detect this attack, we verify that both B_1 and B_2 can collaborate to break fairness for Alice. Without loss of generality we suppose that it is B_1, who gets the advantage. The ATL-formula we verify is thus the following.

$$\langle\langle B_1, B_2, \mathcal{C}om, \mathsf{TimeOut}\rangle\rangle\Diamond(\mathsf{NRO}_1 \wedge \neg\langle\langle \mathsf{A}\rangle\rangle\Diamond\mathsf{NRR}_1)$$

NRO_1 and NRR_1 represent the complete non-repudiation of origin, respectively receipt, evidences related to B_1. TimeOut is a player that controls the clocks of Alice and the TTP, and has to respect fairness conditions on the triggering of the time-outs.

The flawed scenario is as follows. Suppose that the clock of the TTP indicates that the moment t arrived whereas it is not yet the case on the clock of Alice. B_1 can initiate the recovery protocol without having sent the second message of the main protocol. The TTP will consult Alice's public directory and will get a set $\mathcal{B}'$ not containing B_1. If B_1 sends the second message of the main protocol before the moment t on the clock of Alice and before the third message of the recovery protocol, sent by the TTP, arrives to Alice, Alice will include B_1 in $\mathcal{B}'$. Hence, $\mathcal{B}'$ will be different from the one obtained by the TTP. If Alice sends the third message of the main protocol, before receiving the message sent by the TTP, B_1 possesses the message m and the non-repudiation of origin evidence. If B_1 does not send the last message of the main protocol, Alice will receive from the TTP a non-repudiation of receipt evidence for a set of recipients that only contains B_2. Hence, fairness is broken for Alice.

In order to avoid this flaw, Alice must stop taking part in the main protocol as soon as the TTP consults her public directory. Using this additional constraint (hence, modifying the commands of Alice), we did not detect any other flaw.

7.2 The Garay-MacKenzie multi-party contract signing protocol

In 1999, Garay and MacKenzie [68] proposed a multi-party contract signing protocol which, in addition to fairness, would respect abuse-freeness. A first version, which only handles three participants, has already been partially[1] presented by Garay, Jakobsson and MacKenzie [67] when introducing their two-party protocol and the concept of abuse-freeness. As we will see the protocol is rather complicated, which emphasises the need for a formal verification. And the fact is that we succeeded in finding attacks which break both abuse-freeness and fairness.

7.2.1 Protocol description

The protocol presented by Garay and MacKenzie in [68] can be described recursively. For i participants P_1 through P_i to sign a contract m, P_i indicates its willingness to sign the contract to P_1 through P_{i-1}. Now, participants P_1 through P_{i-1} come to an agreement about the contract (with promises, not signatures), letting P_i know about this (again with promises). Then all participants exchange promises to sign the contract. Only when all recursive levels are finished, the participants exchange their signatures on a contract.

To apply this recursive approach, different "strengths" of promises are used, one for each level of recursion. We denote the strength of a promise by an integer "level". Specifically, an "i-level promise from A to B on a message m" is implemented using PCS: $PCS_A((m,i), B, \mathsf{TTP})$.

[1] The abort and recovery protocols were not described in detail.

If there is no misbehaviour by any participant, there is never a need to contact the trusted party TTP, which means that the protocol follows the optimistic approach. If the TTP is contacted through either an abort or a recovery protocol, it needs to store some information to handle the abort and/or resolve messages it may receive. Specifically, if there is a positive resolution, the TTP needs to store a completely signed contract (with the promises possibly converted to universally verifiable signatures by the TTP itself). If there has not been a positive resolution, the TTP needs to store an "abort list" containing indices of all the parties that have aborted, or tried to resolve unsuccessfully. The TTP will also store the first abort message it received, along with another "forced abort list" that will be explained in the protocol below. The protocol is composed of three sub-protocols: a main, an abort and a recovery protocol. A major difference with two-party contract signing protocols and also with the previously described multi-party non-repudiation protocol is that the TTP can overturn his abort decision and recover a previously aborted protocol if the TTP is convinced that the abort has only been sent to dishonest parties. The possibility of overturning an abort is needed to avoid the following kind of scenario. A dishonest party aborts the protocol and continues the optimistic execution until a honest party has sent out his signature. At that moment all the other participants stop the protocol and the honest participant must contact the TTP. If the TTP does not overturn the abort, fairness would be broken. In the two-party case this problem is solved by letting only Alice abort, who is the first one to send out her signature. However, this solution cannot apply to a multi-party case, as several parties must send out their signatures before receiving the other participants signatures. In [68], Garay and MacKenzie give the sketch of an inductive proof showing that overturning abort decisions is unavoidable.

Main protocol. A detailed description of the main protocol is given in protocol 7.3.

At the beginning, P_i waits for all higher participants, i.e. all participants having a higher index, to send their 1-level promise, which confirms their willingness to start the protocol. Hence, P_n, where n is the number of participants, will be the first one to send out a message. Then P_i sends his 1-level promise to all lower participants.

After each participant has sent his 1-level promises to all lower participants, P_i waits for the lower level protocols to finish. Therefore, he must receive an $(i-1)$-level promise from all the lower participants. These promises trigger the i-level protocol to start. Then, P_i sends out his i-level promise to the lower participants and waits for their i-level promises to arrive. When these promises arrive, P_i also sends an i-level promise to P_{i+1} in order to start the $i+1$-level protocol.

More generally, the remaining of the protocol works as follows. P_i sends an $(a-1)$-level promise to P_a, where a is initialised to $i+1$. Now, P_i waits for all higher participants up to a to send their a-level promise. Then P_i sends his a-level promise out to all the lower participants and waits to receive their a-level promises. P_i goes on sending his a-level promise to all higher participants up to a and increases a by 1. This way of acting goes on until a reaches $n+1$. At that point the protocol is slightly changed. In the $(n+1)$-level protocol, signatures on the contract are sent out together with the $(n+1)$-level promises. Moreover, once the promises on the signatures are received from all higher participants, P_i directly sends his $n+1$-level promises and signatures to *all* participants and not only to the lower ones as before. Then P_i waits for the lower participants' promises and signatures in order to end the protocol.

As usually, when some expected messages are not received, P_i may either stop the protocol or contact the TTP, depending on how far the main protocol has already been executed. At the beginning, before having sent out any message, P_i may simply quit the protocol if a message is not received in a timely way. Once he has sent out messages, but has not yet received at least one message from each player, P_i may abort the protocol. Once P_i received at least one message from every other participant, he can launch the recovery protocol.

Abort protocol. In order to take decisions when an abort or recovery request arrives, the TTP maintains two sets S_m and F_m for each contract text m for which he has been contacted. When a first request, with respect to a contract text m arrives, the TTP creates the two sets and initialises them to the empty set. The set S_m contains the indexes of all participants that have received an abort token from the TTP. The intuitive meaning of the set F_m is not clearly stated in [68], but it contains some additional information that should help the TTP in deciding when to overturn an abort.

The details of the abort protocol are given in protocol 7.4. Mainly, if the TTP is contacted for the first time or if the protocol has not already been recovered, the TTP sends back an abort token and updates the sets S_m and F_m. If the protocol has already been successfully recovered, the TTP sends back a signed contract.

Recovery protocol. The details of the recovery protocol are given in protocol 7.5. For P_i to recover, it sends the message

$$S_{P_i}(\{PCS_{P_j}((m, k_j), P_i, \mathsf{TTP})\}_{j \in \{1,\ldots n\} \setminus \{i\}}, S_{P_i}((m, 1)))$$

where

- if $j > i$, k_j is the maximum level of a promise received from P_j on m,
- if $j < i$, k_j is the maximum level of promises received from all participants $P_{j'}$, with $j' < i$, i.e. the min-max of the level of promises from participants with lower index. (For example, if the maximum level of the promises received by P_4 from P_3 and P_2 was 6, and the maximum level received by P_4 from P_1 was 5, then it would send the 5-level promises for P_1, P_2 and P_3.)

The idea of the recovery protocol can be summarised as follows. If the TTP is contacted for the first time with respect to this contract, or if the protocol has already been resolved, the TTP replies with a signed contract which it obtains by converting the PCS given in the promises. Otherwise, if the protocol has already been aborted, the TTP must decide whether to maintain the abort or to overturn it. The decision is based on the contents of the sets S_m and F_m, as described in protocol 7.5.

7.2.2 Analysis

The protocol has several differences with the previously studied one. The most notable difference is that the protocol changes with the number of participants, e.g. the description

Protocol 7.3 GM multi-party contract signing protocol—Main

Wait for all higher recursive levels to start

1. $P_j \rightarrow P_i$: $PCS_{P_j}((m,1), P_i, T)$ $(n \geq j > i)$

   If P_i does not receive 1-level promises from $P_n \ldots P_{i+1}$ in a timely manner, P_i simply quits.

Start recursive level i

2. $P_i \rightarrow P_j$: $PCS_{P_i}((m,1), P_j, T)$ $(i > j \geq 1)$

Wait for recursive level i-1 to finish

3. $P_j \rightarrow P_i$: $PCS_{P_j}((m,i-1), P_i, T)$ $(i > j \geq 1)$

   If P_i does not receive (i-1)-level promises from $P_{i-1} \ldots P_1$ in a timely manner, P_i aborts.

Send i-level promises to all lower parties

4. $P_i \rightarrow P_j$: $PCS_{P_i}((m,i), P_j, T)$ $(i > j \geq 1)$

Finish recursive level i when i-level promises are received

5. $P_j \rightarrow P_i$: $PCS_{P_j}((m,i), P_i, T)$ $(i > j \geq 1)$

   If P_i does not receive i-level promises from $P_{i-1} \ldots P_1$ in a timely manner, P_i recovers.

Complete all higher recursive levels

For $a = i+1$ to n, P_i does the following:

6.1. $P_i \rightarrow P_a$: $PCS_{P_i}((m,a-1), P_a, T)$

6.2. $P_j \rightarrow P_i$: $PCS_{P_i}((m,a), P_j, T)$ $(a \geq j > i)$

   If P_i does not receive a-level promises from $P_a \ldots P_{i+1}$ in a timely manner, P_i recovers.

6.3. $P_i \rightarrow P_j$: $PCS_{P_i}((m,a), P_j, T)$ $(i > j \geq 1)$

6.4. $P_j \rightarrow P_i$: $PCS_{P_i}((m,a), P_j, T)$ $(i > j \geq 1)$

   If P_i does not receive a-level promises from $P_{i-1} \ldots P_1$ in a timely manner, P_i recovers.

6.5. $P_i \rightarrow P_j$: $PCS_{P_i}((m,a), P_j, T)$ $(a \geq j > i)$

Wait for signatures and (n+1)-level promises from higher participants

7. $P_j \rightarrow P_i$: $PCS_{P_j}((m,n+1), P_i, T), S_{P_j}(m,1)$ $(n \geq j > i)$

   If P_i does not receive signatures and (n+1)-level promises from $P_n \ldots P_{i+1}$ in a timely manner, P_i recovers.

Send signatures and (n+1)-level promises to participants

8. $P_i \rightarrow P_j$: $PCS_{P_i}((m,1), P_j, T), S_{P_i}(m,1)$ $(j \neq i)$

9. $P_j \rightarrow P_i$: $PCS_{P_j}((m,n+1), P_i, T), S_{P_j}(m,1)$ $(i > j \geq 1)$

   If P_i does not receive signatures and (n+1)-level promises from $P_{i-1} \ldots P_1$ in a timely manner, P_i recovers.

Protocol 7.4 GM multi-party contract signing protocol—Abort

1. $P_i \rightarrow$ TTP: $S_{P_i}(m, P_i, (P_1, \ldots, P_n), abort)$

   if not validated(m) then

   if $S_m = \emptyset$, TTP stores $S_{\mathsf{TTP}}(S_{P_i}(m, P_i, (P_1, \ldots, P_n), abort))$; $S_m = S_m \cup \{i\}$; if i is larger than the maximum index in S_m, TTP clears F_m

2. TTP $\rightarrow P_i$: $S_T(S_{P_j}(m, P_j, (P_1, \ldots, P_n), abort), S_T(m, S_m, abort))$

   else (validated(m)=false)

3. TTP $\rightarrow P_i$: $\{S_{P_j}((m, k_j))\}_{j \in \{1,\ldots,n\} \setminus \{i\}}$

   where k_j is the level of the promise from P_j that was converted to a universally-verifiable signature during the recovery protocol.

of the first participant is different when the value of n changes. In fact, the number of messages in the protocol increases considerably with the number of participants. For instance, if we have $n = 3$, the main protocol is composed of 20 messages and the protocol proposes 14 different recovery requests. When $n = 4$, we have that the number of messages in the main protocol is 41 and the number of different recovery request is 36. Moreover, the protocol is not symmetric in the sense that the protocol for P_i is different from the protocol for P_j, for all $i \neq j$. For instance, when $n = 4$, P_1 can launch 18 different recovery requests and P_4 only 2.

Due to the high number of messages, we only analyse the *structure* of the protocol. We do not explicitly model any cryptographic primitives, but only the fact that protocol messages can be sent out of order. Hence, a dishonest party can send messages out of order and continue the protocol, even if it is supposed to stop. We manually encode which messages must be known in order to send some other message. Moreover, we did not model the possibility to spy other channels.

Again for efficiency reasons, we model all communication channels to be operational, i.e. receiving messages is possible as soon as they have been sent. Therefore, we can directly model communication through shared variables and avoid to define players representing explicitly the communication channels. As the channels are operational, fairness constraints are not needed. For honest players, we do not use fairness constraints neither, as we did previously to ensure that they cannot *block* the protocol by choosing the idle action forever. However, as we will see below, we slightly adapt the formulae when verifying the properties.

As we are unable to verify parametric systems with Mocha, we simplify our task and verify the protocol only for a given number n of participants. In order to avoid encoding each instance of the protocol using guarded commands, we have written a dedicated C++ program that takes the number n of participants as a parameter and generates the protocol specification.

Although our model is simplified with regard to several aspects, the model seems to be of interest. As already explained in section 5.1.1, fair exchange protocols are generally

Protocol 7.5 GM multi-party contract signing protocol—Recovery

1. $P_i \rightarrow$ TTP: $S_{P_i}(\{PCS_{P_j}((m,k_j),P_i,\mathsf{TTP})\}_{j\in\{1,\ldots n\}\setminus\{i\}}, S_{P_i}((m,1)))$

   if $i \in S_m$, TTP ignores the message

   else if validated(m)

2. TTP $\rightarrow P_i$: $\{S_{P_j}((m,k_j))\}_{j\in\{1,\ldots n\}\setminus\{i\}}$

   where k_j is the level of the promise from P_j that was converted to a universally-verifiable signature.

   else if $S_m = \emptyset$

   validated(m):=true

3. TTP $\rightarrow P_i$: $\{S_{P_j}((m,k_j))\}_{j\in\{1,\ldots n\}\setminus\{i\}}$

   else (validated(m)=false $\wedge$ $S_m \neq \emptyset$)

   a. if $i \notin F_m$, then

      (i) if for any $\ell \in S_m$ there is a $j \in S_m$ such that $j > k_\ell$
      $S_m := S_m \cup \{i\}$
      let a be the maximum value in S_m
      if $a > i$ then $\forall j$, such that $k_j = a - 1 \cdot\ F_m := F_m \cup \{j\}$
      else $F_m := \emptyset$

      4. TTP $\rightarrow P_i$: $S_T(S_{P_j}(m,P_j,(P_1,\ldots,P_n),abort),S_T(m,S_m,abort))$
      where $S_T(S_{P_j}(m,P_j,(P_1,\ldots,P_n),abort))$ corresponds to the stored abort token

      (ii) else
      validated(m):=true

      5. TTP $\rightarrow P_i$: $\{S_{P_j}((m,k_j))\}_{j\in\{1,\ldots n\}\setminus\{i\}}$

   b. else $(i \in F_m)$

      let a be the maximum value in S_m

      (i) if $(\forall j$, such that $i < j \leq a \cdot\ k_j < a) \wedge (\forall j < i \cdot\ k_j \geq a)$
      validated(m):=true

      6. TTP $\rightarrow P_i$: $\{S_{P_j}((m,k_j))\}_{j\in\{1,\ldots n\}\setminus\{i\}}$

      (ii) else
      $S_m := S_m \cup \{i\}$
      if $a > i$ then $\forall j, k_j = a - 1 \cdot\ F_m := F_m \cup \{j\}$
      if $a = i$ then $F_m := \emptyset$

      7. TTP $\rightarrow P_i$: $S_T(S_{P_j}(m,P_j,(P_1,\ldots,P_n),abort),S_T(m,S_m,abort))$
      where $S_T(S_{P_j}(m,P_j,(P_1,\ldots,P_n),abort))$ corresponds to the stored abort token

well protected at the cryptographic level, while concurrency issues are at the origin of most flaws. When analysing the GM multi-party contract signing protocol, we were able to discover an attack on abuse-freeness, when $n = 3$ and several attacks on fairness when $n = 4$. All these attacks are novel. The protocol was published in 1999 and believed secure during four years.

7.2.2.1 Abuse-freeness

To show that the protocol does not respect abuse-freeness, consider a three party instance of the protocol with participants P_1, P_2 and P_3. We will assume that P_3 is honest and P_2 and P_1 are dishonest and collude in order to convince an outsider, Charlie, that they have the power to choose the outcome of the protocol. In particular, if P_2 and P_1 have enough information, they may contact the TTP at points where the protocol specification does not allow them to do so.

Suppose that P_3 starts by sending out his 1-level promises to P_1 and P_2. Upon receiving this promise (or even before), P_2 aborts the protocol, but continues to run the execution of the main protocol. On receiving P_3's 1-level promise, P_2 sends out his 1-level promise to P_1. Now, P_1 has received 1-level promises from P_3 and P_2. Using these first level promises, P_1 sends a recovery request to the TTP. Note that, in the protocol, P_1 is never allowed to abort and the TTP would not accept an abort request from P_1. Due to P_2's initial abort request, P_1's recovery request will be refused and the TTP sends

$$S_{\mathsf{TTP}}(S_{P_2}(m, P_2, (P_1, P_2, P_3), abort))$$

and

$$S_{\mathsf{TTP}}(m, S_m = \{1, 2\}, abort)$$

to P_1. At this point we can make the following observations:

- the abort reply contains the set $S_m = \{1, 2\}$ and is different from the one P_2 received,
- if P_1 receives an abort reply from the TTP, it is always the answer to a recovery request.
- a recovery request always includes a promise from each participant, and the TTP verifies these promises.

From these remarks we can conclude that if P_1 is showing its abort reply to an external observer called Charlie, Charlie will be convinced that P_3 has started the protocol even though Charlie is not able to verify the private contract signature from P_3. In other words, we can say that the TTP has verified the private contract signature for Charlie. In a two-party protocol the abort token is never interpreted as a proof for Charlie, as Alice can generate it without Bob having started the protocol, and Bob can never successfully finish a protocol when he received an abort reply and Alice is honest.

Now, we only need to show that P_1 and P_2 do have the power to determine the outcome of the protocol. In order to choose abort, P_1 and P_2 stop the protocol. P_3 is not able to overturn their abort, as there exists a honest execution for both P_1 and P_2 leading to the given situation. Otherwise, if their abort could be overturned, the protocol would be unfair even for three participants, i.e. the abort of a honest participant would be overturned. To choose a successful outcome, P_1 and P_2 continue the protocol in an optimistic way, i.e. without the TTP's intervention.

The flaw is detected by Mocha when checking the following formula.

$$\begin{array}{ll}\exists\Diamond(\langle\langle P_1, P_2\rangle\rangle\Diamond(& \mathsf{TTP}.\,send(abort)\ to\ P_1 \wedge \\ & \neg\langle\langle P_{3h}\rangle\rangle\Diamond(P_3.S_{P_1}(m) \wedge P_3.S_{P_2}(m)) \wedge \\ & \langle\langle P_1, P_2\rangle\rangle\Diamond(P_3.stop \rightarrow (P_1.S_{P_3}(m) \wedge P_2.S_{P_3}(m)))))\end{array}$$

The formula has been adapted, as in our simplified model, no fairness constraints are used. Moreover, we restricted the predicate prove2C to hold when the boolean variable $\mathsf{TTP}.\,send(abort)\ to\ P_1$ holds. The variables $P_i.S_{P_j}(m)$ reflect that player i has received player j's signature on the contract. More precisely, the formula requires that it is possible to reach a state where players P_1 and P_2 have a strategy to prove to Charlie that

1. the protocol was started by P_3,
2. they have a strategy to choose an unsuccessful outcome, i.e. P_3 cannot get all the other players' signatures,
3. they have a strategy to choose a successful outcome, i.e. when the honest player stops, the dishonest players obtain the honest player's signature.

Due to the differences in the model, we used a weaker formula than the one defined in the two-party case. However, the above described scenario shows that the attack on abuse-freeness is an attack in the sense of the previously given modelling.

The above described vulnerability can easily be addressed by not including the set S_m in the abort reply. In that case, the replies P_2 and P_1 receive are identical. As P_2 can abort a protocol that has not been started by P_3, the abort reply does not prove P_3's participation to Charlie. In [69], Garay and MacKenzie propose a different fix, which consists in letting P_1 abort the protocol, rather then simply quitting. Both fixes, change the prove2C predicate in the same way. With the new definition of prove2C, we did not find any additional attack with Mocha.

7.2.2.2 Fairness

When verifying fairness for the protocol with $n = 3$, we did not discover any flaw. The fairness requirement for P_1 is expressed by the following formula.

$$\forall\Box((P_2.S_{P_1}(m) \vee P_3.S_{P_1}(m)) \rightarrow \exists\Diamond(P_1.S_{P_2}(m) \wedge P_1.S_{P_3}(m))$$

This formula, as well as the similar formulae representing fairness for P_2 and P_3, are validated by Mocha. We use here the CTL formulae, rather than the ATL formulae in order to verify the stronger requirement for fairness. However, we do not model cryptographic primitives. Hence, we can only conclude that the protocol does not contain any *structural* weakness for $n = 3$. Indeed, when relaxing the assumption on the confidentiality of the channels, the attack presented by Shmatikov and Mitchell in [147] on the two-party GJM protocol can be adapted to the multi-party version. We only need a malicious participant to abort and stop the protocol. The honest party would send a recovery, containing his signature on the contract, but should not be able to overturn the abort. If the malicious party can eavesdrop the signature, the honest participant is flawed. With our present modelling, we do not find such flaws, as this requires to eavesdrop channels and to decompose messages. However, the fix proposed by Shmatikov and Mitchell directly applies to the multi-party protocol too. In the remaining of our analysis, we simply suppose that the channels are confidential.

When setting $n = 4$, on the contrary, we discovered several weaknesses. The first attack was discovered by hand, when finding an error in the correctness proof given in the original paper [68]. The discovered flaw encouraged us to make a detailed analysis and, using Mocha, we discovered seven additional and different attacks. We start by giving a detailed description of the first flaw we discovered.

For our attack on fairness to succeed, we need at least four participants, that we denote P_1, P_2, P_3 and P_4. We shall assume that P_2 is honest and the rest of the participants are dishonest and are colluding. This scenario succeeds in letting the TTP send an abort message to P_2 and afterwards, a resolve message to P_4. We shall first describe the scenario in broad terms and then give a detailed description. Roughly speaking, the scenario consists of the following sequence of events.

1. P_3 contacts the TTP at the beginning of the protocol and gets an abort token. P_3 gets added to S_m.
2. At some later point P_1 contacts the TTP, gets an abort token and gets added to S_m. At this point P_1 manages to make the TTP add P_2 to the list F_m.
3. P_2 contacts the TTP at step 6.2. when $a = 4$. At this point P_2 is trying to complete the fourth level protocol and the tests in the protocol indicate that P_2 must receive an abort token in response.
4. P_4 contacts the TTP and gets the abort overturned.

Let us now discuss this scenario in detail.

At the beginning of the protocol, P_3 aborts the protocol and the TTP updates $S_m = \{3\}$. However, unlike dictated by the protocol, dishonest P_3 continues the main protocol execution.

As soon as P_1 has received the second level promise from P_2, he asks the TTP for a resolve by sending

$$S_{P_1}(\{PCS_{P_2}((m,2),P_1,\mathsf{TTP}),PCS_{P_3}((m,1),P_1,\mathsf{TTP}),PCS_{P_4}((m,1),P_1,\mathsf{TTP})\},S_{P_1}((m,1))).$$

The TTP refuses this resolve request, answers with an abort message and updates $S_m = \{1,3\}$ and $F_m = \{2\}$. As P_3 did before, P_1 also continues the protocol.

The main protocol is executed normally until participant P_2 reaches point 6.2. of the protocol with $a = 4$. At that point P_2 has sent out the set of messages

$$\{PCS_{P_2}((m,1),P_1,\mathsf{TTP}),PCS_{P_2}((m,2),P_1,\mathsf{TTP}),PCS_{P_2}((m,2),P_3,\mathsf{TTP}),$$
$$PCS_{P_2}((m,3),P_1,\mathsf{TTP}),PCS_{P_2}((m,3),P_3,\mathsf{TTP}),PCS_{P_2}((m,3),P_4,\mathsf{TTP})\}$$

and has received the set of messages

$$\{PCS_{P_4}((m,1),P_2,\mathsf{TTP}),PCS_{P_3}((m,1),P_2,\mathsf{TTP}),PCS_{P_1}((m,1),P_2,\mathsf{TTP}),\\ PCS_{P_1}((m,2),P_2,\mathsf{TTP}),PCS_{P_3}((m,3),P_2,\mathsf{TTP}),PCS_{P_1}((m,3),P_2,\mathsf{TTP})\}.$$

P_2 is at position 6.2. with $a = 4$ and is waiting for 4-level promises from P_3 and P_4. As P_3 and P_4, who are dishonest, do not reply, P_2 sends the following resolve request to the TTP.

$$S_{P_2}(\{PCS_{P_2}((m,3),P_2,\mathsf{TTP}),PCS_{P_3}((m,3),P_2,\mathsf{TTP}),PCS_{P_4}((m,1),P_2,\mathsf{TTP})\},S_{P_2}((m,1))).$$

As P_2 is in F_m, the tests in the protocol description indicate that the TTP refuses the request, updates $S_m = \{1,2,3\}$ and replies with an abort message.

Now P_4 can launch a resolve request, sending

$$S_{P_4}(\{PCS_{P_1}((m,3),P_4,\mathsf{TTP}),PCS_{P_2}((m,3),P_4,\mathsf{TTP}),PCS_{P_3}((m,3),P_4,\mathsf{TTP})\},S_{P_4}((m,1))).$$

This request will overturn the previous aborts, and hence violates fairness as the TTP sends the signed contract back to P_4.

There exist seven additional attacks which exploit either the same error in the "correctness proof" or exploit the fact that there exists a case which has not been discussed in the proof. Namely, the authors did not consider in their proof the last step when the participants are waiting for the $(n+1)$-level promises. All the attacks we detected follow the very general scenario given hereunder.

1. A dishonest participant aborts the protocol.
2. Another dishonest participant tries to recover, but does not succeed. However, he changes the sets S_m and F_m.
3. The honest participant tries to recover, but is not successful in getting the abort decision overturned.
4. The remaining dishonest participant contacts the TTP and achieves the overturn.

Using Mocha we have shown that the protocol is unfair for participants P_1 through P_3. However, it seems that no attack exists against P_4. Checking fairness for participants P_1 to P_3 using a similar formula as in the case where $n = 3$, but adapted to the 4 player protocol, only tells us the existence of an attack. To get more information about the scenario which is compromising fairness we may alter the formula. For instance, for the attack described above, we can check the following formula.

$$\exists\Diamond(P_4.S_{P_1}(m) \wedge P_{2h}.\,rec(Abort))$$

The fact that this formula is validated indicates that it is possible for P_4 to get P_2's signature, while P_2 gets an abort token. Hence, we already know that it is P_4 who succeeds in overturning the abort. In a similar way, we can know that it is P_3 who sent the initial abort request and figure out the attack scenario.

We now briefly sketch each of the attack scenarios that we discovered.

Attacks against P_1. In a first attack against P_1, P_4 succeeds in getting P_1's signature on the contract, while P_1 does not get P_4's signature.

1. At the beginning P_3 aborts, but continues the protocol. We have that $S_m = \{3\}$ and $F_m = \emptyset$.
2. As soon as possible, P_2 recovers by sending

   $$S_{P_2}(\{PCS_{P_1}((m,2),P_2,\mathsf{TTP}),PCS_{P_3}((m,1),P_2,\mathsf{TTP}),PCS_{P_4}((m,1),P_2,\mathsf{TTP})\},S_{P_2}((m,1)))$$

   but receives an abort reply from the TTP. We have that $S_m = \{3,2\}$ and $F_m = \{1\}$.
3. P_1 is forced to recover by sending

   $$S_{P_1}(\{PCS_{P_2}((m,3),P_1,\mathsf{TTP}),PCS_{P_3}((m,3),P_1,\mathsf{TTP}),PCS_{P_4}((m,1),P_1,\mathsf{TTP})\},S_{P_1}((m,1)))$$

   but receives an abort reply from the TTP. We have that $S_m = \{3,2,1\}$ and $F_m = \{1\}$.
4. P_4 recovers by sending

   $$S_{P_4}(\{PCS_{P_1}((m,3),P_4,\mathsf{TTP}),PCS_{P_2}((m,3),P_4,\mathsf{TTP}),PCS_{P_3}((m,3),P_4,\mathsf{TTP})\},S_{P_4}((m,1)))$$

   and succeeds in overturning the abort decision.

In a second attack against P_1, it is P_2 who succeeds in receiving P_1's signature. Note that the attack is rather different from the previous one, as P_3 is added to F_m in this attack.

1. At the beginning P_4 aborts, but continues the protocol. We have that $S_m = \{4\}$ and $F_m = \emptyset$.
2. As soon as possible, P_2 recovers by sending

   $$S_{P_2}(\{PCS_{P_1}((m,3),P_2,\mathsf{TTP}),PCS_{P_3}((m,3),P_2,\mathsf{TTP}),PCS_{P_4}((m,1),P_2,\mathsf{TTP})\},S_{P_2}((m,1)))$$

   but receives an abort reply from the TTP. We have that $S_m = \{4,2\}$ and $F_m = \{3,1\}$.
3. P_1 is forced to recover by sending

   $$S_{P_1}(\{PCS_{P_2}((m,4),P_1,\mathsf{TTP}),PCS_{P_3}((m,4),P_1,\mathsf{TTP}),PCS_{P_4}((m,4),P_1,\mathsf{TTP})\},S_{P_1}((m,1)))$$

   but receives an abort reply from the TTP. We have that $S_m = \{4,2,1\}$ and $F_m = \{3,1\}$.
4. P_3 recovers by sending

   $$S_{P_3}(\{PCS_{P_1}((m,4),P_3,\mathsf{TTP}),PCS_{P_2}((m,4),P_3,\mathsf{TTP}),PCS_{P_4}((m,5),P_3,\mathsf{TTP})\},S_{P_3}((m,1)))$$

   and succeeds in overturning the abort decision.

There exists also a third attack against P_1. This time, it is P_2 who succeeds in getting P_1's signature.

1. At the beginning P_4 aborts, but continues the protocol. We have that $S_m = \{4\}$ and $F_m = \emptyset$.

2. As soon as possible, P_3 recovers by sending

$$S_{P_3}(\{PCS_{P_1}((m,3),P_3,\mathsf{TTP}),PCS_{P_2}((m,3),P_3,\mathsf{TTP}),PCS_{P_4}((m,3),P_3,\mathsf{TTP})\},S_{P_3}((m,1)))$$

but receives an abort reply from the TTP. We have that $S_m = \{4,3\}$ and $F_m = \{2,1\}$.

3. P_1 is forced to recover by sending

$$S_{P_1}(\{PCS_{P_2}((m,4),P_1,\mathsf{TTP}),PCS_{P_3}((m,4),P_1,\mathsf{TTP}),PCS_{P_4}((m,4),P_1,\mathsf{TTP})\},S_{P_1}((m,1)))$$

but receives an abort reply from the TTP. We have that $S_m = \{4,3,1\}$ and $F_m = \{2,1\}$.

4. P_2 recovers by sending

$$S_{P_2}(\{PCS_{P_1}((m,4),P_2,\mathsf{TTP}),PCS_{P_3}((m,5),P_2,\mathsf{TTP}),PCS_{P_4}((m,5),P_2,\mathsf{TTP})\},S_{P_2}((m,1)))$$

and succeeds in overturning the abort decision.

Attacks against P_2. In addition to the attack on P_2 which we described in details above, there exists an attack where P_3 receives P_2's signature on the contract.

1. At the beginning P_4 aborts, but continues the protocol. We have that $S_m = \{4\}$ and $F_m = \emptyset$.

2. As soon as possible, P_1 recovers by sending

$$S_{P_1}(\{PCS_{P_2}((m,2),P_1,\mathsf{TTP}),PCS_{P_3}((m,3),P_1,\mathsf{TTP}),PCS_{P_4}((m,1),P_1,\mathsf{TTP})\},S_{P_1}((m,1)))$$

but receives an abort reply from the TTP. We have that $S_m = \{4,1\}$ and $F_m = \{2\}$.

3. P_2 is forced to recover by sending

$$S_{P_2}(\{PCS_{P_1}((m,4),P_2,\mathsf{TTP}),PCS_{P_3}((m,4),P_2,\mathsf{TTP}),PCS_{P_4}((m,4),P_2,\mathsf{TTP})\},S_{P_2}((m,1)))$$

but receives an abort reply from the TTP. We have that $S_m = \{4,2,1\}$ and $F_m = \{2\}$.

4. P_3 recovers by sending

$$S_{P_3}(\{PCS_{P_1}((m,4),P_3,\mathsf{TTP}),PCS_{P_2}((m,4),P_3,\mathsf{TTP}),PCS_{P_4}((m,4),P_3,\mathsf{TTP})\},S_{P_3}((m,1)))$$

and succeeds in overturning the abort decision.

There exists a third attack against P_2. This attack is similar to the previous one, in the sense that it is P_3 who obtains P_2's signature. However, the "attack technique" slightly differs, since here P_3 is in F_m, while this is not the case in the previous attack.

1. At the beginning P_4 aborts, but continues the protocol. We have that $S_m = \{4\}$ and $F_m = \emptyset$.

2. As soon as possible, P_1 recovers by sending

$$S_{P_1}(\{PCS_{P_2}((m,3),P_1,\mathsf{TTP}),PCS_{P_3}((m,3),P_1,\mathsf{TTP}),PCS_{P_4}((m,1),P_1,\mathsf{TTP})\},S_{P_1}((m,1)))$$

but receives an abort reply from the TTP. We have that $S_m = \{4,1\}$ and $F_m = \{3,2\}$.

3. P_2 is forced to recover by sending

$$S_{P_2}(\{PCS_{P_1}((m,4),P_2,\mathsf{TTP}),PCS_{P_3}((m,4),P_2,\mathsf{TTP}),PCS_{P_4}((m,4),P_2,\mathsf{TTP})\},S_{P_2}((m,1)))$$

   but receives an abort reply from the TTP. We have that $S_m = \{4,2,1\}$ and $F_m = \{3,2\}$.

4. P_3 recovers by sending

$$S_{P_3}(\{PCS_{P_1}((m,4),P_3,\mathsf{TTP}),PCS_{P_2}((m,4),P_3,\mathsf{TTP}),PCS_{P_4}((m,5),P_3,\mathsf{TTP})\},S_{P_3}((m,1)))$$

   and succeeds in overturning the abort decision.

Attacks against P_3. We also discovered two attacks against P_3. In the first attack, P_2 is able to obtain P_3's signature on the contract test.

1. At the beginning P_4 aborts, but continues the protocol. We have that $S_m = \{4\}$ and $F_m = \emptyset$.

2. As soon as possible, P_1 recovers by sending

$$S_{P_1}(\{PCS_{P_2}((m,2),P_1,\mathsf{TTP}),PCS_{P_3}((m,3),P_1,\mathsf{TTP}),PCS_{P_4}((m,1),P_1,\mathsf{TTP})\},S_{P_1}((m,1)))$$

   but receives an abort reply from the TTP. We have that $S_m = \{4,1\}$ and $F_m = \{3\}$.

3. P_3 is forced to recover by sending

$$S_{P_3}(\{PCS_{P_1}((m,4),P_3,\mathsf{TTP}),PCS_{P_2}((m,4),P_3,\mathsf{TTP}),PCS_{P_4}((m,4),P_3,\mathsf{TTP})\},S_{P_3}((m,1)))$$

   but receives an abort reply from the TTP. We have that $S_m = \{4,3,1\}$ and $F_m = \{3,2\}$.

4. P_2 recovers by sending

$$S_{P_2}(\{PCS_{P_1}((m,4),P_2,\mathsf{TTP}),PCS_{P_3}((m,4),P_2,\mathsf{TTP}),PCS_{P_4}((m,4),P_2,\mathsf{TTP})\},S_{P_2}((m,1)))$$

   and succeeds in overturning the abort decision.

In the second attack on P_3, it is P_1 who succeeds in overturning the abort decision and getting P_3's signed contract.

1. At the beginning P_4 aborts, but continues the protocol. We have that $S_m = \{4\}$ and $F_m = \emptyset$.

2. As soon as possible, P_2 recovers by sending

$$S_{P_2}(\{PCS_{P_1}((m,2),P_2,\mathsf{TTP}),PCS_{P_3}((m,3),P_2,\mathsf{TTP}),PCS_{P_4}((m,1),P_2,\mathsf{TTP})\},S_{P_2}((m,1)))$$

   but receives an abort reply from the TTP. We have that $S_m = \{4,2\}$ and $F_m = \{3\}$.

3. P_3 is forced to recover by sending

$$S_{P_3}(\{PCS_{P_1}((m,4),P_3,\mathsf{TTP}),PCS_{P_2}((m,4),P_3,\mathsf{TTP}),PCS_{P_4}((m,4),P_3,\mathsf{TTP})\},S_{P_3}((m,1)))$$

   but receives an abort reply from the TTP. We have that $S_m = \{4,3,1\}$ and $F_m = \{3\}$.

4. P_1 recovers by sending

$$S_{P_1}(\{PCS_{P_2}((m,4),P_1,\mathsf{TTP}),PCS_{P_3}((m,4),P_1,\mathsf{TTP}),PCS_{P_4}((m,4),P_1,\mathsf{TTP})\},S_{P_1}((m,1)))$$

and succeeds in overturning the abort decision.

We did not detect any attack against P_4 while analysing the protocol. This is probably due to the fact that P_4 can never be added to the set F_m, when $n = 4$. We believe that it is difficult to fix the protocol without applying major changes to the recovery procedure.

7.3 The Baum-Waidner multi-party contract signing protocol

In this section we are analysing a second multi-party contract signing protocol. The protocol was presented in 2000 by Baum and Waidner [22] and was developed independently of the GM protocol, which we discussed in the previous section. The structure of the protocol is much simpler than the GM protocol and it has the advantage of being symmetric, i.e. all the participants follow the same protocol. However, the protocol description depends on the number n of participants. The basic version of the protocol does not provide abuse-freeness. However, the authors propose a generic method to add abuse-freeness to a protocol using only standard cryptographic primitives.

7.3.1 Protocol Description

We start giving a description of the basic protocol, i.e. not providing abuse-freeness. In our description, we suppose $n-1$ potential dishonest participants. The original description includes a threshold, which allows to parameterise the protocol with respect to the number of potentially dishonest parties. For the sake of readability, we only describe the case with $n-1$ dishonest participants, which corresponds to the case that we analyse in the next section.

We use the following additional notations during our description.

- $c = (tid, contr)$: the contract, where tid is a unique transaction identifier and $contr$ is the contract text,
- $\mathcal{P}$: the set of n participants who wish to sign the contract.

Main protocol. A detailed description of the main protocol for each participant $P_i \in \mathcal{P}$ is given in protocol 7.6. The protocol is composed of $n+1$ rounds[2]. During each round, P_i sends a promise to all other participants. Note that these promises are based on universally verifiable signatures and that the protocol does not intend to provide abuse-freeness. The level of the promise is increased in each round, and considered as a signed contract once the round number equals $n+1$. Note that the promise also includes the history of all previously received promises, through the vectors M and X, as defined in protocol 7.6. If any expected message is not received, P_i can decide to launch a recovery protocol.

[2] In [22], the protocol has $t+2$ rounds, where t is the maximum number of dishonest parties.

Protocol 7.6 Baum-Waidner multi-party contract signing protocol—Main

$r := 1$

1. $P_i \Rightarrow \mathcal{P} \setminus \{i\}$: $m_{1,i} = S_{P_i}(c, 1, \mathsf{prev\_round\_ok})$
2. $P_j \rightarrow P_i$: $m_{1,j} = S_{P_j}(c, 1, \mathsf{prev\_round\_ok})$ $(P_j \in \mathcal{P} \setminus \{i\})$

   if P_i times out then recovery(1)

   P_i computes vectors $M_{1,i} := (m_{1,1}, \ldots, m_{1,n})$ and $X_{1,i} := M_{1,i}$

   for $r := 2$ to $n + 1$ do

3. $P_i \Rightarrow \mathcal{P} \setminus \{i\}$: $m_{r,i} = S_{P_i}(M_{r-1,i}, r, \mathsf{vec\_ok}), S_{P_i}(c, r, \mathsf{prev\_round\_ok})$
4. $P_j \rightarrow P_i$: $m_{r,j} = S_{P_j}(M_{r-1,j}, r, \mathsf{vec\_ok}), S_{P_j}(c, r, \mathsf{prev\_round\_ok})$ $(P_j \in \mathcal{P} \setminus \{i\})$

   if P_i times out then recovery(r)

   P_i computes vectors

$$M_{r,i} := (S_{P_1}(c, r, \mathsf{prev\_round\_ok}), \ldots, S_{P_n}(c, r, \mathsf{prev\_round\_ok}))$$

   and

$$X_{1,i} := (S_{P_1}(M_{r-1,1}, r, \mathsf{vec\_ok}), \ldots, S_{P_n}(M_{r-1,n}, r, \mathsf{vec\_ok}))$$

Recovery protocol. The details of the recovery protocol are given in protocol 7.7. To recover, P_i sends a recovery request. If the recovery request is launched in the first round, i.e. P_i did not receive a message from all the participants, the recovery request consists in the first level promise. Otherwise, if $r > 1$, the recovery request contains, via the vector $X_{r-1,i}$, the set of received messages.

The TTP maintains for each contract for which it has been contacted a variable recovered indicating whether the given contract has already been recovered or not, a set con, containing the indexes of the participants that contacted the TTP for the given contract and the set abort_set containing the indexes of the participants for whom the TTP aborted the protocol.

When the TTP receives a valid recovery request, it first verifies whether this participant already contacted the TTP or not. If so, the TTP ignores the request. If not, it checks whether the contract has already been successfully recovered or not. A successful recovery is always maintained. Otherwise, we distinguish two cases. If the recovery request is sent in the first round, the TTP must abort the protocol, as the request does not contain a proof that all participants actually started the protocol. If the recovery request is sent during any later round, the TTP verifies whether all the requests that were aborted previously occurred at least two rounds before. If so the TTP can be sure that the previous abort replies were sent to dishonest players, as they continued the protocol and sent out higher level promises. Hence, the previous abort decision is overturned. Otherwise, the TTP replies with an abort token.

7.3.2 Protocol analysis

To analyse the protocol, we used similar simplifications than in the GM protocol. The protocol is simpler than the GM in the sense that it is symmetric, i.e. the description for each participant is the same. However, the number of rounds also depends on the number of participants. Thus, the protocol description changes with the number of participants. As for the GM protocol, we have written a C++ program, specific to this protocol, which generates the description for a given number of participants.

We have verified the protocol with two and up to five participants, but were not able to find any flaw. Note again, that we only analysed the structure of the protocol. Hence, our results only prove that the protocol is correct in the given model. Nevertheless, we believe that the protocol would also be correct in a more general model as all messages are signed and a unique contract identifier is used.

7.3.3 Adding abuse-freeness

In this section we present the generic method proposed by Baum and Waidner to add abuse-freeness to any contract signing protocol using only standard cryptographic primitives, i.e. no esoteric primitives such as PCS.

The idea is that each participant uses a pair of short-term signing and verification keys (sk_i, pk_i), which are freshly generated for each new protocol session. These short term keys are used to sign a *pre-contract*, that is converted into a real contract at the end of the protocol execution. As the certificate linking these session keys to the corresponding participant is kept secret until the end of the protocol execution, verifying such a signature

Protocol 7.7 Baum-Waidner multi-party contract signing protocol—Recovery

1. $P_i \rightarrow$ TTP: $\mathsf{resolve_{r,i}}$

   where

$$\mathsf{resolve_{r,i}} = \begin{cases} (1, i, S_{P_i}(m_{1,i}, \mathsf{resolve})) & \text{if } r = 1 \\ (r, i, S_{P_i}(X_{r-1,i}, \mathsf{resolve})) & \text{otherwise} \end{cases}$$

   if $i \in$ con then stop

   else if recovered

   con:=con$\cup\{i\}$

2. TTP $\rightarrow P_i$: $\mathsf{signed_{r,i}}$

   where $\mathsf{signed_{r,i}} =$ first_signed

   else ($\neg$recovered)

   a. if $r = 1$
      abort_set:=abort_set $\cup\{(r, i)\}$
      con:=con$\cup\{i\}$

   3. TTP $\rightarrow P_i$: $\mathsf{aborted_{r,i}}$
      where $\mathsf{aborted_{r,i}} = S_{\mathsf{TTP}}(c, r, i, \mathsf{aborted})$

   b. else $(r > 1)$

      (i) if $\forall(s, k) \in$ abort_set $\cdot\ s < r - 1$
      if con=$\emptyset$ then the TTP sets first_signed:=($\mathsf{resolve_{r,i}}, S_{\mathsf{TTP}}(c, r, i, \mathsf{recovered})$)
      recovered:=true
      con:=con$\cup\{i\}$

      4. TTP $\rightarrow P_i$: $\mathsf{signed_{r,i}}$
         where $\mathsf{signed_{r,i}} =$ first_signed

      (ii) else
      abort_set:=abort_set $\cup\{(r, i)\}$
      con:=con$\cup\{i\}$

      5. TTP $\rightarrow P_i$: $\mathsf{aborted_{r,i}}$
         where $\mathsf{aborted_{r,i}} = S_{\mathsf{TTP}}(c, r, i, \mathsf{aborted})$

does not prove anything about its origin. The details of this method can be described in four phases.

1. **Distribution of encrypted certificates.** For each new session, P_i generates (sk_i, pk_i) and computes the corresponding certificate
$$cert_i = S_{P_i}(i, pk_i, tid, contr)$$
and the encryption of the certificate for the TTP
$$ecert_i = E_{\mathsf{TTP}}(r_i, cert_i)$$
where r_i is some random string. Then P_i distributes the $(pk_i, ecert_i, tid, contr)$ to all participants.

2. **Pre-contract signing.** Execute the contract signing protocol using the n session keys $(pk_1, sk_1), \ldots, (pk_n, sk_n)$ to sign the pre-contract
$$c = (tid, contr, (pk_1, ecert_1), \ldots, (pk_n, ecert_n)).$$

3. **Conversion of pre-contract into real contract.** Distribute $(r_i, cert_i)$ to all other participants. If P_i does not receive valid messages from all the other participants, he contacts the TTP.

4. **Recovery from failed conversion.** P_i sends the signed pre-contract to the TTP, who extracts the list $((pk_1, ecert_1), \ldots, (pk_n, ecert_n))$ and tries to decrypt all certificates. If the TTP succeeds to decrypt all certificates and they match the session verification keys, the TTP sends
$$S_{\mathsf{TTP}}(tid, contr, (cert_1, \ldots, cert_n))$$
to P_i. Otherwise the TTP sends a "failed" message to P_i.

Note that a contract is defined to be valid only if all the short term keys match. Therefore, whenever the TTP is not able to extract the certificates, one is sure that the contract will be regarded as invalid by a verifier.

We did not formally analyse the resulting protocol when the above described method is used to provide abuse-freeness. However, we are going to illustrate that the composition of the two protocols is not trivial and may lead to an attack in an efficient implementation. Imagine the following execution. Participants P_1 through P_n are executing the previously defined contract signing protocol, to sign the pre-contract
$$c = (tid, contr, (pk_1, ecert_1), \ldots, (pk_n, ecert_n)).$$
In round number one, dishonest P_1 contacts the TTP and gets the protocol aborted. In round number two, P_2 who is also dishonest contacts the TTP, who will not overturn the abort decision. When the TTP verifies the validity of the recovery request, the TTP must verify a signature for every participant. There exist two ways of proceeding: the TTP can decide to either verify or not to verify the validity of the short-term keys by decrypting

$ecert_i$. The advantage of doing this verification is that an attempt to cheat could be detected much earlier and the protocol would be more efficient. An invalid key would be detected the first time a participant contacts the TTP and the protocol could be stopped immediately, without going until the fourth phase of the described method for ensuring abuse-freeness. However, if this is done, the TTP's abort reply to the second recovery request would provide a proof, which can be shown to Charlie, that all participants have started the protocol. Moreover, the dishonest participants could either stop the protocol or achieve a successful outcome via the optimistic protocol flow. This would be a similar attack than in the GM protocol, where the TTP is used to verify signatures that Charlie would be unable to verify. Therefore it is crucial that the TTP does not check the encrypted certificate at this point of the protocol. In [22], Baum and Waidner do not discuss the details of how to compose the contract signing protocol with the method for providing abuse-freeness. As we have shown, not being explicit about such details may compromise the security of the protocol. We do not claim that the above scenario is an attack, but we want to emphasise that there may be subtle problems due to the fact that some details were left unspecified.

7.4 Conclusion

In this chapter, we have analysed three optimistic multi-party fair exchange protocols. The first one, a non-repudiation protocol, has been analysed using a similar model than in the two-party case. Multi-party contract signing protocols are more complex and for efficiency reasons, we only verified the *structure* of these protocols. Due to the simplified model, MOCHA did not detect the attack against fairness on the three-party version of the GM protocol, but discovered several attacks on the four-party version. Moreover, we showed how to break and fix abuse-freeness on the 3-party version of the GM protocol. We also analysed a second, slightly simpler, but still complex, contract signing protocol due to Baum and Waidner. We did not find any flaw, at least in its structure, but show, how abuse-freeness may be compromised when aiming at an efficient implementation. This is the first formal analysis of multi-party fair exchange protocols and currently we only considered a very simple model due to the great complexity of these protocols. However, as our analysis demonstrated, even this simple model can reveal unknown flaws. Considering more complete models or verifying the parameterised version of the protocols raises new challenges and is left for future works.

CHAPTER 8

Other approaches

In this chapter, we describe some other approaches which were used to study fair exchange protocols. Some of these approaches are not directly related to our work, such as the study of rational exchange [39, 40, 41] or probabilistic protocols [121] and we only briefly describe these works. There exist also approaches which study non-repudiability, a property which we did not aim to analyse in our work. For instance Zhou and Gollmann's work [180] using the SVO logic does only study non-repudiability, but neither fairness nor timeliness. Other approaches, such as the ones by Schneider [141] using CSP or by Bella and Paulson [24] using inductive methods, studied both non-repudiability as well as fairness and timeliness. However, they did not study optimistic protocols. Protocols using an on-line TTP seem to be much more easy to analyse, as the protocols are not branching. Finally, we describe three approaches that were used to study optimistic fair exchange protocols. Boyd and Kearny [31] used a specification animation tool to analyse protocols at a very high level of abstraction. More intensive studies were realized by Shmatikov and Mitchell [146, 147, 148] using the finite state model-checker Murφ and by Chadha, Kanovich and Scedrov [43] as well as Chadha, Mitchell, Shmatikov and Scedrov [44] using inductive methods. These two last approaches are the most complete studies and the closest to our work. Therefore, we describe them in more detail.

8.1 Game-theoretic analysis of rational exchange protocols

In [39, 40, 41], Buttyán et al. use a game-theoretic approach based on the notion of equilibria to analyse rational exchange protocol. Remember that a rational exchange protocols does protect each participant against a malicious party, who is not interested to harm the other party unless cheating will provide some gain. The authors show that rationality in an exchange protocol can be precisely modelled using the notion of Nash equilibrium. A Nash equilibrium is a set of strategies, one for each player, such that any deviation from this strategy implies a lower outcome for the player who is not following the strategy (and possibly also for the other players). A protocol is said to be rational if the strategies dictated by the protocol form a Nash equilibrium. Fairness can be expressed as follows: for all possible strategies of a player, if the outcome corresponds to the expected gain, then the other player's strategy also provides his expected gain. The authors formally prove, by pen and paper, the intuition that fairness is stronger than rationality. However, they consider a synchronous network model. They briefly discuss how to extend their model to an asynchronous network and show that Syverson's protocol [152] would not be rational on such a network, while they prove it to be rational on a synchronous network. When considering an asynchronous network, however, we think that the fairness definition is not strong enough anymore. We have shown in chapter 5, that the strongest version of the fairness property can be expressed in CTL and not in terms of strategies. A similar argument applies to their model. Hence, their model would probably need important modifications to study optimistic fair exchange protocols on an asynchronous network (which was not the aim of their papers).

8.2 Prism

In 2002, Norman and Shmatikov [121] analysed a probabilistic contract signing protocol by Ben-Or et al. [25] using the probabilistic symbolic model checker Prism. This is the first attempt to automatically analyse a probabilistic fair exchange protocol. A simplified version of the protocol, discretising the probabilities, is described using a probabilistic variant of reactive modules, which is the MOCHA input language. As all messages are signed, Norman and Shmatikov only consider the protocol messages in their analysis. The dishonest behaviour of a participant allows delaying and rescheduling messages on the communication channels, as well as contacting the judge whenever this is possible.

In their study, they first observe that the protocol in its initial description does not respect timeliness. Given this description, the model checker Prism is used to determine the maximum probability with which an unfair state can be reached and show that this probability can be arbitrarily small. The authors also show that adding timeliness is not a trivial task. When the TTP is implemented in an efficient way, a straightforward method for adding timeliness results into an unfair protocol, i.e. fairness can be broken with arbitrarily high probability. Their case study shows that using Prism to analyse a simple version of the protocol, with a limited attacker, provide already interesting results.

8.3 SVO

In 1998, Zhou and Gollmann [180] use the SVO logic, which is the most mature of the BAN logic successors, introduced in section 2.4. The aims of their verification are different from our method. They reason about the validity of the produced non-repudiation evidences, rather than properties such as fairness or timeliness. They reason about what a judge J can *believe* when evidences are presented during the dispute resolution procedure, defined with a non-repudiation protocol. Non-repudiability is naturally expressed in such a logic. For instance non-repudiation of origin and receipt can be expressed as follows in SVO.

$$J\ believes\ (A\ said\ M)$$

$$J\ believes\ (B\ received\ M)$$

As this method is applied to different properties, it is complementary to our analysis technique. In [180], Zhou and Gollmann analysed the non-repudiation protocol with online TTP which they previously presented in [177]. We believe that analysing the dispute resolution in an *optimistic* protocol should not be a major problem. However, the fact that different non-repudiation evidences could be generated, depending on whether the TTP did intervene or not, would increase the complexity of the proofs and harm to some extent the advantage of the shortness and simplicity of SVO proofs. One can also note that some contract signing protocols, e.g. [22], have a *contract verification protocol*, in order to decide the validity of a contract. Such contract verification protocols could be subject to a similar analysis. Also weak fairness, as defined in section 3.4.3, could benefit from such an approach, as in a weakly fair protocol an entity should be able to prove to a judge up to which point the protocol has been executed.

8.4 CSP

In 1998, Schneider [141] presents, using the CSP process algebra, the first study of a non-repudiation protocol, i.e. the non-repudiation protocol with online TTP by Zhou and Gollmann [177]. He provides proofs for both non-repudiation evidences, i.e. if a non-repudiation of origin, respectively receipt, evidence is presented to a judge, then Alice has sent, respectively Bob has received, a given message, and for the fairness property. To achieve his proofs, Schneider adapted his techniques previously used for authentication protocols described in section 2.6. There are essentially two main differences with the techniques described previously.

The first difference is that the environment does not contain an attacker but only dishonest participants. Dishonest participants are modelled in a similar way to the intruder in authentication protocols. This is similar to our model, in the sense that we also model dishonest participants rather than an intruder, but Schneider does not restrict the classical Dolev-Yao model. As a consequence, the network is now modelled explicitly and is not part of the intruder. Schneider also noticed that, when verifying the dispute resolution protocol, there was no need to explicitly model the protocol steps, as the honest behaviour is included into the dishonest one. As the judge does not know whether an entity did behave correctly or not, describing explicitly the protocol steps is not needed. However, when verifying fairness, one has to explicitly model the honest behaviours as fairness is only guaranteed to a participant behaving honestly. Indeed a dishonest party could trivially induce executions where fairness is violated for himself. In our approach based on strategies, dishonest behaviours where participants harm themselves are automatically excluded, but as we have seen before when discussing properties expressed in ATL, considering the honest behaviour results into stronger guarantees.

The second difference is that the properties require to be partially expressed as *liveness* properties: while authentication and secrecy properties are safety properties, fairness and non-repudiability are expressed as liveness properties or a combination of fairness and liveness properties. Safety properties can be proved on traces using rank functions, which have been introduced in section 2.6. However, to prove liveness properties, one needs to consider failures. The failures of a process P are a set of trace/refusal pairs, where a refusal is a set of events the process P can avoid or refuse. Liveness can then be expressed as events that are not included in any failure of the process. For instance, a general form of fairness for Alice could be expressed as follows.

$$\forall (tr, X) \in \mathit{failures}(P) \cdot \mathsf{item_B} \ \mathit{in}\ tr \Rightarrow \mathsf{item_A} \notin X$$

This formula states that if Alice received $\mathsf{item_B}$ on a given trace, then it is impossible for Bob not to get $\mathsf{item_A}$, i.e. Bob cannot avoid getting $\mathsf{item_A}$.

In [141], Schneider analysed a protocol with an online TTP. We think that the techniques do not have to be altered to apply to optimistic protocols. As branching is naturally expressed in a process algebra, the modelling of an optimistic protocol should be straightforward. However, although the techniques are the same, we believe that optimistic protocols would require longer and more complex proofs. The main drawback of this approach is the fact that the proofs are complicated and require a great amount of expertise.

8.5 The inductive approach

In 2001, Bella and Paulson [24] used the theorem prover Isabelle, in a similar way as described in section 2.9 to prove the correctness of Zhou and Gollmann's non-repudiation protocol with online TTP. The choice of the protocol was influenced by the existing works of Zhou and Gollmann using SVO, and Schneider's work in CSP. Like Schneider, Bella and Paulson verified both fairness and validity of the non-repudiation evidences. The inductive approach as defined originally by Paulson, only requires minor changes in order to be able to study non-repudiation protocols. The most important extension is probably the possibility to reason about the knowledge of honest participants. In the initial model, it was only possible to reason about the knowledge of the intruder. Their proofs are first carried out considering only honest participants. Then the proofs are extended in the presence of an intruder, who controls a set of "bad" agents, and knows their private keys. This is a significant difference to our approach, as Bella and Paulson do not explicitly model malicious participants. Moreover, in contrast to Zhou and Gollmann, as well as Schneider's work, a judge is not modelled, as the dispute protocol is a different protocol. Instead of sending evidences to a judge, they consider that knowledge of an evidence is sufficient. The theorems proved by Isabelle are similar to the ones proved by Schneider in CSP.

Their case study is carried out on a protocol with online TTP and to the best of our knowledge no effort has been made to study an optimistic protocol. Moreover, the properties that are studied are all expressed on traces. Properties like balance and abuse-freeness can however not be expressed as trace-based properties. In Chadha et al.'s works [43, 44], which will be discussed below, the inductive approach is extended to deal with properties that consider trees of possible execution. However Chadha et al.'s proofs are carried out by hand and we do not know what would be the consequences when trying to automate these proofs using Isabelle.

8.6 Possum animation tool

In 2000, Boyd and Kearny [31] proposed to use the *specification animation* tool Possum to analyse fair exchange protocols. This analysis takes place at a very high level of abstraction. The contents of the messages is completely ignored and only the structure of the protocol is examined. They model a dishonest participant by giving him the possibility to send messages out of order. This very high level description of the protocol is given in the Z specification language. Then, using Possum, it is possible to *step through* the protocol and explore the different reachable states using machine assistance. The advantage of the approach is its simplicity. However, the preconditions about when a message may be sent must be given manually (in terms of other messages that have been received). Encoding by hand when a message can be sent may however lead to an incomplete specification. Nevertheless, their effort has lead to unknown flaws and the reproduction of known ones.

8.7 Murφ

In [146, 147, 148], Shmatikov and Mitchell used the model-checker Murφ to analyse the ASW and the GJM protocol, which we described in chapter 6. The general approach for verifying security protocols, in the case of authentication protocols, has already been discussed in section 2.8. The protocol participants as well as the intruder are described using the Murφ input language by hand. A translation from a more high-level language, like the protocol description language we introduced in chapter 5, is currently not available.

The model must be slightly adapted to the context of fair exchange protocols. Branching is naturally expressed in their modelling language. To model dishonest participants, Shmatikov and Mitchell took a different approach than we did. They model a dishonest participant by letting him collaborate with the intruder, which is a classical Dolev-Yao intruder. They distinguish three levels of dishonesty. A *strongly corrupted* party shares its private keys with the intruder. A *weakly corrupted* party does only sign or cipher the messages that the protocol foresees, but he is willing to accept messages from the intruder which could help him to cheat, without acting openly in a dishonest way. The weakest form of dishonesty, is the *accidentally corrupt* party. In that case the participant enables, probably unintentionally, the intruder to monitor incoming network traffic. Note that the notions of strong and weak corruption differ from strongly, respectively weakly, dishonest participants as defined in our model in chapter 5. Shmatikov and Mitchell also model different kinds of channels. Unlike we did, they do not use fairness constraints to model resilient channels. Instead, they make the invariants used to express properties conditional on the delivery of all messages emitted on resilient channels.

As explained in section 2.8, properties are expressed using invariants. This modelling is sound for fairness, as fairness is a *monotonic* property, i.e. once fairness is violated, it remains violated. Hence, the invariant may be expected to hold in final states. However, the expressiveness of invariants is restricted. A property such as timeliness, which is a liveness property, cannot be expressed. Moreover, modelling abuse-freeness is not straightforward in their model, as this property is not monotonic. Shmatikov and Mitchell model a simplified version of this property, which is later referred to as balance, by adding the notion of a *challenger*. Their model is increased by a challenger who chooses non-deterministically between success or abort of the protocol. The dishonest party must then succeed in realizing the given challenge. Using this "trick", they succeed in verifying balance. However, our modelling is more complete in the sense that we model abuse-freeness and not only balance. Moreover, our notion of strategies seems more natural for this property.

When analysing the ASW contract signing protocol (protocols 6.6–6.8) they detect a rather strange flaw: it is possible, by the mean of a replay attack to get a different version of a same contract. This is due to the fact, that in the original paper, a contract is defined as being the tuple (m_1, m_2, N_A, N_B). Hence, Shmatikov and Mitchell succeed in showing that an external intruder can create a different contract (m_1, m_2, N_A, N'_B), that refers to the same contract text, the same persons Alice and Bob, but is validated by a different nonce. The attack is depicted in protocol 8.1. As also noticed in [146, 148], it is not clear whether the authors of the protocol, deliberately permit this attack or not and whether it should be considered as a flaw. Shmatikov and Mitchell propose to fix this flaw, by redefining the two last messages of the protocol as $m_3 = S_A(N_A, h(N_B))$ and $m_4 = S_B(N_B, h(N_A))$. Unfortunately, with this solution the last two messages need to be signed. The original protocol did aim efficiency and requested only two instead of four signatures. Therefore, we propose an alternative way to prevent this "attack" by changing the definition of what is a contract. We propose to make a distinction between the contract defined by the contract text and a *validator* for a given contract, corresponding to the tuple (m_1, m_2, N_A, N_B). Then, the possibility to generate different validators for a same contract should not be considered as a problem. We completely agree that their "attack" highlights an interesting ambiguity in the definition of what is a signed contract.

Protocol 8.1 A strange replay attack on the ASW contract signing protocol

1. A → B: $m_1 = S_A(A, B, \mathsf{TTP}, t, h(N_A))$
2. B → A: $m_2 = S_B(h(m_1), h(N_B))$
3. A → B: $m_3 = N_A$
4. B → A: $m_4 = N_B$
5. I → B: $m_1 = S_A(A, B, \mathsf{TTP}, t, h(N_A))$
6. B → I: $m'_2 = S_B(h(m_1), h(N'_B))$
7. I → B: $m_3 = N_A$
8. B → I: $m'_4 = N'_B$

During the analysis of the GJM protocol (protocols 6.9–6.11), they discovered a more serious flaw, which we described during our analysis of the GJM protocol. In broad terms, the attack works as follows. After having sent the first message, Alice aborts the protocol. When Bob launches a recovery request, which is refused by the TTP due to the previous recovery, Alice obtains the signed contract, which is included in Bob's request, by spying on the channel. Assuming confidentiality should prevent this flaw. Shmatikov and Mitchell show that if the TTP is accidentally corrupted, the intruder may eavesdrop the incoming network traffic, even if the channels are private. This depends on how privacy is implemented on the channel, at what level a decryption of the data is performed and what exactly an intruder can recover. They conclude that the presented flaw also compromises abuse-freeness and propose to fix the protocol by sending only the PCS to the TTP and no universally verifiable signatures. The fixed version of the protocol is fair but not balanced. However, they conclude that nevertheless abuse-freeness is ensured due to the use of PCS, but do not formalise this last step. As we have seen in the GM multi-party contract signing protocol, the usage of PCS may not be sufficient to ensure abuse-freeness (cf section 7.2.2). Therefore, modelling the moment, when participation in the protocol can be proven to an observer, should be part of the formal analysis. The result that the protocol is not balanced, corresponds to our result in the case where the communication channels are controlled by the dishonest player.

Shmatikov and Mitchell also verified another property, which is called TTP accountability, also sometimes referred to as verifiability of the TTP. The aim of this property is to ensure that, when a TTP misbehaves, then this can be proved to a judge. However, verifiability only holds when the TTP is supposed to send valid responses to all requests. In our opinion, this is an unrealistic assumption in most contexts, as sending invalid responses or not replying at all would be the easiest way to compromise fairness. Asokan [10] proposes that when a TTP is not answering, this could be demonstrated to a judge. This seems not to be an entirely satisfactory solution if the TTP's not replying is sporadic. Shmatikov and Mitchell verified TTP accountability on both the ASW and the GJM contract signing protocol. They argue that a participant can prove the misbehaviour of the TTP if, at some moment in the protocol, the TTP issues contradictory documents, i.e. an abort token and a replacement contract for a same contract. They did not find any attacks compromising this property on the ASW protocol. However, the previously described attack on the GJM protocol, when the TTP is accidentally corrupted, presents a violation of the property: fairness is broken due to the TTP's, although involuntary, leak of information, without however issuing contradictory documents.

8.8 Multi-set rewriting and inductive methods

In 2001, Chadha, Kanovich and Scedrov [43] came up with a different approach to analyse contract signing protocols. They model the protocols using the multi-set rewriting formalism, MSR for short, which is a well studied description level for security protocols [42]. While MSR has the advantage of being well studied, encoding the protocols using MSR may not be trivial for a non-expert. In their model, they define two levels of dishonest participants, a weakly and a strongly dishonest one, in the same way as Shmatikov and Mitchell did when using Murφ. The communication channels are modelled to be *transparent*, i.e. messages are never lost nor delayed. However, the intruder can read all the messages sent on the network. This corresponds to our notion of operational channels.

Chadha et al. studied the GJM protocol using inductive methods. Fairness is defined as follows. In each reachable state, if a dishonest party has a contract, then the honest participant can reach a state, where he also possesses a valid contract. In order to be able to define balance, Chadha et al. need to introduce the notion of strategies. Therefore, they define what is called a *continuation tree*. Intuitively a continuation tree corresponds to all possible executions of the protocol, where the intruder is limited to a finite number of actions. Each node in the tree corresponds to a state of the protocol and each edge to a possible action, associated to a player, in the given state. A strategy for a player is defined to be a tree obtained by letting the player remove a subset of the edges, corresponding to some of his actions, and all of their descendants. Their notion of strategy is closely related to ours, defined in chapter 4. Their definition of balance is expressed in terms of strategies and similar to our definition of balance given in section 5.2.4. They prove, by adapting the inductive proofs to the case of trees and not only traces, that the protocol is balanced on transparent channels. This result can be related to our result in the case where the communication channels are controlled by the honest player. All the proofs are carried out by hand which makes the analysis rather difficult. The advantage is that the proofs are carried out in a less restricted model.

More recently, in 2003, Chadha, Mitchell, Scedrov and Shmatikov [44] presented a new study of contract signing protocols. The model is similar to the one used by Chadha, Kanovich and Scedrov in [43], except one important detail. Players are modelled to be optimistic, i.e. they always wait for a finite amount of time before contacting the TTP. This is modelled by letting the dishonest player decide when a honest player times out. They prove the following impossibility result. Given a fair, optimistic, timely protocol, if Alice is optimistic, then Bob has an advantage on Alice. Having an advantage is defined to have the power to choose the outcome of the protocol. Their result is general in the sense that it is not proven on a given protocol, but on a family of protocols. We can relate their result to our case studies of the ASW and the GJM contract signing protocols. An optimistic player waiting before contacting the TTP is similar to a dishonest player controlling the communication channels and delaying the delivery to the TTP. The only difference is that in their case the sending of the message is delayed, while in our model it is the delivery which is delayed. Hence, their theorem confirms and generalises our results. They conclude that if achieving balance is impossible, the best one can achieve is abuse-freeness. They also give a complete modelling of abuse-freeness, using a notion similar to the prove2C predicate we defined in section 5.2.4. However, while we instantiated this predicate manually for each protocol, they give a definition based on computationally indistinguishable computations. Such a definition could also be applied to our model and better formalise the notion of "proving to Charlie".

8.9 Conclusion

As we have seen, only very few works have been applied to fair exchange protocols, and especially to optimistic ones. Some of these works model non-repudiability, which we did not study. However, non-repudiability could be modelled in a similar way than Bella and Paulson did. We do not think that this is a restriction of our method.

In comparison to Shmatikov and Mitchell's work, our approach has the advantage of having a more expressive logic. This enabled us to have a more natural expression of the properties. Above all, we did not encounter any problems when expressing balance and abuse-freeness. Moreover, we provide a high-level description language, making the study more convenient. Shmatikov and Mitchell studied TTP accountability, which again is a property that our study did not aim. To study TTP accountability, we would have to model a dishonest TTP. Currently, it seems to us that it is not yet well defined to what extent a TTP could be allowed to be dishonest. Investigating this property in more detail is left for future works.

The works by Chadha et al. have the advantage of providing proofs in a less restricted environment, as they do not restrict to the protocol messages, the messages a malicious participant can send out. Moreover, when proving fairness, they did consider multiple runs, although they did not do so when studying balance and abuse-freeness. Two drawbacks of this approach are that the proofs must be carried out by hand and that they only consider transparent channels. As our studies have shown, modelling different levels of quality provides great flexibility and some of our results are related to the fact that we considered resilient channels. An interesting point is the modelling of optimistic players. Adding optimistic players should not be difficult in our model. We could implement an optimistic player by explicitly modelling the time-outs deciding when to contact the TTP. These time-outs should be controlled by the malicious player and their triggering after a finite amount of time would be subject to a fairness constraint. However, we did not try to implement optimistic players yet.

CHAPTER 9

Conclusion

9.1 Summary

To conclude this thesis we summarise its content and emphasise our results and contributions to the field.

We started by giving a brief state of the art of formal methods in the context of security protocols. These techniques concentrate on protocols aiming confidentiality and authentication.

We went on presenting fair exchange protocols. We first concentrated on the properties a fair exchange protocol should provide. We reviewed the definitions proposed in literature and concluded that most properties are either too specific or gather different properties in a single definitions. Therefore we proposed a general and modular definition of the security of a fair exchange protocol. The definition is informal but the goal is to increase our informal understanding of the properties before linking them to a specific formalism.

Next, we presented the evolution of fair exchange protocols, illustrating, from a historical view, the different solutions that have been proposed to achieve the previously presented properties. The most recent protocol propositions are based on an offline TTP and are also said to be optimistic. Finally, we emphasised some fundamental differences between optimistic fair exchange protocols and more classical protocols. A major difference is the adversarial character of the properties which fair exchange protocols have to satisfy, since they imply a different attacker model. While classical security protocols must be secure against an external intruder, fair exchange protocols must be secure, even against a malicious protocol participant. Two other differences are the communication model, which requires at least some of the channels to be resilient, and the existence of subprotocols implying branching which is specific to optimistic fair exchange protocols. These differences motivate the setup of new techniques for the analysis of optimistic fair exchange protocols.

Then, we introduced the underlying formal model, which we used in the following. We have chosen the alternating transition systems and the alternating-time temporal logic as our tools. The advantage of this model is its game semantics, which allowed us to express properties of fair exchange protocols in a natural way. The ATS can be described in a convenient way through a guarded command language simplifying the encoding of the protocols. Moreover, model-checking is provided by the finite state tool Mocha. However, we had to extend the implementation of the model-checking algorithms in order to handle fairness constraints, which are a powerful tool when modelling protocols.

After the introduction of the formal model, we were able to define the protocols and the properties with respect to this model. We presented a protocol specification language where each participant is described by the tree of its possible executions. This language seems to be particularly adequate for branching protocols, such as optimistic fair exchange protocols. Then we gave the semantics of the specification language in terms of ATS. Note that our model makes several simplifications: first we only consider one parallel protocol execution and second we only consider the messages of the protocol to be important. These simplifications are motivated on the one hand by the need to get a sufficiently small model, making model-checking effective, and on the other hand by the fact that in practice fair exchange protocols are well protected at the cryptographic level, including protocol identifiers and extensive use of digital signatures.

We also defined the desired properties in terms of ATL formulae which can be verified on the ATS describing the protocol. Due to the adversarial character of the properties, a formulation in terms of strategies seems to be very natural. Formalising the properties also allowed us to highlight the link between several properties. More specifically, we prove that

- an unfair protocol cannot respect timeliness,
- a balanced protocol is also abuse-free,
- a protocol respecting author-based selective receipt is abuse-free for the protocol initiator.

We also gave a complete specification of abuse-freeness and not only balance as it is customary in literature. To the best of our knowledge, this is historically the first formal definition of abuse-freeness in the context of formal analysis.

In the following we illustrated our techniques on a number of protocols. We showed that the ZG non-repudiation protocol is unfair when the communication channels are resilient, as defined in the original paper. The ASW certified e-mail protocol has several weaknesses compromising fairness, which are related to concurrency problems between subprotocols executed at unexpected moments. Moreover, we showed that, if the communication channels are not confidential, the protocol is unfair, which emphasises the need of this hypothesis. We also presented a variant of the KM non-repudiation protocol. This variant has been constructed to be fair but does not respect timeliness. The motivation to analyse this variant, is to illustrate the existence of such protocols and the importance of the timeliness property, which is often neglected in the literature compared to fairness. Then we analysed the ASW and the GJM contract signing protocols. We noticed that for the ASW contract signing protocol confidentiality is not required, while supposed by the authors. However, the GJM protocol is unfair when channels are not confidential. We also show that, when a malicious party does not completely control the communication channels, the ASW protocol is balanced, and hence abuse-free. This is in contradiction with the common belief that PCS, or similar techniques, are required to provide abuse-freeness. Therefore, we proposed a new definition of abuse-freeness, which we prove to be stronger and which does not depend on the control of the communication channels. With this new definition, we have, as expected, that the ASW protocol is neither balanced nor abuse-free, while the GJM protocol is not balanced but abuse-free.

We also applied our techniques to multi-party fair exchange protocols. To the best of our knowledge, this is the first formal analysis of group protocols of this kind. We analysed the MK multi-party non-repudiation protocol. We showed that if Alice's and the TTP's clocks are not synchronised, such as proposed in the original paper, a race condition on the message transmission can introduce a subtle error. We also analysed two multi-party contract signing protocols. The GM multi-party contract signing protocol, presented in 1999, was believed correct until now, but we found several novel attacks breaking both abuse-freeness and fairness. While the flaw leading to the violation of abuse-freeness is easily fixed, the different attacks on fairness are more fundamental. We believe that the protocol requires serious changes in order to be correct. When analysing the BW multi-party protocol, we did not find any attack compromising fairness. However, an implementation aiming at efficiency could lead to a flaw compromising abuse-freeness.

Finally, we present some other approaches to formally analyse fair exchange protocols. Some of these approaches are not directly related to our work, as they concern probabilistic or rational exchange protocols. The works that are most close to ours are the ones using the model-checker Murφ and the works based on inductive methods and MSR. In comparison to the studies automated using Murφ, our approach has the advantage of using a more expressive logic to model protocol goals. ATL allows us to express liveness properties such as timeliness and, by means of strategies, provides a natural setting for properties such as abuse-freeness and balance. Moreover, our recently developed tool avoids the error-prone process of encoding the protocol and the attacker using a low level specification language. The approach based on inductive methods and MSR has the advantage of considering a more complete attacker model. However, our approach has been automated, while proofs in the MSR setting have to be done completely by hand.

9.2 Future works

Directions for future works include a more complete adversary model. On the one hand, we want to omit the simplification that consists in considering only protocol messages. This is current work on progress and can be achieved using abstract interpretation techniques [79]. On the other hand, relaxing the hypothesis of only one protocol run would be desirable. Using abstractions could be one direction. However, abstracting away entities, rather than only actions of a given player, seems to be difficult. Therefore we would like to investigate two other directions. One consists in adapting Lowe's "small system theorem" [95] to fair exchange protocols, i.e. formally prove our intuition that the session identifiers and the fact that all messages are usually signed are sufficient to safely consider only one parallel run for a wide range of protocols. Another direction would be to extend Comon and Cortier's work [50], and show that, for fair exchange protocols too, we only need a bounded number of entities to capture all attacks. Then, we can again use abstraction techniques to achieve more general results.

For multi-party fair exchange protocols, the definition of a more complete model is even more challenging. We foresee several possible extensions, that may be combined. First we can, as described above, consider all possible messages an adversary can construct and foresee parallel protocol runs. Another interesting extension would be to analyse the protocol as a parametric system, i.e. without fixing a priori the number of participants. Multi-party fair exchange protocols can be divided into three categories, coming each with additional challenges. The first category includes protocols, such as the MK multi-party non-repudiation protocol, where

- the protocol is symmetric, i.e. each entity, except maybe one leader, execute a similar protocol,
- the structure of the protocol does not depend on the parameter.

The second class of protocols includes the BW multi-party contract signing protocol, where

- the protocol is symmetric,
- the structure of the protocol depends on the parameter, e.g. the number of rounds depends on the parameter.

Finally, the third class of protocols includes the GM multi-party contract signing protocol, where

- the protocol is asymmetric, i.e. each entity executes a different protocol,
- the structure of the protocol depends on the parameter.

The first category is rather close to parametric systems which can nowadays be analysed [164]. However, we expect combining these techniques with a Dolev-Yao-like attacker to be non trivial. Analysing the two other categories of protocols seems an even greater challenge.

Finally, we believe that our techniques could be adapted to analyse other properties, such as TTP accountability. It would also be interesting to analyse non-repudiability in a context where public keys can be revoked and techniques such as [169, 173, 181] are used to maintain the evidences. Our techniques could probably also be adapted to analyse other families of protocols where the attacker is a malicious participant, rather than an external intruder. Examples of such protocols are electronic auction and voting protocols.

Bibliography

[1] Sur un cadre communautaire pour les signatures électroniques. Directive 1999/93/CE du Parlement Eurpéen et du Conseil, December 1999.

[2] Martín Abadi and Andrew D. Gordon. A calculus for cryptographic protocols: the spi calculus. Technical report 414, University of Cambridge Computer Laboratory, Cambridge, UK, January 1997.

[3] Martín Abadi and Roger Needham. Prudent engineering practice for cryptographic protocols. In *IEEE Symposium on Research in Security and Privacy*, pages 122–136, May 1994.

[4] Martín Abadi and Phillip Rogaway. Reconciling two views of cryptography (the computational soundness of formal encryption). *Journal of Cryptology*, 15(2):103–127, 2002.

[5] Martín Abadi and Mark Tuttle. A semantics for a logic of authentication. In Luigi Logrippo, editor, *10th Annual ACM Symposium on Principles of Distributed Computing (PODC 1991)*, pages 201–216, Montreal, Quebec, Canada, August 1991. ACM.

[6] Rajeev Alur and Thomas A. Henzinger. Reactive modules. *Formal Methods in System Design*, 15:7–48, 1999.

[7] Rajeev Alur, Thomas A. Henzinger, and Orna Kupferman. Alternating-time temporal logic. In *38th Annual Symposium on Foundations of Computer Science*, pages 100–109. IEEE Computer Society Press, 1997.

[8] Rajeev Alur, Thomas A. Henzinger, Freddy Y.C. Mang, Shaz Qadeer, Sriram K. Rajamani, , and Serdar Tasiran. MOCHA: modularity in model checking. In Alan J. Hu and Moshe Y. Vardi, editors, *Tenth International Conference on Computer-aided Verification (CAV 1998)*, volume 1427 of *Lecture Notes in Computer Science*, pages 521–525, Vancouver, BC, Canada, June 1998. Springer-Verlag.

[9] Ross Anderson and Roger Needham. Robustness principles for public key protocols. In Don Coppersmith, editor, *Advances in Cryptology – Crypto 1995*, volume 963 of *Lecture Notes in Computer Science*, pages 236–247. International Association for Cryptologic Research, Springer-Verlag, 1995.

[10] N. Asokan. *Fairness in Electronic Commerce.* PhD thesis, University of Waterloo, May 1998.

[11] N. Asokan, Birgit Baum-Waidner, Matthias Schunter, and Michael Waidner. Optimistic synchronous multi-party contract signing. Research Report RZ 3089, IBM Research Division, December 1998.

[12] N. Asokan, Matthias Schunter, and Michael Waidner. Optimistic protocols for multi-party fair exchange. Research Report RZ 2892 (# 90840), IBM Research, December 1996.

[13] N. Asokan, Matthias Schunter, and Michael Waidner. Optimistic protocols for fair exchange. In Tsutomu Matsumoto, editor, *4th ACM Conference on Computer and Communications Security*, pages 8–17, Zurich, Switzerland, April 1997. ACM Press.

[14] N. Asokan, Victor Shoup, and Michael Waidner. Asynchronous protocols for optimistic fair exchange. In *IEEE Symposium on Research in Security and Privacy*, pages 86–99, Oakland, CA, May 1998. Technical Committee on Security and Privacy, IEEE Computer Society Press.

[15] N. Asokan, Victor Shoup, and Michael Waidner. Optimistic fair exchange of digital signatures. In *Advances in Cryptology—Eurocrypt 1998*, volume 1403 of *Lecture Notes in Computer Science*, pages 591–606, 1998.

[16] N. Asokan, Victor Shoup, and Michael Waidner. Optimistic fair exchange of digital signatures. *IEEE Journal on Selected Areas in Communications*, 18(4):593–610, April 2000.

[17] Giuseppe Ateniese. Efficient verifiable encryption (and fair exchange) of digital signatures. In *6th ACM Conference on Computer and Communications Security (CSS 1999)*, pages 138–146, Singapore, November 1999. ACM Press.

[18] Giuseppe Ateniese, Breno de Medeiros, and Michael T. Goodrich. TRICERT: A distributed certified E-mail scheme. In *Symposium on Network and Distributed Systems Security (NDSS 2001)*, San Diego, CA, February 2001. Internet Society.

[19] Alireza Bahreman and Doug Tygar. Certified electronic mail. In *Symposium on Network and Distributed Systems Security*, pages 3–19. Internet Society, February 1994.

[20] Feng Bao, Robert H. Deng, and Wenbo Mao. Efficient and practical fair exchange protocols with off-line TTP. In *IEEE Symposium on Research in Security and Privacy*, Research in Security and Privacy, pages 77–85. IEEE Computer Society,Technical Committee on Security and Privacy, IEEE Computer Security Press, May 1998.

[21] Feng Bao, Robert H. Deng, Khanh Quoc Nguyen, and Vijay Varadharajan. Multi-party fair exchange with an off-line trusted neutral party. In *DEXA 1999 Workshop on Electronic Commerce and Security*, Florence, Italy, September 1999.

[22] Birgit Baum-Waidner and Michael Waidner. Round-optimal and abuse free optimistic multi-party contract signing. In Ugo Montanari, José D. P. Rolim, and Emo Welzl, editors, *Automata, Languages and Programming — ICALP 2000*, volume 1853 of *Lecture Notes in Computer Science*, pages 524–535, Geneva, Switzerland, July 2000. Springer-Verlag.

[23] Giampaolo Bella and Lawrence C. Paulson. Kerberos version iv: inductive analysis of the secrecy goals. In *Computer Security — ESORICS 1998*, volume 1485 of *Lecture Notes in Computer Science*, pages 361–375, Louvain-la-Neuve, Belgium, September 1998. Springer-Verlag.

[24] Giampaolo Bella and Lawrence C. Paulson. Mechanical proofs about a non-repudiation protocol. In Richard J. Boulton and Paul B. Jackson, editors, *Theorem Proving in Higher Order Logics*, volume 2152 of *Lecture Notes in Computer Science*, pages 91–104. Springer-Verlag, 2001.

[25] Michael Ben-Or, Oded Goldreich, Silvio Micali, and Ronald L. Rivest. A fair protocol for signing contracts. *IEEE Transaction on Information Theory*, 36(1):40–46, January 1990.

[26] Manuel Blum. How to exchange (secret) keys. *ACM Transactions on Computer Systems*, 1(2):175–193, May 1983. Previously published in ACM STOC 1983 proceedings, pages 440–447.

[27] Alexandre Boisseau. *Abstractions pour la vérification de propriétés de sécurité de protocoles cryptographiques.* PhD thesis, École Normale Supérieure de Cachan, September 2003.

[28] Jean-Paul Boly, Antoon Bosselaers, Ronald Cramer, Rolf Michelsen, Stig Mjlsnes, Frank Muller, Torben Pedersen, Birgit Pfitzmann, Peter de Rooij, Berry Schoenmakers, Matthias Schunter, Luc Valle, and Michael Waidner. The ESPRIT project CAFE — high security digital payment systems. In *Computer Security — ESORICS 1994*, volume 875 of *Lecture Notes in Computer Science*, pages 217–230, Berlin, Germany, 1994.

[29] Dan Boneh and Moni Naor. Timed commitments. In *Advances in Cryptology—Crypto 2000*, volume 1880 of *Lecture Notes in Computer Science*, pages 236–254. Springer-Verlag, 2000.

[30] Colin Boyd and Ernest Foo. Off-line fair payment protocols using convertible signatures. In *Advances in Cryptology—Asiacrypt 1998*, volume 1514 of *Lecture Notes in Computer Science*, pages 271–285. Springer-Verlag, 1998.

[31] Colin Boyd and Peter Kearney. Exploring fair exchange protocols using specification animation. In Joseph Pieprzyk, Eiji Okamoto, and Jennifer Seberry, editors, *Information Security—International Workshop on Information Security*, volume 1975 of *Lecture Notes in Computer Science*, Wollogong, Australia, December 2000. Springer-Verlag.

[32] Steve Brackin, Catherine Meadows, and Jonathan K. Millen. CAPSL interface for the NRL protocol analyzer. In *IEEE Symposium on Application-Specific Systems and Software Engineering Technology (ASSET 1999)*, 1999.

[33] Randal E. Bryant. Graph-based algorithms for Boolean function manipulation. *IEEE Transactions on Computers*, C-35(8):677–691, August 1986.

[34] Michael Burrows, Martín Abadi, and Roger Needham. Authentication: A practical study of belief in action. In Moshe Vardi, editor, *Second Conference on Theoretical Aspects of Reasoning about Knowledge*, pages 325–342, March 1988.

[35] Michael Burrows, Martín Abadi, and Roger Needham. Authentication: A practical study of belief in action. In *1st IEEE Computer Security Foundations Workshop*, June 1988.

[36] Michael Burrows, Martín Abadi, and Roger Needham. A logic of authentication. Technical Report 39, Digital Systems Research Center, February 1989. Revised February 22, 1990.

[37] Michael Burrows, Martín Abadi, and Roger Needham. A logic of authentication. *Operating Systems Review*, 23(5):1–13, December 1989.

[38] Michael Burrows, Martín Abadi, and Roger Needham. A logic of authentication. *Proceedings of the Royal London Society. Series A. Mathematical and Physical Sciences*, 426(1871):233–271, dec 1989.

[39] Levente Buttyán and Jean-Pierre Hubaux. Rational exchange — a formal model based on game theory. In L. Fiege, G. Mühl, and U. Wilhelm, editors, *Electronic Commerce*, volume 2232 of *Lecture Notes in Computer Science*, pages 114–126, Heidelberg, Germany, 2001. Springer-Verlag.

[40] Levente Buttyán, Jean-Pierre Hubaux, and Srdjan Čapkun. A formal analysis of syverson's rational exchange protocol. In Steve A. Schneider, editor, *15th IEEE Computer Security Foundations Workshop*, pages 193–205, Cape Breton, Nova Scotia, Canada, June 2002. IEEE Computer Society Press.

[41] Levente Buttyán, Jean-Pierre Hubaux, and Srdjan Čapkun. A formal model of rational exchange and its application to the analysis of syverson's protocol. *Journal of Computer Security*, 2003. Special Issue of selected papers from CSFW15. To appear.

[42] Iliano Cervesato, Nancy A. Durgin, Patrick D. Lincoln, John C. Mitchell, and Andre Scedrov. A meta-notation for protocol analysis. In Paul Syverson, editor, *12th IEEE Computer Security Foundations Workshop*, pages 55–69, Modarno, Italy, June 1999. IEEE Computer Society Press.

[43] Rohit Chadha, Max Kanovich, and Andre Scedrov. Inductive methods and contract-signing protocols. In Pierangela Samarati, editor, *8th ACM Conference on Computer and Communications Security*, pages 176–185, Philadelphia, PA, USA, November 2001. ACM Press.

[44] Rohit Chadha, John C. Mitchell, Andre Scedrov, and Vitaly Shmatikov. Contract signing, optimism, and advantage. In R. Amadio and D. Lugiez, editors, *CONCUR 2003 — Concurrency Theory*, volume 2761 of *Lecture Notes in Computer Science*, pages 366–382, Marseille, France, September 2003. Springer-Verlag.

[45] David Chaum. Blind signatures for untraceable payments. In R. L. Rivest, A. Sherman, and D. Chaum, editors, *Crypto 1982*, pages 199–203, New York, 1983. Plenum Press.

[46] David Chaum. Designated confirmer signatures. In Alfredo De Santis, editor, *Advances in Cryptology—Eurocrypt 1994*, volume 950 of *Lecture Notes in Computer Science*, pages 86–91. Springer-Verlag, May 1994.

[47] David Chaum. Private signature and proof systems. United States Patent 5,493,614, 1996.

[48] Guang-Huei Chiou and Wen-Tsuen Chen. Secure broadcasting using the secure lock. *IEEE Transactions on Software Engineering*, 15(8):929–934, August 1989.

[49] Tom Coffey and Puneet Saidha. Non-repudiation with mandatory proof of receipt. *ACM Computer Communication Review*, 26, 1996.

[50] Hubert Comon-Lundh and Véronique Cortier. Security properties: two agents are sufficient. In Pierpaolo Degano, editor, *12th European Symposium on Programming*, volume 2618 of *Lecture Notes in Computer Science*, pages 99–113, Warsaw, Poland, April 2003. Springer-Verlag.

[51] Hubert Comon-Lundh and Vitaly Shmatikov. Is it possible to decide whether a cryptographic protocol is secure or not ? *Journal of Telecommunications and Information Technology, special issue on Cryptographic protocol verification*, 4:5–15, 2002.

[52] Ricardo Corin and Sandro Etalle. An improved constraint-based system for the verification of security protocols. In M. V. Hermenegildo and G. Puebla, editors, *Static Analysis Symposium—SAS 2002*, volume 2477 of *Lecture Notes in Computer Science*, pages 326–341, Madrid, Spain, September 2003. Springer-Verlag.

[53] Véronique Cortier, Jonathan K. Millen, and Harald Ruess. Proving secrecy is easy enough. In Steve Schneider, editor, *14th IEEE Computer Security Foundations Workshop*, Cape Breton, Nova Scotia, Canada, jun 2001. IEEE Computer Society Press.

[54] Benjamin Cox, Doug Tygar, and Marvin Sirbu. NetBill security and transaction protocol. In USENIX Association, editor, *First USENIX Workshop of Electronic Commerce*, pages 77–88. USENIX, July 1995.

[55] Robert H. Deng, Li Gong, Aurel A. Lazar, and Weiguo Wang. Practical protocols for certified electronic mail. *Journal of Network and System Management*, 4(3):279–297, 1996.

[56] Yvo Desmedt, Claude Goutier, and Samy Bengio. Special uses and abuses of the Fiat-Shamir passport protocol. In Carl Pomerance, editor, *Advances in Cryptology—Crypto 1987*, volume 293 of *Lecture Notes in Computer Science*, pages 21–39. Springer-Verlag, August 1987.

[57] Yvo Desmedt and Moti Yung. Weaknesses of undeniable signature schemes (extended abstract). In D. W. Davies, editor, *Advances in Cryptology—Eurocrypt 1991*, volume 547 of *Lecture Notes in Computer Science*, pages 205–220. Springer-Verlag, April 1991.

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

[59] Nancy A. Durgin, Patrick D. Lincoln, John C. Mitchell, and Andre Scedrov. Multiset rewriting and the complexity of bounded security protocols. *Journal of Computer Security*, to appear 2002.

[60] Shimon Even. Protocol for signing contracts. In Allen Gersho, editor, *Advances in Cryptology: A Report on Crypto 1981*, pages 148–153, 1982.

[61] Shimon Even, Oded Goldreich, and Abraham Lempel. A randomized protocol for signing contracts. *Communications of the ACM*, 28(6):637–647, June 1985.

[62] Shimon Even and Yacov Yacobi. Relations among public key signature systems. Technical Report 175, Technion, Haifa, Israel, March 1980.

[63] Josep Lluís Ferrer-Gomila, Magdalena Payeras-Capellà, and Llorenç Huguet i Rotger. An efficient protocol for certified electronic mail. In Joseph Pieprzyk, Eiji Okamoto, and Jennifer Seberry, editors, *Information Security—International Workshop on Information Security*, volume 1975 of *Lecture Notes in Computer Science*, pages 237–248, Wollogong, Australia, December 2000. Springer-Verlag.

[64] Michael J. Fischer, Nancy A. Lynch, and Mike S. Paterson. Impossibility of distributed consensus with one faulty processor. *Journal of the ACM*, 32(2):374–382, April 1985.

[65] Matthew K. Franklin and Michael K. Reiter. Fair exchange with a semi-trusted third party. In *4th ACM Conference on Computer and Communications Security*, pages 1–5. ACM Press, April 1997.

[66] Matthew K. Franklin and Gene Tsudik. Secure group barter: Multi-party fair exchange with semi-trusted neutral parties. In Ray Hirschfeld, editor, *Second Conference on Financial Cryptography (FC 1998)*, volume 1465 of *Lecture Notes in Computer Science*, pages 90–102, Anguilla, British West Indies, February 1998. International Financial Cryptography Association (IFCA), Springer-Verlag.

[67] Juan A. Garay, Markus Jakobsson, and Philip D. MacKenzie. Abuse-free optimistic contract signing. In *Advances in Cryptology—Crypto 1999*, volume 1666 of *Lecture Notes in Computer Science*, pages 449–466. Springer-Verlag, 1999.

[68] Juan A. Garay and Philip D. MacKenzie. Abuse-free multi-party contract signing. In Prasad Jayanti, editor, *International Symposium on Distributed Computing*, volume 1693 of *Lecture Notes in Computer Science*, pages 151–165, Bratislava, Slavak Republic, September 1999. Springer-Verlag.

[69] Juan A. Garay and Philip D. MacKenzie. Personnal communication, August 2003.

[70] Felix C. Gärtner, Henning Pagnia, and Holger Vogt. Approaching a formal definition of fairness in electronic commerce. In *International Workshop on Electronic Commerce (WELCOM 1999)*, pages 354–359, Lausanne, Switzerland, October 1999. IEEE Computer Society Press.

[71] Marc Girault. Self-certified public keys. In *Advances in Cryptology—Eurocrypt 1991*, volume 547 of *Lecture Notes in Computer Science*, pages 490–497. Springer-Verlag, 1991.

[72] Dieter Gollmann. What do we mean by entity authentication? In *Symposium on Security and Privacy*, pages 46–54, Oakland, California, USA, May 1996. IEEE Computer Society Press.

[73] Li Gong, Roger Needham, and Raphael Yahalom. Reasoning about belief in cryptographic protocols. In *IEEE Symposium on Research in Security and Privacy*, pages 234–248, Oakland, CA, May 1990. Technical Committee on Security and Privacy, IEEE Computer Society Press.

[74] Nicolás González-Deleito and Olivier Markowitch. Exclusion-freeness in multi-party exchange protocols. In *5th Information Security Conference*, volume 2433 of *Lecture Notes in Computer Science*, pages 200–209. Springer-Verlag, September 2002.

[75] Joshua Guttman and F. Javier Thayer Fabrega. Protocol independence through disjoint encryption. In *13th IEEE Computer Security Foundations Workshop*, pages 24–34, Cambridge, UK, July 2000. IEEE.

[76] Joshua D. Guttman. *Foundations of Security Analysis and Design*, volume 2171 of *Lecture Notes in Computer Science*, chapter Security Goals: Packet Trajectories and Strand Spaces, pages 197–261. Springer-Verlag, 2001.

[77] James A. Heather, Gavin Lowe, and Steve A. Schneider. How to prevent type flaw attacks on security protocols. In *13th Computer Security Foundations Workshop*, pages 255–268, Cambridge, UK, July 2000. IEEE Computer Society Press.

[78] James A. Heather and Steve A. Schneider. Towards automatic verification of authentication protocols on an unbounded network. In *13th Computer Security Foundations Workshop*, pages 132–143, Cambridge, UK, July 2000. IEEE Computer Society Press.

[79] Thomas A. Henzinger, Rupak Manjumdar, Freddy Y.C. Mang, and Jean-François Raskin. Abstract interpretation of game properties. In *SAS 2000: Intertional Symposium on Static Analysis*, Lecture Notes in Computer Science. Springer-Verlag, 2000.

[80] Markus Jakobsson. Blackmailing using undeniable signatures. In A. De Santis, editor, *Advances in Cryptology—Eurocrypt 1994*, volume 950 of *Lecture Notes in Computer Science*. Springer-Verlag, 1995.

[81] Markus Jakobsson. Ripping coins for fair exchange. In Louis C. Guillou and Jean-Jacques Quisquater, editors, *Advances in Cryptology—Eurocrypt 1995*, volume 921 of *Lecture Notes in Computer Science*, pages 220–230. Springer-Verlag, May 1995.

[82] Markus Jakobsson, Kazue Sako, and Russell Impagliazzo. Designated verifier proofs and their applications. In Ueli Maurer, editor, *Advances in Cryptology—Eurocrypt 1996*, volume 1070 of *Lecture Notes in Computer Science*, pages 143–154. Springer-Verlag, May 1996.

[83] Steve Kremer and Olivier Markowitch. A multi-party non-repudiation protocol. In *15th International Conference on Information Security—Sec 2000*, IFIP World Computer Congress, pages 271–280, Beijing, China, August 2000. Kluwer Academic.

[84] Steve Kremer and Olivier Markowitch. Optimistic non-repudiable information exchange. In J. Biemond, editor, *21st Symp. on Information Theory in the Benelux*, pages 139–146, Wassenaar, The Netherlands, May 2000. Werkgemeenschap Informatie- en Communicatietheorie, Enschede.

[85] Steve Kremer and Olivier Markowitch. Selective receipt in certified e-mail. In *Advances in Cryptology—Indocrypt 2001*, Lecture Notes in Computer Science. Springer-Verlag, December 2001.

[86] Steve Kremer and Olivier Markowitch. Fair multi-party non-repudiation. *International Journal on Information Security*, 1(4):223–235, July 2003.

[87] Steve Kremer, Olivier Markowitch, and Jianying Zhou. An intensive survey of non-repudiation protocols. *Computer Communications*, 25(17):1606–1621, November 2002.

[88] Steve Kremer and Jean-François Raskin. A game-based verification of non-repudiation and fair exchange protocols. In Kim G. Larsen and Mogens Nielsen, editors, *Concurrency Theory—CONCUR 2001*, volume 2154 of *Lecture Notes in Computer Science*, pages 551–565, Aalborg, Denmark, August 2001. Springer-Verlag.

[89] Steve Kremer and Jean-François Raskin. Game analysis of abuse-free contract signing. In Steve A. Schneider, editor, *15th IEEE Computer Security Foundations Workshop*, pages 206–220, Cape Breton, Nova Scotia, Canada, June 2002. IEEE Computer Society Press.

[90] Steve Kremer and Jean-François Raskin. A game-based verification of non-repudiation and fair exchange protocols. *Journal of Compuiter Security*, 11(3):399–429, 2003.

[91] Gavin Lowe. An attack on the Needham-Schroeder public-key authentication protocol. *Information Processing Letters*, 56(3):131–133, November 1995.

[92] Gavin Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using FDR. In *Tools and Algorithms for the Construction and Analysis OF Systems—TACAS 1996*, volume 1055 of *Lecture Notes in Computer Science*, pages 147–166. Springer-Verlag, 1996.

[93] Gavin Lowe. Casper: a compiler for the analysis of security protocols. In Simon Foley, editor, *10th Computer Security Foundations Workshop*, pages 18–30, Rockport, Massachusetts, USA, June 1997. IEEE Computer Society Press.

[94] Gavin Lowe. A hierarchy of authentication specifications. In Simon Foley, editor, *10th Computer Security Foundations Workshop*, Rockport, Massachusetts, USA, June 1997. IEEE Computer Society Press.

[95] Gavin Lowe. Towards a completeness result for model checking of security protocols. In Simon Foley, editor, *11th IEEE Computer Security Foundations Workshop*, pages 96–105, Rockport, Massachusetts, USA, June 1998. IEEE Computer Society.

[96] Olivier Markowitch. *Les protocoles de Non-répudiation*. PhD thesis, Université Libre de Bruxelles, January 2001.

[97] Olivier Markowitch, Dieter Gollmann, and Steve Kremer. On fairness in exchange protocols. In Pil Joong Lee and Chae Hoon Lim, editors, *5th International Conference on Information Security and Cryptology (ICISC 2002)*, volume 2587 of *Lecture Notes in Computer Science*, pages 451–464, Seoul, Korea, November 2002. Springer-Verlag.

[98] Olivier Markowitch and Steve Kremer. A multi-party optimistic non-repudiation protocol. In Dongho Won, editor, *3rd International Conference on Information Security and Cryptology (ICISC 2000)*, volume 2015 of *Lecture Notes in Computer Science*, pages 109–122, Seoul, Korea, December 2000. Springer-Verlag.

[99] Olivier Markowitch and Steve Kremer. An optimistic non-repudiation protocol with transparent trusted third party. In *Information Security Conference 2001*, volume 2200 of *Lecture Notes in Computer Science*, pages 363–378, Malaga, Spain, October 2001. Springer-Verlag.

[100] Olivier Markowitch and Yves Roggeman. Probabilistic non-repudiation without trusted third party. In *Second Conference on Security in Communication Networks 1999*, Amalfi, Italy, September 1999.

[101] Olivier Markowitch and Shahrock Saeednia. Cryptanalysis of the Wu-Varadharajan fair exchange protocol. *Information Processing Letter*, 87(3):169–171, August 2003.

[102] Olivier Markowitch and Shahrokh Saeednia. Optimistic fair-exchange with transparent signature recovery. In *5th International Conference, Financial Cryptography 2001*, Lecture Notes in Computer Science. Springer-Verlag, 2001.

[103] MasterCard and Visa. *SET Secure Electronic Transactions Protocol*, version 1.0 edition, May 1997. Book One: Business Specifications, Book Two: Technical Specification, Book Three: Formal Protocol Definition. Available from `http://www.setco.org/set_specifications.html`.

[104] Catherine Meadows. A system for the specification and verification of key management protocols. In *IEEE Computer Society Symposium on Research in Security and Privacy*, pages 182–197, Washington - Brussels - Tokyo, May 1991. IEEE.

[105] Catherine Meadows. Analyzing the Needham-Schroeder public key protocol: A comparison of two approaches. In *Proc. ESORICS 96*, pages 351–364. Springer Verlag, 1996.

[106] Catherine Meadows. The NRL protocol analyzer: An overview. *Journal of Logic Programming*, 26(2):113–131, February 1996.

[107] Catherine Meadows. Analysis of the Internet Key Exchange protocol using the NRL protocol analyzer. In *IEEE Symposium on Research in Security and Privacy*, pages 48–61, Oakland, CA, May 1999. IEEE Computer Society, Technical Committee on Security and Privacy, IEEE Computer Society Press.

[108] Catherine Meadows. Extending formal cryptographic protocol analysis techniques for group protocols and low-level cryptographic primitives. In Pierpaolo Degano, editor, *Workshop on Issues in the Theory of Security (WITS 2000)*, Geneva, Switzerland, July 2000.

[109] Catherine Meadows and Paul Syverson. A formal specification of requirements for payment transactions in the SET protocol. In Ray Hirschfeld, editor, *Second Conference on Financial Cryptography (FC 1998)*, volume 1465 of *Lecture Notes in Computer Science*, pages 122–140, Anguilla, British West Indies, February 1998. International Financial Cryptography Association (IFCA), Springer-Verlag.

[110] Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography.* Series on Discrete Mathematics and its Applications. CRC Press, 1997.

[111] Silvio Micali. Certified E-mail with invisible post offices. Available from author; an invited presentation at the RSA 1997 conference, 1997.

[112] Jonathan Millen and Vitaly Shmatikov. Constraint solving for bounded-process cryptographic protocol analysis. In Pierangela Samarati, editor, *8th ACM Conference on Computer and Communications Security*, pages 166–175, Philadelphia, PA, USA, November 2001. ACM Press.

[113] Jonathan K. Millen. CAPSL Web site. `http://www.csl.sri.com/~millen/capsl`.

[114] Jonathan K. Millen. A CAPSL connector to Athena. In Helmut Veith, Nevin Heintze, and Edmund Clarke, editors, *Workshop of Formal Methods and Computer Security*, 2000.

[115] Robin Milner. *Communicating and mobile systems: the π-calculus.* Cambridge University Press, 1999.

[116] John C. Mitchell, Mark Mitchell, and Ulrich Stern. Automated analysis of cryptographic protocols using Murφ. In *IEEE Symposium on Security and Privacy*, pages 141–153, 1997.

[117] John Mitsianis. A new approach to enforcing non-repudiation of receipt. Manuscript, 2001.

[118] Moni Naor and Kobbi Nissim. Certificate revocation and certificate update. In *7th USENIX Security Symposium (SECURITY 1998)*, pages 217–228. Usenix Association, January 1998.

[119] Roger M. Needham and Michael D. Schroeder. Using encryption for authentication in large networks of computers. *Communications of the ACM*, 21(12):993–999, 1978.

[120] Dan M. Nessett. A critique of the Burrows, Abadi and Needham logic. *ACM Operating Systems Review*, 24(2):35–38, 1990.

[121] Gethin Norman and Vitaly Shmatikov. Analysis of probabilistic contract signing. In *Formal Aspects of Security*, 2002.

[122] Jose Onieva, Jianying Zhou, Mildrey Carbonell, and Javier Lopez. A multi-party non-repudiation protocol for exchange of different messages. In *18th IFIP International Information Security Conference*, Athens, Greece, May 2003. Kluwer.

[123] Henning Pagnia and Felix C. Gärtner. 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, Darmstadt, Germany, March 1999.

[124] Henning Pagnia, Holger Vogt, and Felix C. Gärtner. Fair exchange. *The Computer Journal*, 8(2):55–75, January 2003.

[125] Lawrence C. Paulson. *Isabelle: A Generic Theorem Prover*, volume 828 of *Lecture Notes in Computer Science*. Springer-Verlag, 1994.

[126] Lawrence C. Paulson. Proving properties of security protocols by induction. In *10th IEEE Computer Security Foundations Workshop*, pages 70–83, Washington - Brussels - Tokyo, June 1997. IEEE.

[127] Lawrence C. Paulson. The inductive approach to verifying cryptographic protocols. *Journal of Computer Security*, 6(1/2):85–128, 1998.

[128] Lawrence C. Paulson. Verifying the SET protocol. In *International Joint Conference on Automated Reasoning — IJCAR 2001*, June 2001.

[129] Judy Pearsall, editor. *The concise Oxford Dictionary.* Oxford University Press, tenth edition, 1999.

[130] Birgit Pfitzmann, Matthias Schunter, and Michael Waidner. Optimal efficiency of optimistic contract signing. In *Seventeenth Annual ACM Symposium on Principles of Distributed Computing*, pages 113–122, New York, May 1998. ACM.

[131] Guillaume Poupard and Jacques Stern. Security analysis of a practical "on the fly" authentication and signature generation. In *Advances in Cryptology—Eurocrypt 1998*, volume 1403 of *Lecture Notes in Computer Science*, pages 422–436. Springer-Verlag, 1998.

[132] Michael O. Rabin. Transaction protection by beacons. *Journal of Computer and System Sciences*, 27(2):256–267, October 1983.

[133] James Riordan and Bruce Schneier. A certified E-mail protocol. In *14th Annual Computer Security Applications Conference*. ACM, 1998.

[134] Ronald L. Rivest, Adi Shamir, and David A. Wagner. Time-lock puzzles and timed-release crypto. Technical Report MIT/LCS/TR-684, Massachusetts Institute of Technology, February 1996.

[135] A. W. Roscoe. Modelling and verifying key-exchange protocols using CSP and FDR. In *8th IEEE Computer Security Foundations Workshop*, pages 98–107. IEEE Computer Society, 1995.

[136] A. W. Roscoe. Intensional specifications of security protocols. In *9th IEEE Computer Security Foundations Workshop*, pages 28–38. IEEE Computer Security Press, 1996.

[137] Rainer A. Rueppel. Revocation and revocation certificates. In *Trusted Third Party Workshop*, pages 1–8, Barcelona, Spain, February 1995.

[138] Peter Ryan, Steve A. Schneider, Michael H. Goldsmith, Gavin Lowe, and A. W. Roscoe. *Modelling and Analysis of Security Protocols*. Addison-Wesley, 2001.

[139] Shahrokh Saeednia, Steve Kremer, and Olivier Markowitch. Efficient designated verifier signatures. In Ludo Tolhuyzen, editor, *Twenty-fourth Symposium on Information Theory in the Benelux*, pages 187–194. Werkgemeenschap Informatie- en Communicatietheorie, Enschede (NL), May 2003.

[140] Steve A. Schneider. Verifying authentication protocols with CSP. In Simon Foley, editor, *10th IEEE Computer Security Foundations Workshop*, pages 3–17, Rockport, Massachusetts, USA, June 1997. IEEE Computer Society Press.

[141] Steve A. Schneider. Formal analysis of a non-repudiation protocol. In *11th IEEE Computer Security Foundations Workshop*, pages 54–65, Washington - Brussels - Tokyo, June 1998. IEEE.

[142] Steve A. Schneider and Peter Ryan. An attack on a recursive authentication protocol: A cautionary tale. *Information Processing Letters*, 65:7–10, 1998.

[143] Bruce Schneier. *Applied Cryptography*. John Wiley & Sons, 1996.

[144] Matthias Schunter. *Optimistic Fair Exchange*. PhD thesis, Technische Fakultät der Universität des Saarlandes, Saarbrücken, October 2000.

[145] Vitaly Shmatikov. Probabilistic analysis of anonymity. In Steve A. Schneider, editor, *15th IEEE Computer Security Foundations Workshop*, pages 119–128, Cape Breton, Nova Scotia, Canada, June 2002. IEEE Computer Society Press.

[146] Vitaly Shmatikov and John Mitchell. Analysis of a fair exchange protocol. In *Symposium on Network and Distributed Systems Security (NDSS 2000)*, pages 119–128, San Diego, CA, February 2000. Internet Society.

[147] Vitaly Shmatikov and John Mitchell. Analysis of abuse-free contract signing. In *Financial Cryptography 2000*, volume 1962 of *Lecture Notes in Computer Science*, pages 174–191, Anguilla, British West Indies, February 2000. Springer-Verlag.

[148] Vitaly Shmatikov and John Mitchell. Finite-state analysis of two contract signing protocols. *Theoretical Computer Science, special issue on Theoretical Foundations of Security Analysis and Design*, 283(2):419–450, 2002.

[149] Vitaly Shmatikov and Ulrich Stern. Efficient finite-state analysis for large security protocols. In *11th IEEE Computer Security Foundations Workshop*, pages 106–115, Washington - Brussels - Tokyo, June 1998. IEEE.

[150] Dawn Song. Proving properties of security protocols by induction. In *12th IEEE Computer Security Foundations Workshop*, Washington - Brussels - Tokyo, June 1999. IEEE.

[151] Scott D. Stoller. A bound on attacks on payment protocols. In *Logic in Computer Science*, pages 61–70, Washington - Brussels - Tokyo, June 2001. IEEE Computer Society Press.

[152] Paul Syverson. Weakly secret bit commitment: Applications to lotteries and fair exchange. In *11th IEEE Computer Security Foundations Workshop*, June 1998.

[153] Paul Syverson. Weakly secret bit commitment: Applications to lotteries and fair exchange. In *11th IEEE Computer Security Foundations Workshop*, pages 2–13, June 1998.

[154] Paul Syverson and Illiano Cervesato. *Foundations of Security Analysis and Design*, volume 2171 of *Lecture Notes in Computer Science*, chapter The Logic of Authentication Protocols, pages 63–136. Springer-Verlag, 2001.

[155] Paul Syverson and Catherine Meadows. A formal language for cryptographic protocol requirements. *Designs, Codes and Cryptography*, 7(1/2):27–59, January 1996.

[156] Paul Syverson and Paul C. van Oorschot. On unifying some cryptographic protocol logics. In *IEEE Symposium on Research in Security and Privacy*, pages 14–28, Oakland, CA, May 1994. IEEE Computer Society, Technical Committee on Security and Privacy, IEEE Computer Society Press.

[157] Tom Tedrick. How to exchange half a bit. In David Chaum, editor, *Advances in Cryptology—Crypto 1983*, pages 147–151. Plenum Press, August 1983.

[158] Tom Tedrick. Fair exchange of secrets. In G. R. Blakley and D. C. Chaum, editors, *Advances in Cryptology—Crypto 1984*, volume 196 of *Lecture Notes in Computer Science*, pages 434–438. Springer-Verlag, 1985.

[159] F. Javier Thayer Fabrega, Jonathan C. Herzog, and Joshua D. Guttman. Honest ideals on strand spaces. In *11th IEEE Computer Security Foundations Workshop*, pages 66–78, Washington - Brussels - Tokyo, June 1998. IEEE.

[160] F. Javier Thayer Fabrega, Jonathan C. Herzog, and Joshua D. Guttman. Strand spaces: Why is a security protocol correct ? In *IEEE Symposium on Security and Privacy*, pages 24–34, Washington - Brussels - Tokyo, May 1998. IEEE.

[161] F. Javier Thayer Fabrega, Jonathan C. Herzog, and Joshua D. Guttman. Mixed strand spaces. In *12th IEEE Computer Security Foundations Workshop*, pages 72–82, Washington - Brussels - Tokyo, June 1999. IEEE.

[162] F. Javier Thayer Fabrega, Jonathan C. Herzog, and Joshua D. Guttman. Strand spaces: Proving security protocols correct. *Journal of Computer Security*, 7(2/3):191–230, 1999.

[163] Mathieu Turuani and Michaël Rusinowitch. Protocol insecurity with finite number of sessions is np-complete. In Steve Schneider, editor, *14th IEEE Computer Security Foundations Workshop*, pages 174–187, Cape Breton, Nova Scotia, Canada, jun 2001. IEEE Computer Society Press.

[164] Laurent Van Begin. *Efficient Verification of Counting Abstractions for Parametric Systems*. PhD thesis, Université Libre de Bruxelles, October 2003.

[165] Paul C. van Oorschot. Extending cryptographic logics of belief to key agreement protocols. In Victoria Ashby, editor, *1st ACM Conference on Computer and Communications Security*, pages 232–243, Fairfax, Virginia, November 1993. ACM Press.

[166] Holger Vogt. Asynchronous optimistic fair exchange based on revocable items. In *Financial Cryptography 2003*, Lecture Notes in Computer Science, Gosier, Guadeloupe, January 2003. Springer-Verlag.

[167] Holger Vogt, Henning Pagnia, and Felix C. Gärtner. Modular fair exchange protocols for electronic commerce. In *15th Annual Computer Security Applications Conference*, pages 3–11, Phoenix, Arizona, December 1999. IEEE Computer Society Press.

[168] Chuan-Kun Wu and Vijay Varadharajan. Fair exchange of digital signatures with off-line trusted third party. In Sihan Qing, Tatsuaki Okamoto, and Jianying Zhou, editors, *Information and Communications Security*, volume 2229 of *Lecture Notes in Computer Science*, pages 466–470, Xian, China, November 2001. Springer-Verlag.

[169] Cheng-Hwee You, Jianying Zhou, and Kwok-Yan Lam. On the efficient implementation of fair non-repudiation. *Computer Communication Review*, 28(5):50–60, October 1998.

[170] Ning Zhang and Qi Shi. Achieving non-repudiation of receipt. *The Computer Journal*, 39(10):844–853, 1996.

[171] Jianying Zhou. *Non-repudiation*. PhD thesis, University of London, December 1996.

[172] Jianying Zhou. *Non-repudiation in electronic commerce*. Computer Security Series. Artech House, August 2001.

[173] Jianying Zhou and Robert H. Deng. On the validity of digital signatures. *Computer Communication Review*, 30(2):29–34, April 2000.

[174] Jianying Zhou, Robert H. Deng, and Feng Bao. Evolution of fair non-repudiation with TTP. In *ACISP: Information Security and Privacy: Australasian Conference*, volume 1587 of *Lecture Notes in Computer Science*, pages 258–269. Springer-Verlag, 1999.

[175] Jianying Zhou, Robert H. Deng, and Feng Bao. Some remarks on a fair exchange protocol. In *International Workshop on Practice and Theory in Public Key Cryptography*, volume 1751 of *Lecture Notes in Computer Science*, pages 46–57. Springer-Verlag, January 2000.

[176] Jianying Zhou and Dieter Gollmann. Certified electronic mail. In *Computer Security — ESORICS 1996*, volume 1146 of *Lecture Notes in Computer Science*, pages 160–171, 1996.

[177] Jianying Zhou and Dieter Gollmann. A fair non-repudiation protocol. In *IEEE Symposium on Research in Security and Privacy*, Research in Security and Privacy, pages 55–61. IEEE Computer Society,Technical Committee on Security and Privacy, IEEE Computer Security Press, May 1996.

[178] Jianying Zhou and Dieter Gollmann. An efficient non-repudiation protocol. In *10th IEEE Computer Security Foundations Workshop*, pages 126–132. IEEE Computer Society Press, June 1997.

[179] Jianying Zhou and Dieter Gollmann. Evidence and non-repudiation. *Journal of Network and Computer Applications*, 20:267–281, 1997.

[180] Jianying Zhou and Dieter Gollmann. Towards verification of non-repudiation protocols. In *International Refinement Workshop and Formal Methods Pacific*, pages 370–380, Canberra, Australia, September 1998. Springer-Verlag.

[181] Jianying Zhou and Kwok-Yan Lam. Securing digital signatures for non-repudiation. *Computer Communications*, 22(8):710–716, May 1999.