Foundations of

Security Protocol Analysis

Thesis submitted in
Partial Fulfilment of the
Degree of Doctor of Philosophy (Ph.D.)

by

S.P.Suresh
Theoretical Computer Science Group
The Institute of Mathematical Sciences
Chennai 600 113.

UNIVERSITY OF MADRAS
Chennai 600 005

November 2003

Dedicated to my father
and to the memory of my mother

DECLARATION

I declare that the thesis entitled **"Foundations of Security Protocol Analysis"** submitted by me for the Degree of Doctor of Philosophy is the record of work carried out by me during the period from August 1999 to October 2003 under the guidance of Dr. R. Ramanujam and has not formed the basis for the award of any degree, diploma, associateship, fellowship, titles in this or any other University or other similar institution of higher learning.

November 2003 S.P.Suresh

The Institute of Mathematical Sciences
C.I.T. Campus, Taramani
Chennai 600 113, India.

CERTIFICATE

This is to certify that the Ph.D. thesis submitted by S.P.Suresh to the University of Madras, entitled **"Foundations of Security Protocol Analysis"**, is a record of bonafide research work done during the period 1999-2003 under my guidance and supervision. The research work presented in this thesis has not formed the basis for the award of any degree, diploma, associateship, fellowship, titles in this University or any other University or Institution of Higher Learning.

It is further certified that the thesis represents independent work by the candidate and colloboration when existed was necessitated by the nature and scope of problems dealt with.

R.Ramanujam
Thesis Supervisor

November 2003

The Institute of Mathematical Sciences
C.I.T. Campus, Taramani
Chennai 600 113, India.

Abstract

In this thesis, we study one of the central problems in the automatic verification of security protocols, that of verifying whether a given protocol leaks secrets or not. The central work in the thesis identifies syntactic subclasses of protocols for which the secrecy problem is decidable. The other work in the thesis concerns reasoning about protocols. We introduce a logic using which interesting properties of protocols can be specified and reasoned about.

We start the study by setting up a formal model of security protocols, and proving several important properties about the model. Of particular importance are the properties relating to synth and analz proofs, which formalise the way the agents running a protocol derive new information from old.

We then consider the general secrecy problem and show that it is undecidable both when the set of nonces is infinite (a result first proved in [DLMS99]) and when the length of messages is unbounded (a result proved in [HT96]). We provide relatively simple and uniform proofs for both these results.

We then consider the secrecy problem in the setting of infinitely many nonces but bounded message length. We prove that for a certain syntactic subclass of protocols called *tagged protocols*, the secrecy problem in this setting is decidable.

We then prove that a tagged protocol has a leaky run (a run that leaks a secret) iff it has a leaky run containing only bounded length messages. This enables us to prove that the secrecy problem for tagged protocols is decidable even in the setting where both message length and number of nonces is unbounded.

We finally look at reasoning about security protocols. We define a logic in which we can easily specify several interesting security properties like secrecy, authenticity, etc. We also show some examples which illustrate how to reason about protocols. We then extend some of the undecidability and decidability results of the earlier chapters to the verification problem of the logic.

Acknowledgements

I would like to thank Dr. R. Ramanujam for his guidance and encouragement through the last six years. All the work presented herein was done with him. He has shown remarkable patience with me from the time we embarked on this work together, around four years ago. I have learnt a lot from him in the course of this association. I am especially indebted to him for having taught me most of what I know about logic, and about how to do research in general. The many discussions I have had with him were most enjoyable. I thank him also for his suggestions which have greatly improved this thesis over its earlier versions.

I thank the people in the Theoretical Computer Science groups at IMSc and CMI for welcoming me into their fold. The time I spent here during course-work was one of the most enriching in my life. Learning the many beautiful things in computer science from the people here was a highly enjoyable experience. I am very grateful to all of them — R. Ramanujam, Kamal Lodaya, V. Arvind, Venkatesh Raman, Anil Seth, Meena Mahajan, Madhavan Mukund, Narayan Kumar, P.S.Thiagarajan, and K.V. Subrahmanyam.

I would like to thank Dr. R. Balasubramanian, Director, IMSc, for allowing me to stay beyond my term. I also thank the office and library staff for their friendly and patient help in all matters.

I would also like to thank the organisers of the Marktoberdorf Summer School 2001, the organisers of FOSAD 2002, and the organisers of ETAPS 2003 for providing financial support for my visits to Germany, Italy, and Poland, respectively. I thank Dr. Fabio Martinelli for supporting a visit to I.I.T., Pisa in Italy, and for the useful discussions I had with him. I also thank Dr. Ramesh Bharadwaj of NRL for providing a major part of the financial assistance for my visit to ETAPS 2003.

I would like to record my humble *namaskaarams* to my father for providing me with all the good things in life which he himself couldn't enjoy, and for putting up with me patiently through the years. The period of my stay at Matscience was highly enriching in a different way — it gave me the time to embark on a serious study of Sanskrit, along with my father. My father was mainly instrumental in my efforts in this direction. I would like to also record my salutations to my *gurunaatha*, *Veda Bhashya Ratnam* Dr. R. Krishnamurthi Shastrigal, and his son *Vidvat pravara* Dr. K. Ramasubramaniam for introducing me to the joys of classical Indian philosophy and logic.

I would like to specially thank Dr. Hemalatha Thiagarajan, who has taught me so much in my days I spent at the Regional Engineering College, Tiruchi. She was the person who encouraged me most to take up a career in research. She was one of the few people who supported my move from the software industry to academics. She has been a constant source of support and encouragement over the years. I have also enjoyed many a pleasant stay at her place over the years.

I thank the many friends at Matscience who have made my stay here enjoyable. I had lots of fun with Madhu (from *Ponniyin Selvan* to heated debates on Gödel's theorem). His company was very enjoyable. Meenakshi has been a constant companion from my first days here, both during course-work and after. I enjoyed the many discussions we used to have on topics both academic and nonacademic (especially her tips on cooking). Deepak has also been a positive influence from my early days here. I also had a lot of fun with Rahul Muthu. I thank him for the many chess games. (Hope I am around to harass him during *his* final month at IMSc.) I also thank the other students whose company I enjoyed — Gyan Prakash, S.V.Nagaraj, Vinodchandran, Srinivasa Rao, Piyush, Vijayaraghavan, Narayanan, Jayalal, M. Rajesh, Murugesh and K.R.S. Balaji.

I would like to specially thank the following close friends from my R.E.C. days, who have continued to support and encourage me in all my endeavours, and who have had a most positive influence on me. They would be most happy to see the fruition of my efforts. They are — Shalini Ranganathan, P.V. Kamal Narayan, G. Ramesh, M. Narayanan, and B. Raghuram.

Contents

Chapter 1

Introduction

1.1 Background

Computer security has come to occupy an increasingly central place in our lives over the past twenty years. This has been a direct result of the enormous increase in the development and use of networked and distributed systems over this period. Financial transactions on the Internet is gaining currency now. Distributed financial transactions — even if they are in the simple form of withdrawing money from an ATM — have become part of many peoples' lives today. Even more pervasive is the routine use of electronic mail (which is sometimes even used to share confidential information). The consequences of a misuse of such systems are potentially disastrous. This places a high premium on ensuring that such systems are not misused.

Security can basically be considered as a study of what the potential misuses of such systems are and how they can be averted. A system may be said to be secure if the properties of *confidentiality*, *integrity*, *availability*, *authenticity*, etc. of the various system entities are maintained. Broadly speaking, a system maintains confidentiality if no information can be *accessed* except by those entities which are authorised to access it. Similarly, a system maintains integrity if no information can be *altered* except by those entities which are authorised to alter it. Availability simply means that the desired information (or resource) is available when desired. An entity is said to be authentic if its apparent identity is genuine, i.e., the entity in question does not masquerade as some other entity.

The main challenge in security is to maintain some (or all) of the above attributes in the presence of malicious users, accidental misuse or under some kinds of system failures.

Historically, many different traditions have contributed to developments in computer security. Developments in operating systems, military security, and cryptography have all driven advances in security.

From its early days, research in security has focused on *formal methods* for proving systems correct. This is easily understandable, since the consequences of a security-related error in a system could be disastrous, and thus the utmost care is required in ensuring the security of systems. Formal methods are a useful aid in the design and analysis of such systems.

Research on formal methods related to security has grown so much over the years that it is no longer possible to consider it as a unified whole. Based on the differences in the focus of research and the techniques and tools used, we have several subdisciplines. Our contributions in this thesis lie in the area of security protocols, which we look at in detail in the following sections. Meanwhile, we briefly look at some of the other disciplines below.

Program security: This is a classic area of study in security. The fundamental focus of research in this area is to devise methods which ensure that no program learns information that it is not authorised to know. Examples of programs which learn information in such an unauthorised manner are *viruses* and *Trojan horses*. For high-security systems like those used in the military, it is highly important to check all the programs to see if they have *secure information flow*. Formal methods are of immense help here. The fundamental theoretical problem studied here is whether a given problem has secure information flow ([BL73], [Den77]). A simple definition of a program having secure information flow is as follows: if the variables used in the program are partitioned into high-security and low-security variables, observations of the low-security variables do not reveal any information about the initial values of the high-security variables. Closely related is the problem of detecting *covert flows* [Lam73], where information is leaked indirectly, through variations in program behaviour. The research in this area has focussed on syntactic mechanisms (like typing, see [VSI96] for instance) and semantic methods (see [LJ00], for example), to ensure secure information flows in programs and to detect in-

formation leaks.

Security policy: This is another widely studied area in security, which has its origins in the access control model for confidentiality used in operating systems (see [Lam74], for instance). The central problem here is somewhat similar to that in program security, but is more general. The focus is on ensuring that there is no unauthorised access to information. Most of the solutions depend on restricting the behaviour of the system to achieve security. A classic example is *multilevel security*. Let us assume for simplicity that there are two user levels: *high* and *low*. Let us also assume that there are two security levels for objects: *confidential* and *public*. The typical restrictions on such a system might include *no read-up*: a low user cannot read a confidential file, and *no write-down*: a high user cannot write to a public file. Note that these are restrictions on the *run-time* behaviour of the systems. The fundamental theoretical challenge is to come up with good *security policy models*, which are formal specifications of the desired security-related behaviour of systems. [BL73] and [HRU76] are two early papers dealing with security models. They propose models for confidentiality which are directly based on access control models for operating systems. The model proposed in [BL73] has features for access control as well as *multilevel security*. The current trend of research in this area is to use more abstract models based on the so called *interface models*, which derive from [GM82]. See [McL94] for a good survey of security models.

Database security: The main focus in this line of research is the same as that of the above two — to ensure that every piece of *information* in a database is learnt only by users authorized to know it. This implies much more than protecting *data*, which can be implemented by some kind of access control mechanism. A simple example to illustrate this point involves a salary database where salaries above a certain threshold have to be kept secret. It is easy enough to prevent queries from directly accessing the records which have salary above the given threshold. But there are other kinds of information which could be learned, like the average or sum of the salaries above the threshold. In such cases, it is possible that information about individual records can be inferred by cleverly asking many queries. For instance, if S is a set of employees and $S' = S \cup \{a\}$, then by learning the sum of the salaries of

the employees in S, and the same for the employees in S', a's salary can be learned. In some cases, even the fact that there exists a record of a particular kind is vital information, even if the exact data cannot be accessed. In most of these cases, the operation of aggregation introduces much complexity in the system, by introducing many potential means to learn information. Much of the research has focussed on statistical techniques to prevent the inference of information. A brief introduction to the field (as also a general insight into computer security) can be had from [Gol99].

1.2 Security protocols

Security protocols are specifications of communication patterns which are intended to let agents share secrets over a public network. They are required to perform correctly even in the presence of malicious intruders who listen to the message exchanges that happen over the network and also manipulate the system (by blocking or forging messages, for instance). Obvious correctness requirements include secrecy: an intruder cannot read the contents of a message intended for others, and authenticity: if B receives a message that appears to be from agent A and intended for B, then A indeed sent the same message intended for B in the recent past.

The presence of intruders necessitates the use of encrypted communication. Thus developments in the field of cryptography provide the foundation for the design of security protocols. Research in cryptography has a long and glorious history. The field has come into its own in the past century, with more and more sophisticated mathematical techniques used to develop more and more sophisticated cryptographic schemes. As a result, a wide variety of cryptographic tools are available to the security protocol designer: conventional (shared-key) cryptography, public-key cryptography, digital signature schemes, etc.

The operation of encryption typically involves transforming a given *plaintext* to a *ciphertext* with the use a *key*, such that given the key it is easy to compute the ciphertext from the plaintext and vice versa, and without the key it is hard to compute the plaintext from the ciphertext. The inverse operation of computing the plaintext given the ciphertext and the key, is called *decryption*. The ciphertext is intended to be communicated over a possibly insecure network. Conventional

cryptography uses the same key for both encryption and decryption. Public-key cryptography systems ([DH76], [RSA78]) use a pair of keys for each user of the system (the user's public and private keys), where messages are encrypted using the receiver's public key and decrypted using the receiver's private key. A comprehensive introduction to cryptography can be had from [Sch96b].

Research in cryptography primarily aims at developing new cryptosystems with improved mathematical guarantees. But the focus of research in security protocols is different. It has been widely acknowledged that even the use of the most perfect cryptographic tools does not always ensure the desired security goals. (See [AN95] for an illuminating account.) This situation arises primarily because of logical flaws in the design of protocols.

Quite often, protocols are designed with features like ease of use, efficiency etc. in mind, in addition to some notion of security. For instance, if every message of a protocol were signed in the sender's name and then encrypted with the receiver's public key, it appears as if a lot of the known security flaws do not occur. But it is not usual for every message of a protocol to be signed. This could either be for reasons of efficiency or because frequent use of certain long-term keys might increase the chance of their being broken using cryptanalysis. Great care needs to be exercised in such situations. The following example protocol highlights some of the important issues nicely. It is based on a protocol designed by Needham and Schroeder ([NS78]) and is aimed at allowing two agents A and B to exchange two independent, secret numbers. It uses public-key encryption but does not require agents to sign their messages.

$$\begin{array}{lllll} \text{Msg 1.} & A & \rightarrow & B & : \{x, A\}_{pubk_B} \\ \text{Msg 2.} & B & \rightarrow & A & : \{x, y\}_{pubk_A} \\ \text{Msg 3.} & A & \rightarrow & B & : \{y\}_{pubk_B} \end{array}$$

Here $pubk_A$ and $pubk_B$ are the public keys of A and B, respectively, and $\{x\}_k$ is the notation used to denote x encrypted using key k. In the protocol, x and y are assumed to be newly generated, unguessable (with high probability, of course!), previously unused numbers, also called nonces (nonce stands for "*n*umber *once* used"). In message 2, B includes A's nonce. On seeing it A is assured that B has received message 1, since only B can decrypt the first message and use x in a later message. Similarly on receipt of the third message, B is assured of A's receipt of y.

At the end of a session of the protocol, both A and B share the secrets x and

y and both also know that the other agent knows x and y. But it has been shown ([Low96]) that x and y are not necessarily known *only to A and B*. (Such a property needs to be satisfied if we want to use a combination of x and y as a key shared between A and B, for example.) The attack (called Lowe's attack) is given below:

$$\begin{array}{lrcll}
\text{Msg } \alpha.1. & A & \rightarrow & I & : \{x, A\}_{pubk_I} \\
\text{Msg } \beta.1. & (I)A & \rightarrow & B & : \{x, A\}_{pubk_B} \\
\text{Msg } \beta.2. & B & \rightarrow & (I)A & : \{x, y\}_{pubk_A} \\
\text{Msg } \alpha.2. & I & \rightarrow & A & : \{x, y\}_{pubk_A} \\
\text{Msg } \alpha.3. & A & \rightarrow & I & : \{y\}_{pubk_I} \\
\text{Msg } \beta.3. & (I)A & \rightarrow & B & : \{y\}_{pubk_B}
\end{array}$$

In the above attack, $(I)A \rightarrow B : x$ means that the intruder is sending message x to B in A's name, whereas $A \rightarrow (I)B : x$ means that the intruder is blocking a message sent by A intended for B. The above attack consists of two parallel sessions of the protocol, one (whose messages are labelled with α) involving A as the initiator and I as responder, and the other (whose messages are labelled with β) involving I (in A's name) as the initiator and B as the responder. (This shows that the names A, B, x and y mentioned in the protocol specification are just placeholders or abstract names, which can be concretely instantiated in different ways when the protocol is run. So according to A and B, they have just had a normal protocol session with I and A, respectively. But I knows better!) After the fifth message above, the intruder gets to know y which is the secret generated by B in a session with someone whom B believes to be A. This shows that the protocol does not satisfy the following property: *whenever an agent B engages in a session of the protocol as a responder and B believes that the initiator is A, then the secret generated by B is known only to A and B.* The seriousness of this flaw depends on the kinds of use the protocol is put to. It is worth noting that this attack does not depend on weaknesses of the underlying encryption mechanism (nor even on some keys being guessed by chance). It is also worth noting that this attack on the (simple enough) Needham-Schroeder protocol was discovered seventeen years after the original protocol was proposed. [Low96] also suggests a fix for the protocol:

$$\begin{array}{lrcll}
\text{Msg 1.} & A & \rightarrow & B & : \{x, A\}_{pubk_B} \\
\text{Msg 2.} & B & \rightarrow & A & : \{x, y, B\}_{pubk_A} \\
\text{Msg 3.} & A & \rightarrow & B & : \{y\}_{pubk_B}
\end{array}$$

It is easy to see that the above attack does not happen anymore, but that still doesn't prove that the protocol does not have any vulnerabilities.

The following example illustrates a freshness attack (or replay attack), and also highlights the use of nonces. Consider the following protocol (which is inspired by the Denning-Sacco protocol [DS81]) which uses symmetric (shared-key) encryption, where A is *Aandal*, B is a *bank*, and S is a *key server*. We assume that every agent C shares a key k_{CS} with the server, which only C and S know.

$$\begin{array}{lllll} \text{Msg 1.} & A & \rightarrow & S & : A, B \\ \text{Msg 2.} & S & \rightarrow & A & : \{B, k, \{A, k\}_{k_{BS}}\}_{k_{AS}} \\ \text{Msg 3.} & A & \rightarrow & B & : \{A, k\}_{k_{BS}} \end{array}$$

In message 1, A requests from the server S a key to communicate with B. S generates k and creates message 2. Only A can decrypt this message successfully and learn k, since she alone possesses k_{AS}. She then passes on the component $\{A, k\}_{k_{BS}}$ to B. Now B also learns k. Now A can enter into a session with B using the key k. Since only A and B know k, there is no danger of any information being leaked out, as long as the key k is safe. But unfortunately, there is the following attack:

$$\begin{array}{lllll} \text{Msg } \alpha.1. & A & \rightarrow & S & : A, B \\ \text{Msg } \alpha.2. & S & \rightarrow & A & : \{B, k, \{A, k\}_{k_{BS}}\}_{k_{AS}} \\ \text{Msg } \alpha.3. & A & \rightarrow & B & : \{A, k\}_{k_{BS}} \\ \text{Msg } \beta.3. & (I)A & \rightarrow & B & : \{A, k\}_{k_{BS}} \end{array}$$

The attack is quite simple. Sufficiently long after the session α has happened, the intruder masquerades as A and enters into a session with B with the same old key k. This is possible because all the intruder has to do is to replay message 3 from the old session. There might be a question as to what this achieves, since the intruder cannot continue the session meaningfully unless k is leaked. But this is not a scenario which can be ignored. It might be the case that the key k has actually been compromised by long hours of cryptanalysis, much after the original session was played out. The above attack then gives the intruder a chance for putting this key into use. Or it might be the case that in the original session α, after setting up the key k, A sends the following message:

$$\text{Msg } \alpha.4. \quad A \rightarrow B : \{\text{Deposit Rs. 10000 from my account into } I\text{'s}\}_k$$

(This might well be money which is legitimately owed to I by A.) The intruder, who watches all the communication over the network, infers from the effect of the

above message (Rs. 10000 deposited into I's own account) the content of message $\alpha.4$, and just replays it as part of session β.

$$\text{Msg } \beta.4. \quad (I)A \;\rightarrow\; B \quad : \{\text{Deposit Rs. 10000 from my account into } I\text{'s}\}_k$$

Since the bank thinks that the request is coming from A, I ends up richer by Rs. 10000.

A simple solution to the problem is for A and B to generate fresh nonces at the start of each session, then obtain the key from S and check the timeliness of the key received from S as follows:

$$\begin{array}{lrcl}
\text{Msg 1.} & A & \rightarrow & B \quad : A, B \\
\text{Msg 2.} & B & \rightarrow & A \quad : y \\
\text{Msg 3.} & A & \rightarrow & S \quad : A, B, x, y \\
\text{Msg 2.} & S & \rightarrow & A \quad : \{x, B, k, \{y, A, k\}_{k_{BS}}\}_{k_{AS}} \\
\text{Msg 4.} & A & \rightarrow & B \quad : \{y, A, k\}_{k_{BS}}
\end{array}$$

The use of the fresh nonces prevents the intruder from replaying old messages as new. Of course, it is imperative that for each session a unique, unguessable, random number is chosen as a nonce, since otherwise replay attacks cannot be prevented.

A different kind of problem exists with type-flaw attacks. This is illustrated by the following simple example (see [DMTY97] for more examples of interesting type-flaw attacks), where A sends a fresh, random secret x to B and also gets an assurance that B has received it.

$$\begin{array}{lrcl}
\text{Msg 1.} & A & \rightarrow & B \quad : \{(A, \{x\}_{pubk_B})\}_{pubk_B} \\
\text{Msg 2.} & B & \rightarrow & A \quad : \{x\}_{pubk_A}
\end{array}$$

The intruder can use the structure of message 1 and get the secret generated in place of x leaked, as the following attack shows:

$$\begin{array}{lrcl}
\text{Msg } \alpha.1. & A & \rightarrow & (I)B \quad : \{(A, \{m\}_{pubk_B})\}_{pubk_B} \\
\text{Msg } \beta.1. & I & \rightarrow & B \quad : \{(I, \{(A, \{m\}_{pubk_B})\}_{pubk_B})\}_{pubk_B} \\
\text{Msg } \beta.2. & B & \rightarrow & I \quad : \{(A, \{m\}_{pubk_B})\}_{pubk_I} \\
\text{Msg } \gamma.1. & I & \rightarrow & B \quad : \{(I, \{m\}_{pubk_B})\}_{pubk_B} \\
\text{Msg } \gamma.2. & B & \rightarrow & I \quad : \{m\}_{pubk_I} \\
\text{Msg } \alpha.2. & (I)B & \rightarrow & A \quad : \{m\}_{pubk_A}
\end{array}$$

The important point about this attack is that in session β, the intruder is using the term $\{(A, \{m\}_{pubk_B})\}_{pubk_B}$ in place of x. In the absence of any mechanism to

indicate the type of data being received, B believes that he has received a nonce. By cleverly using the structure of the protocol over two sessions, the intruder learns the secret m at the end of message 2 of session γ. This example also shows that the length of messages occurring in runs of a protocol can be much more than that of the messages occurring in the protocol specifications. Of course, this attack can be simply thwarted by modifying the protocol as follows:

$$\begin{array}{llcll} \text{Msg 1.} & A & \rightarrow & B & : \{(A, x)\}_{pubk_B} \\ \text{Msg 2.} & B & \rightarrow & A & : \{x\}_{pubk_A} \end{array}$$

The above examples illustrate the kinds of attacks which typically happen. Much more details on authentication protocols, attacks on them, and the techniques used to tackle them can be found in the excellent survey article [CJ97].

The above discussion illustrates the pitfalls in security protocol design, and also highlights the need for a systematic approach to protocol design and analysis. There are two possible approaches:

- Development of a design methodology following which we can always generate provably correct protocols. Much work in the protocol design community focuses on this approach. [AN96] gives a flavour of the kinds of useful heuristics which improve protocol design. But there has not been much theoretical development towards formally justifying these design guidelines.

- Development of systematic means of analysing protocols for possible design flaws. The bulk of the work in formal methods for security protocols focuses on this approach. Here again, there are two possibilities:

  - Development of methods for proving the correctness of certain aspects of protocols.

  - Development of systematic methods for finding flaws of those protocols which are actually flawed.

The main contributions in this thesis lie in the field of formal analysis methods for security protocols. We now briefly look at some of the approaches which have been advocated in the literature for proving properties of protocols and detecting flaws in them.

An important stream of work relating to proving protocols right is *automated theorem proving*. The typical approach in this style of work is as follows: a formal protocol model is defined based on an expressive logic like first-order logic or higher-order logic. To every protocol, a theory in the logic is associated. Properties of protocols are also specified using the same logic. A property holds of a protocol if it can be derived from the theory of the protocol using the rules of the logic. Established proof techniques and tools in the logic can now be used to efficiently prove properties of protocols. Examples of this approach include [Pau98] and [Bol97]. The advantage of this approach is that the highly expressive logics in the framework can code up any protocol, and formally prove most of the desired properties. Some possible disadvantages are that it requires expert knowledge to code up a protocol into a theory, and that the theorem proving process is not fully automatic. Expert intervention is needed to guide the proof search. The complexity involved in defining the theory of a protocol introduces further chances for error. Another possible drawback is that the formal proofs are not intuitive, and thus hard for humans to understand and base further developments on them.

An alternative approach is to use belief logics to prove properties of protocols. The pioneering work in this line is [BAN90], in which a modal logic (called the *BAN logic*) was introduced as a tool to specify and reason about properties of protocols. It is based on modalities which seek to formalise the epistemic reasoning of the agents involved in the protocol. This logic has many attractive features, chief among them being that it produces simple and abstract proofs, but there are also some drawbacks. To use the logic, the authors propose a systematic idealisation step, which converts each message of the given protocol into a formula which represents the potential knowledge gained after receipt of the message. This feature introduces a chance for error, since there is a possibility that a wrong idealisation might be used to prove properties of the protocol. [BM93], [GNY90], and [Nes90] are some papers which contain a discussion of this feature and suggest further improvements to the BAN logic. [AT91], [Bie90], and [SvO94] are some papers which attempt to improve the original logic with either new modalities or through new semantic features. While they address some weaknesses of BAN logic, the simplicity of the original logic is lost. More recently, there have been attempts to connect BAN style logics with other formal models for security protocols ([ABV02] and [SC01], for example). There have also been attempts at automated reasoning about protocols using BAN-style logics

([KW96], for instance). [SC01] provides a comprehensive survey of BAN-style logics for authentication protocols. The modalities which these logics concentrate on are fairly abstract, like *belief*, *trust*, *control* etc. While it may not be difficult to formalise these modalities, it is not clear whether they are fundamental to reasoning about security. The iteration of these modalities also brings a lot of complexity in its wake, complicating many of the technical questions regarding these logics. Thus it is worthwhile to look at logics with simpler modalities.

Much of the literature is devoted to methods for detecting flaws in protocols using the so-called *model checking* approach. The main idea is to consider a finite state version (preferably with a small number of states) of the given protocol (by imposing bounds on the set of nonces and keys used) and prove that all states of the finite state system satisfies the desired property. This does not necessarily mean that the protocol itself satisfies the desired property, since use of unboundedly many data might possibly introduce more attacks. But if a violation of the desired property is discovered using the small system, it usually means that the protocol is also flawed. The focus of research in this area is to devise methods which will guarantee that a finite state version of the protocol has most of the errors that the big system has, and to devise techniques for efficiently verifying the small system.

As we will see later, when we model security protocols formally, we get infinite state systems. Thus there is no given finite state system which one can verify. The finite model should be constructed from the protocol specification by using appropriate abstractions. The different subdivisions of research in this line basically reflect the different techniques using which the finite state system can be defined, and the different techniques that can be used to verify it. For example, [Low96], [LR97], [MMS97], [Sch96a], and [Sch98] advocate an approach based on process algebra, in which important security properties are defined using some form of process equivalence. [Mea95], [Mea96a], [Mea96b] advocate an approach based on logic programming, where the protocol is modelled by a set of rules which tell us how each action of the protocol changes the state of the system, and several specialized proof techniques are used to prove that a bad state can never be reached by a protocol. [Bol97] uses standard techniques based on abstract interpretation to define a finite-state system from a protocol. Techniques based on tree automata ([Mon99], [GL00], [CC03], [CCM01]) have been proposed to efficiently represent and manipulate the intruder's state. Typically the intruder's state is the cause of the infinite state na-

ture of protocols, and hence methods of finitely representing the intruder state can help construct a finite state system from a protocol.

The model checking approach has enjoyed great success in unearthing bugs in many protocols, long after they had been put into use. [CJ97] is a good reference for the many attacks which have been uncovered by formal verification tools. But the main drawback in this approach is that the use of a finite state system is not always justified. In fact, the general verification problem for security protocols is undecidable (as we prove in later chapters), and therefore there exist protocols which are not "equivalent" to any system with bounded number of states. In this context, [Low99] proves that for a certain syntactic subclass of protocols and for some particular kinds of properties, checking whether the protocol satisfies those properties amounts to checking whether a particular small system satisfies them. This provides a justification for verification algorithms, most of which define a small system of the above kind from a given protocol, and verify the small system. The decidability results in this thesis are in the same spirit as the results of [Low99].

1.3 Contributions of the thesis

In chapter 2 of the thesis, we describe our formal model for security protocols which will be used in the rest of the thesis. We also highlight the aspects in which the model differs from other models current in the literature. We set up several technical propositions about synth and analz proofs, which formalise the way the agents running the protocols derive new information from old.

We also introduce *the secrecy problem*, which aims to check if there is a run of the given protocol which leaks a secret or not. Our main contribution in the thesis is to identify subclasses of protocols for which it is possible to automatically verify this property.

It turns out that when we model security protocols precisely, we get infinite state systems. There are many sources of unboundedness in the model which contribute to this. The first type of unboundedness occurs because there is no *a priori* bound on the number of sessions occurring in a run, and thus there is no bound on the length of the runs of a protocol as well. Further, requirements such as freshness might necessitate the use of a fresh nonce or key for each session. Since the number of sessions in a run is unbounded, it follows that there is no *a priori* bound on

the number of distinct nonces and keys used in a run of a protocol. Further, as evidenced in the type-flaw attack which was shown earlier, messages occurring in runs of a protocol can be longer than those occurring in the protocol specification. Thus there is no *a priori* bound on the length of the messages which are part of the runs as well.

As such, it is to be expected that it is not possible to verify even simple reachability properties, and thus security properties like secrecy as well, of such systems. It has been formally proved in ([DLMS99], [HT96], [ALV02]) that in fact, such simple problems are undecidable for these systems. Of the factors which lead to unboundedness of these systems, the number of nonces and the message length are of special importance. It is proved in [DLMS99] that even when the message length is restricted to be bounded, allowing an unbounded number of nonces to occur in runs of a protocol leads to undecidability. Dually, in [HT96] and [ALV02], it is proved that even if the nonces and keys come from a fixed finite set, allowing arbitrarily long messages to occur in protocol runs leads to undecidabilty. In chapter 3, we provide simple and uniform proofs for the above two undecidability results.

The literature consists of many proposals to cope with the undecidability results. If there is a bound on the number of nonces as well as the message length, then every run can be shown to be equivalent to a run of bounded length, in terms of the security-relevant information learnt by the various parties at the end of the run. This has been used to prove decidability in [DLMS99]. Another common approach is to place bounds on the number of plays of any run of the protocol, effectively yielding a finite state system. [ALV02], [MS01] and [RT03] contain examples of this approach. There are also approaches which impose restrictions on the way messages can be constructed. Examples of this include [DEK82] and [ALV02] where restrictions are imposed on the way messages are concatenated with one another to form new messages. The work in [CCM01] uses techniques from tree automata to show decidability for a subclass of protocols in which every agent copies at most one piece of any message it receives into any message it sends. The survey article [CS02] gives a nice overview of the various approaches to decidability of security protocol verification, and also the various undecidability results. [ALV02] also provides a nice perspective on the various factors which affect decidability of security protocol verification.

The literature also consists of work where decidability is obtained without placing

such 'external' bounds. For example, the work [Sto02] seeks to identify some simple semantic properties which lead to decidability and argue that these properties are satisfied by a large class of protocols found in the literature. [AC02] introduces checkable syntactic conditions which entail the equivalence of the given protocol to a finite-state system, and then gives methods of checking the finite-state systems for security breaches. A significant work in this line is [Low99], where decidability is proved for a *syntactic* subclass of protocols, under the assumption that message length is bounded but without any assumptions on the number of nonces. Our work in chapter 4 is in this spirit. Assuming that message length is bounded and the set of nonces is not, we prove decidability of the secrecy problem for a syntactic subclass of protocols, the so called tagged protocols. Essentially, these are protocols where the important components of each message have some kind of *type tags* attached to them. The use of tags allows us to prove that for every tagged protocol, there is a run which leaks a secret iff there is a run of bounded length which leaks a secret. This is the key to our decidability result.

We continue the same theme in chapter 5, where we prove that even if we do not place any bound on message length, we can obtain decidability of the secrecy problem for the class of tagged protocols. We achieve this by showing that for tagged protocols, every run is equivalent to a *well-typed run* (under a suitable notion of equivalence which preserves many important security properties). A well-typed run is basically a run in which there is no type-flaw. This means that nonces occurring in the protocol specification are only replaced by nonces in the different sessions of the run, and so on for the other types of data as well. This further means that the length of the messages occurring in a well-typed run is bounded by the length of the messages occurring in the protocol specification. Since every run is equivalent to a well-typed run, the problem reduces in effect to the setting of chapter 4, and thus we get our decidability result.

In chapter 5, we also consider a semantic subclass of protocols based on an equivalence relation of finite index on messages, and prove the decidability of the secrecy problem for this semantic subclass, under the assumption that the nonces and keys come from a fixed finite set.

In chapter 6, we look at methods for reasoning about protocols. We define a logic in which several important properties like secrecy and authentication can be naturally specified. A major portion of the chapter is devoted to examples which

illustrate how to reason about protocols using the logic. We then show that the undecidability results of chapter 3 and the reduction to well-typed runs proved in chapter 5 extend to the verification problem for the logic as well. Using the reduction to well-typed runs, we prove the decidability of the verification problem of the logic in a setting where there are no restrictions on the length of messages occurring in runs of a protocol, but where the nonces and keys come from a fixed finite set.

The research that this thesis is based on was done in cooperation with R. Ramanujam. The work in chapter 4 is based on the papers [RS03a] and [RS03c]. [RS03c] is also the basis for the part of chapter 5 which deals with the reduction to well-typed runs. The semantic decidability result in chapter 5 is based on [RS03b].

Chapter 2

Security protocol modelling

In this chapter, we first discuss the issues involved in modelling security protocols. We then informally introduce our model and compare it with some of the other existing models. We then present a formalization of the model. We close the chapter with some important properties of our models, especially properties relating to the generation of new messages by agents from old information which they possess.

2.1 Discussion

The formal modelling of security protocols is a nontrivial problem in itself. For example, consider the Needham-Schroeder protocol presented in Section 1.2.

- The protocol is specified in terms of two agents A and B and two secrets x and y. But as evidenced in Lowe's attack, these are just abstract names which act as placeholders and can be concretely instantiated with different values to create many different sessions of the protocol.
- It is also evident from Lowe's attack that runs typically contain many parallel sessions.
- Further there could be infinitely many sessions of a given protocol and it is possible that a run consists of unboundedly many sessions.

- A further complication is that the abstract terms in the protocol can be instantiated with arbitrary messages (not just atomic messages) to carry out certain attacks. This was illustrated by the second example of Section 1.2.

So we see that while protocol specifications are finite (usually quite small), the system which generates the set of runs of the protocol needs to remember an unbounded amount of information, and is thus an infinite state system. Thus a formal model for security protocols involves many details which need to be got right. The large gap in complexity between a protocol specification and the system which generates the runs of the protocol makes the task of formally modelling protocols nontrivial.

Further, at every step of defining a model, the modeller is presented with choices which have to be resolved one way or the other. Some of the possible questions that she might face are:

- what should be the structure of the messages?
- how are protocols to be presented?
- what should be the assumptions on intruders?
- how do agents construct new messages from old?
- what is the underlying model of communication?

As always, the manner in which the choices are resolved is driven by the application in hand. Thus it is not surprising that a consensus has still not been reached, and that the literature abounds with many different models for security protocols.

Before a description of our model, we briefly look at some of the other popular styles of modelling security protocols.

Process algebra models Examples of these kinds of models include the CSP-based models of [Low96], [LR97], and [Sch96a], and the the spi calculus model of [AG99]. We look at the spi-calculus model to provide a flavour of these kinds of models. It is an extension of the pi calculus [MPW92] with cryptographic primitives. The basic idea is that every protocol is represented by a spi calculus process (which gives the operational semantics of the protocol, in the sense that the process displays exactly the same run-time behaviour as

the protocol). The process for a protocol is typically a parallel composition of (possibly many different instantiations of) a process for each role of the protocol. The other process algebra models also model the behaviour of the intruder as an *intruder process*, and the process corresponding to a protocol is defined as a parallel composition of the processes for the roles and the intruder process. But the spi calculus differs from them in that it does not fix an intruder process. We will see a little later how intruder behaviour is modelled in the spi calculus. Security properties of protocols can now be translated to properties of the process representing the protocol. These are typically various kinds of observational equivalences between processes, which basically say that no observer interacting with the two processes can distinguish between the two.

For instance, let us say that a protocol which uses an abstract term x is represented by a process $P(x)$. (The notation signifies that the definition of P is parametrized by x.) Let us say that the protocol involves sending x from A to B securely. For every concrete term m, we define $P_{spec}(m)$ to be a process which is "obviously correct" in its behaviour with respect to m. (For instance, it might say that irrespective of what happens after A sends the message m, at some future point of time B (either normally or magically) receives the same message m.) Now a possible definition of secrecy is that for any two distinct messages m and m', $P(m)$ is *observationally equivalent* to $P(m')$. If the secret is not revealed, then no external observer can see any difference between a run of the protocol which uses secret m and one which uses secret m'. A possible definition of authentication is that for all m, $P(m)$ is observationally equivalent to $P_{spec}(m)$. This says that if the A sends the message m, then if at all the receiver receives a message which purports to be from A, the message has to be m.

Since the notion of observational equivalence used in the spi calculus refers to *all processes*, there is no need to explicitly define an intruder process. If there is an attack on a protocol, it will definitely manifest in the form of the two relevant processes being distinguishable by a process coding up the intruder behaviour in the attack.

The main focus of research in spi calculus is to develop generic proof techniques that work for classes of protocols ([AG98], [Aba99], [AFG02]). It is

also possible to use existing tools for the process algebra models and apply them to security. An example is the FDR model checker for CSP, which has been successfully used in discovering attacks on protocols (see [Low96], for example).

The inductive approach This approach was pioneered by [Pau98], which advocates a theorem-proving approach to verifying cryptographic protocols. The theorem prover used in [Pau98] is Isabelle/HOL, which works with higher-order logic.

A protocol is formalised as a set of *traces*, where each trace is a sequence of *events*. Example of events include Says A B X and Notes A X. Says A B X means that A says X to B, it does not imply that B heard what A says. Notes A X means that A learns the message X. The important point is that the set of traces of the protocol is defined inductively, starting with the empty trace, adding "proper" actions for the honest principals, and any "admissible" action for the intruder. "Proper" actions are those which follow the protocol. For instance the fifth message of a role can be sent only after the fourth message. "Admissible" means that the message that is being communicated in the event can be constructed by the agent from the information already learnt by him. The operators synth and analz formalize the way in which new messages are constructed from old.

A protocol is said to satisfy a property if all its traces satisfy the property. This can be verified by letting a theorem-prover inductively check that all traces of the protocol satisfy the said property. If a property does not hold of a protocol, then the failed attempts at a proof lead one to an attack scenario.

The inductive approach has been used as a basis for proving the correctness of some very complicated protocols [Bel99].

Strand spaces This is a model introduced in [FHG99]. In this model, a protocol is assumed to be presented by set of *(parametrized) strands*, which are sequences of send or receive actions. A *node* of a protocol is a pair consisting of an instantiation s of a parametrized strand and an index i which is at most the length of s. A strand space corresponding to a protocol is a graph whose nodes consist of all the nodes of the protocol and whose edges reflect the local and communication dependency between events. A very important component of

the model is the formalisation of the intruder behaviour in terms of *penetrator strands*. Each penetrator strand describes an atomic behaviour of the intruder. Examples of such behaviour include receiving a message, creating a copy of a message that has been received, splitting a message of the form (t, t') to get t, encrypting t using a key k to obtain $\{t\}_k$, and so on. The penetrator strands of this model, the intruder process in the process algebra models, and the intruder theory in the multi-set rewriting model (to be described below) roughly correspond to one another. A *bundle* of a protocol (which basically stands for a run of the protocol) is a finite partially ordered subgraph of the strand space of the protocol, with the condition that for every event in the bundle, its causal past is also included in the bundle. The significant feature of this model is that runs of a protocol are formalised as partially ordered objects.

Significant properties of protocols can now be expressed in terms of the model. An example of an authentication property is the requirement that whenever node n_1 occurs in a bundle, node n_2 should also occur. Secrecy properties are formalised by saying that some kinds of nodes do not occur in any bundle of the protocol. (These are typically nodes which reveal some secret to the intruder). A significant amount of the research here is devoted to developing techniques for proving general bounds on the intruder's abilities in any run of a protocol (or a class of protocols). There have also been attempts at automatic analysis of protocols based on the strand spaces model (see [SBP01], for example). There have also been attempts to provide a semantics for BAN logic in terms of the strand space model ([SC01], for example).

Multi-set rewriting Like the spi calculus and the inductive model, this is also a general-purpose model in which we can embed security protocols. [DM99] is an introduction to the model, whereas [DLMS99] and [CDL+99] present technical results about the framework.

The basic idea here is that a security protocol is given by a *theory* which is a finite set of rules, where each rule is of the form $P_1(\cdots), \ldots, P_k(\cdots) \longrightarrow \vec{\exists}.\, Q_1(\cdots), \ldots, Q_l(\cdots)$. The P's and Q's are atomic formulas (of the predicate calculus). The theory of a protocol is got by composing a theory for each role with a standard intruder theory. A *state* is a finite multiset of atomic

sentences. Rules are allowed to have free variables, but ground instantiations of rules are applied to states to yield new states. A rule application on a state s yields another state s' iff:

- all the preconditions of the rule all belong to s,
- the preconditions which are not postconditions do not belong to s',
- for every copy of a postcondition which is not a precondition, a copy of it is added to s',
- the rest of s is copied into s', and
- each existentially quantified variable is instantiated by a new constant not occurring in s.

In fact, the semantics of rules has close connections with the proof theory of linear logic.

Properties of security protocols can be easily formalised in this framework. For instance, the *secrecy problem* is essentially a state reachability problem (the input for the problem is a theory, an initial state and an atomic sentence). The problem is to determine whether there is a reachable state in which the said atomic sentence holds.

We now describe our model informally. While it does not differ drastically from any of the models described above, still there are differences in emphasis. Our focus is on retaining enough distinctions at the level of protocol specification so that it is easy to define certain syntactic subclasses, for which we later prove the decidability of verifying secrecy.

Protocol specifications: Security protocols are typically specified as a (finite) set of roles (typically with names like challenger, responder and so on). These are abstract patterns of communication which specify what messages are sent when, and how to respond to the receipt of any message. The content of these messages is (usually) not relevant, but the structure is; hence abstract variables suffice to describe the protocol. For example, the Needham-Schroeder can be viewed as consisting of two roles, an initiator role given by

$$A!B{:}\{x, A\}_{pubk_B}; \quad A?B{:}\{x, y\}_{pubk_A}; \quad A!B{:}\{y\}_{pubk_B}$$

and a responder role given by

$$B?A{:}\{x, A\}_{pubk_B}; \quad B!A{:}\{x, y\}_{pubk_A}; \quad B?A{:}\{y\}_{pubk_B}.$$

Roles are typically sequences of actions, which can either be a send action of the form $A!B{:}t$ (which stands for A sending t over the network intended for B) or a receive action of the form $A?B{:}t$ (which stands for A receiving t over the network with some indication that the sender is B).

In our model, we pay close attention to protocol specifications. In fact, the major technical results in this thesis show that the manner in which protocols are specified has a major bearing on problems like verifying secrecy of a given protocol. In fact, the negative results in Chapter 3 point out that the above style of presenting protocols admits too many complicated protocols, which are not representative of the protocols which arise in practice ([CJ97]). So, for our positive results we focus on the more manageable class of protocols which are presented as sequence of communications of the form $A \rightarrow B : t$. This is also the informal style of presenting protocols which is popular in the literature. There are also some admissibility conditions here that are assumed implicitly in the literature. We make them explicit and point out their crucial role in the analysis of protocols. The class of protocols which satisfy these conditions are called well-formed protocols.

Starting from such descriptions of a protocol, we formally define the semantics of each protocol. This is slightly different from the style current in the literature. For instance, in the inductive model, a protocol is formally a set of rules (in higher-order logic) which specify the conditions under which runs of the protocol can be extended by adding an event. In the spi calculus model, a protocol is formally a spi calculus process (which can generate the set of all runs of the protocol). The passage from an informal protocol specification (as a sequence of communications) to the formal object is not given much attention (as that is usually trivially achieved). But formally any finite set of rules (or any process) can be a protocol. The advantage of such an approach is the high expressive power of the model. Any protocol can be coded up as a formal object of the model. A possible disadvantage is that it is sometimes difficult to isolate a certain (syntactic or semantic) class of protocols that we wish to concentrate on. Further, it is sometimes difficult to judge whether a

technical result (like undecidability of verification, for instance) holds because of something inherent to protocols or because it is a general result which holds of the model itself.

Messages: A protocol as specified above is run by a set of agents, who are of two kinds: the malicious intruder and the rest, who are honest. They perform message exchanges as prescribed in the protocol. Following the lead of Dolev and Yao ([DY83]), we will assume that the terms which are communicated in message exchanges come from a free algebra of terms with tupling and encryption operators. This means that we are operating on a space of symbolic terms, abstracting away from the fact that in the underlying system all messages are bit strings.

We work with a simple syntax of messages which allows only atomic keys. We disallow constructed keys, using which one can form messages of the form $\{x\}_{\{k\}_{k'}}$. While this choice certainly limits the applicability of our model and the results, we want to consider key technical questions like the decidability of the secrecy problem in this important setting, before moving on to more complex settings. On the other hand we feel that some of the other extensions to the message syntax, like hashing, can be easily handled and almost all our results will go through with minor modifications.

Cryptographic assumptions: Following the lead of Dolev and Yao ([DY83]) we make the perfect encryption assumption. This means that a message encrypted with key k can be decrypted only by an agent who has the corresponding inverse $\overline{k}$. We thus abstract away cryptographic concerns and treat encryption and decryption as symbolic operators. There is a different tradition to studying security protocols, called the "computational approach". In this approach, protocols are shown correct by reducing the protocol to the underlying cryptography, i.e., it is shown that if there exists an adversary with a significant chance of attacking the protocol, there exists another adversary with a significant chance of breaking the underlying cryptographic scheme itself. The work [BR93] is an example of this approach. We have chosen the more abstract framework which is preferred by most researchers in formal methods for cryptographic protocols. Recently, there has been some important work in reconciling the two approaches to cryptography. (See [AR00], [Her02], [Her03],

for examples of such work.)

We also abstract away the real-life phenomenon in which some honest agents lose their long-term keys. This is modelled in [Pau98], for example, by the notion of an Oops event. This reflects the probabilistic nature of the underlying cryptography, all the current schemes being not absolutely secure but only unbreakable with a very high probability. While we can model more attacks this way, we opt for a more restricted model in which decidability questions are easier to handle. Further our focus is mainly on logical flaws in protocols which exist even under the assumption that cryptography is absolutely unbreakable.

Intruder capabilities: We assume an all-powerful intruder, who can copy every communication in the system, can block any message and can pretend to be any agent. In addition he also has the message building capabilities available to every agent. It is assumed that the intruder has unlimited computational resources and can keep a record of every public system event and utilize it at an arbitrarily later time. However, we assume that the intruder cannot break encryption. These assumptions keep the intruder model technically simple. They are also followed widely in the literature.

The different models in the literature have tended to agree on most aspects of the intruder modelling. Such an intruder is called a Dolev-Yao intruder. Some variations to the above model have been tried but it has been shown that they do not significantly alter the intruder's powers. For example, we might consider a group of colluding intruders rather than a single intruder. But such a collusion cannot cause more attacks than a single intruder acting alone, as has been proved in [CMS00].

Events and runs of a protocol: An event of a protocol is an action of some role of the protocol with a substitution which supplies concrete terms for the abstract placeholders mentioned in the roles. As observed earlier, arbitrary terms can be substituted in place of nonces. An important class of events we will consider are the class of well-typed events which are obtained by substitutions which replace nonces only by nonces. It is clear that there are potentially infinitely many events of a protocol. If the set of nonces and keys is assumed to be infinite, it is possible that even the set of well-typed events is infinite.

A run of a protocol can informally be thought of as a sequence of events which

respects certain admissibility conditions, which will be detailed below. Thus it is seen that we do not place any bounds on the number of plays occurring in a run, or on the number of plays which are active simultaneously (parallel sessions, as we called them earlier). It is to be noted that in [MS01] and [RT03], certain decidability results are obtained by essentially placing bounds on the number of plays that can occur in any run of the protocol. We follow an alternative approach by retaining the more general model and proving the corresponding decidability results for syntactic subclasses of protocols.

We consider sequential runs, like most of the other models in the literature, and unlike the strand spaces model. We choose sequential runs over partially ordered runs since we find it is easier to present the decidability arguments in that setting.

Admissibility: Arbitrary interleavings of plays of a protocol are not counted as runs. They have to be realisable, in the sense that for every action a occurring in the run, if t is the term communicated in a and if agent A is the communicator, t can be constructed from the information which is presented to A in the initial state along with the information learnt by her from the message exchanges preceding a. Another important requirement is that certain secrets which are used as instantiations of *new nonces* (i.e., abstract secret names which are specified as "fresh" by the protocol) should satisfy the property of freshness, i.e. these secrets have not been used before in the run. Thus a record of the secrets used so far in the run has to be necessarily kept. These considerations lead us to the notions of information state of an agent and message construction rules. The agents are supposed to have learnt all the messages which have been communicated to them. Further they can construct new messages from old by tupling, detupling, encryption and decryption using known keys, and by generating new unguessable nonces which have not been previously used by anyone. The formal counterparts of the message generation rules are the operators synth and analz which are at the heart of most of the technical results in the thesis.

It is to be noted that our definition of runs is quite close to that given in [Pau98]. At the level of defining runs, the admissibility conditions are quite standard in the literature. The key element in our model is that we consider

incorporating some of these conditions in the protocol specification itself as a formalisation of a notion of a "well-behaved protocol".

Initial knowledge: This is another feature of security protocol modelling in which the different existing models have tended to display slight differences. One typical approach is to let this be part of the specification of protocols. For instance, we might say that every agent shares a key with the server in the initial state, while the server has (or can generate) all the other keys, which the agents can request and obtain. Or we might say that every agent shares a key with every other agent in the initial state. We follow the technically simple approach of fixing a set of keys known to each of the agents in the initial state, independent of the protocol. This looks restrictive, but the model can be easily adapted to include such protocol specifications. We only need to add a few consistency conditions (for instance, at every state, if a key is available to some agent, then its inverse is also available to some (not necessarily the same) agent) for some of the technical results in Chapter 4 to go through.

Closely related to this is the issue of constant terms of a protocol. Typical names occurring in a protocol specification (like the names A, B, x, etc. of the Needham-Schroeder protocol) are placeholders which can be substituted with any other term to generate runs. But some protocols might refer to some agents like a key server, whose role can be played only by some designated processes. Thus we do not allow the meanings of these names to change during the course of a protocol run. While we usually do not distinguish between the rest of the honest agents either in terms of their initial knowledge or in terms of their computational power, designated agents like the key server might have some extra information in the initial state, and some added computational power as well.

2.2 A formal model for security protocols

2.2.1 Security protocols and their runs

Basic terms

We assume a (potentially infinite) set of *agents* Ag with a special *intruder* $I \in Ag$. The set of *honest agents*, denoted Ho, is defined to be $Ag \setminus \{I\}$. We

assume that the set of *keys* K is given by $K_0 \cup K_1$ where K_0 is a countable set and $K_1 \stackrel{\text{def}}{=} \{k_{AB}, pubk_A, privk_A \mid A, B \in Ag, A \neq B\}$. $pubk_A$ is A's *public key* and $privk_A$ is its *private key*. k_{AB} is the *(long-term) shared key* of A and B. For $k \in K$, $\overline{k}$, the *inverse key* of k, is defined as follows: $\overline{pubk_A} = privk_A$ and $\overline{privk_A} = pubk_A$ for all $A \in Ag$, and $\overline{k} = k$ for all the other keys. For every agent A, the set of keys which are assumed to be always known by A, denoted K_A, is defined to be $\{k_{AB}, k_{BA}, pubk_A, privk_A, pubk_B \mid B \in Ag, B \neq A\}$. We also assume a countable set of *nonces* N. ('Nonce' stands for "*n*umber *once* used"). We also assume a perfect nonce generation mechanism which can generate a *nonguessable*, *unique* nonce on each invocation. Finally we assume a set SN of *sequence numbers* (numbers which are used to associate one message with another). A mechanism to generate sequence numbers is also assumed, which can generate a unique (but not necessarily nonguessable) number on each invocation. $\mathcal{T}_0$, the set of *basic terms*, is defined to be $K \cup N \cup SN \cup Ag$. The set $K_0 \cup N \cup SN \cup Ag$ will also play a special role in the subsequent development. We use the notation $\mathcal{T}_0$ to denote it.

Further we fix the nonce n_0, the sequence number m_0, and the key $\mathsf{k}_0 \in K_0$ for the whole discourse. They will essentially play the role of the intruder's initial knowledge, as will be explained later.

Terms

The set of *information terms* is defined to be

$$\mathcal{T} \quad ::= \quad m \mid (t_1, t_2) \mid \{t\}_k$$

where m ranges over $\mathcal{T}_0$ and k ranges over K. These are the terms used in the message exchanges below.

The notion of subterm of a term is the standard one — $ST(m) = \{m\}$ for $m \in \mathcal{T}_0$; $ST((t_1, t_2)) = \{(t_1, t_2)\} \cup ST(t_1) \cup ST(t_2)$; and $ST(\{t\}_k) = \{\{t\}_k\} \cup ST(t) \cup ST(k)$. t' is an *encrypted subterm* of t if $t' \in ST(t)$ and t' is of the form $\{t''\}_k$. $EST(t)$ denotes the set of encrypted subterms of t. The *size* of terms is inductively defined as follows: $|m| = 1$ for $m \in \mathcal{T}_0$; $|(t_1, t_2)| = |t_1| + |t_2| + 1$; and $|\{t\}_k| = |t| + |k| + 1$.

In the rest of the thesis, we use the notation $|\cdot|$ in three different meanings: as the size of terms, as the size of sets, and as the length of sequences. It is easy to know what is meant by looking at the context.

The term $\{t\}_k$ is an abstract notation where we make no cryptographic assump-

tions about the algorithm used to form $\{t\}_k$ from t and k. It could stand for t encrypted with the key k, or it could stand for t appended with a *signature* using the key k. Following the lead of Dolev and Yao [DY83] we make the perfect encryption assumption. This means that a message encrypted with key k can be decrypted only by an agent who has the corresponding inverse $\overline{k}$. This is reflected in the encrypt and decrypt rules below.

Actions

An *action* is either a *send action* of the form $A!B{:}(M)t$ or a *receive action* of the form $A?B{:}t$ where: $A \in Ho, B \in Ag$ and $A \neq B$; $t \in \mathcal{T}$; and M is a subset of $ST(t) \cap (N \cup K_0 \cup SN)$. In a send action of the form $A!B{:}(M)t$, M is the set of nonces, keys and sequence numbers freshly generated by A just before sending t. For simplicity of notation, we write $A!B{:}t$ instead of $A!B{:}(\emptyset)\ t$. The set of all actions is denoted by Ac, the set of all send actions is denoted by $Send$, and the set of all receive actions is denoted by Rec. Ac_A, the set of A-*actions* is given by $\{C!D{:}(M)t, C?D{:}t \in Ac \mid C = A\}$.

Note that we do not have explicit intruder actions in the model. As will be clear from the definition of updates caused by actions, every send action is implicitly considered to be an instantaneous receive by the intruder, and similarly, every receive action is considered to be an instantaneous send by the intruder. Thus the agent B is (merely) the intended receiver in $A!B{:}(M)t$ and the purported sender in $A?B{:}t$.

For a of the form $A!B{:}(M)t$, $term(a) \stackrel{\text{def}}{=} t$ and $NT(a) \stackrel{\text{def}}{=} M$. For a of the form $A?B{:}t$, $term(a) \stackrel{\text{def}}{=} t$ and $NT(a) \stackrel{\text{def}}{=} \emptyset$. $NT(a)$ stands for *new terms* generated during action a. $ST(a)$ and $EST(a)$ have the obvious meanings, $ST(term(a))$ and $EST(term(a))$ respectively. $terms(\eta) \stackrel{\text{def}}{=} \bigcup_{1 \leq i \leq \ell} term(a_i)$ for $\eta = a_1 \cdots a_\ell \in Ac^*$. $NT(\eta)$, $ST(\eta)$ and $EST(\eta)$ are similarly defined. $\eta\restriction A$, A's view of η, is defined inductively as follows: $\varepsilon\restriction A = \varepsilon$; $(\eta \cdot a)\restriction A = (\eta\restriction A) \cdot a$ if $a \in Ac_A$ and $\eta\restriction A$ otherwise.

Protocol specifications

Definition 2.2.1 *An* information state *s is a tuple $(s_A)_{A \in Ag}$ where $s_A \subseteq \mathcal{T}$ for each agent A. $\mathcal{S}$ denotes the set of all information states. For a state s, we define $ST(s)$ to be $\bigcup_{A \in Ag} ST(s_A)$.*

Definition 2.2.2 *A* **protocol** *is a pair* $\mathsf{Pr} = (\mathsf{C}, \mathsf{R})$ *where:*

- C*, the set of* constants *of* Pr*, denoted* $\mathsf{CT}(\mathsf{Pr})$*, is a subset of* $\mathcal{T}_0$ *with the property that* $\{\mathsf{n}_0, \mathsf{m}_0, \mathsf{k}_0\} \cap \mathsf{C} = \emptyset$*, and*
- R*, the set of* roles *of* Pr*, denoted* $Roles(\mathsf{Pr})$*, is a finite subset of* Ac^+ *such that for each* $\eta \in \mathsf{R}$*, there is an* $A \in Ho$ *with* $\eta \in Ac^+_A$*.*

Definition 2.2.3 *Given a protocol* $\mathsf{Pr} = (\mathsf{C}, \mathsf{R})$*,* $\mathsf{init}(\mathsf{Pr})$*, the* initial state of Pr *is defined to be* $(T_A)_{A \in Ag}$ *where for all* $A \in Ho$*,* $T_A = \mathsf{C} \cup K_A$ *and* $T_I = \mathsf{C} \cup K_I \cup \{\mathsf{n}_0, \mathsf{m}_0, \mathsf{k}_0\}$*.*

This style of presentation of protocols is close to that in the multiset rewriting framework of [CDL+99], [DLMS99], [DM99], etc., and the process algebra framework of [AG99], [Low96], etc. The more usual style of presenting protocols is developed in a later section.

As we have mentioned earlier, we do not explicitly model intruder actions. Thus we do not explicitly model the phenomenon of the intruder generating new nonces in the course of a run, as is done in some other models (for instance, [DLMS99]). An alternative would be to provide an arbitrary set of nonces and keys to the intruder in the initial state. We follow the approach of just providing the intruder with the fixed nonce n_0, the fixed sequence number m_0, and the fixed key k_0 in the initial state. They serve as symbolic names for the set of new data the intruder might generate in the course of a run. This suffices for the analysis we perform in our proofs later. We will ensure as we develop the model that n_0, m_0 and k_0 are not generated as a fresh term by any honest agent in the course of a run of Pr.

Example 2.2.4 A version of the Needham-Schroeder protocol ([NS78]) is presented in this example. The protocol $\mathsf{Pr}_{\mathsf{NS}}$ is given by (C, R) where

- $\mathsf{C} = \emptyset$, and
- $\mathsf{R} = \{\eta_1, \eta_2\}$, where
  - η_1 is the following sequence:

$$\begin{array}{llllll} 1. & A & ! & B & : (x) & \{A, x\}_{pubk_B} \\ 2. & A & ? & B & : & \{x, y\}_{pubk_A} \\ 3. & A & ! & B & : & \{y\}_{pubk_B} \end{array}$$

– η_2 is the following sequence:

$$\begin{array}{llllll} 1. & B & ? & A & : & \{A, x\}_{pubk_B} \\ 2. & B & ! & A & : (y) & \{x, y\}_{pubk_A} \\ 3. & B & ? & A & : & \{y\}_{pubk_B} \end{array}$$

The protocol has two roles: we call η_1 the initiator role and η_2 the responder role. A sends the new nonce x to B as a challenge to prove his (B's) identity. She then receives a response to it as also a challenge from B in the form of a nonce y. She finally responds to B's challenge by sending back y. Since only B can decrypt the contents of the first message, A is at least convinced that B is alive. Similarly, B first receives a challenge from A and responds to it while issuing his own challenge. He finally receives the response to his challenge. Since only A could have decrypted the contents of the message sent by B, the latter is at least convinced that A is alive. □

Example 2.2.5 Here is another example of a protocol, We call this $\mathsf{Pr}_{\mathsf{ut}}$. It is given by (C, R) where:

- $\mathsf{C} = \emptyset$, and
- $\mathsf{R} = \{\zeta_1, \zeta_2\}$ where
  - ζ_1 is the following sequence:

$$\begin{array}{llllll} 1. & A & ! & B & : (x) & \{A, \{x\}_{pubk_B}\}_{pubk_B} \\ 2. & A & ? & B & : & \{x\}_{pubk_A} \end{array}$$

  - ζ_2 is the following sequence:

$$\begin{array}{llllll} 1. & B & ? & A & : & \{A, \{x\}_{pubk_B}\}_{pubk_B} \\ 2. & B & ! & A & : & \{x\}_{pubk_A} \end{array}$$

Here again we can call the role ζ_1 the initiator role and the role ζ_2 the responder role. The initiator issues a challenge, response to which will ensure her at least of the responder's being alive in the network. The responder plays the passive role of just responding to the challenge. □

Substitutions and events of a protocol

A *substitution* σ is a partial map from $\mathcal{T}_0$ to $\mathcal{T}$ such that:

- for all $A \in Ag$, if $\sigma(A)$ is defined then it belongs to Ag,
- for all $k \in K_0$, if $\sigma(k)$ is defined then it belongs to K_0, and
- for all $m \in SN$, if $\sigma(m)$ is defined then it belongs to SN.

An important point to note about substitutions is that nonces can be substituted with arbitrary terms. Thus our formal model allows the possibility of some kinds of type-flaw attacks to be carried out by the intruder. A substitution σ is *well-typed* iff for all $n \in N$, if $\sigma(n)$ is defined then it belongs to N. Given a set $T \subseteq \mathcal{T}_0$, a substitution is said to be a T-substitution iff for all $m \in \mathcal{T}_0$, if $\sigma(m)$ is defined then it belongs to T.

Substitutions are extended to terms, sets of terms, actions and sequences of actions in a straightforward manner, as follows:

- $\sigma(pubk_A)$ and $\sigma(privk_A)$ are defined only if $\sigma(A)$ is defined, in which case they are defined to be $pubk_{\sigma(A)}$ and $privk_{\sigma(A)}$, respectively.
- $\sigma(k_{AB})$ is defined only if $\sigma(A)$ and $\sigma(B)$ are defined and $\sigma(A) \neq \sigma(B)$, in which case it is defined to be $k_{\sigma(A)\sigma(B)}$.
- $\sigma((t, t'))$ is defined only if $\sigma(t)$ and $\sigma(t')$ are defined, in which case it is defined to be $(\sigma(t), \sigma(t'))$.
- $\sigma(\{t\}_k)$ is defined only if $\sigma(t)$ and $\sigma(k)$ are defined, in which case it is defined to be $\{\sigma(t)\}_{\sigma(k)}$.
- $\sigma(T)$ is defined only if $\sigma(t)$ is defined for all $t \in T$, in which case it is defined to be $\{\sigma(t) \mid t \in T\}$.
- $\sigma(A!B{:}(M)t)$ is defined only if $\sigma(A)$, $\sigma(B)$ and $\sigma(t)$ are defined, $\sigma(A) \in Ho$, $\sigma(A) \neq \sigma(B)$, and $\sigma(M \cap N)$ is a subset of N, in which case it is defined to be $\sigma(A)!\sigma(B){:}(\sigma(M))\sigma(t)$.
- $\sigma(A?B{:}t)$ is defined only if $\sigma(A)$, $\sigma(B)$ and $\sigma(t)$ are defined, $\sigma(A) \in Ho$, and $\sigma(A) \neq \sigma(B)$, in which case it is defined to be $\sigma(A)?\sigma(B){:}\sigma(t)$.

- for $\eta = a_1 \cdots a_\ell \in Ac^*$, $\sigma(\eta)$ is defined only if $\sigma(a_i)$ is defined for all $i \leq \ell$, in which case it is defined to be $\sigma(a_1) \cdots \sigma(a_\ell)$.

A substitution σ is said to be *suitable for an action* a iff $\sigma(a)$ is defined, and *suitable for a sequence of actions* η iff $\sigma(\eta)$ is defined. σ is said to be *suitable for* a protocol Pr if $\sigma(t)$ is defined and equal to t for all constants $t \in \mathsf{CT}(\mathsf{Pr})$.

Example 2.2.6

- Here are two substitutions suitable for the protocol $\mathsf{Pr}_{\mathsf{NS}}$, presented in Example 2.2.4:
  - σ_1 given by: $\sigma_1(x) = m$, $\sigma_1(y) = n$, $\sigma(A) = A$ and $\sigma_1(B) = I$.
  - σ_2 given by: $\sigma_2(x) = m$, $\sigma_2(y) = n$ and $\sigma_2(A) = A$ and $\sigma_2(B) = B$.

  Of these σ_1 is suitable for η_1 and σ_2 is suitable for η_2. Notice that σ_1 is not suitable for η_2 since $\sigma_1(B) = I$ and $\eta_2 \in Ac_B^*$.

- Here are three substitutions suitable for the protocol presented in Example 2.2.5.
  - ς_1 given by: $\varsigma_1(x) = m$, $\varsigma_1(A) = A$ and $\varsigma_1(B) = B$.
  - ς_2 given by: $\varsigma_2(x) = (A, \{m\}_{pubk_B})$, $\varsigma_2(A) = I$, and $\varsigma_2(B) = B$.
  - ς_3 given by: $\varsigma_3(x) = m$, $\varsigma_3(A) = I$, and $\varsigma_3(B) = B$.

  Of these ς_1 is suitable for ζ_1 and ς_2 and ς_3 are suitable for ζ_2. Notice that ς_3 is not suitable for ζ_1 since $\zeta_1 \in Ac_A^*$ and $\varsigma_3(A) = I$. ς_2 is not suitable for ζ_1 for the same reason, and also since $x \in NT(\zeta_1)$ but $\varsigma_2(x) \notin \mathcal{T}_0$.

□

An *event* is a triple (η, σ, lp) such that $\eta \in Ac^+$, σ is a substitution suitable for η, and $1 \leq lp \leq |\eta|$. The set of all events is denoted *Events*. An event (η, σ, lp) is said to be *well-typed* iff σ is well-typed. For a set $T \subseteq \mathcal{T}_0$, an event (η, σ, lp) is said to be a *T-event* iff σ is a T-substitution. An event $e = (\eta, \sigma, lp)$ is said to be an event of a protocol Pr if $\eta \in Roles(\mathsf{Pr})$ and σ is suitable for Pr. The set of all events of Pr is denoted $Events(\mathsf{Pr})$.

For an event $e = (\eta, \sigma, lp)$ with $\eta = a_1 \cdots a_\ell$, $act(e) \stackrel{\text{def}}{=} \sigma(a_{lp})$. If $lp < |\eta|$ then $(\eta, \sigma, lp) \rightarrow_\ell (\eta, \sigma, lp + 1)$. For any event e, $LP(e)$, the *local past* of e, is defined to be the set of all events e' such that $e' \stackrel{+}{\rightarrow}_\ell e$. For any event e, $term(e)$ will be used to denote $term(act(e))$ and similarly for $NT(e)$, $ST(e)$, $EST(e)$, etc. For any sequence $\xi = e_1 \cdots e_k$ of events, $terms(\xi) \stackrel{\text{def}}{=} \bigcup_{1 \leq i \leq k} term(e_i)$. $NT(\xi)$, $ST(\xi)$, $EST(\xi)$ etc. are similarly defined.

For any sequence of events $\xi = e_1 \cdots e_k$, $Events(\xi) \stackrel{\text{def}}{=} \{e_1, \ldots, e_k\}$.

Message generation rules

Definition 2.2.7 *A* sequent *is of the form $T \vdash t$ where $T \subseteq \mathcal{T}$ and $t \in \mathcal{T}$.*

An analz*-proof (*synth*-proof) π of $T \vdash t$ is an inverted tree whose nodes are labelled by sequents and connected by one of the* analz*-rules (*synth*-rules) in Figure 2.1, whose root is labelled $T \vdash t$, and whose leaves are labelled by instances of the Ax_a rule (Ax_s rule). For a set of terms T,* analz(T) *(*synth(T)*) is the set of terms t such that there is an* analz*-proof (*synth*-proof) of $T \vdash t$.*

For ease of notation, synth(analz(T)) *is denoted by $\overline{T}$.*

Thus $\overline{T}$ represents the *closure* of T got by first "analysing" all terms in T into their subcomponents, using the analz-rules, and then "synthesizing" new terms using the synth-rules. Later, we will prove that this definition is equivalent to a different way of defining the closure of T, in which the synth and analz-rules are applied in an arbitrary order.

The analz-rule decrypt says that if the abstract term $\{t\}_k$ and $\overline{k}$ can be derived from T, then t can also be derived. This could either mean decrypting the encrypted term $\{t\}_k$ using the inverse key $\overline{k}$, or verifying the *signed term* $\{t\}_k$ using the corresponding *sign verifier* $\overline{k}$. Thus this is an abstract rule in which, depending on the status of k, the concrete algorithm which leads to the derivation of t differs. Similarly, the synth-rule encrypt could either denote either encryption or signing. The rule reduce really says that $\{\{t\}_k\}_{\overline{k}}$ is a different abstract notation which denotes the same term denoted by t. This is again a consequence of the fact that $\{t\}_k$ denotes different cryptographic algorithms — encryption, decryption, signing, verifying signatures, etc.

Example 2.2.8 Let $T = \{t\}$ where $t = (\{\{(m, n)\}_k\}_{k'}, (\overline{k}, \overline{k'}))$. The analz-proof given in Figure 2.2 shows that $m \in$ analz(T). To reduce clutter, we use the notation

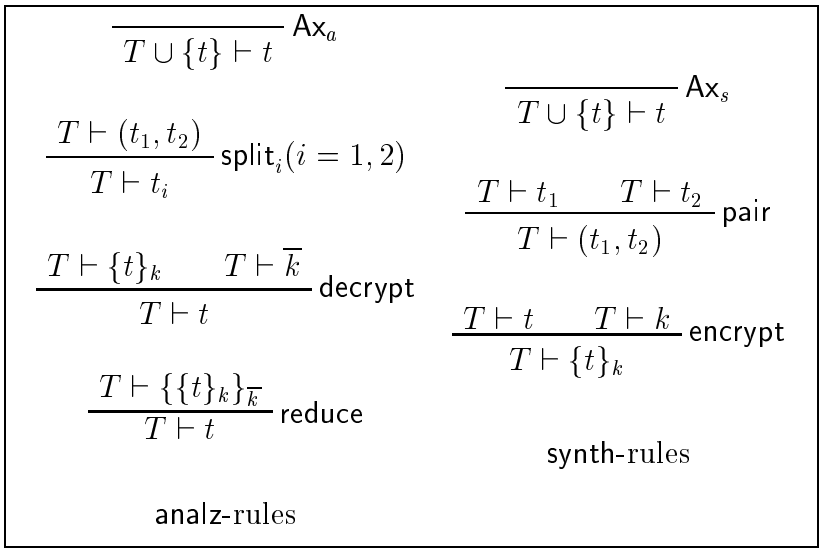

Figure 2.1: analz and synth rules.

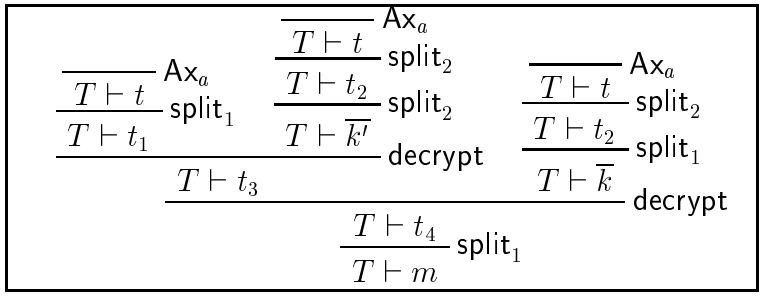

Figure 2.2: An example analz-proof.

t_1 for $\{\{(m,n)\}_k\}_{k'}$, t_2 for $(\overline{k}, \overline{k'})$, t_3 for $\{(m,n)\}_k$ and t_4 for (m,n). □

Example 2.2.9 Let $T = \{m, n, k, k'\}$ and $t = \{\{(m,n)\}_k\}_{k'}$. The synth-proof given in Figure 2.3 shows that $t \in \mathsf{synth}(T)$. For readability, we denote $\{(m,n)\}_k$ by t_1 and (m,n) by t_2. □

Example 2.2.10 Note that when $t' = (\{\{(m,n)\}_k\}_{k'}, (k, \overline{k'}))$, $m \notin \mathsf{analz}(\{t'\})$ unless $k = \overline{k}$. Also note that if $T'' = \{(n,m), k, k'\}$ and $t'' = \{\{(m,n)\}_k\}_{k'}$, $t'' \in \overline{T''}$ but $t'' \notin \mathsf{synth}(T'')$. □

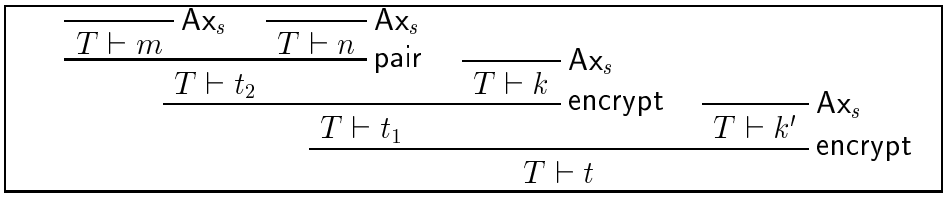

Figure 2.3: An example synth-proof.

Information states and updates

Definition 2.2.11 *The notions of an action enabled at a state and update of a state on an action are defined as follows:*

- $A!B{:}(M)t$ *is* enabled *at* s *iff* $t \in \overline{s_A \cup M}$.
- $A?B{:}t$ *is* enabled *at* s *iff* $t \in \overline{s_I}$.
- $update(s, A!B{:}(M)t) \stackrel{\text{def}}{=} s'$ *where* $s'_A = s_A \cup M \cup \{t\}$, $s'_I = s_I \cup \{t\}$, *and for all* $C \in Ag \setminus \{A, I\}$, $s'_C = s_C$.
- $update(s, A?B{:}t) \stackrel{\text{def}}{=} s'$ *where* $s'_A = s_A \cup \{t\}$ *and for all* $C \in Ag \setminus \{A\}$, $s'_C = s_C$.

$update(s, \varepsilon) = s$, $update(s, \eta \cdot a) = update(update(s, \eta), a)$.

In an action of the form $A!B{:}(M)t$, M is supposed to represent the set of new terms which are generated by the action. For such an action to be enabled at a state s, it is natural to expect that a *freshness* condition should hold, namely that none of the terms in M belong to $ST(s)$. We find it simpler to ensure this condition in the definition of runs (which occurs later in this section) rather than here. Since we usually look at states only in the context of runs, there are no technical problems as well.

Note that we have chosen to let I record only the terms communicated over the network, and not the sender and receiver information as well. This is a slight departure from the usual practice, and also from what was said in our informal discussion of the model. We choose the simpler alternative, since the choice here does not have a bearing on our main results.

Another aspect worth noting here is that the intruder is acting as an unbounded buffer which synchronises with each send and receive event of the honest agents. In effect the intruder is playing the role of the network as well, but there are some vital

differences. The intruder is assumed not to lose any message (even though it might not be sent to the intended recepient). This simplifies much of our analysis since at any point in time, the intruder has all the messages exchanged thus far. In a real-life situation the network (having finite memory) might lose some information and hence our analysis might get more complicated due to consideration of past information.

Definition 2.2.12 *Given an information state s and a sequence of events $\xi = e_1 \cdots e_k$, $infstate(s, e_1 \cdots e_k)$ is defined to be $update(s, act(e_1) \cdots act(e_k))$. An event e is said to be* enabled at (s, ξ) *iff $LP(e) \subseteq \{e_1, \ldots, e_k\}$ and $act(e)$ is enabled at $infstate(s, \xi)$.*

Given a protocol Pr and a sequence $\xi = e_1 \cdots e_k$ of events of Pr, $infstate_{\mathsf{Pr}}(\xi)$ is defined to be $infstate(\mathsf{init}(\mathsf{Pr}), e_1 \cdots e_k)$. We omit the subscript Pr if the context is clear. An event e of Pr is said to be enabled at a sequence ξ of events of Pr iff e is enabled at $(\mathsf{init}(\mathsf{Pr}), \xi)$.

The following two propositions, which state that if an agent A is not "involved" in an action a then a does not affect A's state, are easy consequences of the definition of update.

Proposition 2.2.13 *Suppose s is an information state, η is a finite sequence of actions, $A \in Ho$ and $a \notin Ac_A$. Then $(update(s, \eta))_A = (update(s, \eta \cdot a))_A$. As a consequence, for all information states s, all finite sequences of actions η and for all $A \in Ho$, $(update(s, \eta))_A = (update(s, \eta \restriction A))_A$.*

Proposition 2.2.14 *Suppose s is an information state, η is a finite sequence of actions, and a is a receive action. Then $(update(s, \eta))_I = (update(s, \eta \cdot a))_I$.*

Runs of a protocol

We isolate the sequences of events which can possibly occur as runs of protocols in the following definition. In the next definition, we define the set of runs of a given protocol.

Definition 2.2.15 *A sequence of events $e_1 \cdots e_k$ is said to be a* run *with respect to an information state s iff:*

- *for all $i : 1 \leq i \leq k$, e_i is enabled at $(s, e_1 \cdots e_{i-1})$,*

- *for all* $i : 1 \leq i \leq k$, $NT(e_i) \cap ST(s) = \emptyset$, *and for all* $i < j \leq k$, $NT(e_i) \cap NT(e_j) = \emptyset$. *(This is the* unique origination property *of runs.)*

A run is ξ *is said to be* well-typed *iff every* $e \in Events(\xi)$ *is well-typed. For a given* $T \subseteq \mathfrak{T}_0$, *a run* ξ *is said to be a* T-run *iff every* $e \in Events(\xi)$ *is a* T*-event.*

Definition 2.2.16 *Given a protocol* Pr, *a sequence* ξ *of events of* Pr *is said to be a run of* Pr *iff it is a run with respect to* $\mathsf{init}(\mathsf{Pr})$.

We let $\mathcal{R}(\mathsf{Pr})$ *denote the set of all runs of* Pr, $\mathcal{R}_{\mathsf{wt}}(\mathsf{Pr})$ *denote the set of all* well-typed *runs of* Pr, *and for any given* $T \subseteq \mathfrak{T}_0$, $\mathcal{R}_T(\mathsf{Pr})$ *denote the set of all* T-runs *of* Pr.

Note that in our definition of runs, we do not insist that every send event have a "matching" receive event. These would be the messages which are blocked by the intruder. There is no requirement that every receive should have a "matching" send, as well. These would be the messages which are generated and sent by the intruder (possibly under an assumed identity).

Example 2.2.17

- An example run of $\mathsf{Pr}_{\mathsf{NS}}$ is ξ_1, given below:

$$
\begin{array}{llllllll}
(\eta_1, \sigma_1, 1) & A & ! & I & : & (m) & \{A, m\}_{pubk_I} \\
(\eta_2, \sigma_2, 1) & B & ? & A & : & & \{A, m\}_{pubk_B} \\
(\eta_2, \sigma_2, 2) & B & ! & A & : & (n) & \{m, n\}_{pubk_A} \\
(\eta_1, \sigma_1, 2) & A & ? & I & : & & \{m, n\}_{pubk_A} \\
(\eta_1, \sigma_1, 3) & A & ! & I & : & & \{n\}_{pubk_I} \\
(\eta_2, \sigma_2, 3) & B & ? & A & : & & \{n\}_{pubk_B}
\end{array}
$$

  Here η_1 and η_2 are roles of $\mathsf{Pr}_{\mathsf{NS}}$ defined in Example 2.2.4 and σ_1 and σ_2 are substitutions suitable for $\mathsf{Pr}_{\mathsf{NS}}$ defined in Example 2.2.6.

- An example run of $\mathsf{Pr}_{\mathsf{ut}}$ is ξ_2, given below:

$$
\begin{array}{llllllll}
(\zeta_1, \varsigma_1, 1) & A & ! & B & : & (m) & \{A, \{m\}_{pubk_B}\}_{pubk_B} \\
(\zeta_2, \varsigma_2, 1) & B & ? & I & : & & \{I, \{A, \{m\}_{pubk_B}\}_{pubk_B}\}_{pubk_B} \\
(\zeta_2, \varsigma_2, 2) & B & ! & I & : & & \{A, \{m\}_{pubk_B}\}_{pubk_I} \\
(\zeta_2, \varsigma_3, 1) & B & ? & I & : & & \{I, \{m\}_{pubk_B}\}_{pubk_B} \\
(\zeta_2, \varsigma_3, 2) & B & ! & I & : & & \{m\}_{pubk_I} \\
(\zeta_1, \varsigma_1, 2) & A & ? & B & : & & \{m\}_{pubk_A}
\end{array}
$$

Again ζ_1 and ζ_2 are roles of $\mathsf{Pr}_{\mathsf{ut}}$ defined in Example 2.2.5 and ς_1, ς_2 and ς_3 are substitutions suitable for $\mathsf{Pr}_{\mathsf{ut}}$ defined in Example 2.2.6.

- Let us now look at some non-examples of runs. The following is not a run of $\mathsf{Pr}_{\mathsf{NS}}$ since the second message, which has been sent by the intruder to A, cannot be constructed by I from the rest of available information. Only B can decrypt the first message and learn m, which is a fresh nonce generated by A and so is unavailable to the intruder at any previous time.

$$\begin{array}{llllllll} (\eta_1,\sigma,1) & A & ! & B & : & (m) & \{A,m\}_{pubk_B} \\ (\eta_1,\sigma,2) & A & ? & B & : & & \{m,m\}_{pubk_A} \end{array}$$

  The following is not a run of $\mathsf{Pr}_{\mathsf{NS}}$ for the simple reason that property of unique origination of nonces is not maintained.

$$\begin{array}{llllllll} (\eta_1,\sigma,1) & A & ! & B & : & (m) & \{A,m\}_{pubk_B} \\ (\eta_2,\sigma',1) & B & ? & A & : & & \{A,m\}_{pubk_B} \\ (\eta_2,\sigma',2) & B & ! & A & : & (m) & \{m,m\}_{pubk_A} \\ (\eta_1,\sigma,2) & A & ? & B & : & & \{m,m\}_{pubk_A} \end{array}$$

□

The following is an easy consequence of the definition of runs.

Proposition 2.2.18 *Suppose $\xi = e_1 \cdots e_k$ is a run with respect to a state s. Then for all $i \leq k$, $NT(e_i) \cap ST(infstate(s, e_1 \cdots e_{i-1})) = \emptyset$.*

Proof: We first prove that $ST(infstate(s, e_1 \cdots e_{i-1})) \cap \mathcal{T}_0 = (ST(s) \cap \mathcal{T}_0) \cup NT(e_1 \cdots e_{i-1})$. For this it suffices to prove that for any sequence of actions η, $ST(update(s,\eta)) \cap \mathcal{T}_0 = (ST(s) \cap \mathcal{T}_0) \cup NT(\eta)$. For this, we first observe that for all states s and actions a, $ST(update(s,a)) \cap \mathcal{T}_0 = (ST(s) \cap \mathcal{T}_0) \cup NT(a)$. Now the statement is proved by an easy induction on $|\eta|$. The statement is immediate for $\eta = \varepsilon$. If $\eta = \eta' \cdot a$ then we note that $update(s,\eta) = update(s',a)$ where we denote $update(s,\eta')$ by s'. Therefore $ST(update(s,\eta)) \cap \mathcal{T}_0 = (ST(s') \cap \mathcal{T}_0) \cup NT(a)$. Now $NT(\eta) = NT(\eta') \cup NT(a)$, and by induction hypothesis, $ST(s') \cap \mathcal{T}_0 = ST(s) \cup NT(\eta')$, and thus the statement immediately follows.

Using the above fact, we prove the proposition. Since ξ has the unique origination property, it is clear that $NT(e_i) \cap ST(s) = \emptyset$ and $NT(e_i) \cap NT(e_j) = \emptyset$ for all $j < i$. This implies that $NT(e_i) \cap ST(\mathit{infstate}(s, e_1 \cdots e_{i-1})) = \emptyset$. □

Another aspect of our definition of runs is worth highlighting. We allow events to have more than one occurrence in a run (as long as they do not generate fresh nonces). This is not strictly necessary, since there is no information gain in repeating the same event many times. But we retain this definition, as imposing a condition on unique occurrence of events would make some of our definitions and proofs considerably messier. The following propositions suggest a way of removing duplicate events from a run in such a manner that the reduced run is leaky iff the original run is.

Definition 2.2.19 *The function* $\mathsf{red} : \mathit{Events} \to \mathit{Events}$ *is defined as follows:*

- $\mathsf{red}(\varepsilon) = \varepsilon$.
- $\mathsf{red}(\xi \cdot e) = \begin{cases} \mathsf{red}(\xi) \cdot e & \text{if } e \notin \mathit{Events}(\mathsf{red}(\xi)) \\ \mathsf{red}(\xi) & \text{otherwise} \end{cases}$

$\mathsf{red}(\xi)$ *is called the* reduced form *of* ξ. *We call* ξ *a* reduced run *iff* $\mathsf{red}(\xi) = \xi$.

It is easy to see that for any ξ, $\mathit{Events}(\xi) = \mathit{Events}(\mathsf{red}(\xi))$ and $\mathsf{red}(\xi)$ has at most one occurrence of each event.

Proposition 2.2.20 *Suppose* ξ *is a run with respect to* s_0. *Then:*

1. $\mathit{infstate}(s_0, \xi) = \mathit{infstate}(s_0, \mathsf{red}(\xi))$, *and*
2. $\mathsf{red}(\xi)$ *is also a run with respect to* s.

Proof:

1. This is quite easy to prove. We prove it by induction on the length of ξ. The base case is trivial, since $\mathsf{red}(\varepsilon) = \varepsilon$. For the induction step, there are two cases to consider:

   - Suppose $\xi = \xi' \cdot e$ and $e \in \mathit{Events}(\mathsf{red}(\xi'))$. Then $\mathsf{red}(\xi) = \mathsf{red}(\xi')$. Therefore $\mathit{infstate}(s_0, \mathsf{red}(\xi)) = \mathit{infstate}(s_0, \mathsf{red}(\xi'))$. Since $\mathit{infstate}(s_0, \xi') = \mathit{infstate}(s_0, \mathsf{red}(\xi'))$, by induction hypothesis, the desired result will follow

if we show that $infstate(s_0, \xi) = infstate(s_0, \xi')$. Denote $infstate(s_0, \xi) = s$ and $infstate(s_0, \xi') = s'$ for notational convenience. Let us consider the case when $act(e) = A!B{:}(M)t$. The case when e is a receive event is similarly handled. Since $e \in Events(\mathsf{red}(\xi'))$, $e \in Events(\xi')$ as well. Now if M were not empty, then it would mean that two distinct event occurrences of ξ generate the same new nonce (or key), which would be a violation of the unique origination property of the *run* ξ. Thus $M = \emptyset$. Further it follows from $e \in Events(\xi')$ and the definition of *update* that $t \in s'_A \cap s'_I$. From the definition of *update* and the fact that $M = \emptyset$, we also see that $s_A = s'_A \cup \{t\}$, $s_I = s'_I \cup \{t\}$ and $s_C = s'_C$ for all $C \in Ag \setminus \{A, I\}$. Since $t \in s'_A \cap s'_I$, it is clear that $s = s'$ and we are through.

- Suppose $\xi = \xi' \cdot e$ and $e \notin Events(\mathsf{red}(\xi'))$. Then $\mathsf{red}(\xi) = \mathsf{red}(\xi') \cdot e$. Further since $Events(\xi') = Events(\mathsf{red}(\xi'))$, $e \notin Events(\xi')$ as well. Denote $infstate(s_0, \xi') = s'$ and $infstate(s_0, \mathsf{red}(\xi')) = s'_1$ for notational convenience. Now $infstate(s_0, \xi) = update(s', act(e))$. But by induction hypothesis, $s' = s'_1$ and therefore $update(s', act(e))$ is equal to $update(s'_1, act(e))$, which is the same as $infstate(s_0, \mathsf{red}(\xi))$, by definition.

2. Since $Events(\xi) = Events(\mathsf{red}(\xi))$, $\mathsf{red}(\xi)$ also has the unique origination property. Further from the first part of the proposition, it follows that every event of $\mathsf{red}(\xi)$ is enabled at the end of the sequence of events preceding it.

□

The secrecy problem

Definition 2.2.21 *A basic term $m \in \mathcal{T}_0$ is said to be* secret at state *s iff there exists $A \in Ho$ such that $m \in \mathsf{analz}(s_A) \setminus \mathsf{analz}(s_I)$. Given a protocol* Pr *and $\xi \in \mathcal{R}(\mathsf{Pr})$, m is said to be secret at ξ if it is secret at $infstate(\xi)$. ξ is* leaky *iff there exists a basic term m and a prefix ξ' of ξ such that m is secret at ξ' and not secret at ξ.*

The secrecy problem is the problem of determining for a given protocol Pr *whether some run of* Pr *is leaky. The secrecy problem for well-typed runs is the problem of determining for a given protocol* Pr *whether some* well-typed *run of* Pr *is leaky. For a given $T \subseteq \mathfrak{T}_0$, the secrecy problem for T-runs is the problem of determining for a*

given protocol Pr *whether some* T-run *of* Pr *is leaky.*

Thus we say that a run is leaky if some atomic term is secret at some intermediate state of the run but is revealed to the intruder at the end of the run. It is possible that there are protocols for which leaks of the above form do not constitute a breach of security. A more general notion would be to *allow the user to specify certain secrets which should not be leaked* and check for such leaks. In later chapters, we prove the decidability of the secrecy problem (defined above) for a subclass of protocols. It is still not known whether there is a "reasonable" syntactic subclass of protocols for which the more general secrecy problem (which checks for leaks of user-specified secrets) is decidable.

Example 2.2.22

- The run ξ_1 of Example 2.2.17 is leaky. This is because n is secret at the prefix $\xi_1' = (\eta_1, \sigma_1, 1) \cdot (\eta_2, \sigma_2, 1) \cdot (\eta_2, \sigma_2, 2)$ of ξ_1, whereas it is not secret at ξ_1.
- Similarly, the run ξ_2 of Example 2.2.17 is also leaky, for m is secret at the prefix $\xi_2' = (\zeta_1, \varsigma_1, 1)$ of ξ_2, but it is not secret at ξ_2.

□

2.2.2 Well-formed protocols

In the literature, protocols are informally presented as a sequence of communications of the form $A \rightarrow B : t$. There are also some other "well-formedness" conditions which are implicitly assumed. In this section, we formalise these criteria and explore their consequences. The main property of well-formed protocols is that for each of their roles and plays, every send action in it is enabled by the previous actions. As a result, when we analyse well-formed protocols, checking enabledness of send actions by honest agents is relatively straightforward. If $e_1 \cdots e_k$ is a run of a well-formed protocol Pr and e is a send event such that $LP(e) \subseteq \{e_1, \cdots, e_k\}$, then as a consequence of the propositions proved in this section, e is enabled at ξ. Hence if the new terms introduced in e do not already occur in $e_1 \cdots e_k$, then $e_1 \cdots e_k \cdot e$ is also a run of the protocol. Thus the task of checking whether a send event is permissible at a given stage of a run is much simplified. In analysing a well-formed protocol, it suffices to check the enabledness of the receive actions (corresponding to

intruder sends). This has also been the standard practice in the analysis of security protocols. It can be seen that it is the implicit assumption of well-formedness that justifies this practice.

Well-formed Protocols

A *communication* is of the form $A \rightarrow B : (M)t$ where $A, B \in Ho$, $A \neq B$, $t \in \mathcal{T}$, and $M \subseteq ST(t) \cap (N \cup SN \cup K_0)$. For a communication $c = A \rightarrow B : (M)t$, $act_s(c)$ is defined to be $A!B:(M)t$ and $act_r(c)$ is defined to be $B?A:t$. Thus a communication specifies a send and a corresponding *instantaneous receive*. Communications are not necessarily implementable (because of the presence of the intruder), but nevertheless their use can lead to much simpler specifications of protocols than the role-based specifications.

For a sequence of communications δ, $actseq(\delta)$ is defined by induction as follows: $actseq(\varepsilon) = \varepsilon$; $actseq(\delta \cdot c) = actseq(\delta) \cdot act_s(c) \cdot act_r(c)$. Thus from any given sequence of communications we can obtain a sequence of actions by splitting each communication into a send and a corresponding receive. These sequences are used to obtain the semantics of *linear protocols* (defined below), which are specified in terms of communications. For any communication c, $term(c) \stackrel{\text{def}}{=} term(act_s(c))$. $NT(c)$, $ST(c)$ and $EST(c)$ are similarly defined. For any sequence of communications δ, $terms(\delta) \stackrel{\text{def}}{=} terms(actseq(\delta))$. $NT(\delta)$, $ST(\delta)$ and $EST(\delta)$ are similarly defined.

Definition 2.2.23 *A* linear protocol *is a pair* $\mathsf{Pr} = (\mathsf{C}, \delta)$ *where:*

- C*, the set of* constants *of* Pr*, denoted* $\mathsf{CT}(\mathsf{Pr})$*, is a subset of* $\mathcal{T}_0$ *with the property that* $\{\mathsf{n}_0, \mathsf{m}_0, \mathsf{k}_0\} \cap \mathsf{C} = \emptyset$*, and*
- δ*, the body of the protocol, is a nonempty sequence of communications.*

Given a linear protocol $\mathsf{Pr} = (\mathsf{C}, \delta)$*,* $Roles(\mathsf{Pr})$*, the set of* roles *of* Pr*, is defined to be the set* $\{\eta\restriction A \mid A \in Ho \text{ and } \eta\restriction A \neq \varepsilon\}$ *where* $\eta = actseq(\delta)$*.*

Example 2.2.24 The protocol $\mathsf{Pr}_{\mathsf{NS}}$ presented earlier is a linear protocol, with the following specification:

$$\begin{array}{llllll} 1. & A & \rightarrow & B & : (x) & \{A, x\}_{pubk_B} \\ 2. & B & \rightarrow & A & : (y) & \{x, y\}_{pubk_A} \\ 3. & A & \rightarrow & B & : & \{y\}_{pubk_B} \end{array}$$

The protocol $\mathsf{Pr}_{\mathsf{ut}}$ presented earlier is also a linear protocol, with the following specification:

$$\begin{array}{llllll} 1. & A & \rightarrow & B & : (x) & \{A, \{x\}_{pubk_B}\}_{pubk_B} \\ 2. & B & \rightarrow & A & : & \{x\}_{pubk_A} \end{array}$$

□

Even though the presentations look different, linear protocols can in fact be viewed as a subclass of protocols as defined in Definition 2.2.2, as the following proposition asserts.

Proposition 2.2.25 *If* $\mathsf{Pr} = (\mathsf{C}, \delta)$ *is a linear protocol, then* $(\mathsf{C}, Roles(\mathsf{Pr}))$ *is a protocol.*

The proof is by just observing the definitions. This proposition allows us to freely use the standard notions associated with protocols (like $\mathsf{init}(\mathsf{Pr})$, for instance) for linear protocols as well. Note that the converse of the above proposition is not true. It is possible to come up with protocols which have no representation as a linear protocol.

Definition 2.2.26 *A sequence of actions* $\eta = a_1 \cdots a_\ell$ *is said to be* send-admissible *with respect to a state* s *iff for all* $i \leq \ell$*, if* a_i *is a send action then* a_i *is enabled at* $update(s, a_1 \cdots a_{i-1})$*.* η *is said to be send-admissible with respect to a protocol* Pr *iff it is send-admissible with respect to* $\mathsf{init}(\mathsf{Pr})$*.*

Definition 2.2.27 *A* well-formed protocol *is a linear protocol* $\mathsf{Pr} = (\mathsf{C}, \delta)$ *such that* $actseq(\delta)$ *is send-admissible with respect to* Pr*.*

Proposition 2.2.28 *Suppose* $\mathsf{Pr} = (\mathsf{C}, \delta)$ *is a well-formed protocol. Then all its roles are send-admissible with respect to* Pr*.*

Proof: For simplicity of notation, let s_0 denote $\mathsf{init}(\mathsf{Pr})$. Let $\eta = actseq(\delta)$. Suppose $\eta = a_1 \cdots a_\ell$ and suppose $\zeta = a_{i_1} \cdots a_{i_r}$ is a role of Pr, i.e., $\zeta = \eta \restriction A$ for some $A \in Ho$. By Proposition 2.2.13, it is clear that for all $j : 1 \leq j \leq r$, $(update(s_0, a_1 \cdots a_{i_j}))_A = (update(s_0, a_{i_1} \cdots a_{i_j}))_A$. Since Pr is a well-formed protocol, η is send-admissible with respect to Pr. The send-admissibility of ζ now follows

from the above equality. □

Proposition 2.2.29 *Suppose* $\mathsf{Pr} = (\mathsf{C}, \delta)$ *is a well-formed protocol,* ζ *is a role of* Pr *and* σ *is a substitution suitable for* Pr *and* ζ*. Then* $\sigma(\zeta)$ *is send-admissible with respect to* Pr*.*

Proof: For simplicity of notation, let s_0 denote $\mathsf{init}(\mathsf{Pr})$. Let $\eta = \mathit{actseq}(\delta)$. Note that $\zeta = \eta\restriction A$ for some $A \in \mathit{Ho}$. Since σ is suitable for Pr and ζ, σ is defined on all actions occurring in ζ, and $\sigma(m) = m$ for all $m \in \mathsf{CT}(\mathsf{Pr})$. We first prove for all prefixes ζ' of ζ that $\sigma(s'_A) \subseteq (s'_1)_{\sigma(A)}$ by induction on $|\zeta'|$ (where we denote $\mathit{update}(s_0, \zeta')$ by s' and $\mathit{update}(s_0, \sigma(\zeta'))$ by s'_1):

$\zeta' = \varepsilon$*:* In this case $s' = s'_1 = s_0$. Now it is clear that $\sigma(\mathsf{C}) = \mathsf{C}$ and $\sigma(K_A) = K_{\sigma(A)}$. Since $A \in \mathit{Ho}$, $\sigma((s_0)_A) = \mathsf{C} \cup \sigma(K_A)$. Further $(s_0)_{\sigma(A)} \supseteq \mathsf{C} \cup K_{\sigma(A)}$ (with inequality when $\sigma(A) = I$). It immediately follows that $\sigma(s'_A) \subseteq (s'_1)_{\sigma(A)}$ in this case.

$\zeta' = \zeta'' \cdot a$*:* Note that $\sigma(\zeta') = \sigma(\zeta'') \cdot \sigma(a)$. For simplicity let us denote $\mathit{update}(s_0, \zeta'')$ by s'' and $\mathit{update}(s_0, \sigma(\zeta''))$ by s''_1. We need to prove that $\sigma(s'_A) \subseteq (s'_1)_{\sigma(A)}$ assuming that $\sigma(s''_A) \subseteq (s''_1)_{\sigma(A)}$.

Now if $a = A!B\!:\!(M)t$ then $s'_A = s''_A \cup M \cup \{t\}$. Since $\sigma(s''_A) \subseteq (s''_1)_{\sigma_A}$, and since $\sigma(s'_A) = \sigma(s''_A) \cup \sigma(M) \cup \{\sigma(t)\}$ and $(s'_1)_{\sigma(A)} = (s''_1)_{\sigma_A} \cup \sigma(M) \cup \{\sigma(t)\}$ (because $\sigma(a) = \sigma(A)!\sigma(B)\!:\!(\sigma(M)\sigma(t))$, it follows that $\sigma(s'_A) \subseteq (s'_1)_{\sigma(A)}$.

The case when $a = A?B\!:\!t$ is identically handled. This proves the induction case.

From Proposition 2.2.28 it follows that ζ is send-admissible. Now consider any prefix $\zeta' \cdot a$ of ζ with $a \in \mathit{Send}$. For simplicity let us denote $\mathit{update}(s_0, \zeta')$ by s' and $\mathit{update}(s_0, \sigma(\zeta'))$ by s'_1. We know that $\mathit{term}(a) \in \overline{s'_A \cup \mathit{NT}(a)}$. Therefore $\mathit{term}(\sigma(a)) = \sigma(\mathit{term}(a)) \in \sigma(\overline{s'_A \cup \mathit{NT}(a)})$. But item 3 of Proposition 2.3.6 says that $\sigma(\overline{T}) \subseteq \overline{\sigma(T)}$ for any σ and T. Further $\sigma(s'_A \cup \mathit{NT}(a)) = \sigma(s'_A) \cup \mathit{NT}(\sigma(a))$ and by what has been proved above $\sigma(s'_A) \subseteq (s'_1)_{\sigma(A)}$. Putting all this together we see that $\mathit{term}(\sigma(a)) \in \overline{(s'_1)_{\sigma(A)} \cup \mathit{NT}(\sigma(a))}$. This shows that $\sigma(\zeta)$ is also send-admissible. □

Tagged Protocols

While well-formed protocols enforce a reasonableness condition at the level of protocol specifications, we must note that they still allow for quite unreasonable behaviours. Substituting encrypted terms for nonces can give the intruder the ability to circumvent the protocol. For instance, a communication of the form $A \rightarrow B : \{(A, \{x\}_B)\}_B$ in the protocol allows the intruder to capture it and send it on to B as: $I \rightarrow B : \{(I, \{(A, \{x\}_B)\}_B)\}_B$. On receipt B will interpret $(A, \{x\}_B)$ as a nonce and act accordingly. Depending on the situation, such a possibility might have undesirable consequences. For example, consider the following protocol:

$$\begin{array}{llllll} 1. & A & \rightarrow & B & : (x) & \{x\}_{pubk_B} \\ 2. & B & \rightarrow & A & : & \{x\}_{k_{AB}} \end{array}$$

B receives a nonce encrypted in its own public key and sends it back to the sender encrypted in the key k_{AB} shared by them. Consider the following run now (let η_1 denote the initiator role, and η_2 the responder role):

$$\begin{array}{llllll} (\eta_1, \sigma, 1) & A & ! & B & : (m) & \{m\}_{pubk_B} \\ (\eta_2, \sigma, 1) & B & ? & A & : & \{m\}_{pubk_B} \\ (\eta_2, \sigma, 2) & B & ! & A & : & \{m\}_{k_{AB}} \\ (\eta_2, \sigma', 1) & B & ? & A & : & \{\{m\}_{k_{AB}}\}_{pubk_B} \\ (\eta_2, \sigma', 2) & B & ! & A & : & \{\{m\}_{k_{AB}}\}_{k_{AB}} \end{array}$$

At the end of the run above the intruder manages to learn $\{\{m\}_{k_{AB}}\}_{k_{AB}}$. Since $k_{AB} = \overline{k_{AB}}$, using the reduce rule we say that $m \in \overline{s_I}$, where s is the state at the end of the above run. This situation arises because B interprets $\{m\}_{k_{AB}}$ as a nonce and encrypts it and hands it over to the other party, in effect decrypting the message for the intruder. It is thus useful to look at ways to prevent such attacks from happening. *Tagging* is one such mechanism that seeks to distinguish between terms of different structure and prevent attacks such as the above. More specifically, tags are just constants which act as message identifiers and are attached to some of the encrypted subterms of messages which are communicated during a run. The use of tags has the effect of preventing the intruder from passing off a term $\sigma(\{t\}_k)$ as $\sigma'(\{t'\}_{k'})$ in some run of a protocol while $\{t\}_k$ and $\{t'\}_{k'}$ are intended to be distinct terms in the protocol specification. We also use tagging to associate every receive action occurring in a run with its *corresponding send* (if there exists one).

To precisely highlight the assumptions used in the decidablity proofs in later chapter, we define two tagging schemes, one of which subsumes the other.

Definition 2.2.30 *A well-formed protocol* $\mathsf{Pr} = (\mathsf{C}, \delta)$ *is called a* weakly tagged protocol *iff for all* $t \in EST(\delta)$ *there exists* $\mathsf{c}_t \in \mathsf{C}$ *such that:*

- *for all* $t, t' \in EST(\delta)$, *if* $\mathsf{c}_t = \mathsf{c}_{t'}$ *then* $t = t'$, *and*
- *for all* $t \in EST(\delta)$: $t = \{(\mathsf{c}_t, u)\}_k$ *for some* u *and* k.

Definition 2.2.31 *A well-formed protocol* $\mathsf{Pr} = (\mathsf{C}, \delta)$ *with* $\delta = c_1 \cdots c_\ell$ *is called a* tagged protocol *iff for all* $t \in EST(\delta)$ *there exists* $\mathsf{c}_t \in \mathsf{C}$, *and for all* $i \leq \ell$ *there exists* $n_i \in NT(c_i) \cap SN$ *such that:*

- *for all* $i, j \leq \ell$, $t \in EST(c_i)$, *and* $t' \in EST(c_j)$: *if* $\mathsf{c}_t = \mathsf{c}_{t'}$ *then* $t = t'$ *and* $i = j$, *and*
- *for all* $i \leq \ell$ *and all* $t \in EST(c_i)$: $t = \{(\mathsf{c}_t, (n_i, u))\}_k$ *for some* u *and* k.

It is clear that every tagged protocol is also weakly tagged. Hence all the results which we prove for weakly tagged protocols hold for tagged protocols as well.

The weak tagging scheme which we have presented is essentially derived from the schemes presented in [HLS00] and [BP03], whereas there are some new features in the second tagging scheme that we have presented. Most of the standard protocols occurring in the literature (see [CJ97] for example) can be easily tagged to obtain "equivalent protocols", such that for any run ξ of the original protocol which involves only honest agents, the tagged version of ξ is a run of the transformed protocol, and for all runs ξ of the transformed protocol, the untagged version of ξ is a run of the original protocol. (Thus the transformation does not limit the honest agents' capabilities while at the same time not introducing more attacks). But we should note that for some protocols which contain "blind copies" — like the Woo-Lam protocol Π (as presented in [CJ97]) — the second tagging scheme cannot be effected to get an equivalent tagged protocol. The problem would occur if an agent A cannot decrypt an encrypted term which it is blindly passing on. The second tagging scheme requires a distinct tag to be added for each c_i, but A cannot effect the retagging. But on the other hand, we can always apply the weak tagging scheme to any well-formed protocol to get an equivalent weakly tagged protocol. The problem of blind copies does not arise now, because the tags do not depend on the communication but only the structure of the encrypted terms. So there is no need to change the tags of terms which are blindly passed on.

An important point worth noting here is that including the tags in the protocol specification stage rather than later, in the run generation stage, means that the legality of the runs (with respect to the tagging scheme) can be enforced by checks performed by the honest participants of the protocol.

It should also be noted that *sequence numbers* are used in an essential way in the second tagging scheme. Even though the tagging scheme entails unboundedly many new tags to be used in protocol runs, still it does not involve much cost. Since sequence numbers are not required to be unguessable, even simple schemes like using a counter suffice to generate an unbounded number of them. This is different from generating nonces, where the real hard work is in ensuring unguessability.

The main purpose of the tagging schemes is to ensure the following properties of runs of tagged protocols. These properties are easy consequences of the definition of tagged protocols (and weakly tagged protocols), and are very important for the decidability proofs in the later chapters.

Proposition 2.2.32

- *Suppose* $\mathsf{Pr} = (\mathsf{C}, \delta)$ *is a weakly tagged protocol. Then for all* σ, σ' *suitable for* Pr *and for all* $t, t' \in EST(\delta)$*, if* $\sigma(t) = \sigma'(t')$ *then* $t = t'$*.*

- *Suppose* $\mathsf{Pr} = (\mathsf{C}, c_1 \cdots c_\ell)$ *is a tagged protocol. Then the following statements hold:*

  - *for all* σ, σ' *suitable for* Pr*, for all* $i, j \leq \ell$*, for all* $t \in EST(c_i)$ *and for all* $t' \in EST(c_j)$*, if* $\sigma(t) = \sigma'(t')$ *then* $t = t'$ *and* $i = j$*.*

  - *Suppose* $e_1 \cdots e_r$ *is a well-typed run of* Pr*. For all receive events* $e_k (k \leq r)$*, there is at most one send event* e_i *such that* $EST(e_i) \cap EST(e_k) \neq \emptyset$*.*

Proof:

- Suppose $t, t' \in EST(\delta)$ and σ, σ' suitable for Pr such that $\sigma(t) = \sigma'(t')$. By definition of weakly tagged protocols, it follows that $t = \{(\mathsf{c}_t, u)\}_k$ and $t' = \{(\mathsf{c}_{t'}, u')\}_{k'}$ for some u, u', k and k'. It follows that $\sigma(\mathsf{c}_t) = \sigma'(\mathsf{c}_{t'})$. But since σ and σ' are suitable for Pr, and since $\mathsf{c}_t, \mathsf{c}_{t'} \in \mathsf{C}$, $\sigma(\mathsf{c}_t) = \mathsf{c}_t$ and $\sigma'(\mathsf{c}_{t'}) = \mathsf{c}_{t'}$. Therefore $\mathsf{c}_t = \mathsf{c}_{t'}$. Now it follows from the definition of weakly tagged protocols that $t = t'$.

- We now take up the proofs of the statements relating to tagged protocols.

– Suppose $i, j \leq \ell$, $t \in EST(c_i)$, $t' \in EST(c_j)$ and σ, σ' suitable for Pr such that $\sigma(t) = \sigma'(t')$. By definition of tagged protocols, it follows that $t = \{(\mathsf{c}_t, u)\}_k$ and $t' = \{(\mathsf{c}_{t'}, u')\}_{k'}$ for some u, u', k and k'. It follows that $\sigma(\mathsf{c}_t) = \sigma'(\mathsf{c}_{t'})$. But since σ and σ' are suitable for Pr, and since $\mathsf{c}_t, \mathsf{c}_{t'} \in \mathsf{C}$, $\sigma(\mathsf{c}_t) = \mathsf{c}_t$ and $\sigma'(\mathsf{c}_{t'}) = \mathsf{c}_{t'}$. Therefore $\mathsf{c}_t = \mathsf{c}_{t'}$. Now it follows from the definition of tagged protocols that $t = t'$ and $i = j$.

– Suppose $e_1 \cdots e_r$ is a well-typed run of Pr and suppose there is a receive event e_k and two send events e_i and e_j (with $i \neq j$) such that neither $EST(e_i)$ nor $EST(e_j)$ is disjoint from $EST(e_k)$. Suppose $t_i \in EST(e_i) \cap EST(e_k)$ and $t_j \in EST(e_j) \cap EST(e_k)$. From the definition of tagged protocols it is clear that for all events e of ξ, there exists a nonce n such that for all $t \in EST(e)$, $t = \{(\mathsf{c}_t, (n, u))\}_k$ for some u and k. Further if e is a send event, $n \in NT(e)$. Thus there exist $n_i \in NT(e_i)$ and $n_j \in NT(e_j)$ such that $t_i = \{(\mathsf{c}_{t_i}, (n_i, u_i))\}_{k_i}$ and $t_j = \{(\mathsf{c}_{t_j}, (n_j, u_j))\}_{k_j}$ for some u_i, u_j, k_i and k_j. Now both t_i and t_j belong to $EST(e_k)$, therefore it follows that $n_i = n_j$. But then $n_i \in NT(e_i) \cap NT(e_j)$, which violates the property of unique origination of nonces. This contradicts the fact that ξ is a run.

  This contradiction leads us to conclude that there is at most one i such that $EST(e_i) \cap EST(e_k) \neq \emptyset$.

□

2.3 Properties of synth and analz

In this section, we prove several useful results about synth and analz proofs, which will be used throughout the rest of the thesis.

We start off with the following simple observation:

Fact 2.3.1 *For any set of terms T and any term $t \in \mathsf{synth}(T)$, at least one of the following conditions holds:*

- $t \in T$.
- *t is of the form (t', t'') and $\{t', t''\} \subseteq \mathsf{synth}(T)$.*

- *t is of the form $\{t'\}_k$ and $\{t', k\} \subseteq \mathsf{synth}(T)$.*

This fact follows immediately from the definition of synth-proofs. In many situations, this fact helps us to replace induction on synth-proofs by (the much simpler) induction on structure of terms.

Some basic facts about the synth and analz operators are proved in the following proposition.

Proposition 2.3.2 *Let $T, T' \subseteq \mathcal{T}$ and $t \in \mathcal{T}$. Then the following properties hold:*

1. $T \subseteq \mathsf{analz}(T)$.
2. $T \subseteq \mathsf{synth}(T)$.
3. *If $T \subseteq T'$, then* $\mathsf{analz}(T) \subseteq \mathsf{analz}(T')$.
4. *If $T \subseteq T'$, then* $\mathsf{synth}(T) \subseteq \mathsf{synth}(T')$.
5. $\mathsf{analz}(\mathsf{analz}(T)) = \mathsf{analz}(T)$.
6. $\mathsf{synth}(\mathsf{synth}(T)) = \mathsf{synth}(T)$.
7. $t \in \mathsf{synth}(T)$ *iff* $t \in \mathsf{synth}(T \cap ST(t))$.

Proof: The statements relating to analz are proved by a simple induction on analz-proofs, and the statements relating to synth are proved by a simple induction on the structure of terms. We just prove statements 5 and 6 to give a flavour of the proofs.

Proof of statement 5: It is immediate that $\mathsf{analz}(T) \subseteq \mathsf{analz}(\mathsf{analz}(T))$, from statements 1 and 3. We prove the other inclusion. Suppose $t \in \mathsf{analz}(\mathsf{analz}(T))$. Suppose π is an analz-proof of $\mathsf{analz}(T) \vdash t$. We prove by structural induction that for every subproof ϖ of π with root labelled $\mathsf{analz}(T) \vdash r$, $r \in \mathsf{analz}(T)$. From this it follows that $t \in \mathsf{analz}(T)$ as well.

Suppose ϖ is a subproof of π with root labelled $\mathsf{analz}(T) \vdash r$ such that for all proper subproofs ϖ_1 of ϖ with root labelled $\mathsf{analz}(T) \vdash r_1$, $r_1 \in \mathsf{analz}(T)$. Then we prove that $r \in \mathsf{analz}(T)$ as well.

- Suppose ϖ is the following proof:

$$\frac{}{\mathsf{analz}(T) \vdash r}\ \mathsf{Ax}_a$$

Then $r \in \mathsf{analz}(T)$ by definition and we are through.

- Suppose ϖ is the following proof:

$$\dfrac{\begin{array}{c}(\varpi_1)\\ \vdots \\ \mathsf{analz}(T) \vdash (r, r')\end{array}}{\mathsf{analz}(T) \vdash r}\,\mathsf{split}_1$$

  By induction hypothesis $(r, r') \in \mathsf{analz}(T)$ and thus it immediately follows by definition of analz-proofs that $r \in \mathsf{analz}(T)$ as well.

- Suppose ϖ is the following proof:

$$\dfrac{\begin{array}{cc}(\varpi_1) & (\varpi_2)\\ \vdots & \vdots \\ \mathsf{analz}(T) \vdash \{r\}_k & \mathsf{analz}(T) \vdash \overline{k}\end{array}}{\mathsf{analz}(T) \vdash r}\,\mathsf{decrypt}$$

  By induction hypothesis $\{\{r\}_k, \overline{k}\} \subseteq \mathsf{analz}(T)$ and thus it immediately follows by definition of analz-proofs that $r \in \mathsf{analz}(T)$.

- Suppose ϖ is the following proof:

$$\dfrac{\begin{array}{c}(\varpi_1)\\ \vdots \\ \mathsf{analz}(T) \vdash \{\{r\}_k\}_{\overline{k}}\end{array}}{\mathsf{analz}(T) \vdash r}\,\mathsf{reduce}$$

  By induction hypothesis $\{\{r\}_k\}_{\overline{k}} \in \mathsf{analz}(T)$ and thus it immediately follows by definition of analz-proofs that $r \in \mathsf{analz}(T)$.

Proof of statement 6: It is immediate that $\mathsf{synth}(T) \subseteq \mathsf{synth}(\mathsf{synth}(T))$, from the statements 2 and 4. We now prove by induction on the structure of terms that if $t \in \mathsf{synth}(\mathsf{synth}(T))$ then $t \in \mathsf{synth}(T)$. From Fact 2.3.1, it suffices to consider the following three cases:

$t \in \mathsf{synth}(T)$: Then the conclusion trivially follows.

t is of the form (t', t'') and $\{t', t''\} \subseteq \mathsf{synth}(\mathsf{synth}(T))$: By induction hypothesis, it follows that $\{t', t''\} \subseteq \mathsf{synth}(T)$. It now immediately follows from the definition of synth-proofs that $t \in \mathsf{synth}(T)$.

t is of the form $\{t'\}_k$ and $\{t', k\} \subseteq \mathsf{synth}(T)$: By induction hypothesis, it follows that $\{t', k\} \subseteq \mathsf{synth}(T)$. It now immediately follows from the definition of synth-proofs that $t \in \mathsf{synth}(T)$.

□

It immediately follows from the above proposition that $\overline{T} = \mathsf{synth}(\mathsf{analz}(T))$ is closed under synth. The following proposition says that it is closed under analz as well, thus immediately implying the important statement that $\overline{\overline{T}} = \overline{T}$ for all sets of terms T.

Proposition 2.3.3 *For all $T \subseteq \mathcal{T}$, $\mathsf{analz}(\overline{T}) = \overline{T}$.*

Proof: From item 1 of Proposition 2.3.2, $\overline{T} \subseteq \mathsf{analz}(\overline{T})$. We prove the other inclusion now. Suppose $t \in \mathsf{analz}(\overline{T})$. Suppose π is an analz-proof of $\overline{T} \vdash t$. We prove by structural induction that for every subproof ϖ of π with root labelled $\overline{T} \vdash r$, $r \in \overline{T}$. From this it follows that $t \in \overline{T}$ as well.

Suppose ϖ is a subproof of π with root labelled $\overline{T} \vdash r$ such that for all proper subproofs ϖ_1 of ϖ with root labelled $\overline{T} \vdash r_1$, $r_1 \in \overline{T}$. Then we prove that $r \in \overline{T}$ as well. We only consider the case when the rule applied at the root of ϖ is Ax_a or $\mathsf{decrypt}$. The other cases can be similarly handled.

- Suppose ϖ is the following proof:

$$\frac{}{\overline{T} \vdash r}\,\mathsf{Ax}_a$$

  Then $r \in \overline{T}$ by definition and we are through.

- Suppose ϖ is the following proof:

$$\frac{\begin{array}{cc}(\varpi_1) & (\varpi_2)\\ \vdots & \vdots \\ \overline{T} \vdash \{r\}_k & \overline{T} \vdash \overline{k}\end{array}}{\overline{T} \vdash r}\,\mathsf{decrypt}$$

$$\frac{T \vdash (t_1, t_2)}{T \vdash t_i}\ \mathsf{split}_i (i = 1, 2) \qquad \frac{}{T \cup \{t\} \vdash t}\ \mathsf{Ax}$$

$$\frac{T \vdash \{t\}_k \qquad T \vdash \overline{k}}{T \vdash t}\ \mathsf{decrypt} \qquad \frac{T \vdash t_1 \qquad T \vdash t_2}{T \vdash (t_1, t_2)}\ \mathsf{pair}$$

$$\frac{T \vdash \{\{t\}_k\}_{\overline{k}}}{T \vdash t}\ \mathsf{reduce} \qquad \frac{T \vdash t \qquad T \vdash k}{T \vdash \{t\}_k}\ \mathsf{encrypt}$$

Figure 2.4: yields-rules.

By induction hypothesis $\{\{r\}_k, \overline{k}\} \subseteq \overline{T}$. From the definition of synth-proofs it follows that for all atomic terms m, if $m \in \overline{T} = \mathsf{synth}(\mathsf{analz}(T))$, then $m \in \mathsf{analz}(T)$. Since $\overline{k}$ is an atomic term, it follows that $\overline{k} \in \mathsf{analz}(T)$. Since $\{r\}_k \in \mathsf{synth}(\mathsf{analz}(T))$, it follows by Fact 2.3.1 that either $\{r\}_k \in \mathsf{analz}(T)$ or $\{r, k\} \subseteq \mathsf{synth}(\mathsf{analz}(T))$. In the first case, since $\overline{k} \in \mathsf{analz}(T)$, it follows that $r \in \mathsf{analz}(T) \subseteq \overline{T}$. In the second case also $r \in \overline{T}$ and we are through.

□

Following [Pau98], we have taken $\mathsf{synth}(\mathsf{analz}(T))$ as the set of terms which can be built from T. This means that we are considering only "normal proofs" — in which all the analysis rules are applied before the synth rules — in building up new terms from old. An alternate approach would be to consider proofs which involve synth and analz rules applied in an arbitrary order. This approach is also common in the security protocol literature. (For example, [FHG99] and [DLMS99] follow this approach.) We now show that both the approaches are equivalent.

Definition 2.3.4 *An* yields*-proof* π *of* $T \vdash t$ *is an inverted tree whose nodes are labelled by sequents and connected by one of the* yields*-rules in Figure 2.4, whose root is labelled* $T \vdash t$*, and whose leaves are labelled by instances of the* Ax *rule. For a set of terms* T*,* $\widehat{T}$ *is the set of terms* t *such that there is a* yields*-proof of* $T \vdash t$*.*

Proposition 2.3.5 *For all sets of terms* T*,* $\overline{T} = \widehat{T}$*.*

Proof: The inclusion from left to right is trivial, since both the analz-rules and the synth-rules are included in the yields-rules.

We consider the inclusion from right to left now. From item 6 of Proposition 2.3.2 it follows that $\mathsf{synth}(\overline{T}) \subseteq \overline{T}$. Proposition 2.3.3 says that $\mathsf{analz}(\overline{T}) \subseteq \overline{T}$. It follows as an immediate consequence of this that $\widehat{T} \subseteq \overline{T}$. □

Proposition 2.3.6 *Suppose T is a set of terms and σ is a substitution such that $\sigma(t)$ is defined for all $t \in T$. Then*

1. $\sigma(\mathsf{analz}(T)) \subseteq \mathsf{analz}(\sigma(T))$.
2. $\sigma(\mathsf{synth}(T)) \subseteq \mathsf{synth}(\sigma(T))$.
3. $\sigma(\overline{T}) \subseteq \overline{\sigma(T)}$.

Proof: We first note the following simple facts: if $t \in T$ then $\sigma(t) \in \sigma(T)$; $\sigma((t,t')) = (\sigma(t), \sigma(t'))$; $\sigma(\{t\}_k) = \{\sigma(t)\}_{\sigma(k)}$; $\sigma(\{\{t\}_k\}_{\overline{k}}) = \{\{\sigma(t)\}_{\sigma(k)}\}_{\overline{\sigma(k)}}$.

From these it follows that if $\dfrac{T \vdash t}{T \vdash t}$ is a analz-rule, so is $\dfrac{\sigma(T) \vdash \sigma(t)}{\sigma(T) \vdash \sigma(t')}$. A similar statement holds for binary analz-rules and for synth-rules as well (both unary and binary). Statements 1 and 2 immediately follow from these observations. Statement 3 can now be proved as follows: $\sigma(\overline{T}) = \sigma(\mathsf{synth}(\mathsf{analz}(T))) \subseteq \mathsf{synth}(\sigma(\mathsf{analz}(T))) \subseteq \mathsf{synth}(\mathsf{analz}(\sigma(T))) = \overline{\sigma(T)}$. □

Proposition 2.3.7 *For all sets of terms T and terms t, if $t \in ST(\mathsf{synth}(T))$ then either $t \in ST(T)$ or $t \in \mathsf{synth}(T)$.*

Proof: Suppose $t \in ST(\mathsf{synth}(T))$. We prove by induction on the structure of terms that for all r, if $r \in \mathsf{synth}(T)$ and $t \in ST(r)$ then either $t \in ST(T)$ or $t \in \mathsf{synth}(T)$. Bt Fact 2.3.1, it suffices to consider the following three cases:

$r \in T$: Then clearly $t \in ST(T)$.

r is of the form (r', r'') and $\{r', r''\} \subseteq \mathsf{synth}(T)$: There are two cases to consider. If $t = r = (r', r'')$ then clearly $t \in \mathsf{synth}(T)$. Otherwise $t \in ST(r) = ST(r') \cup ST(r'')$ and now we can apply to the induction hypothesis and conclude that $t \in ST(T)$ or $t \in \mathsf{synth}(T)$.

$r = \{r'\}_k$ *and* $\{r', k\} \subseteq \mathsf{synth}(T)$*:* This case is handled the same way as the previous one.

□

Proposition 2.3.8 *For all sets of terms T and terms $\{t\}_k$, if $\{t\}_k \in ST(\mathsf{synth}(T))$ then either $\{t\}_k \in ST(T)$ or $\{t, k\} \subseteq \mathsf{synth}(T)$.*

Proof: Suppose $\{t\}_k \in ST(\mathsf{synth}(T))$. From Proposition 2.3.7 we conclude that either $\{t\}_k \in ST(T)$ or $\{t\}_k \in \mathsf{synth}(T)$. But if $\{t\}_k \in \mathsf{synth}(T)$ then either $\{t\}_k \in T \subseteq ST(T)$ or $\{t, k\} \subseteq \mathsf{synth}(T)$, from Fact 2.3.1. □

Proposition 2.3.9 *Suppose $T \subseteq \mathcal{T}_0$. Then $ST(\mathsf{synth}(T)) \subseteq \mathsf{synth}(T)$.*

Proof: From Proposition 2.3.7 it follows that $ST(\mathsf{synth}(T)) \subseteq ST(T) \cup \mathsf{synth}(T)$. But since T consists only of atomic terms, $ST(T) = T \subseteq \mathsf{synth}(T)$ and hence the result follows. □

Definition 2.3.10 *A term t is a* minimal term *of a set T of terms iff $t \in T$ and $t \notin \mathsf{synth}(T \backslash \{t\})$, i.e. t cannot be "built" from the other terms in T. $\mathsf{min}(T)$ denotes the set of minimal terms of T.*

The following fact follows immediately from the definition of minimal terms.

Proposition 2.3.11 *Suppose T is a set of terms and $t \in \mathsf{min}(T)$. Then the following conditions hold:*

- *If t is of the form (t', t'') then either $t' \notin T$ or $t'' \notin T$.*
- *If t is of the form $\{t'\}_k$ then either $t' \notin T$ or $k \notin T$.*

Proposition 2.3.12 *Suppose T is a set of terms and $t \in \mathsf{min}(\mathsf{analz}(T))$. Then one of the following conditions hold:*

- *$t \in \mathcal{T}_0$.*
- *$t = \{t'\}_k$ for some t', k such that either $t' \notin \mathsf{analz}(T)$ or $k \notin \mathsf{analz}(T)$.*

Proof: Suppose $t \in \mathsf{min}(\mathsf{analz}(T))$ is of the form (t', t''). Since $t \in \mathsf{analz}(T)$, $\{t', t''\} \subseteq \mathsf{analz}(T)$. But this contradicts item 1 of Proposition 2.3.11. □

Proposition 2.3.13 *For any set of terms T, the following properties hold:*

1. $T \subseteq \mathsf{synth}(\mathsf{min}(T))$.
2. $\mathsf{synth}(T) = \mathsf{synth}(\mathsf{min}(T))$.
3. $\overline{T} = \mathsf{synth}(\mathsf{min}(\mathsf{analz}(T)))$.

Proof: We prove by induction on the structure of terms that for all $t \in T$, t belongs to $\mathsf{synth}(\mathsf{min}(T))$. If $t \in T \cap \mathcal{T}_0$ then clearly $t \in \mathsf{min}(T) \subseteq \mathsf{synth}(\mathsf{min}(T))$. If $t = (t', t'')$ belongs to $\mathsf{min}(T)$ then we are through. Otherwise $\{t', t''\} \subseteq T$ and by induction hypothesis $\{t', t''\} \subseteq \mathsf{synth}(\mathsf{min}(T))$ and therefore $t = (t', t'') \in \mathsf{synth}(\mathsf{min}(T))$ as well. A similar argument works for the case when $t = \{t'\}_k$.

Now it is clear that $\mathsf{min}(T) \subseteq T$ and thus $\mathsf{synth}(\mathsf{min}(T)) \subseteq \mathsf{synth}(T)$. On the other hand, it follows from item 1 above that $\mathsf{synth}(T) \subseteq \mathsf{synth}(\mathsf{synth}(\mathsf{min}(T))) = \mathsf{synth}(\mathsf{min}(T))$. Thus $\mathsf{synth}(T) = \mathsf{synth}(\mathsf{min}(T))$. Substituting $\mathsf{analz}(T)$ in place of T in the above equation, it follows that $\overline{T} = \mathsf{synth}(\mathsf{analz}(T)) = \mathsf{synth}(\mathsf{min}(\mathsf{analz}(T)))$. □

We introduce the following bit of terminology before we get to our next proposition.

Definition 2.3.14 *A set of terms T is said to* unravel *another set of terms T' iff there exists a term t and a key k such that $\{t\}_k \in \mathsf{analz}(T')$ and $\overline{k} \in \mathsf{analz}(T)$. Two sets T and T' are said to be* mutually independent *if neither T nor T' unravels the other.*

Proposition 2.3.15 *Suppose T and T' are two mutually independent sets of terms. Then $\mathsf{analz}(T \cup T') = \mathsf{analz}(T) \cup \mathsf{analz}(T')$.*

Proof: The inclusion from right to left is obvious. We now consider an arbitrary $t \in \mathsf{analz}(T \cup T')$ and show that $t \in \mathsf{analz}(T) \cup \mathsf{analz}(T')$. Suppose π is an analz-proof of $T \cup T' \vdash t$. We prove by structural induction that for every subproof ϖ of π with

root labelled $T \cup T' \vdash r$, $r \in \mathsf{analz}(T) \cup \mathsf{analz}(T')$. Therefore $t \in \mathsf{analz}(T) \cup \mathsf{analz}(T')$ as well.

Suppose ϖ is a subproof of π with root labelled $T \cup T' \vdash r$ such that for all proper subproofs ϖ_1 of ϖ with root labelled $T \cup T' \vdash r_1$, $r_1 \in \mathsf{analz}(T) \cup \mathsf{analz}(T')$. Then we prove that $r \in \mathsf{analz}(T) \cup \mathsf{analz}(T')$ as well. We only consider the case when the rule applied at the root of ϖ is Ax_a or $\mathsf{decrypt}$. The other cases can be handled similarly.

- Suppose ϖ is the following proof:

$$\frac{}{T \cup T' \vdash r}\,\mathsf{Ax}_a$$

  Then $r \in T \cup T' \subseteq \mathsf{analz}(T) \cup \mathsf{analz}(T')$.

- Suppose ϖ is the following proof:

$$\frac{\begin{array}{cc}(\varpi_1) & (\varpi_2)\\ \vdots & \vdots \\ T \cup T' \vdash \{r\}_k & T \cup T' \vdash \overline{k}\end{array}}{T \cup T' \vdash r}\,\mathsf{decrypt}$$

  By induction hypothesis $\{\{r\}_k, \overline{k}\} \subseteq \mathsf{analz}(T) \cup \mathsf{analz}(T')$. Since T and T' are independent, it can neither be the case that $\{r\}_k \in \mathsf{analz}(T)$ and $\overline{k} \in \mathsf{analz}(T')$, nor can it be the case that $\{r\}_k \in \mathsf{analz}(T')$ and $\overline{k} \in \mathsf{analz}(T)$. Hence either $\{\{r\}_k, \overline{k}\} \subseteq \mathsf{analz}(T)$ or $\{\{r\}_k, \overline{k}\} \subseteq \mathsf{analz}(T')$. It follows immediately that $r \in \mathsf{analz}(T) \cup \mathsf{analz}(T')$.

□

Chapter 3

Undecidability results

In this chapter we prove that the secrecy problem for security protocols is in general undecidable. In fact we prove that the secrecy problem is undecidable even when we consider only well-typed runs or when we consider only boundedly many nonces and keys.

It might be surprising at first glance that a simple property like secrecy (which is only slightly more complex than reachability) should turn out to be undecidable. It is all the more surprising since protocol specifications prescribe set patterns of communication for the different agents. Even though factors like unbounded nonces or unbounded message length enter the picture, it seems unlikely at first glance that the protocol specifications can force such unbounded behaviour. If that was possible, it would mean that our "language" for specifying protocols has a considerable amount of inherent programming ability.

We will see in this chapter that one can actually define protocols whose runs can code up an unbounded amount of information. We will see that the style of presenting a protocol as a set of roles hides a lot of programming ability. The crucial point about this style of presentation is that in some situations, the question of whether an instance of a particular action (which occurs in the specification of a protocol) occurs in any run of the protocol can be determined only by run-time considerations (in contrast to well-formed protocols, where we know that for every protocol action, there is always one scenario in which some instance of the action is enabled). This contributes primarily to undecidability.

In fact, in the literature, we have found that the undecidability results are usually proved using a syntax of protocols close to the set-of-roles style of presentation, whereas the linear style of presentation is favoured in work on decidability, or analysis of protocols. Thus the undecidability results provide us with much insight into the modelling of protocols.

The undecidability result for well-typed runs was first proved by [CDL$^+$99] (see also [DLMS99]) in the setting of multi-set rewriting. We use a different reduction from that used in [CDL$^+$99]. Our reduction is much simpler than the ones currently found in the literature. To our knowledge, ours is also the first detailed proof of this result, a fact which can be attributed to the simplicity of our reduction. The undecidability result for unbounded length of messages has been proved in various places, including for instance, [HT96] and [ALV02].

Two-counter machines

Our undecidability results use a reduction from the reachability problem for two-counter machines. We recall the relevant definitions below:

A two-counter machine is a tuple $M = (Q, F, q_0, \delta)$ where:

- Q is a finite set of *states*,
- $F \subseteq Q$ is the set of *final states*,
- $q_0 \in Q$ is the *initial state*,
- $\delta \subseteq Q \times \{0,1\}^2 \times Q \times \{-1,0,1\}^2$ is the *transition relation* with the condition that whenever $(q, i_1, i_2, q', j_1, j_2) \in \delta$ then $j_k = -1$ implies $i_k = 1$, for $k = 1, 2$ (we can decrement a positive counter only).

The other standard notions relating to two-counter machines are defined below:

- A *configuration* of a two-counter machine $M = (Q, F, q_0, \delta)$ is a triple (q, n_1, n_2) with $q \in Q, n_k \in \mathbb{N}$ (the n_k's are counters).
- For a configuration (q, n_1, n_2) of M and a transition $t = (q, i_1, i_2, q', j_1, j_2) \in \delta$, t is *enabled* at (q, n_1, n_2) iff for $k = 1, 2$, $i_k = 0$ iff $n_k = 0$. Whenever t is enabled at (q, n_1, n_2) we have the *reduction* $(q, n_1, n_2) \xrightarrow{t} (q', n_1 + j_1, n_2 + j_2)$.
- A configuration (q, n_1, n_2) is *reachable* if $(q_0, 0, 0) \xrightarrow{*} (q, n_1, n_2)$.

- A configuration (q, n_1, n_2) is *final* if $q \in F$.
- The *reachability problem for two-counter machines* is the problem of determining for a given two-counter machine $M = (Q, F, q_0, \delta)$ whether a final configuration of M is reachable.

We assume the well-known fact that the reachability problem for two-counter machines is undecidable.

3.1 Undecidability for well-typed runs

Let $M = (Q, F, q_0, \delta)$ be an arbitrary two-counter machine. We will define a protocol $\mathsf{Pr}_M = (\mathsf{C}, \mathsf{R})$ such that a final configuration of M is reachable iff there is a well-typed leaky run ξ of Pr_M. As we will see in the proofs which follow, crucial use is made of the fact that there are unboundedly many nonces in N.

Before defining the actual reduction, we set up some basic notation: For simplicity, assume $Q \subseteq N$. Let z and d be fixed nonces from N. We fix honest agents A, B (and therefore the shared key k_{AB}.) Then we define the following terms:

$$\begin{array}{ll} \text{for any } u, u' \in N, \text{ and } q \in Q, & [q, u, u'] \stackrel{\text{def}}{=} \{(q, (u, u'))\}_{k_{AB}}. \\ \text{for any } u, u' \in N, & [u, u'] \stackrel{\text{def}}{=} \{(u, u')\}_{k_{AB}}. \end{array}$$

The protocol Pr_M is defined as follows:

Definition 3.1.1 $\mathsf{Pr}_M \stackrel{\text{def}}{=} (\mathsf{C}, \mathsf{R})$ *where:*

- $\mathsf{C} = Q \cup \{A, B, z, d\}$ *and*
- $\mathsf{R} = \{\eta_0\} \cup \{\eta_t \mid t \in \delta\} \cup \{\eta_f \mid f \in F\}$ *where:*
  - $\eta_0 \stackrel{\text{def}}{=} A!B\text{:}[d, d], [q_0, z, z], [d, d].$
  - *for each transition* $t = (q, i_1, i_2, q', j_1, j_2) \in \delta$, $\eta_t \stackrel{\text{def}}{=} a \cdot a'$ *with:*

$$\begin{array}{l} a = A?B\text{:}[u_1, v_1], [q, w_1, w_2], [u_2, v_2]; \\ a' = A!B\text{:}(M)\ [u_1', v_1'], [q', w_1', w_2'], [u_2', v_2'] \end{array}$$

    where $M = \{v_k' \mid k \in \{1, 2\}$ *and* $j_k = 1\}$, *and the following conditions hold for* $k \in \{1, 2\}$*:*

if $i_k = 0$ *and* $j_k = 0$ *then*
$w_k = w'_k = z$ *and* $u_k = v_k = u'_k = v'_k = d$;
if $i_k = 0$ *and* $j_k = 1$ *then*
$u'_k = w_k = z$, $u_k = v_k = d$, $v'_k = w'_k$, *and*
v'_k *does not belong to* C;
if $i_k = 1$ *and* $j_k = 0$ *then*
$w'_k = w_k = v_k$, $u'_k = v'_k = d$, *and*
u_k *and* v_k *are distinct nonces not belonging to* C;
if $i_k = 1$ *and* $j_k = 1$ *then*
$w_k = v_k = u'_k$, $w'_k = v'_k$, *and*
u_k, v_k *and* v'_k *are distinct nonces not belonging to* C;
if $i_k = 1$ *and* $j_k = -1$ *then*
$w_k = v_k$, $w'_k = u_k$, $u'_k = v'_k = d$, *and*
u_k *and* v_k *are distinct nonces not belonging to* C.

For any η_t *as given above, and* $k \in \{1, 2\}$, *the notation* $\mathsf{inctr}_k(\eta_t)$ *is used to denote* w_k *and the notation* $\mathsf{outctr}_k(\eta_t)$ *is used to denote* w'_k.

– *For each* $f \in F$, $\eta_f \stackrel{\text{def}}{=} a \cdot a' \cdot a''$ *where:*

$a = A?B{:}[f, w_1, w_2]$;
$a' = A!B{:}(\{x\})\ \{x\}_{k_{AB}}$;
$a'' = A!B{:}x$

where x, w_1 *and* w_2 *are distinct nonces not occurring in* C.

The role corresponding to the transition $(q, 0, 1, q', 1, -1)$ is presented by way of example:

$A?B{:}[d, d], [q, z, v_2], [u_2, v_2]$;
$A!B{:}(\{v'_1\})\ [z, v'_1], [q', v'_1, u_2], [d, d]$.

The role corresponding to the transition $(q, 1, 1, q', 1, 1)$ is another example:

$A?B{:}[u_1, v_1], [q, v_1, v_2], [u_2, v_2]$;
$A!B{:}(\{v'_1, v'_2\})\ [v_1, v'_1], [q', v'_1, v'_2], [v_2, v'_2]$.

The role η_0 starts off the simulation of the two-counter machine. The role η_f checks if a final configuration with state f is reached and if so signals it by contriving to "leak" a fresh nonce. The role η_t simulates the transition $t \in \delta$.

Lemma 3.1.2 *Suppose ξ is a run of Pr_M and $s = infstate(\xi)$. Then $k_{AB} \notin \mathsf{analz}(s_I)$ (and hence $k_{AB} \notin \overline{s_I}$ as well).*

Proof: The proof is by induction on $|\xi|$. For $\xi = \varepsilon$, by definition $s_I = (\mathsf{init}(\mathsf{Pr}))_I = K_I \cup \mathsf{C} \cup \{\mathsf{n}_0, \mathsf{m}_0, \mathsf{k}_0\}$ and thus it is clear that $k_{AB} \notin \mathsf{analz}(s_I)$. Suppose $\xi = \xi' \cdot e$ with s' denoting $infstate(\xi')$. By induction hypothesis $k_{AB} \notin \mathsf{analz}(s'_I)$. Further $s_I \subseteq s'_I \cup \{term(e)\}$. But $term(e)$ is a tuple of terms of the form $[q, u, u']$ or $[u, u']$ or $\{x\}_{k_{AB}}$ or x (with $x \in N$). Thus it is clear that s'_I and $\{term(e)\}$ are mutually independent sets of terms (since $k_{AB} \notin \mathsf{analz}(s'_I)$ and $\mathsf{analz}(term(e)) \cap K = \emptyset$). By applying Proposition 2.3.15 and using the fact that $k_{AB} \notin \mathsf{analz}(s'_I) \cup \mathsf{analz}(\{term(e)\})$, we conclude that $k_{AB} \notin \mathsf{analz}(s_I)$. □

Definition 3.1.3

1. *We say that a number n is represented in an information state s by a nonce u if there exist* distinct *nonces $u_0, \ldots, u_n$ such that $u_0 = z$, $u_n = u$, and for all $i < n$, $[u_i, u_{i+1}] \in \overline{s_I}$.*

2. *We say that a configuration (q, n, n') is represented in an information state s by the term $[q, u, u']$ if u represents n in s, u' represents n' in s, and $[q, u, u'] \in \overline{s_I}$.*

3. *We say that a number n is represented in a run ξ of Pr_M by a nonce u if n is represented in $infstate(\xi)$ by u.*

4. *We say that a configuration (q, n, n') is represented in a run ξ of Pr_M by the term $[q, u, u']$ iff (q, n, n') is represented in $infstate(\xi)$ by $[q, u, u']$.*

From the definition it follows that in all states s, z represents only 0 and 0 is represented only by z.

The following lemma states that the role η_t faithfully simulates the transition t.

Lemma 3.1.4 *Suppose ξ is a run of Pr_M with $s = infstate(\xi)$, $t = (q, i_1, i_2, q', j_1, j_2)$ is a transition of M, $\eta_t = a \cdot a'$, and (q, n_1, n_2) is a configuration of M represented in s.*

1. *If t is enabled at (q, n_1, n_2) then there is a* well-typed *substitution σ suitable for Pr_M and η_t such that:*

   - *$\sigma(\mathsf{inctr}_k(\eta_t))$ represents n_k in s (for $k = 1, 2$),*

- $\sigma(a)$ *is enabled in* s *and* $\sigma(a')$ *is enabled at* $update(s, \sigma(a))$, *and*
- $\sigma(\mathsf{outctr}_k(\eta_t))$ *represents* $n_k + j_k$ *in* $update(s, \sigma(\eta_t))$ *(for* $k = 1, 2$*).*

2. *If there is a substitution* σ *suitable for* Pr_M *and* η_t *such that* $\sigma(\mathsf{inctr}_k(\eta_t))$ *represents* n_k *in* s *(for* $k = 1, 2$*) and* $\sigma(a)$ *is enabled in* s*, then* t *is enabled at* (q, n_1, n_2).

Proof: Suppose $t = (q, i_1, i_2, q', j_1, j_2)$ and suppose

$a = A?B{:}[u_1, v_1], [q, w_1, w_2], [u_2, v_2]$;
$a' = A!B{:}(M)\ [u'_1, v'_1], [q', w'_1, w'_2], [u'_2, v'_2]$

1. Suppose t is enabled at (q, n_1, n_2). This means that for $k = 1, 2$, $i_k = 0$ iff $n_k = 0$. Let r_k be a nonce which represents n_k in s. We define a substitution σ suitable for Pr_M and η_t as follows:

   - for $k = 1, 2$, $\sigma(w_k) = r_k$,
   - σ is identity on C,
   - for each distinct $m \in M$, $\sigma(m)$ is a distinct nonce not occurring in $ST(s)$ (Note that here we are crucially using the fact that N is an infinite set.),
   - for $k = 1, 2$, if $i_k = 1$ then $\sigma(u_k) = r'_k$ where r'_k is some nonce representing $n_k - 1$ in s such that $[r'_k, r_k] \in \overline{s_I}$ (since $n_k \neq 0$ and since r_k represents n_k in s, there has to exist at least one such r'_k).

   It is clear that σ is a well-typed substitution suitable for Pr_M and η_t. Let $s' = update(s, \sigma(a))$ and $s'' = update(s', \sigma(a'))$.

   - From the definition it is immediate that $\sigma(\mathsf{inctr}_k(\eta_t))$, which is the same as $\sigma(w_k)$, represents n_k at s, for $k = 1, 2$.
   - We now prove that $\sigma(a)$ is enabled at s and $\sigma(a')$ is enabled at s'. Since $[q, r_1, r_2]$ represents (q, n_1, n_2) in s, $[q, r_1, r_2] \in \overline{s_I}$. For $k = 1, 2$, if $i_k = 0$ then $u_k = v_k = d$ and so $[\sigma(u_k), \sigma(v_k)] = [\sigma(d), \sigma(d)] = [d, d]$. Now from the definition of Pr_M it follows that the first event of any run can only be of the form $(\eta_0, \sigma, 1)$ for some substitution σ. Call this event e. But e is a send event and $[d, d] \in \mathsf{analz}(\{term(e)\})$. Hence it follows that $[\sigma(u_k, \sigma(v_k)] = [d, d] \in \overline{s_I}$. Otherwise, $i_k = 1$ and now $w_k = v_k$

by the definition of Pr_M, and therefore $[\sigma(u_k), \sigma(v_k)] = [\sigma(u_k), \sigma(w_k)] = [r'_k, r_k] \in \overline{s_I}$ (by definition of σ). From this it follows that a is enabled at s. Also by definition of σ, $\sigma(M) \cap ST(s) = \emptyset$. Also it is quite easy to verify that $term(a') \in \overline{term(a) \cup M \cup \{k_{AB}\}}$. But $term(a) \cup M \cup \{k_{AB}\} \subseteq s'_A$ and thus a' is enabled in s'.

- Now we prove that $\sigma(\mathsf{outctr}_k(\eta_t)) = \sigma(w'_k)$ represents $n_k + j_k$ in s'' (for $k = 1, 2$). If $j_k = 0$ then $w_k = w'_k$, for $k = 1, 2$ (by definition of Pr_M). Hence it follows that $\sigma(w'_k)$ represents $n_k + j_k$ in s'. If $j_k = -1$ then by definition of σ, $\sigma(u_k)$ represents $n_k - 1 = n_k + j_k$ in s. By definition of Pr_M, $w'_k = u_k$ and thus it follows that $\sigma(w'_k)$ represents $n_k + j_k$ in s and hence in s'' as well. If $j_k = 1$ then observe that $[\sigma(u'_k), \sigma(v'_k)] \in \overline{s''_I}$, $w'_k = v'_k$, $w_k = u'_k$, and $\sigma(w_k)$ represents n_k in s and hence in s'' as well. Therefore $\sigma(w'_k)$ represents $n_k + j_k = n_k + 1$ in s''.

2. Suppose σ is a substitution suitable for Pr_M and η_t such that for $k = 1, 2$, $\sigma(\mathsf{inctr}_k(\eta_t)) = \sigma(w_k)$ represents n_k at s, and such that $\sigma(a)$ is enabled at s. We need to show that $i_k = 0$ iff $n_k = 0$.

   Suppose $i_k = 0$. Then by definition of Pr_M, $w_k = z$, and hence $\sigma(w_k) = z$. Since z represents only 0 in any state and we are given $\sigma(w_k)$ represents n_k at s, $n_k = 0$.

   Suppose $i_k = 1$. Then by definition of Pr_M, we have that $w_k = v_k$ and $u_k \neq v_k$. Also since $\sigma(a)$ is enabled at s, it follows that $[\sigma(u_k), \sigma(v_k)] \in \overline{s_I}$ and that $[q, \sigma(w_1), \sigma(w_2)] \in \overline{s_I}$. It can be easily seen (from the definition of Pr_M and from Lemma 3.1.2) that for all terms of the form $[q, t, t'] \in ST(s)$, $t \neq d$ and $t' \neq d$. It can also be seen that if $[t, t'] \in ST(s)$ such that $t = t'$ then $t = d$. From these facts and the fact that $\sigma(v_k) = \sigma(w_k)$, we conclude that $\sigma(u_k) \neq \sigma(v_k)$. Again it can be easily checked that for all terms $[t, t'] \in ST(s)$, $t' \neq z$. Thus it follows that $\sigma(v_k) \neq z$ and hence $\sigma(w_k) \neq z$. But we are given that $\sigma(w_k)$ represents n_k in s. Since only z represents 0 in any state, it has to be the case that $n_k \neq 0$.

□

Theorem 3.1.5

1. $(q_0,0,0)\xrightarrow{*}(q,n_1,n_2)$ *iff* (q,n_1,n_2) *is represented in some run of* Pr_M *iff it is represented in some well-typed run of* Pr_M.

2. *A final configuration is reachable in* M *iff there is a leaky run of* Pr_M *iff there is a well-typed leaky run of* Pr_M.

Proof:

1. We first prove that if $(q_0,0,0)\xrightarrow{*}(q,n_1,n_2)$ then there is a *well-typed* run of Pr_M in which (q,n_1,n_2) is represented.

   Let m be the length of the derivation $(q_0,0,0)\xrightarrow{*}(q,n_1,n_2)$. We prove the result by induction on m. The base case is when $m=0$ in which case $q=q_0$ and $n_1=n_2=0$. Then the run $(\eta_0,\sigma,1)$ satisfies the statement of the theorem, for any well-typed substitution σ which is identity on C.

   Suppose $(q_0,0,0)\xrightarrow{*}(q,n_1,n_2)\xrightarrow{t}(q',n_1',n_2')$. It is clear that there is a run ξ of Pr_M in which (q,n_1,n_2) is represented, by the induction hypothesis. Let $s=$ *infstate*(ξ). Let $t=(q,i_1,i_2,q',j_1,j_2)$ and $\eta_t=a\cdot a'$. By lemma 3.1.4, there is a well-typed substitution σ suitable for Pr_M and η_t such that $\sigma(a)$ is enabled at s, $\sigma(a')$ is enabled at $update(s,\sigma(a))$, and $\sigma(\mathsf{outctr}_k(\eta_t))$ represents $n_k+j_k=n_k'$ in $update(s,\sigma(\eta_t))$. Letting $e=(\eta_t,\sigma,1)$ and $e'=(\eta_t,\sigma,2)$ it is easy to see that $\xi\cdot e\cdot e'$ is a well-typed run of Pr_M. Further, since $[q',\sigma(w_1'),\sigma(w_2')]\in\overline{(infstate(\xi\cdot e\cdot e'))_I}$, it is clear that (q',n_1',n_2') is represented in $\xi\cdot e\cdot e'$.

   We now prove that if there is a run of Pr_M in which (q,n_1,n_2) is represented then $(q_0,0,0)\xrightarrow{*}(q,n_1,n_2)$. We prove the result by induction on $|\xi|$, where ξ is a run of Pr_M.

   The base case is when $|\xi|=0$ and then the statement is vacuously true since no configuration is represented in ξ.

   Suppose (q',n_1',n_2') is represented in a run $\xi'=\xi''\cdot e$ of Pr_M. Let s'' and s' denote *infstate*(ξ'') and *infstate*(ξ'), respectively. Let $[q',w_1',w_2']$ represent (q',n_1',n_2') in ξ'. By Lemma 3.1.2 we see that $[q',w_1',w_2']\in\mathsf{analz}(s_I')$. If (q',n_1',n_2') is already represented in ξ'' then by induction hypothesis (q',n_1',n_2') is reachable from $(q_0,0,0)$. Otherwise it follows that $[q,w_1',w_2']\in\mathsf{analz}(s_I')\setminus\mathsf{analz}(s_I'')$. Since a term of the form $[q,w_1',w_2']$ does not occur inside an encryption in any event of the protocol, it follows from the above fact that in fact

$[q', w_1', w_2'] \in \mathsf{analz}(\{term(e')\})$. It is also clear that e' is a send event, so we have to consider only the following two cases:

$e' = (\eta_0, \sigma, 1)$*:* Then it is clear that $(q', n_1', n_2') = (q_0, 0, 0)$ and hence that (q', n_1', n_2') is vacuously reachable from $(q_0, 0, 0)$.

$e' = (\eta_t, \sigma, 2)$ *for some* $t \in \delta$*:* Let $t = (q, i_1, i_2, q', j_1, j_2)$ and let $\eta_t = a \cdot a'$. It is clear that $\sigma(\mathsf{outctr}_k(\eta_t))$ represents n_k' for $k = 1, 2$. Further for $k = 1, 2$, $n_k' = n_k + j_k$ where $\sigma(\mathsf{inctr}_k(\eta_t))$ represents n_k in $infstate(\xi'')$. Since e' is enabled at ξ'', it has to be that $e = (\eta_t, \sigma, 1)$ occurs in ξ''. Further (since $\sigma(\mathsf{outctr}_k(\eta_t))$ represents $n_k + j_k$ in s' for $k = 1, 2$) it is clear that $\sigma(\mathsf{inctr}_k(\eta_t))$ represents n_k in s'' for $k = 1, 2$. In fact, there is a proper prefix ξ_1 of ξ'' such that (q, n_1, n_2) is represented in $infstate(\xi_1)$, and $act(e)$ is enabled at $infstate(\xi_1)$. By induction hypothesis we have that (q, n_1, n_2) is a reachable configuration and by lemma 3.1.4, we know that t is enabled at (q, n_1, n_2). Therefore $(q, n_1, n_2) \xrightarrow{t} (q', n_1 + j_1, n_2 + j_2) = (q', n_1', n_2')$. Thus (q', n_1', n_2') is also a reachable configuration.

2. We first prove that if a final configuration is reachable in M then there is a well-typed leaky run of Pr_M. Suppose a final configuration (f, n_1, n_2) is reachable in M. Then there is a well-typed run ξ of Pr_M representing (f, n_1, n_2). Thus $[f, r_1, r_2] \in \overline{(infstate(\xi))_I}$ for some nonces r_1 and r_2, and hence $e_1 \cdot e_2 \cdot e_3$ is enabled at ξ, where $e_i = (\eta_f, \sigma, i)$ for $i = 1, 2, 3$ and some well-typed σ such that $\sigma(x) \neq \sigma(y)$ for all $y \neq x$. It then follows that $\xi \cdot e_1 \cdot e_2 \cdot e_3$ is also a well-typed run of Pr_M, and by definition of Pr_M this run is patently leaky.

   We now prove that if there is a leaky run of Pr_M then a final configuration is reachable in M. Suppose there is a leaky run ξ of Pr_M. According to the definition of Pr_M, this means that some instance of η_f for $f \in F$ has been played out as part of ξ. But this means that some configuration of the form (f, n_1, n_2) is represented in ξ which implies that a final configuration is reachable in M.

□

The main conclusion of this section is stated below.

Theorem 3.1.6 *The general secrecy problem and the secrecy problem for well-typed runs are undecidable.*

Proof: The statement immediately follows from item 2 of Theorem 3.1.5 and the fact that the reachability problem for two-counter machines is undecidable. □

3.2 Undecidability with bounded nonces

In this section we prove that for any fixed (even finite) $T \subseteq \mathcal{T}_0$, the secrecy problem for T-runs is undecidable. The proof is again a reduction from the reachability problem for two-counter machines. For the purposes of coding up arbitrary two-counter machines, we assume that x, z, u_1 and u_2 are *fixed, distinct nonces* which belong to $T \cap N$.

Let $M = (Q, F, q_0, \delta)$ be a two-counter machine. For simplicity we assume that $Q \subseteq \mathbb{N}$. We will define a protocol $\mathsf{Pr}_M = (\mathsf{C}, \mathsf{R})$ such that a final configuration of M is reachable iff there is a leaky T-run of Pr_M. As we will see in the proofs which follow, crucial use is made of ill-typed substitutions.

Before defining the actual reduction, we set up some basic notation: We fix honest agents A, B and the long-term shared key k_{AB}. Then we define the following terms (coding up natural numbers):

$\underline{0} = z.$
$\underline{i+1} = (\underline{i}, z).$
for any terms t_1, t_2, t_3, $\quad [t_1, t_2, t_3] \stackrel{\text{def}}{=} \{(t_1, (t_2, t_3))\}_{k_{AB}}.$

The protocol Pr_M is defined as follows:

Definition 3.2.1 $\mathsf{Pr}_M \stackrel{\text{def}}{=} (\mathsf{C}, \mathsf{R})$ *where:*

- $\mathsf{C} = \{A, B, z\}$ *and*
- $\mathsf{R} = \{\eta_0\} \cup \{\eta_t \mid t \in \delta\} \cup \{\eta_f \mid f \in F\}$ *where:*
  - $\eta_0 \stackrel{\text{def}}{=} A!B\text{:}[\underline{q_0}, z, z]$,
  - *for each transition* $t = (q, i_1, i_2, q', j_1, j_2) \in \delta$, $\eta_t \stackrel{\text{def}}{=} a \cdot a'$ *with:*

    $A?B\text{:}[\underline{q}, w_1, w_2];$
    $A!B\text{:}[\underline{q'}, w_1', w_2']$

where, for $k \in \{1,2\}$, the following conditions hold:

$$\begin{array}{l}
\textit{if } i_k = 0 \textit{ and } j_k = 0 \textit{ then } w_k = w'_k = z; \\
\textit{if } i_k = 0 \textit{ and } j_k = 1 \textit{ then } w_k = z \textit{ and } w'_k = (z,z); \\
\textit{if } i_k = 1 \textit{ and } j_k = 0 \textit{ then } w_k = w'_k = (u_k,z); \\
\textit{if } i_k = 1 \textit{ and } j_k = 1 \textit{ then } w_k = (u_k,z) \textit{ and } w'_k = ((u_k,z),z); \\
\textit{if } i_k = 1 \textit{ and } j_k = -1 \textit{ then } w_k = (u_k,z) \textit{ and } w'_k = u_k.
\end{array}$$

For any η_t as given above, and $k \in \{1,2\}$, the notation $\mathsf{inctr}_k(\eta_t)$ is used to denote the term w_k and the notation $\mathsf{outctr}_k(\eta_t)$ is used to denote the term w'_k.

– *For each $f \in F$, $\eta_f \stackrel{\text{def}}{=} a \cdot a' \cdot a''$ with:*

$$\begin{array}{l}
a = A?B{:}[\underline{f}, u_1, u_2]; \\
a' = A!B{:}(\{x\})\ \{x\}_{k_{AB}}; \\
a'' = A!B{:}x.
\end{array}$$

The role corresponding to the transition $(q,0,1,q',1,-1)$ is presented by way of example:

$$\begin{array}{l}
A?B{:}[\underline{q}, z, (u_2,z)]; \\
A!B{:}[\underline{q'}, (z,z), u_2].
\end{array}$$

The role corresponding to the transition $(q,1,1,q',1,1)$ is another example:

$$\begin{array}{l}
A?B{:}[\underline{q}, (u_1,z), (u_2,z)]; \\
A!B{:}[\underline{q'}, ((u_1,z),z), ((u_2,z),z)].
\end{array}$$

The role η_0 starts off the simulation of the two-counter machine. The role η_f checks if a final configuration with state f is reached and if so signals it by contriving to "leak"a freshly minted nonce. The role η_t simulates the transition $t \in \delta$.

Lemma 3.2.2 *Suppose ξ is a run of Pr_M and $s = infstate(\xi)$. Then $k_{AB} \notin \mathsf{analz}(s_I)$ (and hence $k_{AB} \notin \overline{s_I}$ as well).*

The proof is on the same lines as the proof of Lemma 3.1.2.

Definition 3.2.3

1. *We say that a configuration (q,n,n') is represented in an information state s if the term $[\underline{q}, \underline{n}, \underline{n'}] \in \overline{s_I}$.*

2. *We say that a configuration* (q, n, n') *is represented in a run* ξ *of* Pr_M *if* (q, n, n') *is represented in* $infstate(\xi)$.

The following lemma states that the role η_t faithfully simulates the transition t.

Lemma 3.2.4 *Suppose* ξ *is a run of* Pr_M, $s = infstate(s_0, \xi)$, $t = (q, i_1, i_2, q', j_1, j_2)$ *is a transition of* M, $\eta_t = a \cdot a'$ *and* (q, n_1, n_2) *is a configuration of* M *represented in* s. *Then* t *is enabled at* (q, n_1, n_2) *iff there is a* T*-substitution* σ *suitable for* Pr_M *and* η_t *such that:*

- $\sigma(\mathsf{inctr}_k(\eta_t))$ *represents* n_k *in* s *(for* $k = 1, 2$*),*
- $\sigma(a)$ *is enabled in* s *and* $\sigma(a')$ *is enabled at* $update(s, \sigma(a))$, *and*
- $\sigma(\mathsf{outctr}_k(\eta_t))$ *represents* $n_k + j_k$ *in* $update(s, \sigma(\eta_t))$ *(for* $k = 1, 2$*).*

Proof: Suppose $t = (q, i_1, i_2, q', j_1, j_2)$ and suppose

$a = A?B{:}[\underline{q}, w_1, w_2]$;
$a' = A!B{:}[\underline{q'}, w_1', w_2']$

Suppose t is enabled at (q, n_1, n_2). This means that for $k = 1, 2$, $i_k = 0$ iff $n_k = 0$. We define a substitution σ as follows:

$$\text{for } k = 1, 2 \quad \sigma(u_k) = \begin{cases} z & \text{if } i_k = 0 \\ \underline{n_k - 1} & \text{if } i_k = 1 \end{cases}$$

Further we let σ be identity on C. It is easily seen that σ is a T-substitution suitable for Pr_M and η_t. (Note that in general σ will be an ill-typed substitution.) Let $s' = update(s, \sigma(a))$ and $s'' = update(s', \sigma(a'))$.

- If $i_k = 0$ then $w_k = z$, and since in this case $n_k = 0$ as well it is immediate that $\sigma(\mathsf{inctr}_k(\eta_t))$ represents n_k in s. If $i_k = 1$ then $w_k = (u_k, z)$, and since $\sigma(u_k) = \underline{n_k - 1}$ it is clear that $\sigma(\mathsf{inctr}_k(\eta_t))$ represents n_k in s.
- We are given that (q, n_1, n_2) is represented in s, i.e., $[\underline{q}, \underline{n_1}, \underline{n_2}] \in \overline{s_I}$. But since $\sigma(w_k) = \underline{n_k}$ for $k = 1, 2$, it is easy to see that $\sigma(a)$ is enabled at s. Since $\sigma(term(a')) \in \overline{\{z, k_{AB}\}}$, it is immediate that $\sigma(a')$ is enabled at $update(s, \sigma(a))$.
- If $j_k = 0$ then $\mathsf{outctr}_k(\eta_t) = \mathsf{inctr}_k(\eta_t)$ and thus $\sigma(\mathsf{outctr}_k(\eta_t))$ represents $n_k = n_k + j_k$ in s''. If $j_k = 1$ then $\mathsf{outctr}_k(\eta_t) = (\mathsf{inctr}_k(\eta_t), z)$ and thus $\sigma(\mathsf{outctr}_k(\eta_t))$ represents $n_k + 1 = n_k + j_k$ in s''. If $j_k = -1$ then $\mathsf{inctr}_k(\eta_t) = (\mathsf{outctr}_k(\eta_t), z)$ and thus $\sigma(\mathsf{outctr}_k(\eta_t))$ represents $n_k - 1 = n_k + j_k$ in s''.

Suppose σ is a substitution suitable for Pr_M and η_t such that for $k = 1, 2$, $\sigma(\mathsf{inctr}_k(\eta_t)) = \sigma(w_k)$ represents n_k at s. We need to show that $i_k = 0$ iff $n_k = 0$.

Suppose $i_k = 0$. Then by definition of Pr_M, $w_k = z$, and hence $\sigma(w_k) = z$. Since z represents only 0 in any state and we are given $\sigma(w_k)$ represents n_k at s, $n_k = 0$.

Suppose $n_k = 0$. Since $\sigma(w_k)$ represents $n_k = 0$ at s and since only z represents 0 in any state, $\sigma(w_k) = z$. But according to definition of Pr_M, either $w_k = z$ or $w_k = (u_k, z)$. So $\sigma(w_k) = z$ only when $w_k = z$, and this happens only when $i_k = 0$. □

Theorem 3.2.5

1. $(q_0, 0, 0) \xrightarrow{*} (q, n_1, n_2)$ *iff there is a* T*-run* ξ *of* Pr_M *in which* (q, n_1, n_2) *is represented.*

2. *A final configuration is reachable in* M *iff there is a leaky* T*-run of* Pr_M*.*

Proof:

1. We first prove that if $(q_0, 0, 0) \xrightarrow{*} (q, n_1, n_2)$ then there is a T-run of Pr_M in which (q, n_1, n_2) is represented.

   Let m be the length of the derivation $(q_0, 0, 0) \xrightarrow{*} (q, n_1, n_2)$. We prove the result by induction on m. The base case is when $m = 0$ in which case $q = q_0$ and $n_1 = n_2 = 0$. Then the run $(\eta_0, \sigma, 1)$ satisfies the statement of the theorem, for any T-substitution σ which is identity on C.

   Suppose $(q_0, 0, 0) \xrightarrow{*} (q, n_1, n_2) \xrightarrow{t} (q', n_1', n_2')$. It is clear that there is a run ξ of Pr_M in which (q, n_1, n_2) is represented, by induction hypothesis. Let $s = \mathit{infstate}(\xi)$. Let $t = (q, i_1, i_2, q', j_1, j_2)$ and $\eta_t = a \cdot a'$. By lemma 3.2.4, there is a T-substitution σ suitable for Pr_M and η_t such that $\sigma(a)$ is enabled at s, $\sigma(a')$ is enabled at $\mathit{update}(s, \sigma(a))$, and $\sigma(\mathsf{outctr}_k(\eta_t))$ represents $n_k + j_k = n_k'$ in $\mathit{update}(s, \sigma(\eta_t))$. Letting $e = (\eta_t, \sigma, 1)$ and $e' = (\eta_t, \sigma, 2)$ it is easy to see that $\xi \cdot e \cdot e'$ is a well-typed run of Pr_M. Further, since $[q', \sigma(w_1'), \sigma(w_2')] \in \overline{(\mathit{infstate}(\xi \cdot e \cdot e'))_I}$, it is clear that (q', n_1', n_2') is represented in $\xi \cdot e \cdot e'$.

   We now prove that if there is a run of Pr_M in which (q, n_1, n_2) is represented then $(q_0, 0, 0) \xrightarrow{*} (q, n_1, n_2)$. We prove the result by induction on $|\xi|$, where ξ is a run of Pr_M.

The base case is when $|\xi| = 0$ and then the statement is vacuously true since no configuration is represented in ξ.

Suppose (q', n_1', n_2') is represented in a run $\xi' = \xi'' \cdot e$ of Pr_M. Let s'' and s' denote $infstate(\xi'')$ and $infstate(\xi')$, respectively. Let $[q', w_1', w_2']$ represent (q', n_1', n_2') in ξ'. By Lemma 3.2.2 we see that $[q', w_1', w_2'] \in \mathsf{analz}(s_I')$. If (q', n_1', n_2') is already represented in ξ'' then by induction hypothesis (q', n_1', n_2') is reachable from $(q_0, 0, 0)$. Otherwise it follows that $[q, w_1', w_2'] \in \mathsf{analz}(s_I') \setminus \mathsf{analz}(s_I'')$. Thus it must be the case that $[q', w_1', w_2'] \in \mathsf{analz}(\{term(e')\})$. It is also clear that e' is a send event, so we have to consider only the following two cases:

- $e' = (\eta_0, \sigma, 1)$*:* Then it is clear that $(q', n_1', n_2') = (q_0, 0, 0)$ and hence that (q', n_1', n_2') is vacuously reachable from $(q_0, 0, 0)$.
- $e' = (\eta_t, \sigma, 2)$ *for some* $t \in \delta$*:* Let $t = (q, i_1, i_2, q', j_1, j_2)$ and let $\eta_t = a \cdot a'$. It is clear that $\sigma(\mathsf{outctr}_k(\eta_t))$ represents n_k' for $k = 1, 2$. Further for $k = 1, 2$, $n_k' = n_k + j_k$ where $\sigma(\mathsf{inctr}_k(\eta_t))$ represents n_k in $infstate(\xi'')$. Since e' is enabled at ξ'', it has to be that $e = (\eta_t, \sigma, 1)$ occurs in ξ''. Further (since $\sigma(\mathsf{outctr}_k(\eta_t))$ represents $n_k + j_k$ in s' for $k = 1, 2$) it is clear that $\sigma(\mathsf{inctr}_k(\eta_t))$ represents n_k in s'' for $k = 1, 2$. In fact, there is a proper prefix ξ_1 of ξ'' such that (q, n_1, n_2) is represented in $infstate(\xi_1)$, and $act(e)$ is enabled at $infstate(\xi_1)$. By induction hypothesis we have that (q, n_1, n_2) is a reachable configuration and by lemma 3.2.4, we know that t is enabled at (q, n_1, n_2). Therefore $(q, n_1, n_2) \xrightarrow{t} (q', n_1 + j_1, n_2 + j_2) = (q', n_1', n_2')$. Thus (q', n_1', n_2') is also a reachable configuration.

2. We first prove that if a final configuration is reachable in M then there is a leaky T-run of Pr_M. Suppose a final configuration (f, n_1, n_2) is reachable in M. Then there is a T-run ξ of Pr_M representing (f, n_1, n_2). Thus $[f, r_1, r_2] \in \overline{(infstate(\xi))_I}$ for some nonces r_1 and r_2, and hence $e_1 \cdot e_2 \cdot e_3$ is enabled at ξ, where $e_i = (\eta_f, \sigma, i)$ for $i = 1, 2, 3$ and some T-substitution σ such that $\sigma(x) \notin \mathsf{C}$. It then follows that $\xi \cdot e_1 \cdot e_2 \cdot e_3$ is also a T-run of Pr_M, and by definition of Pr_M this run is patently leaky.

   We now prove that if there is a leaky run of Pr_M then a final configuration is reachable in M. Suppose there is a leaky run ξ of Pr_M. According to the definition of Pr_M, this means that some instance of η_f for $f \in F$ has

been played out as part of ξ. But this means that some configuration of the form (f, n_1, n_2) is represented in ξ which implies that a final configuration is reachable in M.

□

The main conclusion of this section is stated below.

Theorem 3.2.6

The secrecy problem for T-runs is undecidable.

Proof: This immediately follows from item 2 of Theorem 3.2.5 and the fact that the reachability problem for two-counter machines is undecidable. □

3.3 Discussion

The idea of using two-counter machines in the undecidability results is from [ALV02], where the undecidability result for unbounded message length is proved using them. The reduction used in our proof is slightly different — we code up numbers using repeated tupling, whereas in [ALV02], they are coded up using repeated encryption.

The use of two-counter machines in the other undecidability result is a new idea. Existing proofs of this result use reductions from Turing machines or some problems in logic, and the reductions in those proofs are considerably harder than ours.

An interesting point about the proofs in this chapter is that the protocols which were used to code up two-counter machines do not use our definition of secrecy in an essential manner. Reachability is all that really matters. Let us formally define the reachability problem for security protocols:

Definition 3.3.1 (The reachability problem) *Given a protocol* $\mathsf{Pr} = (\mathsf{C}, \mathsf{R})$ *and an action* a*, we say that* a *is* reachable *in* Pr *iff there is a role* η *of* Pr*, a substitution* σ *suitable for* Pr *and* η*, a number* $lp \leq |\eta|$*, and a run* ξ *of* Pr *such that* $\eta(lp) = a$ *and* $(\eta, \sigma, lp) \in Events(\xi)$.

The reachability problem *is to determine whether a is reachable in* Pr*, given a protocol* Pr *and an action a.*

The reachability problem for well-typed runs (T-runs for a fixed T) *is defined similarly by restricting the set of runs under consideration.*

The reachability problem for well-typed runs (as well as that for all runs, and all T-runs for fixed T) is undecidable. The same reduction used earlier suffices to prove the undecidability of this problem as well. *We only have to appeal to the fact that the following problem is undecidable: Given a two-counter machine M and a state q of M, determine whether a configuration with state q is reachable in M.*

In fact, for any logic which is powerful enough to express the reachability property, its verification problem is undecidable in the same settings considered in this chapter.

Chapter 4

Decidability with unboundedly many nonces

In this chapter, we deal with the problem of unbounded nonces. We prove that the tagging scheme introduced in Definition 2.2.31 ensures the decidability of the secrecy problem for well-typed runs, even in the presence of unboundedly many nonces.

4.1 The bounded length case

We first prove the decidability of a restricted secrecy problem — that of checking for a given protocol Pr and a number r whether there is some *well-typed* leaky run of Pr *of length bounded by* r. The trouble is that the set of such runs is still infinite. We show that we can always suitably rename nonces and keys occurring in runs with nonces and keys from a fixed finite set. Since there can only be finitely many well-typed runs which can be thus formed, we get the desired decidability result.

Fix a tagged protocol $\mathsf{Pr} = (\mathsf{C}, \delta)$ for the rest of the section. For any number r, $\mathcal{R}_r(\mathsf{Pr}) \stackrel{\text{def}}{=} \{\xi \text{ is a well-typed run of } \mathsf{Pr} \mid |\xi| \leq r\}$. For any $T \subseteq \mathcal{T}_0$ and any number r, we define $\mathcal{R}_r^T(\mathsf{Pr})$ to be $\{\xi \mid \xi \text{ is a well-typed } T\text{-run of } \mathsf{Pr} \text{ of length at most } r\}$.

Suppose we fix a *finite* $T \subseteq \mathcal{T}_0$ and a number r. It is clear that there are at most $b_1 = (|T|)^{|EST(\delta) \cap \mathcal{T}_0|}$ T-substitutions suitable for Pr. $|EST(\delta) \cap \mathcal{T}_0|$ is an upper bound on the number of basic terms which occur in a role and hence are in the

domain of some T-substitution suitable for Pr. It now follows that there are at most $b_2 = 2 \cdot \ell \cdot b_1$ T-events, where ℓ is the length of δ. This bound easily follows from the fact that the set of distinct (η, i) pairs where η is a role of Pr and $1 \leq |i| \leq |\eta|$ is $2 \cdot \ell$. This coupled with the number of T-substitutions gives us b_2. From this it easily follows that there are at most $(b_2 + 1)^r$ runs in $\mathcal{R}_r^T(\mathsf{Pr})$. Thus we see that $\mathcal{R}_r^T(\mathsf{Pr})$ is a finite, effectively constructible set, and therefore the problem of checking whether there is a leaky run in $\mathcal{R}_r^T(\mathsf{Pr})$ is decidable.

Below we explain how to define a finite set $T(r)$ for any given number r such that $\mathcal{R}_r(\mathsf{Pr})$ has a leaky run iff $\mathcal{R}_r^{T(r)}(\mathsf{Pr})$ has a leaky run. Suppose w is the maximum size of any term occurring in the specification of Pr, and suppose p is the maximum length of any role of Pr. Given r, fix four sets $NT(r) \subseteq N \backslash \mathsf{C}$, $SN(r) \subseteq SN \backslash \mathsf{C}$, $K_0(r) \subseteq K_0 \backslash \mathsf{C}$ and $Ag(r) \subseteq Ag \setminus \mathsf{C}$ such that $|N(r)| = |SN(r)| = |K_0(r)| = |Ag(r)| = r \cdot p \cdot (w+2)$. (The reason for choosing this specific number will become clear as we develop the proof of the following lemma.) $T(r)$ is defined to be $N(r) \cup SN(r) \cup K_0(r) \cup Ag(r) \cup \mathsf{CT}(\mathsf{Pr})$.

Lemma 4.1.1 *For any $r \in \mathbb{N}$, if $\mathcal{R}_r(\mathsf{Pr})$ has a leaky run then so does $\mathcal{R}_r^{T(r)}(\mathsf{Pr})$.*

Proof: We first set up some notation which we use locally in this proof: for any action a of the form $A!B{:}(M)t$ or $A?B{:}t$, *parties*(a) (the set of *apparent (not actual) participants* in the action a), is defined to be $\{A, B\}$. For any sequence of actions $\eta = a_1 \cdots a_\ell$, *parties*$(\eta) = \bigcup_{1 \leq i \leq \ell}$ *parties*(a_i). Let us define the *domain of* η for any $\eta \in \mathsf{Pr}$ to be $(ST(\eta) \cup \mathit{parties}(\eta)) \cap \mathcal{T}_0$. Note that for all $\eta \in \mathsf{R}$, the domain of η contains at most $p \cdot (w+2)$ terms. It clearly suffices to consider events of Pr of the form (η, σ, lp) where the domain of σ is restricted to the domain of η. Let us call such events as *domain-restricted events*. A run composed only of bounded-domain events is called a *domain-restricted run*.

Let us define the *range of a run* ξ to be the union of the ranges of all substitutions σ such that $(\eta, \sigma, lp) \in \mathit{Events}(\xi)$ for some η and lp. (Note that by range of a substitution σ, we mean the set $\{\sigma(x) \mid x \in \mathcal{T}_0 \text{ and } \sigma(x) \text{ is defined}\}$.) If we consider a domain-restricted well-typed run ξ of length at most r, then it is clear that the range of ξ has at most $r \cdot p \cdot (w+2)$ terms. Now $T(r)$ contains $r \cdot p \cdot (w+2)$ nonces and the same number of sequence numbers, keys and agent names. Therefore there exists at least one *injective, well-typed substitution* from the range of ξ to $T(r)$.

Fix one such substitution τ_ξ for each such bounded-domain run $\xi \in \mathcal{R}_r(\mathsf{Pr})$.

(It is the *renaming map associated with* ξ.) For any such run $\xi = e_1 \cdots e_k$ with $e_i = (\eta_i, \sigma_i, lp_i)$ for each $i \leq k$, define $\tau_\xi(\xi)$ to be the run $\tau_\xi(e_1) \cdots \tau_\xi(e_k)$ where $\tau_\xi(e_i) = (\eta_i, \tau_\xi \circ \sigma_i, lp_i)$ for each $i \leq k$ (for each $x \in \mathcal{T}_0$, $(\tau_\xi \circ \sigma_i)(x)$ is defined to be $\tau_\xi(\sigma_i(x))$).

Now for every bounded-domain run $\xi \in \mathcal{R}_r(\mathsf{Pr})$, it is a simple matter to check that for any prefix ξ' of ξ, $A \in Ag$ and $t \in \mathcal{T}$, we have $t \in (infstate(\xi'))_A$ iff $\tau_\xi(t) \in (infstate(\tau_\xi(\xi')))_A$. Also t is leaked in ξ iff $\tau_\xi(t)$ is leaked in $\tau_\xi(\xi)$. From this it easily follows that $\tau_\xi(\xi)$ is in fact a run of Pr (and so belongs to $\mathcal{R}_r^{T(r)}(\mathsf{Pr})$) and that it is leaky if and only if ξ is leaky.

Thus we have shown that if there is a leaky run in $\mathcal{R}_r(\mathsf{Pr})$, then there is also a leaky run in $\mathcal{R}_r^{T(r)}(\mathsf{Pr})$. □

From the above discussion we conclude the following:

Theorem 4.1.2 *The problem of checking for a given protocol* Pr *and a given bound* r *whether there is a well-typed leaky run of* Pr *of length bounded by* r*, is decidable.*

Note that we can also take $p = \ell$ in the above proof. So if we fix Pr with its parameters ℓ and w, and if we fix an r, then the size of $|T(r)|$ is $4 \cdot r \cdot \ell \cdot (w+2) + |\mathsf{CT}(\mathsf{Pr})|$. If we now let $b_1 = (|T(r)|)^{|EST(\delta) \cap \mathcal{T}_0|}$ and $b_2 = 2 \cdot \ell \cdot b_1$, then it suffices to search at most $(b_2 + 1)^r$ runs to see if there is a leak. Letting c_{Pr} be the maximum of $|EST(\delta) \cap \mathcal{T}_0|$, w and $|\mathsf{CT}(\mathsf{Pr})|$, we see that it suffices to search $O((\ell \cdot r \cdot c_{\mathsf{Pr}})^{r \cdot c_{\mathsf{Pr}}})$ runs for a leak.

4.2 Decidability for good runs

In this section, we define the notion of a *good run* and prove some basic properties of good runs. We also prove that the problem of checking whether there is a good leaky run of a given *tagged protocol* is decidable.

Definition 4.2.1 *Suppose* $\mathsf{Pr} = (\mathsf{C}, \delta)$ *is a tagged protocol and* $\xi = e_1 \cdots e_k$ *is a well-typed run of* Pr*. For* $i, j \leq k$*,* e_j *is called a* good successor *of* e_i *(and* e_i *a* good predecessor *of* e_j*) iff* $i < j$ *and at least one of the following conditions holds:*

- $e_i \rightarrow_\ell e_j$.
- e_i *is a send event,* e_j *is a receive event, and* $EST(e_i) \cap EST(e_j) \neq \emptyset$.

For $i \leq k$, e_i is called a good event *in ξ iff either $i = k$ or there is some $j > i$ such that e_j is a good successor of e_i. e_i is called a bad event iff it is not a good event. A run ξ is called a* good run *iff all its events are good. A subsequence $e_1 \cdots e_r$ of ξ is called a* good path *iff for all $j < r$, e_{j+1} is a good successor of e_j.*

Note that a good successor of a send event need not necessarily be a "matching" receive event. Also note that there might be multiple occurrences of the same event in a good run. This might look a bit strange at first glance. But the right way to view this definition is that a bad event definitely signifies something "bad" in terms of the intruder behaviour. In particular, it means that the intruder is playing an active role (generating a new message, or tampering with some earlier message) with regard to that particular event, and is not simply relaying it from someone else to the receiver. Such bad behaviour on the part of the intruder also makes it hard to compute bounds on the length of runs. While good runs do not necessarily eliminate all such "bad" behaviour, enough bad behaviour is eliminated so as to ease the task of computing bounds on the length of good runs, as we will see in the rest of the section.

Note that all good runs are well-typed by diktat. In a later section we will prove that if a tagged protocol has a well-typed leaky run then it has a good leaky run. The following propositions list some useful properties of good runs.

Proposition 4.2.2 *Suppose $\mathsf{Pr} = (\mathsf{C}, c_1 \cdots c_\ell)$ is a tagged protocol and ξ is a well-typed run of Pr. Then all good paths in ξ are of length at most $2 \cdot \ell$.*

Proof: For convenience, define the following notation: for all $i : 1 \leq i \leq \ell$, $a_{2 \cdot i-1} \stackrel{\text{def}}{=} act_s(c_i)$ and $a_{2 \cdot i} \stackrel{\text{def}}{=} act_r(c_i)$. Note that $actseq(c_1 \cdots c_\ell) = a_1 \cdots a_{2 \cdot \ell}$. Suppose $e_1 \cdots e_r$ is a good path in ξ with $e_i = (\zeta_i, \sigma_i, lp_i)$ for all $i \leq r$. Since for all $j \leq r$, e_j is an event of Pr, it is clear that there exists some $i_j \leq 2 \cdot \ell$ such that $\zeta_j(lp_j) = a_{i_j}$.

We now show that for all $j < r$, $i_j < i_{j+1}$, using the fact that e_{j+1} is a good successor of e_j. There are two cases to consider:

$e_j \rightarrow_\ell e_{j+1}$: In this case it is clear that $\zeta_j = \zeta_{j+1}$, $\sigma_j = \sigma_{j+1}$ and $lp_{j+1} = lp_j + 1$. Now ζ_j is a role of Pr and hence a subsequence of $a_1 \cdots a_{2 \cdot \ell}$. Thus a_{i_j} occurs earlier in $a_1 \cdots a_{2 \cdot \ell}$ than $a_{i_{j+1}}$ and hence $i_j < i_{j+1}$.

$act(e_j) \in Send$, $act(e_{j+1}) \in Rec$ and $EST(e_j) \cap EST(e_{j+1}) \neq \emptyset$: It is clear now that

a_{i_j} is a send action and $a_{i_{j+1}}$ is a receive action, and also that $a_{i_{j+1}-1}$ is a send action with $term(a_{i_{j+1}-1}) = term(a_{i_{j+1}})$. Thus it follows that there exist $t \in EST(a_{i_j})$ and $t' \in EST(a_{i_{j+1}-1})$ such that $\sigma_j(t) = \sigma_{j+1}(t')$. But from item 1 of Proposition 2.2.32, it follows that $t = t'$ and $\lfloor i_j \rfloor = \lfloor i_{j+1} - 1 \rfloor$. Since both a_{i_j} and $a_{i_{j+1}-1}$ are send actions, both the indices are odd. Hence it follows that $i_j = i_{j+1} - 1$. This shows that $i_j < i_{j+1}$.

From this it follows that there is a sequence $i_1 < \cdots < i_r \leq 2 \cdot \ell$ such that for all $j \leq r$, $\zeta_j(lp_j) = a_{i_j}$. This suffices to prove that $r \leq 2 \cdot \ell$. □

Lemma 4.2.3 *Suppose* $\mathsf{Pr} = (\mathsf{C}, c_1 \cdots c_\ell)$ *is a tagged protocol and* ξ *is a good run of* Pr. *Then* $|\xi| \leq 2^{2 \cdot \ell + 1} - 1$.

Proof: Suppose $\xi = e_1 \cdots e_k$. Since ξ is a good run of Pr, all the events e_i $(i \leq k)$ are good. This means that for all $i < k$, there is some $j : i < j \leq k$ such that e_j is a good successor of e_i. It easily follows that for all $i < k$, there is a good path frm e_i to e_k. For all $i : 0 \leq i \leq 2 \cdot \ell$, define the set E_i to be the set of events e occurring in ξ such that the shortest good path from e to e_k is of length i. From Proposition 4.2.2 we know that all good paths of ξ are of length at most $2 \cdot \ell$. Thus the set of events occurring in ξ is partitioned by the sets $E_0, \ldots, E_{2 \cdot \ell}$. Now since every good run is also a well-typed run by definition, we can apply item 2 of Proposition 2.2.32 and conclude that for every receive event e occurring in ξ there is at most one send event e' in ξ such that $EST(e) \cap EST(e') \neq \emptyset$. Further for every event e there is at most one e' such that $e' \rightarrow_\ell e$. Thus every event occurring in ξ has at most two good predecessors, and thus for all $i < 2 \cdot \ell$, $|E_{i+1}| \leq 2 \cdot |E_i|$. Thus it is easy to see by induction that for all $i \leq 2 \cdot \ell$, $|E_i| \leq 2^i$, and that $|\xi| \leq 2^{2 \cdot \ell + 1} - 1$. □

Lemma 4.2.3 and Theorem 4.1.2 immediately imply the following theorem.

Theorem 4.2.4 *The problem of checking for a given tagged protocol* Pr *whether there is a* good leaky run *of* Pr *is decidable.*

4.3 Reduction to good runs

In this section we prove that if a tagged protocol has a *well-typed leaky run* then it has a *good leaky run.* As proved in the previous section, checking whether a tagged

protocol has a good leaky run is decidable, and hence the reduction presented in this section yields the decidability of checking whether a tagged protocol has a well-typed leaky run. In the next chapter we prove that if a tagged protocol has a leaky run then it has a well-typed leaky run, thus proving the decidability of the secrecy problem for tagged protocols.

Suppose ξ is a well-typed bad run of a tagged protocol Pr and e is a bad event. The key to eliminating this event is to prove that, under certain conditions, the messages of ξ which come after e can be constructed by the intruder using just the basic terms learned from e instead of $term(e)$. Therefore we first look at how terms can be eliminated appropriately.

4.3.1 How to eliminate terms

Suppose T is a set of terms and u is a term such that $u \in \overline{T}$. Can we remove a term t (with the property that $EST(t) \cap EST(u) = \emptyset$) from T but add a set of *atomic terms* T' such that it is still the case that $u \in \overline{(T \setminus \{t\}) \cup T'}$? The following lemmas show that under some additional assumptions this is possible. They will be crucially used later in the reduction to good runs. We split the task mentioned above into two parts, first handling the case when $u \in \mathsf{analz}(T)$ and then considering what happens when $u \in \overline{T}$. The additional assumptions in the following lemmas are not strong enough to prove that if $u \in \mathsf{analz}(T)$ then $u \in \mathsf{analz}((T \setminus \{t\}) \cup T')$, but we can still prove that either $u \in \mathsf{analz}((T \setminus \{t\}) \cup T')$ or $u \in ST(t)$. Fortunately this suffices to prove that whenever $u \in \overline{T}$, $u \in \overline{(T \setminus \{t\}) \cup T'}$.

Lemma 4.3.1 *Suppose* $T = (\mathsf{analz}(S_1 \cup \{t\}) \setminus \mathsf{analz}(S_1)) \cap \mathcal{T}_0$ *for some* $S_1, S_2 \subseteq \mathcal{T}$ *and* $t \in \mathcal{T}$.

1. *Let* u *be a term and let* π *be an* analz*-proof of* $S_1 \cup S_2 \cup \{t\} \vdash u$ *such that for all* $k \in ST(S_1 \cup \{t\}) \cap K$ *for which* $\overline{k}$ *labels a non-root node of* π, $\overline{k} \in \mathsf{analz}(S_1 \cup \{t\})$. *Then* $u \in (\mathsf{analz}(S_1 \cup \{t\}) \cap ST(t)) \cup \mathsf{analz}(S_1 \cup S_2 \cup T)$.

2. *Let* u *be a term such that* $u \in \mathsf{synth}((\mathsf{analz}(S_1 \cup \{t\}) \cap ST(t)) \cup \mathsf{analz}(S_1 \cup S_2 \cup T))$ *and* $EST(u) \cap EST(t) = \emptyset$. *Then* $u \in \overline{S_1 \cup S_2 \cup T}$.

Proof:

1. Suppose π is an analz-proof of $S_1 \cup S_2 \cup \{t\} \vdash u$. We prove by structural induction that for every subproof ϖ of π with root labelled $S_1 \cup S_2 \cup \{t\} \vdash w$,

we have $w \in (\mathsf{analz}(S_1 \cup \{t\}) \cap ST(t)) \cup \mathsf{analz}(S_1 \cup S_2 \cup T)$. Suppose ϖ is a subproof of π with root labelled $S_1 \cup S_2 \cup \{t\} \vdash w$ such that for all proper subproofs ϖ_1 of ϖ the statement of the lemma holds. Then we prove that it holds for ϖ as well. We only consider the cases when the rule applied at the root of ϖ is Ax_a or $\mathsf{decrypt}$. The other cases can be handled by a routine application of the induction hypothesis.

- Suppose ϖ is the following proof:

$$\frac{}{S_1 \cup S_2 \cup \{t\} \vdash w}\,\mathsf{Ax}_a$$

  Then $w \in S_1 \cup S_2 \cup \{t\}$. If $w = t$ then $w \in \mathsf{analz}(S_1 \cup \{t\}) \cap ST(t)$. If $w \in S_1 \cup S_2$ then $w \in \mathsf{analz}(S_1 \cup S_2 \cup T)$.

- Suppose ϖ is the following proof:

$$\frac{\begin{array}{cc}(\varpi_1) & (\varpi_2)\\ \vdots & \vdots \\ S_1 \cup S_2 \cup \{t\} \vdash \{w\}_k & S_1 \cup S_2 \cup \{t\} \vdash \overline{k}\end{array}}{S_1 \cup S_2 \cup \{t\} \vdash w}\,\mathsf{decrypt}$$

  By induction hypothesis $\{w\}_k \in \mathsf{analz}(S_1 \cup \{t\}) \cup \mathsf{analz}(S_1 \cup S_2 \cup T)$ and $\overline{k} \in \mathsf{analz}(S_1 \cup \{t\}) \cup \mathsf{analz}(S_1 \cup S_2 \cup T)$.

  $\{w\}_k \in \mathsf{analz}(S_1 \cup S_2 \cup T)$*:* If $\overline{k} \in \mathsf{analz}(S_1 \cup S_2 \cup T)$ then w is in the same set as well and we are done. If on the other hand $\overline{k} \in \mathsf{analz}(S_1 \cup \{t\})$, then $\overline{k} \in K \cap (\mathsf{analz}(S_1) \cup (\mathsf{analz}(S_1 \cup \{t\}) \setminus \mathsf{analz}(S_1)))$. But this implies that $\overline{k} \in \mathsf{analz}(S_1 \cup T) \subseteq \mathsf{analz}(S_1 \cup S_2 \cup T)$ and hence w is also in the same set.

  $\{w\}_k \in \mathsf{analz}(S_1 \cup \{t\}) \cap ST(t)$*:* It is evident that $k \in ST(S_1 \cup \{t\})$. Thus by assumption $\overline{k} \in \mathsf{analz}(S_1 \cup \{t\})$ and hence w is also in the same set. Clearly $w \in ST(t)$ as well.

2. Let us denote by W the set $((\mathsf{analz}(S_1 \cup \{t\}) \cap ST(t)) \cup \mathsf{analz}(S_1 \cup S_2 \cup T)) \cap ST(u)$. It is clear that $u \in \mathsf{synth}(W)$. Now $w \in ST(u)$ for every $w \in W$, and since $EST(u) \cap EST(t) = \emptyset$ it is also the case that $EST(w) \cap EST(t) = \emptyset$. We prove below that $W \subseteq \overline{S_1 \cup S_2 \cup T}$; this suffices to prove that $u \in \overline{S_1 \cup S_2 \cup T}$.

   So suppose $w \in W$. Then $w \in \mathsf{analz}(S_1 \cup S_2 \cup T) \cup (\mathsf{analz}(S_1 \cup \{t\}) \cap ST(t))$. If $w \in \mathsf{analz}(S_1 \cup S_2 \cup T)$ we are done. Suppose $w \in \mathsf{analz}(S_1 \cup \{t\}) \cap ST(t)$. In this

case, as observed above $EST(w) \cap EST(t) = \emptyset$, and hence from $w \in ST(t)$ it follows that $EST(w) = \emptyset$. This means that w is just a tuple of atomic terms. In this case it is clear that $w \in \mathsf{synth}(\mathsf{analz}(\{w\}) \cap \mathcal{T}_0)$. But then $\mathsf{analz}(\{w\}) \cap \mathcal{T}_0 \subseteq \mathsf{analz}(S_1 \cup \{t\}) \cap \mathcal{T}_0 \subseteq \mathsf{analz}(S_1 \cup T)$. This implies that $w \in \overline{S_1 \cup S_2 \cup T}$ and the proof is done.

□

The following lemma is vital in proving that if m is secret at a run ξ of a protocol Pr, then m is also secret at ξ', where ξ' is got by eliminating some events and renaming some atomic terms of ξ.

Lemma 4.3.2 *Suppose S is a set of terms and $T \subseteq \mathsf{analz}(S) \cap \mathcal{T}_0$. Suppose τ is a well-typed substitution with the property that for all $x \in \mathcal{T}_0 \setminus T$, $\tau(x) = x$ and for all $x \in T$, $\tau(x) \in S$. Then for all $t \in \mathsf{analz}(\tau(S))$, there exists $r \in \mathsf{analz}(S)$ such that $\tau(r) = t$.*

Proof: Suppose π is an analz-proof of $\tau(S) \vdash t$. We prove by structural induction that for every subproof ϖ of π with root labelled $\tau(S) \vdash w$, there exists $r \in \mathsf{analz}(S)$ such that $\tau(r) = w$. Suppose ϖ is a subproof of π with root labelled $\tau(S) \vdash w$ such that for all proper subproofs ϖ_1 of ϖ the statement of the lemma holds. Then we prove that it holds for ϖ as well. We only consider the cases when the rule applied at the root of ϖ is Ax_a or $\mathsf{decrypt}$. The other cases can be handled by a routine application of the induction hypothesis.

- Suppose ϖ is the following proof:

$$\frac{}{\tau(S) \vdash w}\ \mathsf{Ax}_a$$

  Then $w \in \tau(S)$ which means that there exists $r \in S \subseteq \mathsf{analz}(S)$ such that $\tau(r) = w$.

- Suppose ϖ is the following proof:

$$\frac{\begin{array}{cc}(\varpi_1) & (\varpi_2) \\ \vdots & \vdots \\ \tau(S) \vdash \{w\}_k & \tau(S) \vdash \overline{k}\end{array}}{\tau(S) \vdash w}\ \mathsf{decrypt}$$

By induction hypothesis there exist $r', r'' \in \mathsf{analz}(S)$ such that $\tau(r') = \{w\}_k$ and $r'' = \overline{k}$. Since τ is well-typed, r' is of the form $\{r\}_{k'}$ with $\tau(r) = w$ and $\tau(k') = k$, and r'' is of the form k''. We need to prove that $r \in \mathsf{analz}(S)$.

- Suppose $k' \in T$. It then follows that $k' \in K_0$ and hence it follows that $k' = \overline{k'}$ and that $\overline{k'} \in \mathsf{analz}(S)$ (since $T \subseteq \mathsf{analz}(S)$). Coupled with the fact that $\{r\}_{k'} \in \mathsf{analz}(S)$, we have that $r \in \mathsf{analz}(S)$.
- Suppose $k' \notin T$. From the definition of τ we see that $k' = k$. Thus $\{r\}_k \in \mathsf{analz}(S)$.

  If $k'' \in T$, then since $\tau(T) \subseteq S \subseteq \mathsf{analz}(S)$ it follows that $\overline{k} \in \mathsf{analz}(S)$.
  If $k'' \notin T$, from the definition of τ it follows that $k'' = \overline{k}$, and thus it is again clear that $\overline{k} \in \mathsf{analz}(S)$.

  Coupled with $\{r\}_k \in \mathsf{analz}(S)$, this implies that $r \in \mathsf{analz}(S)$, as desired.

□

4.3.2 Reduction to good runs

In this subsection we proceed to prove the reduction to good runs using the properties proved in the previous subsection.

Lemma 4.3.3 *Suppose* $\mathsf{Pr} = (\mathsf{C}, c_1 \cdots c_\ell)$ *is a tagged protocol which has a* well-typed leaky run. *Then it also has a* good leaky run.

Proof: We fix the following notation for the rest of the proof. Fix a *well-typed leaky run* $\xi = e_1 \cdots e_k$ of Pr, none of whose *proper* prefixes is leaky. Let $e_j = (\eta_j, \sigma_j, lp_j)$ for $j \leq k$. For any $j \leq k$, $t_j = term(e_j)$. For any $j : 1 \leq j \leq k$, ξ_j denotes $e_1 \cdots e_j$, s_j denotes $infstate(\xi_j)$ and T_j denotes $(s_j)_I$. For $i, j : 1 \leq i \leq j \leq k$, ξ_j^{-i} denotes $e_1 \cdots e_{i-1} e_{i+1} \cdots e_j$ if $i < j$ and ξ_{i-1} if $i = j$, s_j^{-i} denotes $infstate(\xi_j^{-i})$ and T_j^{-i} denotes $(s_j^{-i})_I$. We also denote $\mathsf{init}(\mathsf{Pr})$ by s_0 and $(s_0)_I$ by T_0.

Suppose ξ is not a good run. This means that there is a bad event in ξ. Let $r = max(\{i \leq k \mid e_i \text{ is a bad event of } \xi\})$; that is, r is the index of the latest bad event in ξ. Notice that by definition e_k is a good event, and hence $r < k$. Define T to be $(\mathsf{analz}(T_r) \setminus \mathsf{analz}(T_{r-1})) \cap \mathcal{T}_0$. Since ξ_r is not leaky, it follows that no $m \in T$ is secret at ξ_{r-1}. Thus it has to be the case that $T \subseteq NT(e_r) \subseteq N \cup SN \cup K_0$.

Let τ be a substitution which maps every $n \in T \cap N$ to n_0, every $m \in T \cap SN$ to m_0 and every $k \in T \cap K_0$ to k_0 and is identity on all the other terms. (Recall that n_0, m_0, and k_0 are fixed constants in the intruder's initial state.) For all $j \leq k$, we define e'_j to be $(\eta_j, \tau \circ \sigma_j, lp_j)$, where $(\tau \circ \sigma_j)(t) = \tau(\sigma_j(t))$ for all t. We define $\xi' = e'_1 \cdots e'_k$. Analogous to the notations based on ξ, we define the notations t'_j, ξ'_j, s'_j, T'_j, $(\xi')^{-i}_j$, $(s')^{-i}_j$ and $(T')^{-i}_j$ based on ξ'.

We now show that $(\xi')^{-r}_k$) is a (well-typed) run of Pr and that it is leaky; but the index of the latest bad event (if any) in $(\xi')^{-r}_k$ is less than r, and hence we can repeat the process on the new run, eventually obtaining a good run.

We now prove that $(\xi')^{-r}_k$ is a run of Pr and that it is leaky, thus concluding the proof of the theorem.

Claim: $(\xi')^{-r}_k$ is a run of Pr:

Proof of Claim: Since ξ is a run, it follows that $NT(e_i) \cap ST(\mathsf{init}(\mathsf{Pr})) = \emptyset$ for all $i \leq k$, and that $NT(e_i) \cap NT(e_j) = \emptyset$ for all $i < j \leq k$. Since $T \subseteq NT(e_r)$ it follows that $T \cap NT(e_q) = \emptyset$ for all $q \neq r$. It thus follows that $NT(e'_q) = NT(e_q)$ for all $q \neq r$. It is now easy to see that for all $i \leq k, i \neq r$, $NT(e'_i) \cap ST(\mathsf{init}(\mathsf{Pr})) = \emptyset$ and that for all $i < j \leq k, i, j \neq r$, $NT(e'_i) \cap NT(e'_j) = \emptyset$. Thus $(\xi')^{-r}_k$ satisfies the unique origination property. We concentrate on proving that all its events are enabled at the end of the preceding events.

By definition of bad events it follows that $e_r \neq e_k$ and for all $q : r < q \leq k$, e_q is not a good successor of e_r. This implies in particular that for all $q : r < q \leq k$, $\neg(e_r \rightarrow_\ell e_q)$. From this it also follows that for all $q : r < q \leq k$, $\neg(e_r \stackrel{+}{\rightarrow}_\ell e_q)$, i.e., $e_r \notin LP(e_q)$.

- We first consider the case when e_r is a receive event. Then by Proposition 2.2.14, $T_r = T_{r-1}$ and thus $T = \emptyset$. Then it is clear that τ is the identity map on terms. Hence $\xi' = \xi$. It suffices to prove that ξ^{-r}_k is a run of Pr. Firstly it is clear that ξ_{r-1} is a run of Pr. Consider a q such that $r < q \leq k$. Since all events in $LP(e_q)$ occur in ξ_{q-1} and $e_r \notin LP(e_q)$, it follows that all events in $LP(e_q)$ occur in ξ^{-r}_{q-1}.

  Now if e_q is a receive event, then since $T_r = T_{r-1}$ it is clear that $T^{-r}_{q-1} = T_{q-1}$ and hence $t_q \in T^{-r}_{q-1}$. This suffices to show that e_q is enabled at ξ^{-r}_{q-1}. If e_q is a send event, then since plays of Pr are send-admissible, e_q is enabled at ξ^{-r}_{q-1}.

- Let us now consider the case when e_r is a send event. We first show that ξ'_{r-1}

is a run of Pr. Since $T \subseteq NT(e_r)$ and since $NT(e_r) \cap ST(s_{r-1}) = \emptyset$, τ does not affect any term occurring in ξ_{r-1}. Hence it follows that for all $q < r$, $t_q = t'_q$, $s_q = s'_q$, and $T_q = T'_q$. Thus for all $q < r$, e'_q is enabled at ξ'_{q-1}. This means that ξ'_{r-1} is a run of Pr.

We now show that for all $q : r < q \leq k$, e'_q is enabled at $(\xi')^{-r}_{q-1}$). We first note that for any $i < j \leq k$, $e_i \rightarrow_\ell e_j$ iff $e'_i \rightarrow_\ell e'_j$, $e_i \in LP(e_j)$ iff $e'_i \in LP(e'_j)$, and $EST(e_i) \cap EST(e_j) \neq \emptyset$ iff $EST(e'_i) \cap EST(e'_j) \neq \emptyset$. These statements immediately follow from the definitions.

Fix a q such that $r < q \leq k$. There are two cases to consider:

- If e_q is a receive event, then it is clear that $t_q \in \mathsf{synth}(U)$ where $U = \mathsf{analz}(T_{q-1}) \cap ST(t_q)$. Consider some $u \in U$ and an analz-proof π of $T_{q-1} \vdash u$. It is clear that for all keys k, if $k \in (s_0)_A$ for some $A \in Ag$ then $\overline{k} \in (s_0)_B$ for some $B \in Ag$. Further for any index i, if $k \in NT(e_i)$, then $k \in K_0$ and hence $k = \overline{k}$. So we can say that for any $k \in K$, if $k \in (s_i)_A$ for some $A \in Ag$ then $\overline{k} \in (s_i)_B$ for some $B \in Ag$. Further note that if $k \in ST(s_i)$ then $k \in (s_i)_A$ for some $A \in Ag$, and therefore $\overline{k} \in (s_i)_A$ as well. Now since ξ_{q-1} is not leaky, it follows that whenever $k \in ST(s_r)$ for some $r < q$ and $\overline{k} \in \mathsf{analz}(T_{q-1})$ then $\overline{k} \in \mathsf{analz}(T_r)$. Thus T_{r-1}, $T_{q-1} \setminus T_r$, t_r, T, u and π play the role of S_1, S_2, t, T, u, and π respectively in item 4.3.1 of Lemma 4.3.1 and we get $u \in (\mathsf{analz}(T_r) \cap ST(t_r)) \cup \mathsf{analz}(T^{-r}_{q-1})$. Thus $t_q \in \mathsf{synth}((\mathsf{analz}(T_r) \cap ST(t_r)) \cup \mathsf{analz}(T^{-r}_{q-1} \cup T))$. Now since e_r is not a good predecessor of e_q, $EST(t_q) \cap EST(t_r) = \emptyset$. Thus the conditions of item 2 of Lemma 4.3.1 are fulfilled, and hence $t_q \in \overline{T^{-r}_{q-1} \cup T}$. Applying Proposition 2.3.2 and using the fact that $\tau(T) \subseteq T_0$, we conclude that $t'_q = \tau(t_q) \in \overline{\tau(T^{-r}_{q-1}) \cup \tau(T)} = \overline{(T')^{-r}_{q-1}}$. Hence e'_q is enabled at $(\xi')^{-r}_{q-1}$.
- If e_q is a send event then e'_q is also a send event. Now since plays of Pr are send-admissible it immediately follows that $t'_q \in \overline{(T')^{-r}_{q-1}}$. Hence e'_q is enabled at $(\xi')^{-r}_{q-1}$.

This proves that $(\xi')^{-r}_k$ is a run of Pr.

Claim: $(\xi')^{-r}_k$ is leaky.

Proof of Claim: We first prove that some m which is secret at ξ_{k-1} belongs to $\mathsf{analz}(T^{-r}_k \cup T)$. If e_r is a receive event, then by Proposition 2.2.14 it follows that $T_k = T^{-r}_k$ and hence there is some m which is secret at ξ_{k-1} and which belongs to

$\mathsf{analz}(T_k^{-r})$. (This follows from the fact that ξ is itself leaky). Suppose now that e_r is a send event. Consider an analz-proof of $T_k \vdash m'$ for some m' which is secret at ξ_{k-1}. Let π be a subproof of this proof with the property that the root of π is labelled by some m which is secret at ξ_{k-1} and none of the m'' labelling the nonroot nodes of π is secret at ξ_{k-1}. Then it is clear that T_{r-1}, $T_k \setminus T_r$, t_r, T, m and π play the role of S_1, S_2, t, T, u and π respectively in item 4.3.1 of Lemma 4.3.1 (if $\overline{k}$ labels a node of π and if $k \in ST(s_r)$ then since $\overline{k}$ is not secret at ξ_{k-1} it follows that $\overline{k} \in \mathsf{analz}(T_r)$) and we get $m \in (\mathsf{analz}(T_r) \cap ST(t_r)) \cup \mathsf{analz}(T_k^{-r} \cup T)$. But since ξ_r is not leaky, $m \notin \mathsf{analz}(T_r)$. Thus $m \in \mathsf{analz}(T_k^{-r} \cup T)$. From this it follows that $\tau(m) \in \mathsf{analz}((T')_k^{-r})$.

We now prove that $\tau(m)$ is secret at $(\xi')_{k-1}^{-r}$. Since m is secret at ξ_{k-1} and $T \subseteq \mathsf{analz}(T_r) \subseteq \mathsf{analz}(T_{k-1})$, it follows that $m \notin T$. Therefore $\tau(m) = m$. Since m is secret at ξ_{k-1}, it is clear that $m \notin \mathsf{analz}(T_{k-1})$. Now we observe that $T \subseteq \mathsf{analz}(T_r) \cap \mathcal{T}_0 \subseteq \mathsf{analz}(T_{k-1}) \cap \mathcal{T}_0$. Further τ is a well-typed substitution such that for all $x \in \mathcal{T}_0 \setminus T$, $\tau(x) = x$ and for all $x \in T$, $\tau(x) \in T_{k-1}$. Thus T_{k-1}, T and τ satisfy the conditions of Lemma 4.3.2, and we thus see that whenever $t \in \mathsf{analz}(T'_{k-1})$ there exists $r \in \mathsf{analz}(T_{k-1})$ with $\tau(r) = t$. When $t = m$, it immediately follows that $r = m$ as well. This coupled with the fact that $m \notin \mathsf{analz}(T_{k-1})$ implies that $m \notin \mathsf{analz}(T'_{k-1})$. From this it follows that $m \notin \mathsf{analz}((T')_{k-1}^{-r})$ as well, and thus that $\tau(m) = m$ is secret at $(\xi')_{k-1}^{-r}$. This concludes the proof that $(\xi')_k^{-r}$ is leaky.

We have thus proved the reduction to good runs. □

Lemma 4.3.3 and Theorem 4.2.4 immediately yield us the following theorem.

Theorem 4.3.4 *The problem of checking for a given tagged protocol* Pr *whether there is a well-typed leaky run of* Pr *is decidable.*

We conclude this section by some remarks on the complexity of the problem and on the generalisability of the result.

We saw that the length of a good run of a protocol $\mathsf{Pr} = (\mathsf{C}, \delta)$ with $|\delta| = \ell$ is $2^{2\cdot\ell+1} - 1$. Further at the end of Section 4.1 we saw that for checking a leak in well-typed runs of Pr of length bounded by r, we have to search $O((\ell \cdot r \cdot c_{\mathsf{Pr}})^{r \cdot c_{\mathsf{Pr}}})$ runs for a leak, where c_{Pr} is a constant depending on the protocols. (We can assume that it is at most ℓ, for simplicity). From this we see that the complexity of the secrecy problem for tagged protocols is $2^{2^{O(\ell)}}$. Thus we see that a naive implementation of

the above decision procedure gives a double exponential algorithm.

When the secrecy problem was defined in Section 2.2, it was remarked that a more general notion of secrecy is to allow the user to specify the secret which should not be leaked. In fact, in Chapter 6 we define a logic using which we can specify such a more general notion of secrecy, and other interesting properties like authentication as well. We also prove in Section 6.4 that some of the results proved in Section 5.1 (which are specific to the secrecy problem as defined in Section 2.2) generalise to the logic introduced in Chapter 6.

We would ideally like to similarly extend the results of this chapter. But not all the proofs in this chapter can be adapted to the generalised situation. For instance, the proof of Lemma 4.3.3 crucially uses the fact that we start out with a leaky well-typed run of the given protocol, *none of whose proper prefixes is leaky.* We then show that if this is not a good run, we can do some transformations to eliminate a bad event and still have a leaky run. Among the many secrets which are leaked in the original run, it is possible that some are not leaked in the new run. This can happen especially if its being leaked depends on an eliminated bad event. We are only assured that at least one secret is leaked in the new run as well. So if we allow the user to specify the secret which should not be leaked, it is possible that there is some bad run which leaks the secret but on eliminating some bad events, the new run no longer leaks that particular secret (even though it is guaranteed to leak some other secret). A further difficulty is that even the proof which shows that we can eliminate a bad event to form a new *run* of the protocol depends on our starting out with a run none of whose proper prefixes are leaky. Notwithstanding these difficulties, we still believe that the decidability result of this chapter can be generalised appropriately, and that the ideas introduced in this chapter will lead us to new insights which will help solve the generalised problem.

Chapter 5

Decidability with unbounded message length

In this chapter, we deal with the problem of unbounded message length, which causes undecidability even if we assume a fixed finite set of nonces, as proved in Section 3.2. Even though protocol specifications contain only messages of bounded length, still the intruder can force runs to contain unboundedly long messages by repeated use of ill-typed substitutions. This is the heart of the problem.

In the first section, we prove that the tagging scheme which we have introduced earlier ensures that we can work only with well-typed runs. Specifically, we prove that every run of a tagged protocol has an "equivalent" well-typed run, with the property that the original run is leaky iff its well-typed counterpart is leaky. This proves that the general secrecy problem (with no restrictions on the set of runs considered) is decidable for the class of tagged protocols.

In the second section, we approach the problem of unbounded message length from a different angle. We define a semantically motivated equivalence relation on the set of terms, with the property that it is of finite index if we assume only a fixed finite set of nonces and keys. The crucial property of the equivalence relation is that if two terms are equivalent then the set of basic terms which can be "learnt" from either of them is the same. The equivalence also leads to a notion of normal terms, and thence to a notion of normal runs. We then prove the following semantic result: if every run of Pr is equivalent to a normal run of Pr, then we need only consider a finite set of runs of Pr to check for leakiness. This yields the decidability of the

secrecy problem for the semantic subclass of protocols whose set of runs has this kind of closure property.

5.1 Reduction to well-typed runs

We prove in this section (in Subsection 5.1.2, to be more specific) that if a tagged protocol has a leaky run then it has a well-typed leaky run.

We use the following basic definition throughout this section. For any substitution σ and any nonce z, define σ_z (which is easily seen to be well-typed) as follows:

$$\forall x \in \mathcal{T}_0 : \sigma_z(x) = \begin{cases} z & \text{if } x \in N \text{ and } \sigma(x) \notin N \\ \sigma(x) & \text{otherwise} \end{cases}$$

5.1.1 Typing proofs

In this subsection, we introduce a notion of *type* for analz-proofs and prove some basic properties of them. Of special interest are the so-called *well-typed proofs*. They prove useful in coming up with a well-typed run "equivalent" to a given run of a tagged protocol.

Definition 5.1.1 *A* type *is a pair of the form (σ, r) where r is a term and σ is a substitution suitable for r. Given a set of types P, $terms(P) \stackrel{\text{def}}{=} \{\sigma(r) \mid (\sigma, r) \in P\}$ and for any $z \in N$, $terms_z(P) \stackrel{\text{def}}{=} \{\sigma_z(r) \mid (\sigma, r) \in P\}$.*

By definition, σ is suitable for r iff $\sigma(r)$ is defined. Throughout this section, we will implicitly use the fact that if $\sigma(r)$ is defined, then $\sigma(r_1)$ is defined for any $r_1 \in ST(r)$.

Definition 5.1.2 *A type (σ, r) matches a term t at the outermost level iff $\sigma(r) = t$ and $r \in N \Rightarrow t \in N$.*

The following lemma is a trivial observation which follows from the definition above and the definition of substitutions:

Lemma 5.1.3 *Let (σ, r) match t at the outermost level. Then the following conditions hold:*

- *if $t \in K$ then $r \in K$,*

- *if* $t \in SN$ *then* $r \in SN$,
- *if* t *is of the form* (t', t'') *then* r *is of the form* (r', r''), *and*
- *if* t *is of the form* $\{t'\}_{k'}$ *then* r *is of the form* $\{r''\}_{k''}$.

Definition 5.1.4 *Suppose* P *is a set of types and* π *is an* analz*-proof of* $terms(P) \vdash t$ *for some term* t. *We define* $types_P(\pi)$ *(the* types of π with respect to P*) by induction as follows.*

We also observe the following properties which can be trivially checked by following the definition: for all $(\sigma, r) \in types_P(\pi)$*:*

1. $\sigma(r) = t$,
2. *there exists a term* u *such that* $r \in ST(u)$ *and* $(\sigma, u) \in P$, *and*
3. *for all* $z \in N$, $\sigma_z(r) \in \mathsf{analz}(terms_z(P))$.

- *Suppose* π *is the following proof:*

$$\frac{}{terms(P) \vdash t}\ \mathsf{Ax}_a$$

  Then $(\sigma, r) \in types_P(\pi)$ *iff* $(\sigma, r) \in P$ *and* $\sigma(r) = t$.

- *Suppose* π *is the following proof:*

$$\frac{\begin{array}{c}(\pi_1)\\ \vdots \\ terms(P) \vdash (t, t')\end{array}}{terms(P) \vdash t}\ \mathsf{split}_1$$

  Then $(\sigma, r) \in types_P(\pi)$ *iff there exists* r' *such that* $(\sigma, (r, r')) \in types_P(\pi_1)$.

- *Suppose* π *is the following proof:*

$$\frac{\begin{array}{cc}(\pi_1) & (\pi_2)\\ \vdots & \vdots \\ terms(P) \vdash \{t\}_k & terms(P) \vdash \overline{k}\end{array}}{terms(P) \vdash t}\ \mathsf{decrypt}$$

Then $(\sigma, r) \in types_P(\pi)$ *iff there exist keys* k', k'' *and a substitution* σ'' *such that* $(\sigma, \{r\}_{k'}) \in types_P(\pi_1)$ *and* $(\sigma'', k'') \in types_P(\pi_2)$

- *Suppose* π *is the following proof:*

$$\frac{\begin{array}{c}(\pi_1)\\ \vdots \\ terms(P) \vdash \{\{t\}_k\}_{\overline{k}}\end{array}}{terms(P) \vdash t}\ \mathsf{reduce}$$

Then $(\sigma, r) \in types_P(\pi)$ *iff there exists a key* k' *such that* $(\sigma, \{\{r\}_{k'}\}_{\overline{k'}} \in types_P(\pi_1)$.

π *is said to be* well-typed *with respect to* P *if there exists a type* $(\sigma, r) \in types_P(\pi)$ *such that* r *matches* t *at the outermost level.*

We note the following trivially provable consequence of the definition of *types*.

Lemma 5.1.5 *Suppose that* P *and* P' *are sets of types such that* $P \subseteq P'$ *and* t *is a term such that there exists a proof of* $terms(P) \vdash t$ *which is well-typed with respect to* P. *Then there exists a proof of* $terms(P') \vdash t$ *which is well-typed with respect to* P'. *(We will refer to this as the* upward closure property of well-typed proofs*).*

Lemma 5.1.6 *Suppose* P *is a set of types, and* $u_1 \in \mathsf{analz}(terms_z(P))$ *for some* $z \in N$. *Then there exists* $(\sigma, r) \in P$ *and* $r_1 \in ST(r)$ *such that* $\sigma_z(r_1) = u_1$ *and* $\sigma(r_1) \in \mathsf{analz}(terms(P))$.

Proof: Letting T denote $terms(P)$ and T_z denote $terms_z(P)$, we prove by induction on analz-proofs that for any analz-proof π whose root is labelled $T_z \vdash u_1$ there exists $(\sigma, r) \in P$ and $r_1 \in ST(r)$ such that $\sigma(r_1) \in \mathsf{analz}(T)$ and $\sigma_z(r_1) = u_1$. We only look at the cases when the rule applied at the root of π is Ax_a and decrypt. The other cases are handled by a routine application of the induction hypothesis.

- Suppose π is the following proof:

$$\frac{}{T_z \vdash u_1}\ \mathsf{Ax}_a$$

Then it follows that $u_1 \in T_z$, i.e., there exists $(\sigma, r_1) \in P$ such that $\sigma_z(r_1) = u_1$. But $(\sigma, r_1) \in P$ implies that $\sigma(r_1) \in T \subseteq \mathsf{analz}(T)$, and we are through.

- Suppose π is the following proof:

$$\dfrac{\begin{array}{cc}(\pi_1) & (\pi_2)\\ \vdots & \vdots \\ T_z \vdash \{u_1\}_k & T_z \vdash \overline{k}\end{array}}{T_z \vdash u_1}\ \mathsf{decrypt}$$

  By induction hypothesis there exists $(\sigma, r) \in P$ and $r_2 \in ST(r)$ such that $\sigma(r_2) \in \mathsf{analz}(T)$ and $\sigma_z(r_2) = \{u_1\}_k$. From this it clear that r_2 is of the form $\{r_1\}_{k'}$. Therefore $\sigma(r_2) = \sigma(\{r_1\}_{k'}) = \{\sigma(r_1)\}_{\sigma(k')}$. It is also clear that there exists $(\sigma', r') \in P$ and $r_1' \in ST(r')$ such that $\sigma(r_1') \in \mathsf{analz}(T)$ and $\sigma_z'(r_1') = \overline{k}$. From this and the definition of σ_z it follows that $\sigma(r_1') = \overline{k}$. Also from the fact that $\sigma_z(k') = k$ it follows that $\sigma(k') = k$. Thus we have that $\{\sigma(r_1)\}_k \in \mathsf{analz}(T)$ and $\overline{k} \in \mathsf{analz}(T)$ and it follows that $\sigma(r_1) \in \mathsf{analz}(T)$. Since $\sigma_z(r_2) = \{u\}_k$ it also follows that $\sigma_z(r_1) = u_1$.

□

Definition 5.1.7 *A set of types P is said to be* confusion-free *iff for all (σ, r) and (σ', r') belonging to P and for all $r_1 \in EST(r)$ and $r_1' \in EST(r')$, $\sigma(r_1) = \sigma'(r_1') \Rightarrow r_1 = r_1'$.*

Lemma 5.1.8 *Suppose $P \cup \{(\varsigma, u)\}$ is a* confusion-free set of types *such that every t belonging to* $\mathsf{min}(\mathsf{analz}(terms(P)))$ *has an* analz*-proof that is well-typed with respect to P. Suppose further that $\varsigma(u) \in \overline{terms(P)}$. Then for any $z \in N$, $\varsigma_z(u) \in \overline{terms_z(P) \cup \{z\}}$.*

Proof: We fix a z and let T denote $terms(P)$ and T_z denote $terms_z(P)$ throughout this proof. Note that $\varsigma(u) \in \overline{T} = \mathsf{synth}(\mathsf{min}(\mathsf{analz}(T)))$. We now prove that for all $t_1 \in \mathsf{synth}(\mathsf{min}(\mathsf{analz}(T)))$ such that $t_1 = \varsigma(u_1)$ for some $u_1 \in ST(u)$, $\varsigma_z(u_1) \in \overline{T_z \cup \{z\}}$. Now we do an induction on the structure of terms, based on Fact 2.3.1. (We recall that according to Fact 2.3.1, whenever $t \in \mathsf{synth}(T)$ then $t \in T$, or $t = (t', t'')$ and $\{t', t''\} \subseteq \mathsf{synth}(T)$, or $t = \{t'\}_k$ and $\{t', k\} \subseteq \mathsf{synth}(T)$.)

We first consider the case when $t_1 \in \mathsf{min}(\mathsf{analz}(T))$. For any such t_1, it follows by assumption that there is an analz-proof ϖ of $T \vdash t_1$ that is well-typed with respect

to P. Let $(\sigma, r_1) \in \mathit{types}_P(\varpi)$ be a type which matches t_1 at the outermost level. It follows from the definition of *types* that $\sigma_z(r_1) \in \mathsf{analz}(T_z) \subseteq \overline{T_z \cup \{z\}}$. It is also clear from the definition of *types* that $\sigma(r_1) = t_1 = \varsigma(u_1)$. Now there are two cases to consider, by Proposition 2.3.12 (which, we may recall, says that if $t \in \mathsf{min}(\mathsf{analz}(T))$ then $t \in \mathcal{T}_0$ or t is an encrypted term):

$t_1 \in \mathcal{T}_0$*:* It has to be the case that $r_1 \in \mathcal{T}_0$. Since (σ, r_1) matches t_1 at the outermost level, it follows that $r_1 \in N \Rightarrow t_1 \in N$. Thus it follows that $\sigma_z(r_1) = t_1$. Now either $\varsigma_z(u_1) = z$ or $\varsigma_z(u_1) = \varsigma(u_1) = t = \sigma_z(r_1)$. So in either case $\varsigma_z(u_1) \in \overline{T_z \cup \{z\}}$.

$t_1 \in \mathit{EST}(T)$*:* Here there are two cases to consider:

- $u_1 \in N$*:* Then it is clear that $\varsigma_z(u_1) = z$. It immediately follows that $\varsigma_z(u_1) = z \in \overline{T_z \cup \{z\}}$.
- $u_1 \in \mathit{EST}(u)$*:* Since (σ, r_1) matches t_1 at the outermost level, it follows that r_1 is of the form $\{r_2\}_k$, from Lemma 5.1.3. From the definition of *types* it follows that there exists r such that $(\sigma, r) \in P$ and $r_1 \in \mathit{EST}(r)$. Now since the set $P \cup \{(\varsigma, u)\}$ is $\mathsf{confusion\text{-}free}$ and $\sigma(r_1) = \varsigma(u_1)$, it follows that $r_1 = u_1$. It is thus clear that for all $x \in ST(r_1) \cap \mathcal{T}_0$, $\sigma(x) = \varsigma(x)$, and therefore $\sigma_z(x) = \varsigma_z(x)$. From this it follows that $\varsigma_z(u_1) = \varsigma_z(r_1) = \sigma_z(r_1)$. Therefore $\varsigma_z(u_1) \in \overline{T_z \cup \{z\}}$.

Now we consider the case when t_1 is of the form (t_1', t_1'') and $t_1'$and t_1'' belong to $\mathsf{synth}(\mathsf{min}(\mathsf{analz}(T)))$. Now either $u \in N$ or u is of the form (u', u''). If $u \in N$ then $\varsigma_z(u) = z \in \overline{T_z \cup \{z\}}$. Otherwise $\varsigma(u') = t_1'$ and $\varsigma(u'') = t_1''$, and by induction hypothesis both $\varsigma_z(u')$ and $\varsigma(u'')$ belong to $\overline{T_z \cup \{z\}}$. But now it immediately follows that $\varsigma(u) = (\varsigma(u'), \varsigma(u'')) \in \overline{T_z \cup \{z\}}$.

The case when t_1 is of the form $\{t_1'\}_k$ is identically handled. This concludes the induction step and the proof. □

5.1.2 Reduction to well-typed runs

We prove the following lemma in this subsection.

Lemma 5.1.9 *If a weakly tagged protocol* Pr *has a leaky run, then it has a well-typed leaky run.*

For the rest of this section, we fix a weakly tagged protocol $\mathsf{Pr} = (\mathsf{C}, \delta)$ and a run $\xi = e_1 \cdots e_k$ of Pr with $e_i = (\eta_i, \sigma_i, lp_i)$ for all $i \leq k$. We also fix the following notations related to ξ for the rest of the discussion. For any $j : 1 \leq j \leq k$, ξ_j denotes $e_1 \cdots e_j$, s_j denotes $infstate(\xi_j)$, T_j denotes $(s_j)_I$, a_j denotes $\eta_j(lp_j)$, r_j denotes $term(a_j)$, and t_j denotes $\sigma_j(r_j)$. Similarly $(e_j)_{\mathsf{n}_0}$ denotes $(\eta_j, (\sigma_j)_{\mathsf{n}_0}, lp_j)$, $(\xi_j)_{\mathsf{n}_0}$ denotes $(e_1)_{\mathsf{n}_0} \cdots (e_j)_{\mathsf{n}_0}$, $(s_j)_{\mathsf{n}_0}$ denotes $infstate((\xi_j)_{\mathsf{n}_0})$, $(T_j)_{\mathsf{n}_0}$ denotes $((s_j)_{\mathsf{n}_0})_I$, and $(t_j)_{\mathsf{n}_0}$ denotes $(\sigma_j)_{\mathsf{n}_0}(r_j)$. T_0 and $(T_0)_{\mathsf{n}_0}$ denote $(s_0(\mathsf{Pr}))_I$; and σ_0 and $(\sigma_0)_{\mathsf{n}_0}$ denote the identity substitution. Further, for each $i : 0 \leq i \leq k$, we define a set of types P_i as follows: $P_0 = \{(\sigma_0, m) \mid m \in T_0\}$; for $i : 1 \leq i \leq k$, $P_i = P_{i-1} \cup \{(\sigma_i, r_i)\}$.
Proof: We aim to prove that the sequence $(\xi)_{\mathsf{n}_0} \stackrel{\text{def}}{=} (\xi_k)_{\mathsf{n}_0}$ is a run of Pr which is leaky iff ξ is leaky. It is well-typed by construction. We only have to prove that it is a run of Pr and it is leaky if and only if ξ is leaky.

Claim: $(\xi)_{\mathsf{n}_0}$ is a run of Pr.

> **Proof of Claim:** Firstly we observe that the run ξ has the unique origination property. Further $NT(e_i) = NT((e_i)_{\mathsf{n}_0})$ for all $i \leq k$. Thus it immediately follows that $(\xi)_{\mathsf{n}_0}$ also has the unique origination property. We now concentrate on proving the enabledness of the events in $(\xi)_{\mathsf{n}_0}$.
>
> It is clear that for all $i \leq k$, $(e_i)_{\mathsf{n}_0}$ is an event of Pr, since it is clear from the definitions that $(\sigma_i)_{\mathsf{n}_0}$ is suitable for Pr and η_i. We only have to prove that for all $i \leq k$, $(e_i)_{\mathsf{n}_0}$ is enabled at $(e_1)_{\mathsf{n}_0} \cdots (e_{i-1})_{\mathsf{n}_0}$. Suppose e_i is a send event. Send-admissibility of plays of well-formed protocols ensures that $(e_i)_{\mathsf{n}_0}$ is enabled at $(\xi_{i-1})_{\mathsf{n}_0}$.
>
> So we only need to consider the case when e_i is a receive event. We need to prove that $(t_i)_{\mathsf{n}_0} \in \overline{(T_{i-1})_{\mathsf{n}_0}}$. For this, observe that $\sigma_i(r_i) = t_i \in \overline{T_{i-1}}$. Now it follows from Proposition 2.2.32 (an immediate consequence of the weak tagging scheme) that P_i is a confusion-free set of types. Further it follows from Lemma 5.1.10 (to be proved later) that for all t belonging to $\mathsf{min}(\mathsf{analz}(T_{i-1}))$, there is an analz-proof of $T_{i-1} \vdash t$ that is well-typed with respect to P_i. Thus we can apply Lemma 5.1.8 and it follows that $(t_i)_{\mathsf{n}_0} = (\sigma_i)_{\mathsf{n}_0}(r_i) \in \overline{(T_{i-1})_{\mathsf{n}_0} \cup \{\mathsf{n}_0\}}$. But $\mathsf{n}_0 \in T_0$ and hence $\mathsf{n}_0 \in (T_{i-1})_{\mathsf{n}_0}$. Thus it follows that $(t_i)_{\mathsf{n}_0} \in \overline{(T_{i-1})_{\mathsf{n}_0}}$.

Claim: $(\xi)_{\mathsf{n}_0}$ is leaky iff ξ is leaky.

> **Proof of Claim:** We prove this by showing that for all $i : 1 \leq i \leq k$, $\overline{T_i} \cap \mathcal{T}_0 = \overline{(T_i)_{\mathsf{n}_0}} \cap \mathcal{T}_0$. Since the initial states of both runs and the new nonces generated at each event of both runs are the same, it immediately follows that ξ_{n_0} is leaky iff ξ is.
>
> Suppose $m \in \overline{T_i} \cap \mathcal{T}_0$. Then it is clear that $m \in \mathsf{min}(\mathsf{analz}(T_i))$. From Lemma 5.1.10 it is clear that there is an analz-proof π of $T_i \vdash m$ that is well-typed with respect to P_i. Let $(\sigma, r) \in \mathit{types}_{P_i}(\pi)$. It is clear that $r \in N$ as well and that $\sigma_{\mathsf{n}_0}(r) = m$. But now it follows from the definition of *types* that $m \in \mathsf{analz}(T_{\mathsf{n}_0})$. This shows that $\overline{T_i} \cap \mathcal{T}_0 \subseteq \overline{(T_i)_{\mathsf{n}_0}} \cap \mathcal{T}_0$.
>
> Now suppose $m \in \mathsf{analz}((T_i)_{\mathsf{n}_0}) \cap \mathcal{T}_0$. By Lemma 5.1.6 it follows that there exists $(\sigma, r) \in P_i$ and $r_1 \in ST(r)$ such that $\sigma(r_1) \in \mathsf{analz}(T_i)$ and $\sigma_{\mathsf{n}_0}(r_1) = m$. Now if $m = \mathsf{n}_0$ then $m \in T_0$. If $m \neq \mathsf{n}_0$ then it follows that $\sigma(r_1) = \sigma_{\mathsf{n}_0}(r_1) = m$. But then we have that $m \in \mathsf{analz}(T_i)$. This shows that $\overline{(T_i)_{\mathsf{n}_0}} \cap \mathcal{T}_0 \subseteq \overline{T_i} \cap \mathcal{T}_0$ and hence the claim follows.

This completes the proof of the lemma, assuming Lemma 5.1.10. □

Lemma 5.1.10 *For all $i : 1 \leq i \leq k$ and for all $t \in \mathsf{min}(\mathsf{analz}(T_i))$, there is an analz-proof of $T_i \vdash t$ that is well-typed with respect to P_i.*

Proof: The proof is by induction on i.

Base case: $i = 0$: If $t \in \mathsf{analz}(T_0)$ then for any analz-proof π of $T_0 \vdash t$, (σ_0, t) belongs to $\mathit{types}_{P_0}(\pi)$. Clearly (σ_0, t) matches t at the outermost level and thus π is an analz-proof of $T_0 \vdash t$ that is well-typed with respect to P_0.

Induction case: Assume that $i > 0$ and that for all $j < i$ and $t \in \mathsf{min}(\mathsf{analz}(T_j))$, there is an analz-proof of $T_j \vdash t$ that is well-typed with respect to P_j. By the upward closure property of well-typed proofs, we see that for all such t, there is an analz-proof of $T_i \vdash t$ that is well-typed with respect to P_i. Now suppose $t \in \mathsf{min}(\mathsf{analz}(T_i)) \setminus \mathsf{analz}(T_{i-1})$ and π is an analz-proof of $T_i \vdash t$. Then we prove by induction on proofs that for all subproofs ϖ of π with root labelled $T_i \vdash u$, either $u \in \overline{T_{i-1}}$ or there is an analz-proof of $T_i \vdash u$ that is well-typed with respect to P_i. For this we assume that for all proper subproofs ϖ' of

ϖ with root labelled $T_i \vdash u'$, u' has this property and use it to prove that u itself has this property. Once we prove this the desired result follows, since it cannot be the case that t, which is assumed to be a *minimal* term in $\mathsf{analz}(T)$, belongs to $\mathsf{synth}(\mathsf{analz}(T_{i-1})) \subseteq \mathsf{synth}(\mathsf{analz}(T_i) \setminus \{t\})$.

- Suppose ϖ is the following proof:

$$\frac{}{T_i \vdash u}\,\mathsf{Ax}_a$$

  Then $u \in T_i$. By definition of *types*, $types_{P_i}(\varpi) \neq \emptyset$. By Lemma 5.1.11 (which is proved next) it follows that either $u \in \overline{T_{i-1}}$ or ϖ is well-typed with respect to P_i, and we are through.

- Suppose ϖ is the following proof:

$$\frac{\begin{array}{c}(\varpi_1)\\ \vdots \\ T_i \vdash (u, u')\end{array}}{T_i \vdash u}\,\mathsf{split}_1$$

  By induction hypothesis either $(u, u') \in \overline{T_{i-1}}$ or there is an analz-proof ρ_1 of $T_i \vdash (u, u')$ that is well-typed with respect to P_i. In the first case $u \in \mathsf{analz}(\overline{T_{i-1}}) = \overline{T_{i-1}}$ and we are done. In the second case, we have the following proof ρ of $T_i \vdash u$:

$$\frac{\begin{array}{c}(\rho_1)\\ \vdots \\ T_i \vdash (u, u')\end{array}}{T_i \vdash u}\,\mathsf{split}_1$$

  By definition of *types*, $types_{P_i}(\rho) \neq \emptyset$. It follows from Lemma 5.1.11 that either $u \in \overline{T_{i-1}}$ or ρ is well-typed with respect to P_i, and we are through.

- Suppose ϖ is the following proof:

$$\frac{\begin{array}{cc}(\varpi_1) & (\varpi_2)\\ \vdots & \vdots \\ T_i \vdash \{u\}_k & T_i \vdash \overline{k}\end{array}}{T_i \vdash u}\,\mathsf{decrypt}$$

By induction hypothesis either there is an analz-proof of $T_i \vdash \overline{k}$ that is well-typed with respect to P_i or $\overline{k} \in \overline{T_{i-1}}$. In the first case we are done. In the second case, we note that $\overline{k}$ is a basic term, and hence $\overline{k} \in \overline{T_{i-1}} \Rightarrow \overline{k} \in \mathsf{min}(\mathsf{analz}(T_{i-1}))$. The induction hypothesis (on $i-1$) and the upward closure property of well-typed proofs assure us that there is an analz-proof ρ_2 of $T_i \vdash \overline{k}$ that is well-typed with respect to P_i in this case also. Similarly, by induction hypothesis either $\{u\}_k \in \overline{T_{i-1}}$ or there is an analz-proof ρ_1 of $T_i \vdash \{u\}_k$ that is well-typed with respect to P_i. In the case where $\{u\}_k \in \overline{T_{i-1}}$, if $u \in \overline{T_{i-1}}$ we are done. Otherwise $\{u\}_k \in \mathsf{min}(\mathsf{analz}(T_{i-1}))$, and the induction hypothesis (on $i-1$) and the upward closure property of well-typed proofs assure us that there is an analz-proof ρ_1 of $T_i \vdash \{u\}_k$ that is well-typed with respect to P_i. Given ρ_1 and ρ_2, we can build the proof ρ as follows:

$$\dfrac{\begin{array}{cc}(\rho_1) & (\rho_2)\\ \vdots & \vdots \\ T_i \vdash \{u\}_k & T_i \vdash \overline{k}\end{array}}{T_i \vdash u}\ \mathsf{decrypt}$$

By definition of *types* it is clear that $types_{P_i}(\rho) \neq \emptyset$. It follows from Lemma 5.1.11 that either $u \in \overline{T_{i-1}}$ or ρ is well-typed with respect to P_i, and we are through.

- Suppose ϖ is the following proof:

$$\dfrac{\begin{array}{c}(\varpi_1)\\ \vdots \\ T_i \vdash \{\{u\}_k\}_{\overline{k}}\end{array}}{T_i \vdash u}\ \mathsf{reduce}$$

By induction hypothesis either $\{\{u\}_k\}_{\overline{k}} \in \overline{T_{i-1}}$ or there is an analz-proof ρ_1 of $T_i \vdash \{\{u\}_k\}_{\overline{k}}$ that is well-typed with respect to P_i. In the first case, it is clear that $u \in \mathsf{analz}(\overline{T_{i-1}}) = \overline{T_{i-1}}$. We now show that the second case cannot arise at all for the following reason: by induction hypothesis there exists $(\sigma, r) \in types_{P_i}(\rho_1)$ which matches $\{\{u\}_k\}_{\overline{k}}$ at the outermost level. So r is of the form $\{r'\}_{k'}$. But then since Pr is a tagged protocol and $\{r'\}_{k'} \in EST(\delta)$, r' is of the form (c, r'') for some $\mathsf{c} \in \mathsf{C}$ and some r''. It also follows from the definition of *types* that $\sigma(r) = \{\{u\}_k\}_{\overline{k}}$, but this

would mean that $\sigma(\mathsf{c}, r'') = \{u\}_k$, an impossibility. Thus the second case cannot arise at all and we are done.

This concludes the induction step and the proof. The lemma is thus proved, assuming Lemma 5.1.11. □

Lemma 5.1.11 *Suppose* $1 \leq i \leq k$ *and* $t \in \mathsf{analz}(T_i)$ *such that there is an* analz*-proof* π *of* $T_i \vdash t$ *with* $types_{P_i}(\pi) \neq \emptyset$. *Then either* π *is well-typed with respect to* P_i *or* $t \in \overline{T_{i-1}}$.

Proof: Suppose $(\sigma, r) \in types_{P_i}(\pi)$. If (σ, r) matches t at the outermost level, then π is well-typed with respect to P_i. Otherwise it has to be the case that $r \in N$ and $t \notin N$. Since $\sigma(r) = t$ and $r \neq t$, it cannot be the case that $\sigma = \sigma_0$. Hence $\sigma = \sigma_j$ for some $j \geq 1$. It is clear from the definition of *types* that there exists u such that $r \in ST(u)$ and $(\sigma, u) \in P_i$. Since $\sigma = \sigma_j$, $u = r_j$. But now $r \in ST(r_j) \cap N$ and $\sigma_j(r) \notin N$, so it follows from Lemma 5.1.15 (which is proved later) that $t = \sigma_j(r) \in \overline{T_{j-1}} \subseteq \overline{T_{i-1}}$. Thus the lemma is proved, assuming Lemma 5.1.15. □

The following definition and the next two lemmas are preparatory to proving Lemma 5.1.15.

Definition 5.1.12 *We say that a term* t originates at $i \leq k$ *in* ξ *iff* $t \in ST(e_i)$ *and for all* $j < i$, $t \notin ST(e_j)$.

Lemma 5.1.13 *Suppose* e_i *is a* send event *for some* $i : 1 \leq i \leq k$ *and there exists* $n \in ST(r_i) \cap N$ *such that* $\sigma_i(n) \notin N$. *Then* $i > 1$ *and there exists* $j : 1 \leq j < i$ *such that* $n \in ST(r_j)$ *and* $\sigma_i(n) = \sigma_j(n)$.

Proof: Since $\sigma_i(n) \notin N$, it follows from definitions that $n \notin NT(a_i)$ (otherwise σ_i would not be suitable for a_i and hence e_i would not be an event). Also the run ξ has the property of unique origination of nonces, and hence, it follows that $n \notin \mathsf{CT}(\mathsf{Pr})$. But the fact that $n \in ST(r_i)$ implies (again by the send-admissibilty of roles of well-formed protocols) that $n \in ST(\eta_i(lp))$ for some $lp < lp_i$. But then, since $LP(e_i) \subseteq \{e_1, \ldots, e_{i-1}\}$, it follows that $e = (\eta_i, \sigma_i, lp) \in \{e_1, \ldots, e_{i-1}\}$ and thus there exists $j : 1 \leq j < i$ such that $n \in ST(r_j)$ and $\sigma_i(n) = \sigma_j(n)$. □

Lemma 5.1.14 *Suppose a term t* originates at a receive event *e_i for some $i \leq k$. Then $t \in \overline{T_{i-1}}$, and further, if $t = \{u\}_k$ for some u and k then $\{u, k\} \subseteq \overline{T_{i-1}}$.*

Proof: It is clear from the definition of runs that since e_i is a receive event, $t_i \in \overline{T_{i-1}}$. It is also clear that $t \in ST(t_i) \subseteq ST(\overline{T_{i-1}})$ and therefore by Proposition 2.3.7 it follows that $t \in ST(\mathsf{analz}(T_{i-1}))$ or $t \in \overline{T_{i-1}}$. (Recall that according to Proposition 2.3.7, whenever $r \in ST(\mathsf{synth}(T))$ then $r \in \mathsf{synth}(T) \cup ST(T)$.) Now $\mathsf{analz}(T) \subseteq ST(T)$ (and hence $ST(\mathsf{analz}(T)) = ST(T)$) for any set of terms T, and therefore it follows that either $t \in ST(T_{i-1})$ ot $t \in \overline{T_{i-1}}$. Now since t originates at e_i, it cannot be the case that $t \in ST(T_{i-1})$. Therefore $t \in \overline{T_{i-1}}$. Further if $t = \{u\}_k$ we can apply Proposition 2.3.8 to t and $\mathsf{analz}(T_{i-1})$ and conclude that $\{u, k\} \subseteq \mathsf{synth}(\mathsf{analz}(T_{i-1})) = \overline{T_{i-1}}$. (Recall that according to Proposition 2.3.8, whenever $\{r\}_k \in ST(\mathsf{synth}(T))$ then $r \in ST(T)$ or $\{r, k\} \subseteq \mathsf{synth}(T)$. Further, in the present case $t \notin ST(T_{i-1}) = ST(\mathsf{analz}(T_{i-1}))$. Hence the conclusion.) □

Lemma 5.1.15 *If $\sigma_i(n) \notin N$ for some $i : 1 \leq i \leq k$ and $n \in ST(r_i) \cap N$, then $\sigma_i(n) \in \overline{T_{i-1}}$.*

Proof: The proof is by induction on i.

Base case: $i = 1$: Suppose there exists $n \in ST(r_i) \cap N$ such that $\sigma_i(n) \notin N$. We first note that e_i cannot be a send event for then, by Lemma 5.1.13, it would follow that $i > 1$, contradicting the fact that $i = 1$. Thus e_i is a receive event, and hence $t_i \in \overline{T_{i-1}}$ and since $T_{i-1} = T_0 \subseteq \mathcal{T}_0$ it follows from Proposition 2.3.9 that $t \in \overline{T_{i-1}}$ for all $t \in ST(t_i)$ and in particular $\sigma_i(n) \in \overline{T_{i-1}}$. (Recall that according to Proposition 2.3.9, whenever $T \subseteq \mathcal{T}_0$, $ST(\mathsf{synth}(T)) \subseteq \mathsf{synth}(T)$.)

Induction case: Suppose $i > 1$ and the statement of the lemma holds for all $j < i$. Suppose there exists an $n \in ST(r_i) \cap N$ such that $\sigma_i(n) \notin N$. There are two cases to consider here:

e_i is a receive event: In this case it is clear that $t_i = \sigma_i(r_i) \in \overline{T_{i-1}}$. Now if n occurs unencrypted in r_i, $\sigma_i(n) \in \overline{T_{i-1}}$ as well and the induction case is through. Otherwise let $\{u\}_k$ be the smallest encrypted subterm of r_i containing n. Let $\sigma_i(\{u\}_k)$ originate at some $j \leq i$. There are two cases to consider here:

e_j is a receive event: In this case, it follows from Lemma 5.1.14 that $\sigma_i(u) \in \overline{T_{j-1}}$ and since $\{u\}_k$ is a minimum encrypted term containing n as a subterm, $n \in \mathsf{analz}(u)$ and hence $\sigma_i(n) \in \overline{T_{j-1}} \subseteq \overline{T_{i-1}}$.

e_j is a send event: Now it cannot be the case that $\sigma_i(\{u\}_k) \in ST(\sigma_j(m))$ for some $m \in ST(r_j) \cap N$, since it is in violation of Lemma 5.1.13. It also cannot be the case that there is some $\{u'\}_{k'} \in ST(r_j)$ such that $\{u'\}_{k'} \neq \{u\}_k$ and $\sigma_i(\{u\}_k) = \sigma_j(\{u'\}_{k'})$, since it is in violation of Proposition 2.2.32. The only remaining case is that $\{u\}_k \in ST(r_j)$ and $\sigma_i(\{u\}_k) = \sigma_j(\{u\}_k)$ in which case it follows that $\sigma_i(n) = \sigma_j(n)$. Also note that since e_i is a receive event, $j < i$. Hence by induction hypothesis $\sigma_i(n) \in \overline{T_{j-1}} \subseteq \overline{T_{i-1}}$.

e_i is a send event: Since $\sigma_i(n) \notin N$, it follows from Lemma 5.1.13 that there is a $j < i$ such that $n \in ST(r_j)$ and $\sigma_j(n) = \sigma_i(n)$. Thus it follows by induction hypothesis that $\sigma_i(n) \in \overline{T_{j-1}} \subseteq \overline{T_{i-1}}$.

This completes the proof of the lemma. □

Of course the statement of Lemma 5.1.9 holds for tagged protocols as well. This combined with Theorem 4.3.4 leads to the following result, which is the central result of the thesis.

Theorem 5.1.16 . *The general secrecy problem (with no restriction on the set of runs considered) is decidable for the class of tagged protocols.*

5.2 An approach based on equivalence on terms

As mentioned earlier, we approach the problem of unbounded message length in a different manner in this section. We define an equivalence relation on terms based on which we obtain a subclass of protocols for which the secrecy problem is decidable, under the assumption that the keys and nonces used come from a fixed finite set.

The equivalence relation is based on the following semantic motivations: In typical protocols the term (t, t) is not construed as conveying more information than the term t alone. Even in the rare case where it conveys more information, it does so only in an indirect manner. For instance, the same term repeated twice in a

message might signify some control information. In that case, we can use some more direct scheme to convey that information. A similar argument holds for repeated encryptions with the same key as well. Extending this line of thinking, we see that a term of the form $\{\{(\{m,n\}_k,m)\}_{k'}\}_k$ conveys really the same information that $\{\{m,n\}_{k'}\}_k$ does. It can be seen that it is reasonable to equate the two terms, since an agent with a given set of keys learns the same basic terms from both these terms.

These considerations lead us to our definition of the equivalence relation, which is meant to enforce a reasonableness condition on the kinds of messages that can be constructed. We leave open the question of how these rules can be implemented so that only reasonable messages are used. Even if we restrict the protocol specifications to refer only to *normal terms* (which formally stand for "reasonable messages"), the runs of the protocol might not contain only normal terms. It can be seen that such a situation might arise only due to the actions of an unrestricted intruder. One possible way of enforcing the use of normal terms in all the runs is to offer only some restricted kinds of message building capabilities to the users of the protocol, at the implementation level. There are many other ways of achieving the same result, and the decidability result that we prove in this section applies irrespective of the specific scheme used to implement this. The result is proved for a general semantic class of protocols (informally, these are protocols which have "normal representatives" for any of their runs).

We set up the following notation and terminology for this section: We say that *a key k encrypts in a term t* if $\exists t' : \{t'\}_k \in ST(t)$.

Given a term t and a key k define t_{-k} by induction as follows: for $m \in \mathcal{T}_0$, $m_{-k} = m$; $(t,t')_{-k} = (t_{-k}, t'_{-k})$; and $(\{t\}_{k'})_{-k}$ is defined to be t_{-k} if $k = k'$, and $\{t_{-k}\}_{k'}$ otherwise. Thus t_{-k} is the term t with all encryptions by key k removed.

The encryption depth of a term is defined by induction as follows:

$encdepth(m) = 0$ for $m \in \mathcal{T}_0$;
$encdepth((t,t')) = max(encdepth(t), encdepth(t'))$; and
$encdepth(\{t\}_k) = encdepth(t) + 1$.

We also fix a finite set $T \subseteq \mathcal{T}_0$ of size B. Throughout this section we will only consider terms t with the property that $ST(t) \subseteq T$.

Definition 5.2.1 *An $\equiv$-proof is an inverted tree whose nodes are labelled by equations of the form $r \sim r'$ and connected by one of the* rules *in Figure 5.1 and whose*

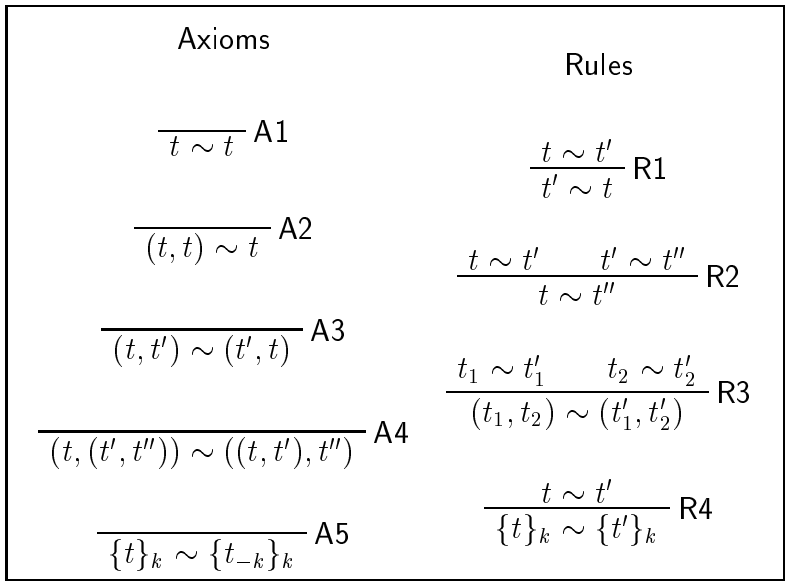

Figure 5.1: Axioms and rules for $\equiv$-proofs.

leaves are labelled by instances of the axioms *in Figure 5.1.*

We say that $t \equiv t'$ iff there is an $\equiv$-proof whose root is labelled by $t \sim t'$. We say that $t \equiv_1 t'$ iff there is an $\equiv$-proof whose root is labelled by $t \sim t'$, and none of whose leaves are labelled by the axioms A2 *and* A5.

Definition 5.2.2 *Any term which has a subterm of the form (r, r) or of the form $\{r\}_k$ with k encrypting in r is said to be a* redex. *A term t is said to be* normal *if there is no t' such that $t \equiv_1 t'$ and t' is a redex. A substitution σ is normal iff for all $x \in \mathcal{T}_0$: if $\sigma(x)$ is defined then it is normal. An event $e = (\eta, \sigma, lp)$ is normal if σ is normal, and a sequence of events ξ is normal iff all the events occurring in it are normal.*

The main function of the equivalence relation is to ensure two things: the tupling operator works with sets of terms now rather than lists, which is ensured by Axioms A2 to A4; the depth of the encryption operator is bounded. The latter is achieved by the axiom A5, which ensures that if we consider a basic term m occurring in two equivalent terms t and t', the same keys encrypt m in both t and t'. Thus it easily follows that for any set of terms T, $\mathsf{analz}(T \cup \{t\}) \cap \mathcal{T}_0 = \mathsf{analz}(T \cup \{t'\}) \cap \mathcal{T}_0$. This property is crucial for our later development.

We first observe the following property which follows immediately from the def-

initions.

Proposition 5.2.3 *For any two terms t and t', if $t \equiv_1 t'$ then t is normal iff t' is normal.*

Lemma 5.2.4 *For any normal term t, $encdepth(t) \leq B$.*

Proof: This is quite easy to see. Firstly note there are at most B keys in T. Now the result can be proved by a a trivial induction on the structure of terms as follows:

If $t \in T$ then of course $encdepth(t) = 0 \leq B$.

Suppose t is of the form (r, r'). We first claim that r and r' are normal terms. For, suppose r were not a normal term, for example. Then there is a redex u such that $r \equiv_1 u$. But now $(r, r') \equiv_1 (u, r')$. Since u is a redex, (u, r') is also a redex, and hence t would itself be a nonnormal term. This contradiction leads us to the fact that r and r' are normal terms. Therefore $encdepth(r) \leq B$ and $encdepth(r') \leq B$, by induction hypothesis. Thus $encdepth(t) = max(encdepth(r), encdepth(r')) \leq B$.

Suppose t is of the form $\{r\}_k$. Then as before we can show that r is a normal term. So $encdepth(r) \leq B$. But since t is a normal term, it follows that it is not a redex. From this it follows that k does not encrypt in r. Thus $encdepth(r)$ is strictly less than B. From this it follows that $encdepth(t) \leq B$. □

Lemma 5.2.5 *The equivalence relation $\equiv$ on terms is of finite index. Further there is a bound on the size of normal terms.*

Proof: It is easy to see that every term is equivalent to a normal term. We now show that the set of normal terms is finite, which will immediately imply the statement of the proposition. We will also simultaneously prove that each normal term is of bounded size (which depends only on T.)

Recall that $|T| = B$. Let us denote by N_i the set of normal terms of encryption depth i. We show below that there is a bound f_i on the size of the terms in N_i. Since all normal terms are encryption depth at most B, the number f_B is a bound on the size of normal terms.

Consider a term t in N_0. Clearly t is built up using only the pairing construct, with no basic term having more than one occurrence. Thus t can be viewed as a binary tree with at most B leaves. The size of such a tree can be at most $2 \cdot B$. Thus we can let $f_0 = 2 \cdot B$.

Consider a term t in N_i. Suppose the set N_{i-1} is of size at most g_{i-1}. Now we note that any term in N_i can be built from terms of the form $\{r\}_k$ (with $r \in N_{i-1}$) using the pairing construct repeatedly. The number of terms of the form $\{r\}_k$ with $r \in N_{i-1}$ is at most $B \cdot g_{i-1}$ (since any of at most B keys can be used to encrypt any of the at most g_{i-1} terms from N_{i-1}). Now since t is normal, it follows that there is at most one occurrence of each of the above $B \cdot g_{i-1}$ terms in t. Thus t can again be viewed as a binary tree with at most $B \cdot g_{i-1}$ leaves. The size of t cannot exceed $2 \cdot B \cdot g_{i-1}$. This number can be chosen as f_i.

We now show how to determine g_i from f_i, for each i. We first look at the different "structures" of size f_i that can occur. A loose upper bound is the number of binary trees with at most f_i leaves. This gives us a bound of $f_i^{O(f_i)}$. Now we can map each of the leaves of these trees to any one of the B basic terms to form terms in N_i, so we get an estimate of $B^{f_i^{O(f_i)}}$ for g_i.

This completes the proof of this lemma. □

While the bounds arrived at in the above lemma suffice for our decidability results, they are clearly not practical. More work needs to be done in coming up with protocol-specific equivalences which yield practical bounds.

We now come to the second part of our endeavour, which is to prove that if ξ and ξ' are equivalent runs, then ξ is leaky iff ξ' is. We say that $\sigma \equiv \sigma'$ for two substitutions σ and σ' iff their domains of definition are the same and for all $x \in \mathcal{T}_0$, if $\sigma(x)$ is defined then $\sigma'(x) \equiv \sigma(x)$. We say that $(\eta, \sigma, lp) \equiv (\eta', \sigma', lp')$ iff $\eta = \eta'$, $lp = lp'$, and $\sigma \equiv \sigma'$. Given two sequences of events $\xi = e_1 \cdots e_k$ and $\xi' = e'_1 \cdots e'_k$, we say that $\xi \equiv \xi'$ iff for all $i \leq k$, $e_i \equiv e'_i$.

We now prove the crucial semantic property of the equivalence on runs. Preparatory to that is the following property of equivalent terms.

Proposition 5.2.6 *Suppose t and t' are two terms with $t \equiv t'$. Suppose U is a set of basic terms. Then* $\mathsf{analz}(U \cup \{t\}) \cap T = \mathsf{analz}(U \cup \{t'\}) \cap T$.

Proof: We note that it suffices to prove the statement when t is of the form $\{r\}_k$ and t' is of the form $\{r_{-k}\}_k$. Then a trivial induction on $\equiv$-proofs yields the desired result.

We now proceed to prove that $\mathsf{analz}(U \cup \{\{r\}_k\}) \cap T = \mathsf{analz}(U \cup \{\{r_{-k}\}_k\}) \cap T$. At the outset there are two cases to be considered:

- Suppose $\overline{k} \notin U$. Then $\mathsf{analz}(U \cup \{\{r\}_k\}) = \mathsf{analz}(U \cup \{\{r_{-k}\}_k\}) = \emptyset$, so we get our result.

- Suppose $\overline{k} \in U$. We now prove by induction on the structure of terms that $\mathsf{analz}(U \cup \{r\}) \cap T = \mathsf{analz}(U \cup \{r_{-k}\}) \cap T$. The desired result follows since the presence of $\overline{k}$ in U ensures that $\mathsf{analz}(U \cup \{r_{-k}\}) \cap T = \mathsf{analz}(U \cup \{\{r_{-k}\}_k\}) \cap T$.

  When $r \in T$ then $r_{-k} = r$, so it immediately follows that $\mathsf{analz}(U \cup \{r\}) = \mathsf{analz}(U \cup \{\{r_{-k}\}_k\})$.

  When $r = (u, u')$ then $r_{-k} = (u_{-k}, u'_{-k})$. By induction hypothesis we know that $\mathsf{analz}(U \cup \{u\}) \cap T = \mathsf{analz}(U \cup \{u_{-k}\}) \cap T$, and that a similar property holds for u'. The result now follows by noting that $\mathsf{analz}(U \cup \{(u, u')\}) \cap T = (\mathsf{analz}(U \cup \{u\}) \cup \mathsf{analz}(U \cup \{u'\})) \cap T$, and that a similar property holds for (u_{-k}, u'_{-k}).

  When $r = \{u\}_{k'}$, there are two cases to consider. If $k' = k$ then $r_{-k} = u_{-k}$. By induction hypothesis $\mathsf{analz}(U \cup \{u\}) \cap T = \mathsf{analz}(U \cup \{u_{-k}\}) \cap T$. But the presence of $\overline{k}$ in U ensures that $\mathsf{analz}(U \cup \{u\}_k) \cap T = \mathsf{analz}(U \cup \{u\}) \cap T$. From this the desired result follows. If $k' \neq k$ then $r_{-k} = \{u_{-k}\}_{k'}$. By induction hypothesis $\mathsf{analz}(U \cup \{u\}) \cap T = \mathsf{analz}(U \cup \{u_{-k}\}) \cap T$. Again a case analysis based on whether $\overline{k'}$ belongs to U or not yields the desired result.

□

Proposition 5.2.7 *Suppose* Pr *is a protocol and* ξ *and* ξ' *are runs of* Pr *such that* $\xi \equiv \xi'$. *Then* $\overline{(infstate(\xi))_A} \cap T = \overline{(infstate(\xi'))_A} \cap T$ *for all* $A \in Ag$. *Further* ξ *is leaky iff* ξ' *is leaky.*

Proof: We prove the proposition by induction on the length of the runs. In the base case $\xi = \xi' = \varepsilon$ and therefore clearly $infstate(\xi) = infstate(\xi') = \mathsf{init}(\mathsf{Pr})$ and the proposition is true. For the induction step suppose that $\xi = \xi_1 \cdot e$ and $\xi' = \xi'_1 \cdot e'$ with $e \equiv e'$ and $\xi_1 \equiv \xi'_1$. Fix an $A \in Ag$. By induction hypothesis we see that $\overline{(infstate(\xi_1))_A} \cap T = \overline{(infstate(\xi'_1))_A} \cap T$. Let this set be denoted by U. Now we only consider the case when e is a receive event by A. Let $t = act(e)$ and $t' = act(e')$. Clearly $t \equiv t'$. Then we note that $\overline{(infstate(\xi))_A} \cap T = \mathsf{analz}(U \cup \{t\}) \cap T$, and that

a similar property holds for ξ'. It immediately follows from Proposition 5.2.6 that $\overline{(infstate(\xi))_A} \cap T = \overline{(infstate(\xi'))_A} \cap T$.

We now claim that if $e_1 \cdots e_k \equiv e'_1 \cdots e'_k$ then for all $i \leq k$, $NT(e_i) = NT(e'_i)$. This is easy to see. If we let $e_i = (\eta_i, \sigma_i, lp_i)$ and $e'_i = (\eta'_i, \sigma'_i, lp'_i)$, then for all $m \in NT(\eta_i(lp_i))$, $\sigma(m) \in T$. But $\sigma(m) \equiv \sigma'(m)$ and, since $m \in \mathcal{T}_0$, it can only be the case that $\sigma(m)$ is the same as $\sigma'(m)$. This shows that $NT(e_i) = NT(e'_i)$.

The above two facts immediately imply that ξ is leaky iff ξ' is leaky. □

We now define a semantic subclass of protocols, the class of $\equiv$-*invariant* protocols.

Definition 5.2.8 *A protocol* Pr *is said to be* $\equiv$*-invariant iff for all runs* ξ *of* Pr*, there is a normal run of* ξ' *of* Pr *such that* $\xi \equiv \xi'$.

It immediately follows that, given an $\equiv$-invariant protocol Pr, checking whether there is a leaky run of Pr boils down to checking whether there is a normal leaky run of Pr. Now the set of normal events of Pr is bounded in number (the bound depending on the number f_B derived in Lemma 5.2.5 and the specification of Pr). But this does not mean that the set of normal runs of Pr is a finite set. The problem arises because the same event may occur many times in a run (as long as it does not generate any new nonces), and so there is no bound on the length of the runs that we have to consider. A solution to this problem is provided in the proof of the following theorem.

Theorem 5.2.9 *The problem of checking whether a given* $\equiv$*-invariant protocol has a leaky run is decidable.*

Proof: Given an $\equiv$-invariant protocol Pr, it suffices to check whether there is a normal leaky run of Pr or not. We now show that this is equivalent to checking whether there is a *reduced* normal leaky run of Pr or not. We recall that a reduced run is a run with all duplicate occurrences of events removed. Since there are only boundedly many normal events, and since there is at most one occurrence of any event in a reduced run, the set of reduced normal runs of Pr is finite, and thus we obtain decidability.

It follows from Proposition 2.2.20 that if ξ is a run of Pr so is $\mathsf{red}(\xi)$. We now prove that ξ is leaky iff $\mathsf{red}(\xi)$ is leaky. Suppose ξ is leaky. This means that

there is a basic term m and a prefix ξ' of ξ such that m is secret at ξ' and not secret at ξ. From Proposition 2.2.20 we see that $\mathit{infstate}(\xi) = \mathit{infstate}(\mathsf{red}(\xi))$ and $\mathit{infstate}(\xi') = \mathit{infstate}(\mathsf{red}(\xi'))$. Thus it follows that m is secret at $\mathsf{red}(\xi')$ and not secret at $\mathsf{red}(\xi)$. Further it is clear from the definitions that $\mathsf{red}(\xi')$ is a prefix of $\mathsf{red}(\xi)$. Thus $\mathsf{red}(\xi)$ is also leaky.

Suppose on the other hand that $\mathsf{red}(\xi)$ is leaky. This means that there is a basic term m which is secret at some prefix of $\mathsf{red}(\xi)$ but not secret at $\mathsf{red}(\xi)$. We now use the fact (which immediately follows from definitions) that any prefix of $\mathsf{red}(\xi)$ is of the form $\mathsf{red}(\xi')$ for some prefix ξ' of ξ. Thus we see that m is secret at $\mathsf{red}(\xi')$ and not secret at $\mathsf{red}(\xi)$. From Proposition 2.2.20, it follows that m is secret at ξ' but not secret at ξ. This means that ξ is leaky.

So we see that there is a normal leaky run of Pr iff there is a reduced normal leaky run of Pr, and this completes the proof of the theorem. □

The work in this section suggests an approach to the verification of security protocols. To make this relevant to practice, much more work needs to be done to yield better bounds on the size of terms. This might entail changing the definition of the equivalence relation suitably (perhaps with some specific classes of protocols in mind). Further we need to come up with syntactic conditions on protocols which ensure that they are $\equiv$-invariant. It is needed because as of now we do not have any method of effectively checking whether a given protocol is $\equiv$-invariant or not. We conclude by saying that the development in this section sets up a framework for the verification of security protocols, and that there is still some way to go before we obtain results which are relevant to practice.

Chapter 6

Reasoning about security protocols

In this chapter, we develop a logic for specifying interesting properties of protocols and reasoning about them. We also show that some of the decidability results of the earlier chapters extend to the verification problem for the logic.

6.1 Motivation

In chapter 1, we briefly saw some of the approaches to logical reasoning of security protocols: namely, automated theorem proving and belief logics. We also pointed out some of the strengths and drawbacks of each approach. We take a fresh look at these approaches in the light of the developments and results of the preceding chapters.

We saw in Chapter 2 that modelling security protocols is fairly intricate. The technical results proved in the other chapters also rest on some nontrivial analysis based on the model. In such a situation, an automatic choice for reasoning about protocols is a highly expressive logic like first-order logic or higher-order logic (which are typically used by automated theorem provers). But as was already pointed out, it requires expert knowledge to work with these logics. A further drawback is that the added expressive power usually brings undecidability in its wake, and thus a fully automated approach to protocol verification cannot be based on such a logic.

On the other hand, as we already pointed out, belief logics work with fairly abstract modalities like *knowledge*, *belief*, *awareness*, etc. It is not clear whether these are at the core of reasoning about security protocols. The analysis involved in the proofs of the various technical results that we saw earlier suggest that the explicit information present in the agents' state is crucial to much of the reasoning about protocols. We base our logic on this. Thus ours is an explicit-information based logic in that we focus on the explicit information available in each agent's state at any point of a protocol run, rather than on the epistemic attitudes of the different agents. The crucial security properties also involve a notion of time, so the logic needs some way of referring to the future and past. Here again, we see that temporal modalities like the nexttime and until modalities of LTL, and complex temporal reasoning involving them are not crucial to the analysis of protocols. We thus choose to endow the logic with the simple tense logic modailties F (referring to some time in the future) and P (referring to some time in the past).

[RS01] is an attempt to develop a simple modal logic along these lines. The main feature of the logic is the modality has, which refers to the explicit information available to an agent at a state. For instance, the formula A has m says that the term m is in A's database in the current state. More interestingly, the formula A has (B has m) says that A has explicit information about B having access to m. But the technical treatment in [RS01] is unnecessarily complicated because has is treated as a *modality*, and can thus be iterated. It is also not clear whether iterating the has modality lies at the core of reasoning about security protocols.

The logic which we describe in this chapter follows the information based approach, but does not treat has as a modality. Instead it is a special kind of atomic proposition. Our aim in defining this logic is to come up with a core logic for security protocols with the property that most of the technical results proved in the earlier chapters (about the secrecy problem) generalise to the logic. But at the same time the logic should have enough expressive power such that the basic security properties can be naturally expressed in it. The different choices made in defining the elements of the logic have the above two requirements in mind.

Before we define the logic proper (in the next section), we motivate it by describing a much simpler logic which helps us understand the issues involved. The syntax of the logic has basic propositions of the form A has m and a where $A \in Ag$, $m \in \mathcal{T}_0$ and $a \in Ac$. Further the set of formulas is closed under the usual boolean operators,

the *future modality* F, and the *past modality* P. The formulas are interpreted over instants of runs of a protocol, i.e., (ξ, i) where ξ is a run of a protocol and $0 \leq i \leq |\xi|$. We say that the formula A has m is satisfied at (ξ, i) iff $m \in \overline{(infstate(\xi_i))_A}$ (where ξ_i is the prefix of ξ of length i). (ξ, i) satisfies a iff $act(e_i) = a$ (e_i being the ith event of ξ). The formula $\mathsf{F}\alpha$ is satisfied at (ξ, i) iff α is satisfied at (ξ, j), for some $j \geq i$. Similarly, $\mathsf{P}\alpha$ is satisfied at (ξ, i) iff α is satisfied at (ξ, j), for some $j \leq i$. The dual modalities G and H are defined by: $\mathsf{G}\alpha \stackrel{\text{def}}{=} \neg\mathsf{F}\neg\alpha$ and $\mathsf{H}\alpha \stackrel{\text{def}}{=} \neg\mathsf{P}\neg\alpha$. A protocol Pr satisfies a formula α if $(\xi, 0)$ satisfies α for all runs ξ of Pr. This is basically a tense logic with the past operator and some specialised atomic propositions to talk about security.

Several basic security properties can be specified in this logic. The formula $\neg\mathsf{F}(I\ \mathsf{has}\ m)$ says that the basic term m is never learnt by the intruder in the course of a run. This is a rudimentary form of secrecy. A rudimentary form of authentication is specified by the formula $\mathsf{G}(A?B{:}t \supset \mathsf{P}(B!A{:}t))$. This says that if A receives t purportedly from B at some point of a run, then B actually sent it intended for A at some time in the past. We can even define more complicated forms of authentication in the logic. With respect to the Needham-Schroeder protocol $\mathsf{Pr_{NS}}$ the following formula α says that if some instantiation of the responder role is played, then an appropriate instantiation of the initiator role has also been played to completion.

$$\alpha \stackrel{\text{def}}{=} \mathsf{G}[B?A{:}\{n\}_{pubk_B} \supset \mathsf{P}(A!B{:}\{n\}_{pubk_B} \wedge \mathsf{P}(A?B{:}\{m, n\}_{pubk_A} \wedge \mathsf{P}(A!B{:}(m)\{m\}_{pubk_B})))]$$

This is just representative of the kind of properties that can be specified. Other forms of protocol-specific authentication properties can be specified using the logic. But the main drawback of the logic is that the formulas mention concrete terms actually communicated during a run. This makes the task of specifying abstract security properties in the logic much harder. Further, since there are potentially infinitely many concrete terms, we need a logical device like quantification over terms to express properties about all terms. In the logic that we introduce next, we solve these problems by mentioning only abstract terms mentioned in the protocol specification. Further, instead of a quantification on terms we have a quantification over substitutions. Recall that substitutions are the unknown elements at the level of protocol specifications, since they serve to introduce different terms in the protocol runs. These features enable the proposed logic to naturally specify abstract properties of protocols with reference to the runs of the protocol. Thus our approach

combines some of the advantages of BAN-style logics (ability to specify abstract properties) with some of the advantages of the logic presented above (formulas can be easily and naturally interpreted over runs of a protocol, even though concrete terms not in mentioned in the formula (or the protocol specification) occur in the run).

6.2 A modal logic for security protocols

In this section, we develop a logic keeping the points raised in the above discussion in mind. The logic is designed to specify abstract properties of protocols. Thus the formulas need to talk about terms, actions, etc. but in an abstract way.

Syntax

We assume a countable set $\mathcal{AS}$ of abstract substitution names. For a term $m \in \mathcal{T}_0$, we define $\mathsf{type}(m)$ to be nonce if $m \in N$, sequence-number if $m \in SN$, key if $m \in K$ and agent if $m \in Ag$.

The set of formulas Φ is given by:

$$
\begin{aligned}
\Phi ::= \quad & \iota{\cdot}A \text{ has } \iota'{\cdot}m \qquad (A \in Ag, m \in \mathcal{T}_0, \iota, \iota' \in \mathcal{AS}) \\
| \ & \iota{\cdot}a \qquad (a \in Ac, \iota \in \mathcal{AS}) \\
| \ & \iota{\cdot}x = \iota'{\cdot}x' \qquad (x, x' \in \mathcal{T}_0, \mathsf{type}(x) = \mathsf{type}(x'), \iota, \iota' \in \mathcal{AS}) \\
| \ & \neg\alpha \\
| \ & \alpha \vee \beta \\
| \ & \mathsf{F}\alpha \\
| \ & \mathsf{P}\alpha \\
| \ & (\exists\iota)\alpha
\end{aligned}
$$

We introduce the other standard operators as follows: $\alpha \wedge \beta \stackrel{\text{def}}{=} \neg(\neg\alpha \vee \neg\beta)$, $\alpha \supset \beta \stackrel{\text{def}}{=} \neg\alpha \vee \beta$, $\alpha \equiv \beta \stackrel{\text{def}}{=} (\alpha \supset \beta) \wedge (\beta \supset \alpha)$, $\mathsf{G}\alpha \stackrel{\text{def}}{=} \neg\mathsf{F}\neg\alpha$, $\mathsf{H}\alpha \stackrel{\text{def}}{=} \neg\mathsf{P}\neg\alpha$, $(\forall\iota)\alpha \stackrel{\text{def}}{=} \neg(\exists\iota)\neg\alpha$.

The set of subformulas, the set of free substitution names, and the set of "subterms" of a formula are all easily defined:

- $SF(\iota{\cdot}A \text{ has } \iota'{\cdot}m) = \{\iota{\cdot}A \text{ has } \iota'{\cdot}m\}$;
  $FSN(\iota{\cdot}A \text{ has } \iota'{\cdot}m) = \{\iota, \iota'\}$;

$ST(\iota{\cdot}A \text{ has } \iota'{\cdot}m) = \{\iota{\cdot}A, \iota'{\cdot}m\}$;

- $SF(\iota{\cdot}a) = \{\iota{\cdot}a\}$;
  $FSN(\iota{\cdot}a) = \{\iota\}$;
  $ST(\iota{\cdot}a) = \{\iota{\cdot}m \mid m \in ST(a) \cap \mathcal{T}_0\}$;

- $SF(\iota{\cdot}x = \iota'{\cdot}x') = \{\iota{\cdot}x = \iota'{\cdot}x'\}$;
  $FSN(\iota{\cdot}x = \iota'{\cdot}x') = \{\iota, \iota'\}$;
  $ST(\iota{\cdot}x = \iota'{\cdot}x') = \{\iota{\cdot}x, \iota{\cdot}x'\}$;

- $SF(\neg\alpha) = \{\neg\alpha\} \cup SF(\alpha)$;
  $FSN(\neg\alpha) = FSN(\alpha)$;
  $ST(\neg\alpha) = ST(\alpha)$;

- $SF(\alpha \vee \beta) = \{\alpha \vee \beta\} \cup SF(\alpha) \cup SF(\beta)$;
  $FSN(\alpha \vee \beta) = FSN(\alpha) \cup FSN(\beta)$;
  $ST(\alpha \vee \beta) = ST(\alpha) \cup ST(\beta)$;

- $SF(\mathsf{F}\alpha) = \{\mathsf{F}\alpha\} \cup SF(\alpha)$;
  $FSN(\mathsf{F}\alpha) = FSN(\alpha)$;
  $ST(\mathsf{F}\alpha) = ST(\alpha)$;

- $SF(\mathsf{P}\alpha) = \{\mathsf{P}\alpha\} \cup SF(\alpha)$;
  $FSN(\mathsf{P}\alpha) = FSN(\alpha)$;
  $ST(\mathsf{P}\alpha) = ST(\alpha)$;

- $SF((\exists\iota)\alpha) = \{(\exists\iota)\alpha\} \cup SF(\alpha)$;
  $FSN((\exists\iota)\alpha) = FSN(\alpha) \setminus \{\iota\}$;
  $ST((\exists\iota)\alpha) = ST(\alpha)$.

A formula α is said to be *closed* iff $FSN(\alpha) = \emptyset$.

Semantics

A *structure* is a pair $\mathcal{A} = (\mathsf{Pr}, \mathbb{S})$ where Pr is a protocol and $\mathbb{S}$ is a set of substitutions suitable for Pr. (Note that $\mathbb{S}$ need not necessarily be the set of *all* substitutions σ suitable for Pr.) An $\mathcal{A}$-*run* ξ is a run of Pr such that for all $(\eta, \sigma, lp) \in Events(\xi)$, $\sigma \in \mathbb{S}$. An $\mathcal{A}$-*assignment* θ is a map which associates each substitution name ι in $\mathcal{A}\mathbb{S}$ to a substitution $\theta_\iota \in \mathbb{S}$. (Note that for ease of notation we write θ_ι rather than

$\theta(\iota)$.) Given a structure $\mathcal{A} = (\mathsf{Pr}, \mathcal{S})$, an $\mathcal{A}$-assignment θ and a substitution $\sigma \in \mathcal{S}$ we define $\theta[\iota := \sigma]$ to be the assignment θ' with the property that $\theta'_\iota = \sigma$ and $\theta'_{\iota'} = \theta_{\iota'}$ for $\iota' \neq \iota$.

A *model* is a pair $\mathcal{M} = (\mathcal{A}, \theta)$ where $\mathcal{A}$ is a structure and θ is an $\mathcal{A}$-assignment. We say that ξ is an $\mathcal{M}$-*run* if it is an $\mathcal{A}$-run. A model $\mathcal{M} = (\mathcal{A}, \theta)$ is said to be *compatible with a formula* α iff for all $\iota{\cdot}m \in ST(\alpha)$, $\theta_\iota(m)$ is defined and $\mathsf{type}(\theta_\iota(m)) = \mathsf{type}(m)$.

Given a sequence of events ξ, an *instant* in ξ is a number i such that $0 \leq i \leq |\xi|$.

Given a formula α, a model $\mathcal{M} = ((\mathsf{Pr}, \mathcal{S}), \theta)$ compatible with α, an $\mathcal{M}$-run ξ and an instant i in ξ, we define the satisfaction relation $\mathcal{M}, (\xi, i) \models \alpha$. Suppose that $\xi = e_1 \cdots e_k$, where for each $i \leq k$, $e_i = (\eta_i, \sigma_i, lp_i)$. Let s_i denote $infstate(e_1 \cdots e_i)$, for any $i \leq k$. We now give the inductive definition of $\mathcal{M}, (\xi, i) \models \alpha$.

- $\mathcal{M}, (\xi, i) \models \iota{\cdot}A$ has $\iota'{\cdot}m$ iff $n \in \overline{(s_i)_C}$ (where $\theta_{\iota'}(m) = n$ and $\theta_\iota(A) = C$);
- $\mathcal{M}, (\xi, i) \models \iota{\cdot}a$ iff $i > 0$, $\eta_i(lp_i) = a$ and $\theta_\iota(a) = \sigma_i(a)$;
- $\mathcal{M}, (\xi, i) \models \iota{\cdot}x = \iota'{\cdot}x'$ iff $\theta_\iota(x) = \theta_{\iota'}(x')$;
- $\mathcal{M}, (\xi, i) \models \neg\alpha$ iff $\mathcal{M}, (\xi, i) \not\models \alpha$;
- $\mathcal{M}, (\xi, i) \models \alpha \vee \beta$ iff $\mathcal{M}, (\xi, i) \models \alpha$ or $\mathcal{M}, (\xi, i) \models \beta$;
- $\mathcal{M}, (\xi, i) \models \mathsf{F}\alpha$ iff there exists $j \geq i$ such that $\mathcal{M}, (\xi, j) \models \alpha$;
- $\mathcal{M}, (\xi, i) \models \mathsf{P}\alpha$ iff there exists $j \leq i$ such that $\mathcal{M}, (\xi, j) \models \alpha$;
- $\mathcal{M}, (\xi, i) \models (\exists\iota)\alpha$ iff $\mathcal{M}', (\xi, i) \models \alpha$, where $\mathcal{M}' = (\mathcal{A}, \theta[\iota := \sigma])$ for some substitution $\sigma \in \mathcal{S}$ and $\mathcal{M}'$ is compatible with α.

A formula α is *satisfiable* iff there exists a model $\mathcal{M}$ compatible with α, an $\mathcal{M}$-run ξ, and an instant i in ξ such that $\mathcal{M}, (\xi, i) \models \alpha$. A formula α is *valid* iff $\mathcal{M}, (\xi, i) \models \alpha$ for all models $\mathcal{M}$ compatible with α, all $\mathcal{M}$-runs ξ, and all instants i in ξ.

Note that a formula α is valid iff $\neg\alpha$ is not satisfiable.

The interesting validities involve interaction of the quantifiers and modalities. Note that $(\forall\iota)\mathsf{G}\alpha \equiv \mathsf{G}(\forall\iota)\alpha$ and $(\exists\iota)\mathsf{F}\alpha \equiv \mathsf{F}(\exists\iota)\alpha$ are validities. Similarly for the past modalities. On the other hand note that $(\exists\iota)\mathsf{G}\alpha \supset \mathsf{G}(\exists\iota)\alpha$ and $\mathsf{F}(\forall\iota)\alpha \supset (\forall\iota)\mathsf{F}\alpha$ are validities, but the implications do not hold the other way. A similar statement can be made about the past modalities. This behaviour is typical of the interaction of the quantifiers and the modalities.

Note that even though the logic has both quantifiers and modalities, the semantics is more restricted than that of first-order modal logic. The typical feature of first-order modal logic is that the possible worlds are different first-order structures (even under the so-called *constant-domain semantics*, the different worlds only share the domain while the interpretations of the relations and constants usually vary). In our framework, a single *structure* remains constant across many worlds. In this respect, the logic presented here can be thought of as a kind of quantified propositional logic with modalities. The quantification over substitutions can be considered as a special form of quantification over propositions.

For a structure $\mathcal{A} = (\mathsf{Pr}, \mathcal{S})$ and a formula α, we say that $\mathcal{A} \models \alpha$ iff $\mathcal{M}, (\xi, 0) \models \alpha$ for all $\mathcal{A}$-assignments θ such that $\mathcal{M} = (\mathcal{A}, \theta)$ is compatible with α, and all $\mathcal{A}$-runs ξ. Suppose α is a formula, $\mathcal{A}$ is a structure, and $\mathcal{M} = (\mathcal{A}, \theta)$ and $\mathcal{M}' = (\mathcal{A}, \theta')$ are two models compatible with α such that for all $\iota \in FSN(\alpha)$, $\theta_\iota = \theta'_\iota$. Then $\mathcal{M}, (\xi, i) \models \alpha$ iff $\mathcal{M}', (\xi, i) \models \alpha$ for all $\mathcal{M}$-runs ξ and all instants i in ξ. It follows from this that given a structure $\mathcal{A}$ and a formula α, to check whether $\mathcal{A} \models \alpha$, it suffices to consider $\mathcal{A}$-assignments restricted to $FSN(\alpha)$.

We now define several notions of validity with respect to a fixed protocol Pr.

We say that $\mathsf{Pr} \models \alpha$ iff $(\mathsf{Pr}, \mathcal{S}_{\mathsf{Pr}}) \models \alpha$, where $\mathcal{S}_{\mathsf{Pr}}$ is the set of *all* substitutions σ suitable for Pr.

We say that $\mathsf{Pr} \models_{wt} \alpha$ iff $(\mathsf{Pr}, \mathcal{S}_{\mathsf{Pr},wt}) \models \alpha$, where $\mathcal{S}_{\mathsf{Pr},wt}$ is the set of all *well-typed* substitutions σ suitable for Pr.

For a fixed set $T \subseteq \mathcal{T}_0$, we say that $\mathsf{Pr} \models^T \alpha$ iff $(\mathsf{Pr}, \mathcal{S}_{\mathsf{Pr},T}) \models \alpha$, where $\mathcal{S}_{\mathsf{Pr},T}$ is the set of all T-substitutions suitable for Pr.

We say that $\mathsf{Pr} \models_{wt}^T \alpha$ iff $(\mathsf{Pr}, \mathcal{S}_{\mathsf{Pr},wt,T}) \models \alpha$, where $\mathcal{S}_{\mathsf{Pr},wt,T}$ is the set of all well-typed T-substitutions suitable for Pr.

A feature of the semantics that needs a little discussion is that the satisfaction relation $\mathcal{M}, (\xi, i) \models \alpha$ is defined only if $\mathcal{M}$ is compatible with α. Recall that the core logic that we presented in Section 6.1 works with formulas of the form A has m, where $m \in \mathcal{T}_0$. The logic we are working with is supposed to be an abstraction of the core logic. Consider a formula of the form $\iota \cdot A$ has $\iota' \cdot m$. If we interpret this formula on some model $(\mathcal{A}, \theta)$ such that $\theta_{\iota'}(m) \notin \mathcal{T}_0$, then we would be indirectly referring to a nonatomic term t using our formula. The definition of $\mathcal{M}$ being compatible with α disallows such an indirect reference to nonatomic terms.

Note that the logic has both quantification over substitution names and equality.

As the examples in the next section show, a combination of these two features of the logic is crucially used in specifying properties of and reasoning about protocols. The logic would not be as effective even if one of the two features were not present. In the absence of the equality operator, there would be no means of relating substitution names with one another. In the absence of quantification, the logic would not have the ability to refer to all the substitutions of the model (there might possibly be infinitely many of them). For instance, a typical authentication requirement would be that for any instantiation of a responder role occurring in a run with A as the purported initiator and B as the responder, there is an instantiation of the initiator role in the same run with A as the initiator and B as the intended responder. Note the crucial use of the of quantifiers (for every responder role, there is an initiator role) and of equality (which constrain the initiator role to correspond to the responder role).

6.3 Examples

Let us look at some examples which illustrate the use of the logic. Without loss of generality we assume that for all models $\mathcal{M} = (\mathcal{A}, \theta)$ compatible with a formula α, $\theta_\iota(I) = I$ for all $\iota{\cdot}I \in ST(\alpha)$. This means that we can use the name I in formulas without prefixing it with any substitution name.

6.3.1 The Needham-Schroeder protocol

We look at the Needham-Schroeder protocol in detail now, stating several of its properties in our logic, demonstrating that some of them are true in all runs of the protocol, and also showing that some crucial properties fail.

The protocol is given by (C, δ) where $\mathsf{C} = \emptyset$ and δ is the following sequence of communications.

$$\begin{array}{llllll} 1. & A & \rightarrow & B & : (x) & \{A, x\}_{pubk_B} \\ 2. & B & \rightarrow & A & : (y) & \{x, y\}_{pubk_A} \\ 3. & A & \rightarrow & B & : & \{y\}_{pubk_B} \end{array}$$

There are two roles in this protocol. The *initiator role* η_1 is given below:

$$\begin{array}{llllll} 1. & A & ! & B & : (x) & \{A, x\}_{pubk_B} \\ 2. & A & ? & B & : & \{x, y\}_{pubk_A} \\ 3. & A & ! & B & : & \{y\}_{pubk_B} \end{array}$$

The *responder role* η_2 is given below:

$$\begin{array}{llllll} 1. & B & ? & A & : & \{A, x\}_{pubk_B} \\ 2. & B & ! & A & : (y) & \{x, y\}_{pubk_A} \\ 3. & B & ? & A & : & \{y\}_{pubk_B} \end{array}$$

We will use the notation a_i to denote $\eta_1(i)$ and b_i to denote $\eta_2(i)$, for $1 \leq i \leq 3$.

The following is an immediate and trivial validity for this protocol, which just says that any event in a run is preceded by its local past.

$$(\forall \iota)\mathsf{G}[\bigwedge_{i=2,3}((\iota{\cdot}a_i \supset \mathsf{P}(\iota{\cdot}a_{i-1})) \wedge (\iota{\cdot}b_i \supset \mathsf{P}(\iota{\cdot}b_{i-1})))].$$

Example specifications

One of the most immediate properties that we desire of this protocol is that of *secrecy*. There are two desirable secrecy requirements in this case. *Secrecy for the initiator* says that all fresh nonces that are instantiated for x and not intended for the intruder are not leaked to the intruder. It is expressed by the following formula:

$$secrecy_{init} \stackrel{\text{def}}{=} (\forall \iota)\mathsf{G}[(\iota{\cdot}a_1 \wedge \neg(\iota{\cdot}B = I)) \supset \mathsf{G}\neg I \text{ has } \iota{\cdot}x].$$

Secrecy for the responder says that all fresh nonces that are instantiated for y and are not intended for the intruder are not leaked to the intruder. It is expressed by the following formula:

$$secrecy_{resp} \stackrel{\text{def}}{=} (\forall \iota)\mathsf{G}[(\iota{\cdot}b_2 \wedge \neg(\iota{\cdot}A = I)) \supset \mathsf{G}\neg I \text{ has } \iota{\cdot}y].$$

Authentication for the initiator says that for every play of the initiator role (with an apparently honest responder) in a run of the protocol, there is a corresponding play of the responder role in that run.

$$\begin{aligned} auth_{init} \stackrel{\text{def}}{=} (\forall \iota)\mathsf{G}[(\iota{\cdot}a_2 \wedge \neg(\iota{\cdot}B = I)) \supset (\exists \iota')[\iota{\cdot}x = \iota'{\cdot}x \wedge \iota{\cdot}y = \iota'{\cdot}y \wedge \\ \iota{\cdot}A = \iota'{\cdot}A \wedge \iota{\cdot}B = \iota'{\cdot}B \wedge \mathsf{P}(\iota'{\cdot}b_2)]]. \end{aligned}$$

Authentication for the responder says that for every play of the responder role (with an apparently honest initiator) in a run of the protocol, there is a corresponding play of the initiator role in that run.

$$auth_{resp} \stackrel{\text{def}}{=} (\forall \iota)\mathsf{G}[(\iota{\cdot}b_3 \wedge \neg(\iota{\cdot}A = I)) \supset (\exists \iota')[\iota{\cdot}x = \iota'{\cdot}x \wedge \iota{\cdot}y = \iota'{\cdot}y \wedge \iota{\cdot}A = \iota'{\cdot}A \wedge \iota{\cdot}B = \iota'{\cdot}B \wedge \mathsf{P}(\iota'{\cdot}a_3)]].$$

The notable feature of the formulas is that they are quite simple and intuitive to write, not requiring us to name any actual terms that are substituted.

Lowe's attack

Of the above properties, secrecy for the responder is not guaranteed by the protocol, i.e., $\mathsf{Pr}_{\mathsf{NS}} \not\models secrecy_{resp}$. This can be evidenced by the following run ξ. In the following, σ_1 is a substitution such that $\sigma_1(A) = A$, $\sigma_1(B) = I$, $\sigma_1(x) = m$, and $\sigma_1(y) = n$; and σ_2 is a substitution such that $\sigma_2(A) = A$, $\sigma_2(B) = B$, $\sigma_2(x) = m$, and $\sigma_2(y) = n$.

$$\begin{array}{llllllll}
(\eta_1, \sigma_1, 1) & A & ! & I & : & (m) & \{A, m\}_{pubk_I} \\
(\eta_2, \sigma_2, 1) & B & ? & A & : & & \{A, m\}_{pubk_B} \\
(\eta_2, \sigma_2, 2) & B & ! & A & : & (n) & \{m, n\}_{pubk_A} \\
(\eta_1, \sigma_1, 2) & A & ? & I & : & & \{m, n\}_{pubk_A} \\
(\eta_1, \sigma_1, 3) & A & ! & I & : & & \{n\}_{pubk_I} \\
(\eta_2, \sigma_2, 3) & B & ? & A & : & & \{n\}_{pubk_B}
\end{array}$$

Suppose $\mathcal{A} = (\mathsf{Pr}, \mathcal{S}_{\mathsf{Pr}})$ and θ is an $\mathcal{A}$-assignment such that $\theta_\iota = \sigma_2$. Suppose $\mathcal{M} = (\mathcal{A}, \theta)$. Then it is clear that $\mathcal{M}, (\xi, 3) \models \iota{\cdot}b_2 \wedge \neg(\iota{\cdot}A = I)$. But on the other hand it can be easily seen that $\mathcal{M}, (\xi, 5) \models I \mathsf{\ has\ } \iota{\cdot}y$. This is easy to see since $n \in \overline{s_I}$, where s is the information state at the end of the first five events of ξ. From these two facts it follows that $\mathcal{M}, (\xi, 0) \not\models secrecy_{resp}$ and hence that $\mathsf{Pr}_{\mathsf{NS}} \not\models secrecy_{resp}$ as well. In fact, this also shows that $\mathsf{Pr}_{\mathsf{NS}} \not\models_{wt} secrecy_{resp}$. This is the famous Lowe's attack on the Needham-Schroeder protocol.

The above attack also shows that $\mathsf{Pr}_{\mathsf{NS}} \not\models_{wt} auth_{resp}$. It is clear that $\mathcal{M}, (\xi, 6) \models \iota{\cdot}b_3 \wedge \neg(\iota{\cdot}A = I)$. But it is also true that $\mathcal{M}, (\xi, 0) \models (\forall \iota')\mathsf{G}[\iota'{\cdot}a_3 \supset \iota'{\cdot}B \neq \iota{\cdot}B]$. This shows that $\mathcal{M}, (\xi, 0) \not\models auth_{resp}$ and hence that $\mathsf{Pr}_{\mathsf{NS}} \not\models_{wt} auth_{resp}$.

Secrecy for the initiator

Even though $\mathsf{Pr}_{\mathsf{NS}} \not\models_{wt} secrecy_{resp}$, it can be argued that $\mathsf{Pr}_{\mathsf{NS}} \models_{wt} secrecy_{init}$. The reasoning is as follows: We assume that $\mathsf{Pr}_{\mathsf{NS}} \not\models_{wt} secrecy_{init}$ and arrive at a contradiction. The assumption means that $\mathcal{M}, (\xi, 0) \not\models secrecy_{init}$ for some $\mathcal{M} =$

$((\mathsf{Pr}, \mathcal{S}_{\mathsf{Pr}_{\mathsf{NS}},wt}), \theta)$ compatible with $secrecy_{init}$, and some well-typed run $\xi = e_1 \cdots e_k$ of $\mathsf{Pr}_{\mathsf{NS}}$. Let s_i denote $infstate(e_1 \cdots e_i)$ for $i \leq k$. Also let $e_i = (\zeta_i, \sigma_i, lp_i)$, for $i \leq k$.

1. We are given that $\mathcal{M}, (\xi, 0) \not\models secrecy_{init}$. This means that there exist $i \geq 0$ and $\iota \in \mathcal{AS}$ such that $\mathcal{M}, (\xi, i) \models \iota{\cdot}a_1 \wedge \neg(\iota{\cdot}B = I)$ and $\mathcal{M}, (\xi, i) \models \mathsf{F}(I \ \mathsf{has} \ \iota{\cdot}x)$.

2. Since $\mathcal{M}, (\xi, i) \models \iota{\cdot}a_1$, it follows that $\zeta_i(lp_i) = a_1$ and $\sigma_i(a_1) = \theta_\iota(a_1)$.

3. Since $x \in NT(\eta_1(1))$, it is clear that $\theta_\iota(x) \in NT(e_i)$, and hence it follows from the unique origination property of runs that $\mathcal{M}, (\xi, i') \models \neg(I \ \mathsf{has} \ \iota{\cdot}x)$ for all $i' < i$. Since only $\{\theta_\iota(A), \theta_\iota(x)\}_{pubk_{\theta_\iota(B)}}$ is added to the intruder's state by e_i, and since $\theta_\iota(B) \neq I$, it follows that $\mathcal{M}, (\xi, i) \models \neg(I \ \mathsf{has} \ \iota{\cdot}x)$ as well.

4. Since $\mathcal{M}, (\xi, i) \models \mathsf{F}(I \ \mathsf{has} \ \iota{\cdot}x)$, there is a least $j \geq i$ such that $\mathcal{M}, (\xi, j) \models I \ \mathsf{has} \ \iota{\cdot}x$. Clearly $j > i$ and $\mathcal{M}, (\xi, j') \models \neg(I \ \mathsf{has} \ \iota{\cdot}x)$ for all $j' < j$.

5. Since there is a change in the intruder's state at the jth instant, it must be the case that e_j is a send event. A further perusal of the protocol specification tells us that e_j can only take one of the following forms:

   (a) $(\eta_1, \sigma, 1)$ with $\sigma(x) = \theta_\iota(x)$ and $\sigma(B) = I$.

   This means that $\theta_\iota(x) \in NT(e_j)$ but that cannot happen because of the property of unique origination. Hence this case cannot arise at all.

   (b) $(\eta_1, \sigma, 3)$ with $\sigma(y) = \theta_\iota(x)$ and $\sigma(B) = I$.

   In this case it is clear that there exists $\ell < j$ such that $e_\ell = (\eta_1, \sigma, 2)$. Suppose $\sigma(x) = n$ and $\sigma(y) = m$. Then $term(e_\ell) = \{n, m\}_{pubk_{\sigma(A)}}$. Since e_ℓ is a receive event, $\{n, m\}_{pubk_{\sigma(A)}} \in \overline{(s_{\ell-1})_I}$. It should be noted that $m \in NT(e_i)$ and $term(e_i) = \{\theta_\iota(A), m\}_{pubk_{\theta_\iota(B)}}$, and therefore by the unique origination property of ξ, it is not possible that there is a send event e with $term(e) = \{n, m\}_{pubk_{\sigma(A)}}$ (since $m \in NT(e)$ would hold in that case). Thus $\{n, m\}_{pubk_{\sigma(A)}} \notin \mathsf{analz}((s_{\ell-1})_I)$, in particular. But this term belongs to $\overline{(s_{\ell-1})_I}$, and hence it follows that $m \in \overline{(s_{\ell-1})_I}$. But then $\mathcal{M}, (\xi, \ell - 1) \models I \ \mathsf{has} \ \iota{\cdot}x$. Since $\ell - 1 < j$, this is a contradiction to the fact that j is the *least* instant in ξ such that $\mathcal{M}, (\xi, j) \models I \ \mathsf{has} \ \iota{\cdot}x$. Therefore this case is also not possible.

   (c) $(\eta_2, \sigma, 2)$ with $(\sigma(y) = \theta_\iota(x)$ or $\sigma(x) = \theta_\iota(x))$ and $\sigma(A) = I$.

If $\sigma(y) = \theta_\iota(x)$ then it means that $\theta_\iota(x) \in NT(e_j)$ but that cannot happen because of the property of unique origination. Hence it has to be the case that $\sigma(x) = \theta_\iota(x)$.

In this case it is clear that there exists $\ell < j$ such that $e_\ell = (\eta_2, \sigma, 1)$. Suppose $\sigma(x) = m$. Then $term(e_\ell) = \{I, m\}_{pubk_{\sigma(B)}}$. Since e_ℓ is a receive event, $\{I, m\}_{pubk_{\sigma(B)}} \in \overline{(s_{\ell-1})_I}$. It should be noted that $m \in NT(e_i)$ and $term(e_i) = \{\theta_\iota(A), m\}_{pubk_{\theta_\iota(B)}}$ with $\theta_\iota(A) \in Ho$, and therefore by the unique origination property of ξ, it is not possible that there is a send event e with $term(e) = \{I, m\}_{pubk_{\sigma(B)}}$. Thus $\{I, m\}_{pubk_{\sigma(B)}} \notin \mathsf{analz}((s_{\ell-1})_I)$, in particular. But this term belongs to $\overline{(s_{\ell-1})_I}$, and hence it follows that $m \in \overline{(s_{\ell-1})_I}$. But then $\mathcal{M}, (\xi, \ell - 1) \models I \ \mathsf{has}\ \iota{\cdot}x$. Since $\ell - 1 < j$, this is a contradiction to the fact that j is the *least* instant in ξ such that $\mathcal{M}, (\xi, j) \models I \ \mathsf{has}\ \iota{\cdot}x$. Therefore this case is also not possible.

This concludes the proof that $\mathsf{Pr_{NS}} \models_{wt} secrecy_{init}$.

Secrecy for the responder

Even though $\mathsf{Pr_{NS}} \not\models_{wt} secrecy_{resp}$, it can be shown that the following slightly weaker guarantee holds for the responder:

$$secrecy'_{resp} \stackrel{\text{def}}{=} (\forall\iota)[(\forall\iota')\neg(\iota{\cdot}y = \iota'{\cdot}y \ \wedge\ \iota'{\cdot}B = I \ \wedge\ \mathsf{F}(\iota'{\cdot}a_1)) \ \supset \mathsf{G}[(\iota{\cdot}b_2 \wedge \neg(\iota{\cdot}A = I)) \ \supset\ \mathsf{G}\neg I \ \mathsf{has}\ \iota{\cdot}y]].$$

The proof is as before. We assume that $\mathsf{Pr_{NS}} \not\models_{wt} secrecy'_{resp}$ and arrive at a contradiction. The assumption means that $\mathcal{M}, (\xi, 0) \not\models secrecy'_{resp}$ for some $\mathcal{M} = ((\mathsf{Pr}, \mathcal{S}_{\mathsf{Pr_{NS}}, wt}), \theta)$ compatible with $secrecy'_{resp}$, and some well-typed run $\xi = e_1 \cdots e_k$ of $\mathsf{Pr_{NS}}$. Let s_i denote $infstate(e_1 \cdots e_i)$ for $i \leq k$. Also let $e_i = (\zeta_i, \sigma_i, lp_i)$, for $i \leq k$.

Reasoning along the lines of items 1 to 4 in the previous proof, we can show that there exists $\iota \in \mathcal{AS}$ such that $\mathcal{M}, (\xi, 0) \models (\forall\iota')\neg(\iota{\cdot}y = \iota'{\cdot}y \ \wedge\ \iota'{\cdot}B = I \ \wedge\ \mathsf{F}(\iota'{\cdot}a_1))$, $i \geq 0$ such that $\mathcal{M}, (\xi, i) \models \iota{\cdot}b_2 \wedge \neg(\iota{\cdot}A = I)$ and $\mathcal{M}, (\xi, i) \models \mathsf{F}(I \ \mathsf{has}\ \iota{\cdot}y)$, and $j > i$ such that $\mathcal{M}, (\xi, j) \models I \ \mathsf{has}\ \iota{\cdot}y$ and $\mathcal{M}, (\xi, j') \models \neg(I \ \mathsf{has}\ \iota{\cdot}y)$ for all $j' < j$.

Reasoning along the lines of item 5, we see that e_j can only be one of the following forms:

(a) $(\eta_1, \sigma, 1)$ with $\sigma(x) = \theta_\iota(y)$ and $\sigma(B) = I$.

It can be shown that this case cannot arise, reasoning along the lines of item 5(a) of the previous proof.

(b) $(\eta_1, \sigma, 3)$ with $\sigma(y) = \theta_\iota(y)$ and $\sigma(B) = I$.

In this case it is clear that there exists $\ell < j$ such that $e_\ell = (\eta_1, \sigma, 1)$. Thus $\mathcal{M}, (\xi, 0) \models (\exists \iota')(\iota \cdot y = \iota' \cdot y \ \wedge \ \iota \cdot B = I \ \wedge \ \mathsf{F}(\iota' \cdot a_1))$, which is a contradiction to our assumption. Therefore this case cannot arise. Note that this case is actually the problem with Lowe's attack. If it is possible for honest agents to initiate sessions with the intruder (this is not an improbable situation), then Lowe's attack exists. If we rule out this possibility (which is what the extra assumptions in $secrecy'_{resp}$ do), then Lowe's attack does not exist any more.

(c) $(\eta_2, \sigma, 2)$ with $(\sigma(y) = \theta_\iota(y)$ or $\sigma(x) = \theta_\iota(y))$ and $\sigma(A) = I$.

It can be shown that this case cannot arise as well, reasoning along the lines of item 5(c) of the previous proof.

Authentication for the initiator

We now show that $\mathsf{Pr}_{\mathsf{NS}} \models_{wt} auth_{init}$. Consider some well-typed run $\xi = e_1 \cdots e_k$ of $\mathsf{Pr}_{\mathsf{NS}}$. Let s_i denote $infstate(e_1 \cdots e_i)$, for $i \leq k$. Also let $e_i = (\zeta_i, \sigma_i, lp_i)$ for $i \leq k$. Consider a model $\mathcal{M} = ((\mathsf{Pr}_{\mathsf{NS}}, \mathcal{S}_{\mathsf{Pr}_{\mathsf{NS}}, wt}), \theta)$ compatible with $auth_{init}$. We prove below that $\mathcal{M}, (\xi, 0) \models auth_{init}$.

1. Suppose now that there exists $\iota \in \mathcal{AS}$ and $i \geq 0$ such that $\mathcal{M}, (\xi, i) \models \iota \cdot a_2 \wedge \neg(\iota \cdot B = I)$.

2. It easily follows that there exists an $i' < i$ such that $\mathcal{M}, (\xi, i') \models \iota \cdot a_1$. Using the fact that $\mathsf{Pr}_{\mathsf{NS}} \models_{wt} secrecy_{init}$, we can conclude that $\mathcal{M}, (\xi, i') \models \mathsf{G}\neg(I \ \mathsf{has} \ \iota \cdot x)$.

3. Since $x \in NT(\eta_1(1))$, it follows from the unique origination property of runs that $\mathcal{M}, (\xi, i'') \models \neg(I \ \mathsf{has} \ \iota \cdot x)$ for all $i'' < i'$. Thus we can conclude that $\mathcal{M}, (\xi, 0) \models \mathsf{G}\neg(I \ \mathsf{has} \ \iota \cdot x)$.

4. Let $\theta_\iota(A) = C$, $\theta_\iota(x) = m$ and $\theta_\iota(y) = n$. Then $term(e_i) = \{m, n\}_{pubk_C}$. Clearly $\{m, n\}_{pubk_C} \in \overline{(s_{i-1})_I}$. But since $m \notin \overline{(s_{i-1})_I}$, it has to be the case that there is some send event e_j $(j < i)$ with $term(e_j) = term(e_i)$. But then e_j is of the form $(\eta_2, \sigma, 2)$ with $\sigma(A) = \theta_\iota(A)$, $\sigma(x) = \theta_\iota(x)$ and $\sigma(y) = \theta_\iota(y)$. Our proof would be complete if we showed that $\sigma(B) = \theta_\iota(B)$. Suppose $\sigma(B) = D$. It is clear that there exists $\ell < j$ such that $e_\ell = (\eta_2, \sigma, 1)$. Here again $term(e_\ell) = \{C, m\}_{pubk_D}$. This term belongs to $\overline{(s_{\ell-1})_I}$, but since

$m \notin \overline{(s_{\ell-1})_I}$ it follows that there is a send event $e_{\ell'}$ with $term(e_{\ell'}) = term(e_\ell)$. Then it would be the case that $m \in NT(e_{\ell'})$, and by the unique origination property of ξ, it follows that $i' = \ell'$. From this it follows that $\sigma(B) = \theta_\iota(B)$, and we are through.

6.3.2 The Needham-Schroeder-Lowe protocol

This is a slight modification of the Needham-Schroeder protocol, with a correction proposed by Gavin Lowe. The change in this protocol is that the responder's identity is included in the message sent by the responder.

The protocol is given by $\mathsf{Pr}_{\mathsf{NSL}} = (\mathsf{C}, \delta)$ where $\mathsf{C} = \emptyset$ and δ is the following sequence of communications.

$$\begin{array}{llllll} 1. & A & \rightarrow & B & : (x) & \{A, x\}_{pubk_B} \\ 2. & B & \rightarrow & A & : (y) & \{B, x, y\}_{pubk_A} \\ 3. & A & \rightarrow & B & : & \{y\}_{pubk_B} \end{array}$$

There are two roles in this protocol. The *initiator role* η_1 is given below:

$$\begin{array}{llllll} 1. & A & ! & B & : (x) & \{A, x\}_{pubk_B} \\ 2. & A & ? & B & : & \{B, x, y\}_{pubk_A} \\ 3. & A & ! & B & : & \{y\}_{pubk_B} \end{array}$$

The *responder role* η_2 is given below:

$$\begin{array}{llllll} 1. & B & ? & A & : & \{A, x\}_{pubk_B} \\ 2. & B & ! & A & : (y) & \{B, x, y\}_{pubk_A} \\ 3. & B & ? & A & : & \{y\}_{pubk_B} \end{array}$$

As before, we will use the notation a_i to denote $\eta_1(i)$ and b_i to denote $\eta_2(i)$, for $1 \leq i \leq 3$.

Secrecy for the initiator and responder, and authentication for the initiator and responder, are given by the four formulas $secrecy_{init}$, $secrecy_{resp}$, $auth_{init}$ and $auth_{resp}$ respectively. These formulas have the same definitions as earlier, except for the change in the actions a_2 and b_2. It can be seen that the attack which leads to the violation of $secrecy_{resp}$ and $auth_{resp}$ does not work anymore, with the addition of the responder's name in the action b_2, but we have to still prove that no other attacks are possible.

One can prove that $\mathsf{Pr}_{\mathsf{NSL}} \models_{wt} secrecy_{init}$ and $\mathsf{Pr}_{\mathsf{NSL}} \models_{wt} auth_{init}$ in exactly the same manner as before. The nice thing is that $\mathsf{Pr}_{\mathsf{NSL}} \models_{wt} secrecy_{resp}$ also holds now. The proof is exactly along the lines of the proof of secrecy for the initiator in the Needham-Schroeder protocol.

Authentication for responder

We now show that $\mathsf{Pr}_{\mathsf{NSL}} \models_{wt} auth_{resp}$ as well. Consider some well-typed run $\xi = e_1 \cdots e_k$ of $\mathsf{Pr}_{\mathsf{NSL}}$. Let s_i denote $infstate(e_1 \cdots e_i)$, for $i \leq k$. Also let $e_i = (\zeta_i, \sigma_i, lp_i)$ for $i \leq k$. Consider a model $\mathcal{M} = ((\mathsf{Pr}_{\mathsf{NSL}}, \mathcal{S}_{\mathsf{Pr}_{\mathsf{NSL}},wt}), \theta)$ compatible with $auth_{resp}$. We prove below that $\mathcal{M}, (\xi, 0) \models auth_{resp}$.

1. Suppose now that there exists $\iota \in \mathcal{AS}$ and $i \geq 0$ such that $\mathcal{M}, (\xi, i) \models \iota \cdot b_3 \wedge \neg(\iota \cdot A = I)$.

2. It easily follows that there exists an $i' < i$ such that $\mathcal{M}, (\xi, i') \models \iota \cdot b_2$. Using the fact that $\mathsf{Pr}_{\mathsf{NS}} \models_{wt} secrecy_{resp}$, we can conclude that $\mathcal{M}, (\xi, i') \models \mathsf{G}\neg(I \ \mathsf{has} \ \iota \cdot y)$.

3. Since $y \in NT(\eta_2(2))$, it follows from the unique origination property of runs that $\mathcal{M}, (\xi, i'') \models \neg(I \ \mathsf{has} \ \iota \cdot y)$ for all $i'' < i'$. Thus we can conclude that $\mathcal{M}, (\xi, 0) \models \mathsf{G}\neg(I \ \mathsf{has} \ \iota \cdot y)$.

4. Arguing in the lines of item 4 of the proof of authentication for the initiator in the Needham-Schroeder protocol, we can show that there exists some $j' < i$ and $\iota' \in \mathcal{AS}$ such that $\mathcal{M}, (\xi, j') \models \iota' \cdot a_3 \wedge \iota' \cdot B = \iota \cdot B \wedge \iota' \cdot y = \iota \cdot y$. It follows immediately from this that there exists $j < j'$ such that $\mathcal{M}, (\xi, j) \models \iota' \cdot a_2$. Now we note that $\theta_\iota(B) \neq I$, since $act(e_i) \in Ac_{\theta_\iota(B)}$, and by definition $\theta_\iota(B) \in Ho$. Thus $\mathcal{M}, (\xi, j) \models \neg(\iota \cdot B = I)$. Now we use the fact that $\mathsf{Pr}_{\mathsf{NSL}} \models_{wt} auth_{init}$. Thus there exists $\iota'' \in \mathcal{AS}$ such that $\mathcal{M}, (\xi, j) \models \iota' \cdot A = \iota'' \cdot A \ \wedge \ \iota' \cdot B = \iota'' \cdot B \ \wedge \ \iota' \cdot x = \iota'' \cdot x \ \wedge \ \iota' \cdot y = \iota'' \cdot y \ \wedge \ \mathsf{P}(\iota'' \cdot b_2)$. Let $\ell < j$ be such that $\mathcal{M}, (\xi, \ell) \models \iota'' \cdot b_2$. It is clear that $\theta_\iota(y) \in NT(e_\ell)$. But recall that $e_{i'} = (\eta_2, \theta_\iota, 2)$ and thus $\theta_\iota(y) \in NT(e_{i'})$ as well. By the unique origination of ξ, it follows that $\ell = i'$, and thus it also follows that $\theta_\iota = \theta''_\iota$. This proves the desired result.

6.4 Decidability

In this section we study the verification problem of the logic in different settings and see that all the undecidability results and some of the decidability results which we saw in the earlier chapters go through for the logic as well.

The undecidability results are easy to show, since the reachability property (defined at the end of Chapter 3) can be trivially expressed in our logic. Suppose we are given a protocol $\mathsf{Pr} = (\mathsf{C}, \mathsf{R})$, and an action a. Consider the following formula:

$$\alpha_{reach} \stackrel{\text{def}}{=} \neg(\exists \iota)\mathsf{F}(\iota \cdot a).$$

Then it is clear that $\mathsf{Pr} \not\models_{wt} \alpha_{reach}$ iff Pr and a form a positive instance of the reachability problem for well-typed runs. From this it follows that the problem of checking whether $\mathsf{Pr} \models_{wt} \alpha$ is undecidable. Reasoning on exactly the same lines, we can conclude that the problem of checking whether $\mathsf{Pr} \models^T \alpha$ is undecidable, even for finite T (of some reasonable size — the proof in Section 3.2 requires T to be of size at least 6). We summarize the results in the following theorem.

Theorem 6.4.1 *The problem of checking whether* $\mathsf{Pr} \models_{wt} \alpha$ *given a protocol* Pr *and a formula* α *is undecidable.*

For a fixed $T \subseteq \mathcal{T}_0$ *(which might even be finite), the problem of checking whether* $\mathsf{Pr} \models^T \alpha$ *given a protocol* Pr *and a formula* α *is undecidable.*

We now prove that the reduction to well-typed runs described in Section 5.1 extends to our logic as well. In the proof we crucially use the following fact proved in Section 5.1, in the proof of Lemma 5.1.9: if $\xi = e_1 \cdots e_k$ is a run of a weakly-tagged protocol, then for all $i \leq k$, $\overline{(s_i)_I} \cap \mathcal{T}_0 = \overline{(s'_i)_I} \cap \mathcal{T}_0$ (where $s_i = \mathit{infstate}(e_1 \cdots e_i)$ and $s'_i = \mathit{infstate}((e_1)_{\mathsf{n}_0} \cdots (e_i)_{\mathsf{n}_0})$. *We claim that it can be proved along the same lines that* $\overline{(s_i)_A} \cap \mathcal{T}_0 = \overline{(s'_i)_A} \cap \mathcal{T}_0$ *for all* $A \in Ag$*, provided that* n_0 *is added to all the agents' initial states.* We therefore make the assumption that for all protocols Pr and for all $A \in Ag$, $\mathsf{n}_0 \in (\mathsf{init}(\mathsf{Pr}))_A$.

Lemma 6.4.2 *For any fixed* $T \subseteq \mathcal{T}_0$ *such that* $\mathsf{n}_0 \in T$*, for any weakly tagged protocol* $\mathsf{Pr} = (\mathsf{C}, \delta)$ *such that* $\mathsf{C} \subseteq T$*, and for any formula* $\alpha \in \Phi$, $\mathsf{Pr} \models^T \alpha$ *iff* $\mathsf{Pr} \models^T_{wt} \alpha$.

Proof: Fix a set $T \subseteq \mathcal{T}_0$ such that $\mathsf{n}_0 \in T$. Fix a weakly tagged protocol $\mathsf{Pr} = (\mathsf{C}, \delta)$ such that $\mathsf{C} \subseteq T$, and fix a formula α_0. Fix a T-run $\xi = e_1 \cdots e_k$ of Pr with $e_i = (\eta_i, \sigma_i, lp_i)$ for all $i : 1 \leq i \leq k$. Let $s_i = \mathit{infstate}(e_1 \cdots e_i)$, for $i \leq k$. It is clear that

$\xi_{\mathsf{n}_0} = (e_1)_{\mathsf{n}_0} \cdots (e_k)_{\mathsf{n}_0}$ is a well-typed T-run. Let us denote $\mathit{infstate}((e_1)_{\mathsf{n}_0} \cdots (e_i)_{\mathsf{n}_0})$ by $(s_i)_{\mathsf{n}_0}$, for all $i \leq k$. Let $\mathcal{A} = (\mathsf{Pr}, \mathcal{S}_{\mathsf{Pr},T})$ and $\mathcal{A}_{wt} = (\mathsf{Pr}, \mathcal{S}_{\mathsf{Pr},wt,T})$. (*Note that we work with only well-typed substitutions in* $\mathcal{A}_{wt}$.) For every $\mathcal{A}$-assignment θ, let θ_{n_0} be a map such that $\theta_{\mathsf{n}_0}(\iota) = (\theta(\iota))_{\mathsf{n}_0}$ for all $\iota \in \mathcal{AS}$. Since $\theta_{\mathsf{n}_0}(\iota)$ is a well-typed substitution for all $\iota \in \mathcal{AS}$, it is clear that θ_{n_0} is an$\mathcal{A}_{wt}$-assignment. It is also clear that a model $\mathcal{M} = (\mathcal{A}, \theta)$ is compatible with a formula α iff $\mathcal{M}_{\mathsf{n}_0} = (\mathcal{A}_{wt}, \theta_{\mathsf{n}_0})$ is compatible with α. Throughout the proof we will also use the fact that any model compatible with α is also compatible with any subformula of α.

We now prove by induction that for all subformulas α of α_0, and for all $\mathcal{A}$-assignments θ such that $\mathcal{M} = (\mathcal{A}, \theta)$ is compatible with α, for all $\mathcal{A}$-runs ξ, and for all instants i in ξ: $\mathcal{M}, (\xi, i) \models \alpha$ iff $\mathcal{M}_{\mathsf{n}_0}, (\xi_{\mathsf{n}_0}, i) \models \alpha$.

- Suppose α is of the form $\iota{\cdot}A$ has $\iota'{\cdot}m$. Suppose $\theta(\iota') = \sigma$. Then $\theta_{\mathsf{n}_0}(\iota') = \sigma_{\mathsf{n}_0}$. Since $\mathcal{M}$ is compatible with α_0, and since $\iota{\cdot}m \in ST(\alpha_0)$, it follows that $\mathsf{type}(\sigma(m)) = \mathsf{type}(m)$. Hence it follows that $\sigma(m) = \sigma_{\mathsf{n}_0}(m) \in \mathcal{T}_0$. Finally note that $\overline{(s_i)_A} \cap \mathcal{T}_0 = \overline{((s_i)_{\mathsf{n}_0})_A} \cap \mathcal{T}_0$ (as explained in the discussion preceding this lemma).

  Now $\mathcal{M}, (\xi, i) \models \alpha$ iff $\sigma(m) \in \overline{(s_i)_A} \cap \mathcal{T}_0$ iff $\sigma_{\mathsf{n}_0}(m) \in \overline{((s_i)_{\mathsf{n}_0})_A} \cap \mathcal{T}_0$ iff $\mathcal{M}_{\mathsf{n}_0}, (\xi_{\mathsf{n}_0}, i) \models \alpha$.

- Suppose α is of the form $\iota{\cdot}a$. Suppose $\theta(\iota) = \sigma$. Then $\theta_{\mathsf{n}_0}(\iota) = \sigma_{\mathsf{n}_0}$. Since $\mathcal{M}$ is compatible with α_0 and since $\{\iota{\cdot}m \mid m \in ST(a) \cap \mathcal{T}_0\} \subseteq ST(\alpha_0)$, it follows that $\mathsf{type}(\sigma(m)) = \mathsf{type}(m)$ for all $m \in ST(a) \cap \mathcal{T}_0$. Hence it follows that $\sigma(a) = \sigma_{\mathsf{n}_0}(a)$. It also follows that for all $j \leq k$, $\sigma_j(a) = (\sigma_j)_{\mathsf{n}_0}(a)$.

  Now $\mathcal{M}, (\xi, i) \models \alpha$ iff $\eta_i(lp_i) = a$ and $\sigma_i(a) = \sigma(a)$ iff $\sigma_{\mathsf{n}_0}(a) = (\sigma_i)_{\mathsf{n}_0}(a)$ and $\eta_i(lp_i) = a$ iff $\mathcal{M}_{\mathsf{n}_0}, (\xi_{\mathsf{n}_0}, i) \models \alpha$.

- Suppose α is of the form $\iota{\cdot}x = \iota'{\cdot}x'$. Suppose $\theta(\iota) = \sigma$ and $\theta(\iota') = \sigma'$. Then $\theta_{\mathsf{n}_0}(\iota) = \sigma_{\mathsf{n}_0}$ and $\theta_{\mathsf{n}_0}(\iota') = \sigma'_{\mathsf{n}_0}$. Also note that $\mathsf{type}(\sigma(x)) = \mathsf{type}(x)$ and $\mathsf{type}(\sigma'(x')) = \mathsf{type}(x')$. Therefore $\sigma_{\mathsf{n}_0}(x) = \sigma(x)$ and $\sigma'_{\mathsf{n}_0}(x') = \sigma'(x')$.

  Now $\mathcal{M}, (\xi, i) \models_\theta \alpha$ iff $\sigma(x) = \sigma'(x')$ iff $\sigma_{\mathsf{n}_0}(x) = \sigma'_{\mathsf{n}_0}(x')$ iff $\mathcal{M}_{\mathsf{n}_0}, (\xi_{\mathsf{n}_0}, i) \models \alpha$.

- Suppose α is of the form $\neg\beta$. Now $\mathcal{M}, (\xi, i) \models \alpha$ iff (by semantics) $\mathcal{M}, (\xi, i) \not\models \beta$ iff (by induction hypothesis) $\mathcal{M}_{\mathsf{n}_0}, (\xi_{\mathsf{n}_0}, i) \not\models \beta$ iff (by semantics) $\mathcal{M}_{\mathsf{n}_0}, (\xi_{\mathsf{n}_0}, i) \models \alpha$.

- Suppose α is of the form $\beta \vee \gamma$. Now by semantics $\mathcal{M}, (\xi, i) \models \alpha$ iff $\mathcal{M}, (\xi, i) \models \beta$ or $\mathcal{M}, (\xi, i) \models \gamma$. By induction hypothesis, this happens exactly when $\mathcal{M}_{\mathsf{n}_0}, (\xi_{\mathsf{n}_0}, i) \models \beta$ or $\mathcal{M}_{\mathsf{n}_0}, (\xi_{\mathsf{n}_0}, i) \models \gamma$. But by semantics this happens exactly when $\mathcal{M}_{\mathsf{n}_0}, (\xi_{\mathsf{n}_0}, i) \models \alpha$.

- Suppose α is of the form $\mathsf{F}\beta$.

  If $\mathcal{M}, (\xi, i) \models \alpha$ then (by semantics) there exists $j \geq i$ such that $\mathcal{M}, (\xi, j) \models \beta$. This implies (by induction hypothesis) that $\mathcal{M}_{\mathsf{n}_0}, (\xi_{\mathsf{n}_0}, j) \models \beta$. But now (by semantics) $\mathcal{M}_{\mathsf{n}_0}, (\xi_{\mathsf{n}_0}, i) \models \alpha$. In a similar manner we can prove that if $\mathcal{M}_{\mathsf{n}_0}, (\xi_{\mathsf{n}_0}, i) \models \alpha$ then $\mathcal{M}, (\xi, i) \models \alpha$.

- Suppose α is of the form $\mathsf{P}\beta$.

  If $\mathcal{M}, (\xi, i) \models \alpha$ then (by semantics) there exists $j \leq i$ such that $\mathcal{M}, (\xi, j) \models \beta$. This implies (by induction hypothesis) that $\mathcal{M}_{\mathsf{n}_0}, (\xi_{\mathsf{n}_0}, j) \models \beta$. But now (by semantics) $\mathcal{M}_{\mathsf{n}_0}, (\xi_{\mathsf{n}_0}, i) \models \alpha$. In a similar manner we can prove that if $\mathcal{M}_{\mathsf{n}_0}, (\xi_{\mathsf{n}_0}, i) \models \alpha$ then $\mathcal{M}, (\xi, i) \models \alpha$.

- Suppose α is of the form $(\exists \iota)\beta$.

  If $\mathcal{M}, (\xi, i) \models \alpha$ then (by semantics) there exists $\sigma \in \mathcal{S}_{\mathsf{Pr},T}$ such that $\mathcal{M}' = (\mathcal{A}, \theta[\iota := \sigma])$ is compatible with β and $\mathcal{M}', (\xi, i) \models \beta$. This implies (by induction hypothesis) that $\mathcal{M}'_{\mathsf{n}_0}, (\xi_{\mathsf{n}_0}, i) \models \beta$. But now it is clear that $\sigma_{\mathsf{n}_0} \in \mathcal{S}_{\mathsf{Pr},wt,T}$ and thus (by semantics and the fact that $\mathcal{M}'_{\mathsf{n}_0} = (\mathcal{A}_{wt}, \theta_{\mathsf{n}_0}[\iota := \sigma_{\mathsf{n}_0}])$),it follows that $\mathcal{M}_{\mathsf{n}_0}, (\xi_{\mathsf{n}_0}, i) \models \alpha$.

  If $\mathcal{M}_{\mathsf{n}_0}, (\xi_{\mathsf{n}_0}, i) \models \alpha$ then (by semantics) there exists $\sigma \in \mathcal{S}_{\mathsf{Pr},wt,T}$ such that $\mathcal{M}'' = (\mathcal{A}_{wt}, \theta[\iota := \sigma])$ is compatible with β and $\mathcal{M}'', (\xi_{\mathsf{n}_0}, i) \models \beta$. But θ_ι for all $\iota \in \mathcal{AS}$ and σ are well-typed substitutions, which implies that $\theta = \theta_{\mathsf{n}_0}$ and $\sigma = \sigma_{\mathsf{n}_0}$. Thus, letting $\mathcal{M}' = (\mathcal{A}, \theta[\iota := \sigma])$, we see that $\mathcal{M}'' = \mathcal{M}'_{\mathsf{n}_0}$. Thus we have that $\mathcal{M}'_{\mathsf{n}_0}, (\xi_{\mathsf{n}_0}, i) \models \beta$. By induction hypothesis it follows that $\mathcal{M}', (\xi, i) \models \beta$. Thus by semantics it follows that $\mathcal{M}, (\xi, i) \models \alpha$.

Suppose now that $\mathsf{Pr} \models^T_{wt} \alpha$ for some formula α. We claim that $\mathsf{Pr} \models^T \alpha$ as well. Let $\mathcal{A} = (\mathsf{Pr}, \mathcal{S}_{\mathsf{Pr},T})$ and let ξ be an $\mathcal{A}$-run. Consider any $\mathcal{A}$-assignment θ and let $\mathcal{M} = (\mathcal{A}, \theta)$ be compatible with α. By what has been proved above $\mathcal{M}, (\xi, 0) \models \alpha$ iff $\mathcal{M}_{\mathsf{n}_0}, (\xi_{\mathsf{n}_0}, 0) \models \alpha$. Since $\mathsf{Pr} \models^T_{wt} \alpha$, $\mathcal{M}_{\mathsf{n}_0}, (\xi_{\mathsf{n}_0}, 0) \models \alpha$. Therefore $\mathcal{M}, (\xi, 0) \models \alpha$ as well. Since ξ is an arbitrary $\mathcal{A}$-run and θ is an arbitrary $\mathcal{A}$-assignment, this proves that $\mathsf{Pr} \models^T \alpha$.

Suppose now that $\mathsf{Pr} \models^T \alpha$ for some formula α. We claim that $\mathsf{Pr} \models^T_{wt} \alpha$ as well. Let $\mathcal{A}' = (\mathsf{Pr}, \mathcal{S}_{\mathsf{Pr},wt,T})$and let ξ be an $\mathcal{A}'$-run. Of course $\mathcal{A}' = \mathcal{A}_{wt}$ where $\mathcal{A} = (\mathsf{Pr}, \mathcal{S}_{\mathsf{Pr},T})$. Further $\xi = \xi_{\mathsf{n}_0}$. Let θ be a $\mathcal{A}'$-assignment and let $\mathcal{M}' = (\mathcal{A}', \theta)$ be compatible with α. Again it is obvious that $\theta = \theta_{\mathsf{n}_0}$ and thus $\mathcal{M}' = \mathcal{M}_{\mathsf{n}_0}$ where $\mathcal{M} = (\mathcal{A}, \theta)$. By what has been proved above $\mathcal{M}, (\xi, 0) \models \alpha$ iff $\mathcal{M}_{\mathsf{n}_0}, (\xi_{\mathsf{n}_0}, 0) \models \alpha$. Since $\mathsf{Pr} \models^T \alpha$, $\mathcal{M}, (\xi, 0) \models \alpha$. Therefore it follows that $\mathcal{M}_{\mathsf{n}_0}, (\xi, 0) \models \alpha$ as well. Since ξ is an arbitrary $\mathcal{A}'$-run and θ is an arbitrary $\mathcal{A}'$-assignment, this proves that $\mathsf{Pr} \models^T_{wt} \alpha$.

This completes the proof of the lemma. □

The above lemma shows that once we fix a $T \subseteq \mathcal{T}_0$, it suffices to consider *well-typed* T runs of any given protocol. Of course, if we fix a finite $T \subseteq \mathcal{T}_0$, then for any protocol Pr, there are only finite many well-typed T-events. But there might still be infinitely many well-typed T-runs of Pr, since the same event may repeat many times in a run. To get decidability in such a setting, we show that for every protocol Pr and formula α, there is a finite-state automaton $\mathscr{A}_{\mathsf{Pr},\alpha}$ with alphabet $\mathit{Events}(\mathsf{Pr})$ such that $\xi \in \mathscr{L}(\mathscr{A}_{\mathsf{Pr},\alpha})$ iff there is some $(\mathsf{Pr}, \mathcal{S}_{\mathsf{Pr},wt,T})$-assignment θ such that $\mathcal{M} = ((\mathsf{Pr}, \mathcal{S}_{\mathsf{Pr},wt,T}), \theta)$ is compatible with α and $\mathcal{M}, (\xi, 0) \models \alpha$.

We now fix a finite set $T \subseteq \mathcal{T}_0$, a weakly tagged protocol Pr (and therefore the structure $\mathcal{A}_0 = (\mathsf{Pr}, \mathcal{S}_{\mathsf{Pr},wt,T})$), and a formula α_0 for the rest of the section, and take up the construction of the automaton $\mathscr{A}_{\mathsf{Pr},\alpha_0}$. As observed earlier, given a structure $\mathcal{A}$ and a formula α, to see whether $\mathcal{A} \models \alpha$, it suffices to consider $\mathcal{A}$-assignments restricted to $\mathit{FSN}(\alpha)$. In the case of $\mathcal{A}_0$, we need to consider only finitely many such $\mathcal{A}_0$-assignments (since $\mathcal{S}_{\mathsf{Pr},wt,T}$ and $\mathit{FSN}(\alpha_0)$ are finite sets, whose sizes depend only on the sizes of Pr, α_0 and T). For the rest of the section we assume that $\theta_1, \ldots, \theta_r$ is an enumeration of all the $\mathcal{A}_0$-assignments θ restricted to $\mathit{FSN}(\alpha_0)$ such that $(\mathcal{A}_0, \theta)$ is compatible with α_0. We let $\mathcal{M}_i = (\mathcal{A}_0, \theta_i)$, for all $i \leq r$.

Let SF denote $\mathit{SF}(\alpha_0)$. We define $\neg\mathit{SF}$ to be the set $\{\alpha \mid \neg\alpha \in \mathit{SF}\} \cup \{\neg\alpha \mid \alpha \in \mathit{SF}$ and α is not of the form $\neg\beta\}$. We define CL to be $\mathit{SF} \cup \neg\mathit{SF}$.

An *atom* Ψ is any subset of CL which satisfies the following conditions:

- for all $\neg\alpha \in \mathit{CL}$, $\neg\alpha \in \Psi$ iff $\alpha \notin \Psi$;
- for all $\alpha \vee \beta \in \mathit{CL}$, $\alpha \vee \beta \in \Psi$ iff $\alpha \in \Psi$ or $\beta \in \Psi$;
- for all $\mathsf{F}\alpha \in \mathit{CL}$, if $\alpha \in \Psi$ then $\mathsf{F}\alpha \in \Psi$;

- for all $\mathsf{P}\alpha \in CL$, if $\alpha \in \Psi$ then $\mathsf{P}\alpha \in \Psi$.

Given two atoms Ψ_1 and Ψ_2, we say that $\Psi_1 \longrightarrow \Psi_2$ iff:

- for all $\mathsf{F}\alpha \in CL$:
  - if $\mathsf{F}\alpha \in \Psi_2$ then $\mathsf{F}\alpha \in \Psi_1$, and
  - if $\mathsf{F}\alpha \in \Psi_1$ and $\alpha \notin \Psi_1$ then $\mathsf{F}\alpha \in \Psi_2$;
- for all $\mathsf{P}\alpha \in CL$:
  - if $\mathsf{P}\alpha \in \Psi_1$ then $\mathsf{P}\alpha \in \Psi_2$, and
  - if $\mathsf{P}\alpha \in \Psi_2$ and $\alpha \notin \Psi_2$ then $\mathsf{P}\alpha \in \Psi_1$.

An atom Ψ_1 is an *initial atom* iff:

- for all $\mathsf{P}\alpha \in CL$, if $\mathsf{P}\alpha \in \Psi$ then $\alpha \in \Psi$, and
- for all formula $\alpha \in CL$ of the form $\iota \cdot a$, $\alpha \notin \Psi$.

(The last clause reflects the fact that a formula of the form $\iota \cdot a$ is true only at *positive* instants.)

An atom Ψ_1 is a *final atom* iff for all $\mathsf{F}\alpha \in CL$, if $\mathsf{F}\alpha \in \Psi$ then $\alpha \in \Psi$.

For $i, j \leq r$ and $\iota \in FSN(\alpha_0)$, we say that θ_i and θ_j are ι-variants if for all $\iota' \in FSN(\alpha_0)$ such that $\iota' \neq \iota$: $\theta_i(\iota') = \theta_j(\iota')$.

A *molecule* is a tuple of the form $(\xi, \Psi_1, \cdots, \Psi_r)$ such that:

- ξ is a *reduced* well-typed T-run of Pr;
- for all $i \leq r$, Ψ_i is an atom such that for all atomic formulas $\alpha \in CL$ of the form $\iota \cdot A$ has $\iota' \cdot m$ and $\iota \cdot x = \iota' \cdot x'$: $\alpha \in \Psi_i$ iff $\mathfrak{M}_i, (\xi, |\xi|) \models \alpha$;
- for all $i \leq r$ and for all $(\exists \iota)\alpha \in CL$, $(\exists \iota)\alpha \in \Psi_i$ iff there exists $j \leq r$ such that θ_i and θ_j are ι-variants and $\alpha \in \Psi_j$.

Note that since there are only finitely many reduced well-typed T-runs of Pr, and since CL is a finite set, there are only finitely many molecules. We denote the set of molecules by $\mathscr{M}$.

Given two molecules $\chi = (\xi, \Psi_1, \cdots, \Psi_r)$ and $\chi' = (\xi', \Psi'_1, \cdots, \Psi'_r)$, and an event $e \in Events(\mathsf{Pr})$, we say that $\chi \xrightarrow{e} \chi'$ iff:

- $\xi' = \mathsf{red}(\xi \cdot e)$;
- for all $i \leq r$, $\Psi_i \longrightarrow \Psi_i'$;
- for all $i \leq r$ and all atomic formulas $\alpha \in CL$ of the form $\iota \cdot a$, $\alpha \in \Psi_i'$ iff $\mathcal{M}_i, (\xi \cdot e, |\xi \cdot e|) \models \alpha$.

A molecule $\chi = (\xi, \Psi_1, \cdots, \Psi_r)$ is said to be an *initial molecule* iff:

- $\xi = \varepsilon$,
- for all $i \leq r$, Ψ_i is an initial atom, and
- there exists $i \leq r$ such that $\alpha_0 \in \Psi_i$.

The set of initial molecules is denoted by $\mathscr{I}$.

A molecule $\chi = (\xi, \Psi_1, \cdots, \Psi_r)$ is said to be a *final molecule* iff for all $i \leq r$, Ψ_i is a final atom. The set of final molecules is denoted by $\mathscr{F}$.

We are now all set to define the automaton.

Definition 6.4.3 (The automaton $\mathscr{A}_{\mathsf{Pr},\alpha_0}$) *The automaton $\mathscr{A}_{\mathsf{Pr},\alpha_0}$ is given by the tuple $(\mathscr{M}, \longrightarrow, \mathscr{I}, \mathscr{F})$ where:*

- *$\mathscr{M}$, the set of molecules, forms the finite set of* states *of the automaton,*
- *The relation $\longrightarrow$ defined on molecules forms the* transition relation *of the automaton, and*
- *$\mathscr{I}$ forms the set of* initial states *and $\mathscr{F}$ forms the set of* final states *of the automaton.*

An *accepting run of the automaton* on a sequence $\xi = e_1 \cdots e_k$ from $(\mathit{Events}(\mathsf{Pr}))^*$ is a sequence of molecules $\chi_0 \cdots \chi_k$ such that:

- χ_0 is an initial molecule and χ_k is a final molecule, and
- for all $i : 1 \leq i \leq k$, $\chi_{i-1} \xrightarrow{e_i} \chi_i$.

The language accepted by $\mathscr{A}_{\mathsf{Pr},\alpha_0}$, denoted $\mathscr{L}(\mathscr{A}_{\mathsf{Pr},\alpha_0})$ is the set of $\xi \in (\mathit{Events}(\mathsf{Pr}))^*$ such that there is an accepting run of the automaton on ξ.

The following technical lemma shows the correctness of the automaton construction and immediately implies Theorem 6.4.5.

Lemma 6.4.4 *For any sequence $\xi \in (Events(\mathsf{Pr}))^*$, $\xi \in \mathscr{L}(\mathscr{A}_{\mathsf{Pr},\alpha_0})$ iff ξ is an $\mathcal{A}_0$-run and there exists $i \leq r$ such that $\mathcal{M}_i, (\xi, 0) \models \alpha_0$.*

Proof: Fix a $\xi = e_0 \cdots e_k \in (Events(\mathsf{Pr}))^*$. For all $j \leq k$, let ξ_j denote $e_1 \cdots e_j$.
$(\Rightarrow)$:
We first prove that if ξ is in the language of the automaton then ξ is a run of Pr and for some $i \leq r$, $\mathcal{M}_i, (\xi, 0) \models \alpha$. Suppose $\xi \in \mathscr{L}(\mathscr{A}_{\mathsf{Pr},\alpha_0})$. This means that there is an accepting run of the automaton of the form $\chi_0 \cdots \chi_k$. Let $\chi_j = (\rho_j, \Psi_1^j, \cdots, \Psi_r^j)$, for all $j \leq k$.

Claim: ξ is an $\mathcal{A}_0$-run.

> **Proof of Claim:** We now prove that for all $j \leq k$, $\rho_j = \mathsf{red}(\xi_j)$. From this it would follow that $\mathsf{red}(\xi) = \rho_k$, and since ρ_k is a run, it is easy to see that ξ is a run as well.
>
> Since $\xi_0 = \rho_0 = \varepsilon$, $\mathsf{red}(\xi_0) = \rho_0$. Suppose $\rho_{j-1} = \mathsf{red}_{j-1}$ for some $j : 1 \leq j \leq k$. Now $\xi_j = \xi_{j-1} \cdot e_j$. But since $\chi_{j-1} \xrightarrow{e_j} \chi_j$, it follows from the definitions that $\rho_j = \mathsf{red}(\rho_{j-1} \cdot e_j)$. But it is an easy consequence of the definition of red that $\mathsf{red}(\xi \cdot e) = \mathsf{red}(\mathsf{red}(\xi) \cdot e)$, and from this it follows that $\mathsf{red}(\xi_j) = \rho_j$. This completes the induction step and the proof of the claim as well.

Claim: $\mathcal{M}_i, (\xi, 0) \models \alpha_0$ for some $i \leq r$.

> **Proof of Claim:** We now prove that for all $j \leq k$, all $\alpha \in CL$ and all $i \leq r$, $\alpha \in \Psi_i^j$ iff $\mathcal{M}_i, (\xi, j) \models \alpha$. Since χ_0 is an initial molecule, by definition $\alpha_0 \in \Psi_i^0$ for some $i \leq r$, and it immediately follows that $\mathcal{M}_i, (\xi, 0) \models \alpha_0$.
>
> Fix $j \leq k$ and $i \leq r$. We prove by induction on the structure of formulas that $\alpha \in \Psi_i^j$ iff $\mathcal{M}_i, (\xi, j) \models \alpha$.
>
> - If α is of the form $\iota \cdot A$ has $\iota' \cdot m$ or $\iota \cdot x = \iota' \cdot x'$ then it follows from the definition of molecules that $\alpha \in \Psi_i^j$ iff $\mathcal{M}_i, (\rho_j, |\rho_j|) \models \alpha$. But since $\rho_j = \mathsf{red}(\xi_j)$, it follows that $infstate(\xi_j) = infstate(\rho_j)$. It now immediately follows that $\alpha \in \Psi_i^j$ iff $\mathcal{M}_i, (\xi, j) \models \alpha$.
> - Suppose α is of the form $\iota \cdot a$. If $j = 0$ then it follows from the semantics that $\mathcal{M}_i, (\xi, j) \not\models \alpha$, and it follows from the definition of initial atoms that $\alpha \notin \Psi_i^j$. If $j \geq 1$, then it follows from $\chi_{j-1} \xrightarrow{e_j} \chi_j$ that $\alpha \in \Psi_i^j$ iff $\mathcal{M}_i, (\rho_{j-1} \cdot e_j, |\rho_{j-1} \cdot e_j|) \models \alpha$. But the semantics of a formula of this kind

depends only on the last event e_j and not on the other events in ρ_j. It thus immediately follows that $\alpha \in \Psi_i^j$ iff $\mathcal{M}_i, (\xi, j) \models \alpha$.

- The boolean cases are handled by a routine application of the induction hypothesis, using the fact that atoms are propositionally consistent.
- Suppose α is of the form $\mathsf{F}\beta$. We prove by induction on $k-j$ that if $\alpha \in \Psi_i^j$ then $\mathcal{M}_i, (\xi, j) \models \alpha$. Suppose $\alpha = \mathsf{F}\beta \in \Psi_i^k$. Then by definition of final atom, $\beta \in \Psi_i^k$. By induction hypothesis (on the formulas) $\mathcal{M}_i, (\xi, k) \models \beta$, and hence $\mathcal{M}_i, (\xi, k) \models \alpha$. Suppose $j < k$ and $\alpha \in \Psi_i^j$. If $\beta \in \Psi_i^j$, then by induction hypothesis (on the formulas) $\mathcal{M}_i, (\xi, j) \models \beta$ and hence $\mathcal{M}_i, (\xi, j) \models \alpha$. If $\beta \notin \Psi_i^j$, then since $\Psi_i^j \longrightarrow \Psi_i^{j+1}$, it follows that $\alpha \in \Psi_i^{j+1}$. By induction hypothesis (on $k-j$), it follows that $\mathcal{M}_i, (\xi, j+1) \models \alpha$, and hence $\mathcal{M}_i, (\xi, j) \models \alpha$ as well.

  We now prove by induction on $k - j$ that if $\mathcal{M}_i, (\xi, j) \models \alpha$ then $\alpha \in \Psi_i^j$. If $\mathcal{M}_i, (\xi, k) \models \alpha$, then by semantics $\mathcal{M}_i, (\xi, k) \models \beta$ as well. Therefore by induction hypothesis (on formulas), it follows that $\beta \in \Psi_i^k$, and by definition of atoms it follows that $\alpha \in \Psi_i^k$ as well. Suppose $j < k$ and $\mathcal{M}_i, (\xi, j) \models \alpha$. If $\mathcal{M}_i, (\xi, j) \models \beta$ then $\beta \in \Psi_i^j$ (by induction hypothesis on formulas). It follows from the definition of atoms that $\alpha \in \Psi_i^j$ as well. If $\mathcal{M}_i, (\xi, j) \not\models \beta$ then $\mathcal{M}_i, (\xi, j+1) \models \alpha$ and hence by induction hypothesis on $k-j$, $\alpha \in \Psi_i^{j+1}$. Since $\Psi_i^j \longrightarrow \Psi_i^{j+1}$, it follows from the definitions that $\alpha \in \Psi_i^j$ as well.
- The case when α is of the form $\mathsf{P}\beta$ is handled similarly as above.
- Suppose α is of the form $(\exists \iota)\beta$. Then $\alpha \in \Psi_i^j$ iff (by definition of molecules) there is $i' \leq r$ such that θ_i and $\theta_{i'}$ are ι-variants and $\beta \in \Psi_{i'}^j$ iff (by induction hypothesis) there is $i' \leq r$ such that θ_i and $\theta_{i'}$ are ι-variants and $\mathcal{M}_{i'}, (\xi, j) \models \beta$ iff (by semantics) $\mathcal{M}_i, (\xi, j) \models \alpha$.

($\Leftarrow$) :

We now prove that if ξ is an $\mathcal{A}_0$-run and $\mathcal{M}_i, (\xi, 0) \models \alpha_0$ for some $i \leq r$, then $\xi \in \mathscr{L}(\mathscr{A}_{\mathsf{Pr},\alpha_0})$. For all $i \leq r$ and $j \leq k$, let $\Psi_i^j = \{\alpha \in CL \mid \mathcal{M}_i, (\xi, j) \models \alpha\}$. For all $j \leq k$, let $\rho_j = \mathsf{red}(\xi_j)$. Let $\chi_j = (\rho_j, \Psi_1^j, \cdots, \Psi_r^j)$. We claim that $\chi_0 \cdots \chi_k$ is an accepting run of $\mathscr{A}_{\mathsf{Pr},\alpha_0}$ on the sequence ξ.

It is straightforward to check that for all $i \leq r$ and $j \leq k$, Ψ_i^j is an atom. Further from the fact that ξ is a run, ρ_j is a reduced *run* for all $j \leq k$. It now follows by

the semantics that χ_j is a molecule for all $j \leq k$. From the semantics it also follows that $\chi_{j-1} \xrightarrow{e_j} \chi_j$ for all $j : 1 \leq j \leq k$, and it also follows that χ_0 is an initial molecule and χ_k is a final molecule. Thus $\chi_0 \cdots \chi_k$ is an accepting run of the automaton on ξ. Therefore $\xi \in \mathscr{L}(\mathscr{A}_{\mathsf{Pr},\alpha_0})$.

This completes the proof of the lemma. □

Thus we see that checking whether $\mathsf{Pr} \models^T_{wt} \alpha_0$ reduces to checking whether $\mathscr{L}(\mathscr{A}_{\mathsf{Pr},\neg\alpha_0})$ is empty. Since the emptiness problem for finite state automata is decidable, it follows that checking whether $\mathsf{Pr} \models^T_{wt} \alpha$ is decidable. This coupled with Lemma 6.4.2 yields the following theorem, the main technical result of this chapter.

Theorem 6.4.5 *For a fixed finite $T \subseteq \mathfrak{T}_0$, the problem of checking whether $\mathsf{Pr} \models^T \alpha$ given a weakly tagged protocol and a formula α is decidable.*

Chapter 7

Conclusions

We summarise the work done in the thesis below:

- We introduced a model for security protocols in Chapter 2, where we highlighted the role of properties like send admissibility in analysis of protocols. We also introduced the important notions of well-formed protocols and tagged protocols, and proved some important consequences of our tagging scheme. We also looked at important properties of the synth and analz operators.

- We gave proofs of the undecidability of the secrecy problem, both under the setting of unboundedly many nonces but bounded message length, and boundedly many nonces but unbounded message length, in Chapter 3. We provided simple and uniform proofs for both the resuts.

- In Chapter 4, we proved that the secrecy problem for tagged protocols is decidable, when we consider only well-typed runs. We also saw a decision procedure for solving the problem with a double exponential upper bound (in terms of the number of communications in the protocol specification).

- In Chapter 5, we proved that for weakly tagged protocols, presence of a leaky run implies the presence of a well-typed leaky run. We derived the fact that the general secrecy problem for tagged protocols is decidable as a consequence of the above result. We also looked at a semantic approach to decidability based on an equivalence relation on terms.

- In Chapter 6, we introduced a logic using which we could express many interesting security properties. We saw many examples of reasoning using the logic. We then extended some of the results of Chapter 5 to the logic.

Future directions

The most immediate improvement over the work in this thesis involves extending the decidability result in Chapter 4 to cover other notions of secrecy and authentication. We feel that obtaining a decidable logic in the presence of unbounded nonces will be a significant result and that it will provide significant insight into the nature of the problem itself. We believe that such a result is eminently possible, if the logic itself does not force undecidability. This is because the undecidability results have to do with the inherent power of protocols to code up computations and do not have much to do with the properties we are checking for. Since the well-formedness conditions and other restrictions on tagged protocols restrict the intruder's power to code up such computations, we believe that the decidability result will extend to the logic. But more insight needs to be developed before we can tackle the problem formally.

Another important direction of work is to convert the decision procedure of Chapter 4 into a practical verification algorithm which is efficient in practice. It is possible that some notions introduced in Chapter 6 like abstract substitution names might be of help in this endeavour.

Much more work needs to be done on *formal reasoning about protocols*. The examples which we presented in Chapter 6 involved semantic reasoning. In future work, we aim to formalise this process by introducing axioms and (probably protocol-specific) rules using which we can carry out the reasoning in the logic. There are further interseting technical questions like formally characterising classes of protocols in the logic, various axiomatisability questions, decidability of satisfiability etc.

An important extension would involve extending some of the features of our basic model. The most important of these is to consider constructed keys. In the presence of constructed keys, $\mathsf{synth}(\mathsf{analz}(T))$ no longer represents the closure of the set of terms T. For instance, letting $T = \{\{m\}_{\{n\}_k}, n, k\}$, m does not belong to $\mathsf{synth}(\mathsf{analz}(T))$ but (once we set up the synth and analz-rules for constructed keys properly) it can be seen that $\{n\}_k$ belongs to $\mathsf{synth}(T)$ and that m belongs to

$\mathsf{analz}(\mathsf{synth}(T))$. The usual style in such a setting is to use a combined proof system which incorporate both synthesis and analysis rules. Several of our proofs have to be modified considerably in this new setting. We believe that the results of Chapter 5 can be easily extended in this new setting as well. But the reduction to good runs has to be reworked to an extent. The key to proving these results would be to derive some normal forms for these new proofs.

We hope that the ideas and results presented in this thesis will form a basis for further improvements and eventually find their use in practical verification of security protocols.

Publications

[RS01] R. Ramanujam and S.P. Suresh. Information based reasoning about security protocols In *Proceedings of LACPV'01 (Logical Aspects of Cryptographic Protocol Verification)*, volume 56 of *Electronic Notes in Theoretical Computer Science*, pages 89–104, 2001.

[RS03a] R. Ramanujam and S.P. Suresh. A decidable subclass of unbounded security protocols In Roberto Gorrieri, editor, *Proceedings of WITS'03 (Workshop on Issues in the Theory of Security)*, pages 11–20, Warsaw, Poland, April 2003.

[RS03b] R. Ramanujam and S.P. Suresh. An equivalence on terms for security protocols In Ramesh Bharadwaj, editor, *Proceedings of AVIS'03 (Workshop on Automatic Verification of Infinite-State Systems)*, pages 45–56, Warsaw, Poland, April 2003.

[RS03c] R. Ramanujam and S.P. Suresh. Tagging makes secrecy decidable for unbounded nonces as well In *Proceedings of 23rd FST&TCS*, Mumbai, India, December 2003. To appear.

Bibliography

[Aba99] Martin Abadi. Secrecy by Typing in Security Protocols. *Journal of the ACM*, 46(5):749–786, 1999.

[ABV02] Rafael Accorsi, David Basin, and Luca Viganò. Modal Specifications of Trace-Based Security Properties. In Klaus Fischer and Dieter Hutter, editors, *Proceedings of the Second International Workshop on Security of Mobile Multiagent Systems*, pages 1–11, July 2002.

[AC02] Roberto M. Amadio and Witold Charatonik. On name generation and set-based analysis in Dolev-Yao model. Technical Report 4379, INRIA, January 2002. Extended abstract in *Proceedings of CONCUR'02*, Springer-Lecture Notes in Computer Science 2421, pages 499–514, 2002.

[AFG02] Martin Abadi, Cédric Fournet, and Georges Gonthier. Secure Implementation of Channel Abstractions. *Information and Computation*, 174(1):37–83, April 2002.

[AG98] Martin Abadi and Andrew D. Gordon. A Bisimulation Method for Cryptographic Protocols. *Nordic Journal of Computing*, 5(4):267–303, 1998.

[AG99] Martin Abadi and Andrew D. Gordon. A calculus for cryptographic protocols: the spi calculus. *Information and Computation*, 148(1):1–70, 1999.

[ALV02] Roberto M. Amadio, Denis Lugiez, and Vincent Vanackère. On the symbolic reduction of processes with cryptographic functions. *Theoretical Computer Science*, 290(1):695–740, 2002. Also *INRIA Research Report* 4147, March 2001.

[AN95] Ross Anderson and Roger M. Needham. Programming Satan's computer. In *Computer Science Today*, volume 1000 of *Lecture Notes in Computer Science*, pages 426–441, 1995.

[AN96] Martin Abadi and Roger M. Needham. Prudent engineering practices for cryptographic protocols. *IEEE Transactions on Software Engineering*, 22:6–15, 1996.

[AR00] Martin Abadi and Phillip Rogaway. Reconciling two views of cryptography (the computational soundness of formal encryption). In *Proceedings of the IFIP International Conference on TCS (IFIP TCS2000)*, volume 1872 of *Lecture Notes in Computer Science*, pages 3–22, 2000.

[AT91] Martin Abadi and Mark Tuttle. A Semantics for a Logic of Authentication. In *Proceedings of the Tenth Annual ACM Symposium on Principles of Distributed Computing*, pages 201–216, August 1991.

[BAN90] Michael Burrows, Martin Abadi, and Roger M. Needham. A logic of authentication. *ACM Transactions on Computer Systems*, 8(1):18–36, Feb 1990.

[Bel99] Giampaolo Bella. Modelling Security Protocols Based on Smart Cards. In *Proceedings of the International Workshop on Cryptographic Techniques & E-Commerce*, pages 139–146, 1999.

[Bie90] Pierre Bieber. A logic of communication in a hostile environment. In *Proceedings of 3rd Computer Security Foundations Workshop*, pages 14–22. IEEE Press, 1990.

[BL73] David E. Bell and Leonard J. LaPadula. Secure computer systems: Mathematical foundations and model. Technical Report M74-244, MITRE Corporation, Bedford, Massachussets, 1973.

[BM93] Colin Boyd and Wenbo Mao. On a limitation of BAN logic. In *Proceedings of Eurocrypt'93*, Lecture Notes in Computer Science, pages 240–247, 1993.

[Bol97] Dominique Bolignano. Towards a mechanization of cryptographic protocol verification. In *Proceedings of CAV'97*, volume 1254 of *Lecture Notes in Computer Science*, pages 131–142, 1997.

[BP03] Bruno Blanchet and Andreas Podelski. Verification of Cryptographic Protocols: Tagging Enforces Termination. In Andrew D. Gordon, editor, *Proceedings of FoSSaCS'03*, volume 2620 of *Lecture Notes in Computer Science*, pages 136–152, 2003.

[BR93] Mihir Bellare and Phillip Rogaway. Entity authentication and key distribution. In D. Stinson, editor, *Advances in Cryptography-Crypto93*, volume 773 of *Lecture Notes in Computer Science*, pages 232–249, 1993.

[CC03] Hubert Comon and Véronique Cortier. Tree automata with one memory, set constraints, and cryptographic protocols. *Theoretical Computer Science*, 2003. To appear.

[CCM01] Hubert Comon, Véronique Cortier, and John C. Mitchell. Tree automata with One Memory, Set Constraints, and Ping-Pong Protocols. In *Proceedings of ICALP 2001*, volume 2076 of *Lecture Notes in Computer Science*, pages 682–693, 2001.

[CDL+99] Iliano Cervesato, Nancy A. Durgin, Patrick D. Lincoln, John C. Mitchell, and Andre Scedrov. A Meta-notation for Protocol Analysis. In P. Syverson, editor, *Proceedings of the 12th IEEE Computer Security Foundations Workshop*, pages 35–51. IEEE Computer Society Press, 1999.

[CJ97] John Clark and Jeremy Jacob. A survey of authentication protocol literature. Available at `http://www.cs.york.ac.uk./~jac`, 1997.

[CMS00] Iliano Cervesato, Catherine A. Meadows, and Paul F. Syverson. Dolev-Yao is no better than Machiavelli. In P. Degano, editor, *Proceedings of WITS'00*, pages 87–92, July 2000.

[CS02] Hubert Comon and Vitaly Shmatikov. Is it possible to decide whether a cryptographic protocol is secure or not? *Journal of Telecommunications and Information Technology*, 4:5–15, 2002.

[DEK82] Danny Dolev, Shimon Even, and Richard M. Karp. On the Security of Ping-Pong Protocols. *Information and Control*, 55:57–68, 1982.

[Den77] Dorothy E. Denning. A lattice model of secure information flow. *Communications of the ACM*, 19(5):236–243, May 1977.

[DH76] Whitfield Diffie and Martin E. Hellman. New Directions in Cryptography. *IEEE Transactions on Information Theory*, IT-22(6):644–654, November 1976.

[DLMS99] Nancy A. Durgin, Patrick D. Lincoln, John C. Mitchell, and Andre Scedrov. The undecidability of bounded security protocols. In *Proceedings of the Workshop on Formal Methods and Security Protocols (FMSP'99)*, 1999.

[DM99] Nancy A. Durgin and John C. Mitchell. Analysis of security protocols. In *Calculational System Design*, volume 173 of *Series F: Computer and System Sciences*, pages 369–395. IOS Press, 1999.

[DMTY97] Mourad Debbabi, Mohamed Mejri, Nadia Tawbi, and Imed Yahmadi. Formal automatic verification of authentication protocols. In *Proceedings of the First IEEE International Conference on Formal Engineering Methods (ICFEM97)*, pages 50–59. IEEE Press, 1997.

[DS81] Dorothy E. Denning and Giovanni M. Sacco. Timestamps in Key Distribution Protocols. *Communications of the ACM*, 24(8):533–536, August 1981.

[DY83] Danny Dolev and Andrew Yao. On the Security of public-key protocols. *IEEE Transactions on Information Theory*, 29:198–208, 1983.

[FHG99] F. Javier Thayer Fábrega, Jonathan Herzog, and Joshua Guttman. Strand Spaces: Proving Security Protocols Correct. *Journal of Computer Security*, 7:191–230, 1999.

[GL00] Jean Goubault Larrecq. A method for automatic cryptographic protocol verification. In *Proceedings of the 15th IPDPS Workshops 2000*, volume 1800 of *Lecture Notes in Computer Science*, pages 977–984, 2000.

[GM82] Joseph Goguen and Josè Meseguer. Security policies and security models. In *Proceedings of the 1982 IEEE Symposium on Research in Security and Privacy*, pages 11–20. IEEE Computer Society Press, 1982.

[GNY90] Li Gong, Roger Needham, and Raphael Yahalom. Reasoning About Belief in Cryptographic Protocols. In Deborah Cooper and Teresa Lunt, editors, *Proceedings 1990 IEEE Symposium on Research in Security and Privacy*, pages 234–248. IEEE Computer Society, 1990.

[Gol99] Dieter Gollmann. *Computer Security*. John Wiley & Sons Ltd., 1999.

[Her02] Jonathan Herzog. Computational soundness for formal adversaries. Master's thesis, Massachussets Institute of Technology, October 2002.

[Her03] Jonathan Herzog. A Computational Interpretation of Dolev-Yao Adversaries. In R. Gorrieri, editor, *Proceedings of WITS'03*, pages 146–155, April 2003.

[HLS00] James Heather, Gavin Lowe, and Steve Schneider. How to Prevent Type Flaw Attacks on Security Protocols. In *Proceedings of the 13th IEEE Computer Security Foundations Workshop (CSFW 13)*, pages 255–268, July 2000.

[HRU76] Michael Harrison, Walter Ruzzo, and Jeffrey Ullman. Protection in operating systems. *Communications of the ACM*, 19(8):461–471, 1976.

[HT96] Nevin Heintze and Doug Tygar. A model for secure protocols and their composition. *IEEE Transactions on Software Engineering*, 22:16–30, 1996.

[KW96] Darrell Kindred and Jeannette Wing. Fast, automatic checking of security protocols. In *Proceedings of the 2nd USENIX workshop on e-commerce*, pages 41–52, 1996.

[Lam73] Butler W. Lampson. A note on the confinement problem. *Communications of the ACM*, 16(10):613–615, October 1973.

[Lam74] Butler W. Lampson. Protection. *ACM Operating Systems Review*, 8(1):18–24, 1974.

[LJ00] K. Rustan M. Leino and Rajeev Joshi. A Semantic Approach to Secure Information Flow. *Science of Computer Programming*, 37(1–3):113–138, 2000.

[Low96] Gavin Lowe. Breaking and fixing the Needham-Schroeder public key protocol using FDR. In *Proceedings of TACAS'96*, volume 1055 of *Lecture Notes in Computer Science*, pages 147–166, 1996.

[Low99] Gavin Lowe. Towards a completeness result for model checking of security protocols. *Journal of computer security*, 7:89–146, 1999.

[LR97] Gavin Lowe and Bill Roscoe. Using CSP to detect errors in the TMN protocol. *IEEE Transactions of Software Engineering*, 23(10):659–669, 1997.

[McL94] John McLean. Security models. In John Marciniak, editor, *Encyclopedia of Software Engineering*. Wiley & Sons Inc., 1994.

[Mea95] Catherine A. Meadows. Formal Verification of Cryptographic Protocols: (A Survey). In *Asiacrypt '94*, volume 917 of *Lecture Notes in Computer Science*, pages 133–150, 1995.

[Mea96a] Catherine A. Meadows. Analyzing the Needham-Schroeder public-key protocol. In E. Bertino, H. Kurth, G. Martella, and E. Montolivo, editors, *ESORICS'96*, volume 1146 of *Lecture Notes in Computer Science*, pages 351–364, 1996.

[Mea96b] Catherine A. Meadows. The NRL Protocol Analyzer: An overview. *Journal of Logic Programming*, 26(2):113–131, 1996.

[MMS97] John C. Mitchell, Mark Mitchell, and Ulrich Stern. Automated analysis of cryptographic protocols using Murφ. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 141–153, 1997.

[Mon99] David Monniaux. Abstracting cryptographic protocols with tree automata. In *Static analysis symposium*, volume 1694 of *Lecture Notes in Computer Science*, pages 149–163, 1999.

[MPW92] Robin Milner, Joachim Parrow, and David Walker. A calculus of mobile processes: parts I and II. *Information and Computation*, 100(1):1–77, 1992.

[MS01] Jonathan K. Millen and Vitaly Shmatikov. Constraint solving for bounded-process cryptographic protocol analysis. In *ACM Conference on Computer and Communications Security*, pages 166–175, 2001.

[Nes90] D. M. Nessett. A critique of the Burrows, Abadi and Needham logic. *ACM Operating systems review*, 24(2):35–38, 1990.

[NS78] 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.

[Pau98] Lawrence C. Paulson. The inductive approach to verifying cryptographic protocols. *Journal of computer security*, 6:85–128, 1998.

[RS01] R. Ramanujam and S.P. Suresh. Information based reasoning about security protocols. In *LACPV'01 (Logical Aspects of Cryptographic Protocol Verification)*, volume 56 of *Electronic Notes in Theoretical Computer Science*, pages 89–104, 2001.

[RS03a] R. Ramanujam and S.P. Suresh. A decidable subclass of unbounded security protocols. In Roberto Gorrieri, editor, *Proceedings of WITS'03*, pages 11–20, Poland, Warsaw, April 2003.

[RS03b] R. Ramanujam and S.P. Suresh. An equivalence on terms for security protocols. In Ramesh Bharadwaj, editor, *Proceedings of AVIS'03*, pages 45–56, Poland, Warsaw, April 2003.

[RS03c] R. Ramanujam and S.P. Suresh. Tagging makes secrecy decidable for unbounded nonces as well. In *Proceedings of 23rd FST&TCS*, Mumbai, India, December 2003. To appear.

[RSA78] Ronald L. Rivest, Adi Shamir, and Leonard M. Adelman. A Method for Obtaining Digital Signatures and Public-key Cryptosystems. *Communications of the ACM*, 21(2):120–126, February 1978.

[RT03] Michaël Rusinowitch and Mathieu Turuani. Protocol Insecurity with Finite Number of Sessions and Composed Keys is NP-complete. *Theoretical Computer Science*, 299:451–475, 2003.

[SBP01] Dawn Xiaodong Song, Sergey Berezin, and Adrian Perrig. Athena: A Novel Approach to Efficient Automatic Security Protocol Analysis. *Journal of Computer Security*, 9(1/2):47–74, 2001.

[SC01] Paul F. Syverson and Iliano Cervesato. The logic of authentication protocols. In Ricardo Focardi and Roberto Gorrieri, editors, *Foundations of Security Analysis and Design*, volume 2171 of *Lecture Notes in Computer Science*, pages 63–106, 2001.

[Sch96a] Steve Schneider. Security properties and CSP. In *Proceedings of the IEEE Computer Society Symposium on Security and Privacy*, pages 174–187, 1996.

[Sch96b] Bruce Schneier. *Applied Cryptography*. John Wiley & Sons, second edition, 1996.

[Sch98] Steve Schneider. Verifying Authentication Protocols in CSP. *IEEE Transactions on Software Engineering*, 24(9):741–758, 1998.

[Sto02] Scott D. Stoller. A Bound on Attacks on Authentication Protocols. In *Proceedings of the 2nd IFIP International Conference on Theoretical Computer Science*, pages 588–600, 2002. An extended version appeared as Indiana University, CS Dept., Technical Report 526, July 1999 (revised January 2001).

[SvO94] Paul F. Syverson and P.C. van Oorschot. On unifying some cryptographic protocol logics. In *Proceedings of the 13th IEEE Symposium on security and privacy*, pages 14–28. IEEE Press, 1994.

[VSI96] Dennis Volpano, Geoffrey Smith, and Cynthia Irvine. A sound type system for secure flow analysis. *Journal of Computer Security*, 4(3):1–21, 1996.