LOGICAL ANALYSIS AND COMPLEXITY OF SECURITY PROTOCOLS

A DISSERTATION
SUBMITTED TO THE DEPARTMENT OF COMPUTER SCIENCE
AND THE COMMITTEE ON GRADUATE STUDIES
OF STANFORD UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

Nancy A. Durgin
March 2003

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

John C. Mitchell
(Principal Advisor)

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

David Dill

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

Patrick Lincoln
(SRI)

Approved for the University Committee on Graduate Studies.

Abstract

We formalize the Dolev-Yao model of security protocols using two different notations, one based on multiset rewriting with existentials, and the other based on a process calculus combined with Hoare logic-style logical assertions. The goal is to provide simple formal notations for describing and proving security properties about protocols that use nonces (randomly generated numbers that uniquely identify a protocol session) and public- and symmetric-key cryptography. In the first part of the paper, we formalize the assumptions of the Dolev-Yao model and analyze the complexity of the secrecy problem under various restrictions. We use multiset rewriting to prove that, even for the case where we restrict the size of messages and the depth of message encryption, the secrecy problem is undecidable for the case of an unrestricted number of protocol roles and an unbounded number of new nonces. We also identify several decidable classes, including a DEXP-complete class when the number of nonces is restricted, and an NP-complete class when both the number of nonces and the number of roles is restricted. In the second part of the paper we define a logic, designed around a process calculus with actions for each possible protocol step, that consists of axioms about protocol actions and inference rules that yield assertions about protocols composed of multiple steps. Although assertions are written using only steps of the protocol, the logic is sound in a stronger sense: each provable assertion about an action or sequence of actions holds in any run of the protocol that contains the given actions and arbitrary additional actions by a malicious attacker. This approach lets us prove security properties of protocols under attack while reasoning only about the sequence of actions taken by honest parties to the protocol.

Acknowledgements

I would like to thank my advisor, Professor John Mitchell, for his guidance and support during my time at Stanford. Thanks also to many of the Computer Science Department faculty, particularly Dan Boneh, David Dill, Daphne Koller, Serge Plotkin, Eric Roberts, and Cynthia Dwork for their support and advice.

Thanks to the Aurora folks – Tom Dokter, Carl Stehle, Mark Isaak, Geof Cooper, Steve Anderson for their support and friendship, and especially to Luis Trabb-Pardo for his encouragement and support in my decision to return to school and pursue my "phud".

I would like to thank my co-authors: The results in Chapter 2 are from joint work with Iliano Cervesato, Pat Lincoln, John Mitchell and Andre Scedrov, and the results in Chapter 3 are from joint work with John Mitchell and Dusko Pavlovic. Thanks also to Rohit Chadha and Vitaly Shmatikov and the anonymous referees for their helpful comments.

My research at Stanford was supported by a Junglee Fellowship under the Stanford Graduate Fellowship program. The work in Chapter 2 was supported by DoD MURI "Semantic Consistency in Information Exchange," ONR Grant N00014-97-1-0505 and by the DoD University Research Initiative (URI) program administered by the Office of Naval Research under Grant N00014-01-1-0795. The work in Chapter 3 was supported by the Kestrel Institute, ONR MURI "Semantic Consistency in Information Exchange," N00014-97-1-0505, and ONR Grant "Games and Security in Systems of Autonomous Agents," N0014-00-C0495.

And finally, I would like to thank my family – my parents Mona and Chuck, and my sister Doranna – for their love and support.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Protocols based on cryptographic primitives are commonly used to protect access to computer systems and to protect transactions over the internet. Two well-known examples are the Kerberos authentication scheme [KNT94, KN93], used to manage encrypted passwords on clusters of interconnected computers, and the Secure Sockets Layer [FKK96], used by internet browsers and servers to carry out secure internet transactions.

Security protocol design and analysis is a difficult problem. Some of the difficulties come from subtleties of cryptographic primitives. Further difficulties arise because security protocols are required to work properly when multiple instances of the protocol are carried out in parallel, where a malicious intruder may combine data from separate sessions in order to confuse honest participants. Moreover, although the protocols themselves are often very simple, the security properties they are supposed to achieve are rather subtle and should be formulated with great care. Many security protocols have been published with subtle flaws that may be traced to insufficient rigor in formulating the premises about capabilities of participants.

In the literature on security protocol design and analysis, protocols are commonly described using an informal notation that leaves many properties of a protocol unspecified. For example, a short challenge-response section of a protocol might be written as:

$$\begin{aligned} A \longrightarrow B: &\quad \{n\}_K \\ B \longrightarrow A: &\quad \{f(n)\}_K \end{aligned}$$

In this notation, a message of the form $\{x\}_y$ consists of a plaintext x encrypted with key y. In this example protocol, Alice chooses a random number n and sends its encryption to Bob. There is no specific indication of how Bob determines what to send in response, but we can see that Bob returns a message that contains the encryption of $f(n)$. By analogy with familiar protocols, we might assume that he decrypts the message he receives to determine n, then applies f to n and returns the result to Alice (encrypted with the same key).

As written, the protocol description only gives an intended trace or family of traces involving the honest principals. There is no standard way of determining the initial conditions or assumptions about shared information, nor can we see how the principals will respond to messages that differ from those explicitly written. For example, in the case at hand, we must explain in English that K is assumed to be a shared key and that n is generated by Alice. Otherwise, it is a perfectly reasonable interpretation of the two lines above that Alice and Bob initially share a number n. In this case, Alice might send $\{n\}_K$ to Bob, with Bob returning $\{f(n)\}_K$ to Alice only if he receives precisely $\{n\}_K$. While the two readings of the protocol give the same sequence of messages when no one interferes with network transmission, the effects are different if an intruder intercepts the message from Alice to Bob and replaces it with another message. Hence it seems fair to say that the notation commonly found in the literature does not provide a rigorous basis for security protocol analysis.

In recent years, a variety of methods have been developed for analyzing and reasoning about security protocols. These approaches include specialized logics such as BAN logic [BAN89], special-purpose tools designed for cryptographic protocol analysis [KMM94], as well as theorem-proving [Pau97a, Pau97b] and model-checking methods using general purpose tools [Low96, Mea96a, MMS97, Ros95, Sch96, CJM98, SS98].

Although there are many differences among these approaches, most current formal approaches use the same basic model of adversary capabilities, which appears to have developed from positions taken by Needham and Schroeder [NS78] and a model presented by Dolev and Yao [DY83]. In this idealized setting, a protocol adversary is allowed to nondeterministically choose among possible actions. Messages are composed of indivisible abstract values, not sequences of bits, and encryption is modeled in an idealized way. The adversary may only send messages comprised of data it "knows" as the result of overhearing past transmissions.

In this thesis, we will present two formalisms for reasoning about security protocol in the Dolev-Yao model. In Chapter 2, we present a formalism based on multiset rewriting with existentials. We show how to use this formalism to describe security protocols and to explicitly model a generic intruder with the abilities described in the Dolev-Yao model. We also show how to reason about the properties of a specific protocol and about the meta-properties of security protocols in general. In Chapter 3, we define a logic designed around a process calculus with actions for each possible protocol step that consists of axioms about protocol actions and inference rules that yield assertions about protocols composed of multiple steps. We show that this logic is sound in the sense that each provable assertion about an action or sequence of actions holds in any run of a protocol that contains the given actions, plus the actions of honest participants and the arbitrary actions of a malicious attacker. This approach lets us prove security properties of protocols under attack while reasoning only about the sequence of actions taken by honest honest parties to the protocol.

Chapter 2

Multiset Rewriting

2.1 Introduction

The Dolev-Yao abstraction makes symbolic reasoning about cryptographic protocols a viable approach. Perhaps the simplest approach in this regard is to consider protocols as a form of rewriting, so that protocol execution could be carried out symbolically. This observation was sharpened to a rigorous, formal definition of the Dolev-Yao model by means of multiset rewriting with existential quantification, MSR, introduced in [Mit98, DM99b, CDL$^+$99]. In addition to rewriting to effect state transitions, we also needed a way to choose new values, such as nonces or keys. While this seems difficult to achieve directly in standard rewriting formalisms, the proof rules associated with existential quantification appear to be just what is required. Therefore, we have adopted a notation that may be regarded as an extension of multiset rewriting (see, e.g., [BM93, BB90]) with existential quantification. This formalism is quite palatable and quite close to the informal, traditional way of describing protocol message exchange, described above. Since its inception in [DM99b, CDL$^+$99], the MSR formalism has been applied and extended in several ways. MSR has been incorporated into a high-level specification language for authentication protocols, CAPSL [DM99a]. A typed version of MSR is studied in [Cer01c]. MSR has been successfully applied in the analysis of widely used protocols such as Kerberos 5 [BCJS02]. MSR is used as a formal setting for a game-based analysis of contract-signing protocols in [RC01].

The importance of existential quantification, for security protocols, is that it provides a direct mechanism for choosing a new value that is different from other values used in the execution of a system. Since many protocols involve choosing fresh nonces, fresh encryption keys, and so on, existential quantification seems like a useful primitive for describing security protocols. While existential quantification does not semantically imply there exist "new" values with certain properties, standard proof rules for manipulating existential quantifiers require introduction of fresh symbols (sometimes called Skolem constants). The way that existential quantification is used in our formalism is based on the standard existential elimination rule from natural deduction. If we have an existentially quantified axiom, $\exists x.\phi$, then this rule says that if we wish to prove some formula ψ, we can choose a new symbol y for the "x that is presumed to exist" and proceed to derive ψ from $[y/x]\phi$. The side condition "y not free in any other hypothesis in the proof of ψ" means that the only hypothesis in the proof of ψ that can contain y is the hypothesis $[y/x]\phi$.

Our multiset rewriting framework with existential quantification (MSR) may also be viewed as the existential Horn fragment of first-order linear logic [Gir87a]. The close connection between standard multiset rewriting (without existential quantification) and simple fragments of linear logic has been studied extensively [Asp87, MOM91, GG90b, Kan94] and extended in [CDKS00] to include parameters and existential quantification. Under this correspondence, every MSR transition sequence corresponds to a linear logic derivation in normal form, and conversely.

A linear logical framework automated tool *LLF* [CP96] may be used to simulate the execution of protocols, detect attacks, and construct formal proofs about protocol transformations [CDL+99]. A similar fragment of linear logic is used in [KOS98] as a basis for a specification language for real-time systems. Linear logic is also used to model the state-transition aspect of protocols, but not existential quantification for nonces, in [CD98, DMT98].

As presented in [DM99b, CDL+99], a protocol theory consists of three parts: a bounded phase describing protocol initialization that distributes keys or establishes other shared information, a role generation theory that designates possibly multiple roles that each principal may play in a protocol (such as initiator, responder, client, or server), and a disjoint union of bounded subtheories that each characterize a possible role. Encryption is typed, which prevents arbitrarily nested encryption terms. These syntactic restrictions, which are discussed in detail in the first part this chapter, make it possible to distinguish, in precise terms, protocols from general rewrite systems. This particular feature of the MSR formalism is a novel contribution to security protocol analysis; it seems to have no counterpart in the richer formalisms such as [AG99]. Furthermore, this feature of the MSR formalism allows us to identify two important parameters of a protocol itself: the number of roles and the number of new data (such as nonces or keys) introduced by the protocol.

Using our precise form of protocol theory, in the second part of this chapter we discuss in detail several decidability and complexity results regarding the secrecy property for protocols, most of which were established in [DM99b, CDL$^+$99]. Informally, a protocol satisfies secrecy if some privileged information (fixed in advance) will never be released to the adversary. In MSR this property may be stated as *unreachability*: global configuration in which the intruder is in possession of the specified secret is not reachable by protocol execution steps. Hence the failure of secrecy is stated as *reachability*. We show that secrecy is an undecidable property even if data constructors, message depth, message width, number of distinct roles, role length, and depth of encryption are bounded by constants. DEXPTIME-completeness of the failure of secrecy is shown for protocols further restricted to allow only a fixed number of new data. Furthermore, NP-completeness of the failure of secrecy is shown for protocols restricted even further to have a fixed number of roles. The latter upper bound has been recently strenghtened considerably in [RT01].

In some ways, undecidability might not be expected for protocols. The reason is that there is only a finite number of possible messages, except for the unbounded number of new nonces that repeated runs of a protocol might generate. However, our undecidability proof shows that nonces may be used as a form of "pointer," linking together messages that contain only simple data. However innocuous they may seem, nonces are at the heart of the problem in analyzing this class of security protocols. In the undecidability proof, the intruder stores encryptions of all atomic formulas derivable from a given existential Horn theory without function symbols, replaying these messages as needed in order for the protocol steps to carry out an arbitrary deduction. The undecidability of the implication problem for existential Horn clauses without function symbols follows from [CLM81] and may also be obtained directly by axiomatizing a Cook's-theorem-style Turing machine tableau [DLMS99]. If protocols are further restricted to generate no new data during execution, then DEXPTIME-hardness follows by the same encoding of Horn formulas (Datalog programs) as in our undecidability proof, applied to Horn clauses without function symbols and without existential quantification. For these Horn theories, DEXPTIME-hardness of the implication problem (measured as a function of the size of the theory) is implicit in [Imm86, Var82], as explained in [DEGV97].

Multiset rewriting formalism (MSR) is presented in Section 2.2 by means of several increasingly complex examples. In Section 2.3 we show how to represent security protocol theories in MSR, introducing the modeling of nonces, roles, the intruder, and encryption. A detailed example of the Needham-Schroeder Public Key Protocol in MSR is in Section 2.4. In Section 2.5 we show complexity results for security protocols under various restrictions. In Section 2.6 we show some examples of protocols that demonstrate some of the lower bounds in a more practical setting. In Section 2.7 we discuss related work, and finally in Section 2.8 we present conclusions.

2.2 Multiset Rewriting with Existential Quantification

2.2.1 Protocol Notation

We introduce a formalism for describing a class of nondeterministic infinite-state systems. The formalism is similar in many respects to standard rewrite systems [Klo87, Mit96], with two main differences. The first is that instead of representing information by a single expression, we use multisets of first-order atomic formulas. (A multiset is similar to a set, but with counting of duplicates.) The second main difference is that the formalism has a basic mechanism for choosing "new" symbols. This is important for modeling protocols that choose a new nonce or generate encryption keys.

Our formalism can also be viewed as a Horn fragment of linear logic [Gir87b, Asp87, MOM89, GG90a, Kan94, Cer95]. A similar fragment of linear logic is used in [KOS98] to represent real-time finite-state systems. Two other efforts using linear logic to model the state-transition aspect of protocols (but not existential quantification for nonces) are [CD98, DMT98].

The multiset rewriting notation is also related to the Chemical Abstract Machine formalism [BB90], with the primary difference being the addition of existentials.

The syntax involves terms, facts and rules. If we want to represent a system in this formalism, we begin by choosing a vocabulary, or *first-order signature.* This is a standard notion from multi-sorted first-order logic [End72].

Signatures A first-order signature consist of a set of sorts, together with function symbols and predicate symbols with specific sorts. The sorts indicate the kinds of data that will be used in the model. For example, the sorts used in a protocol may be key, msg, nonce for encryption keys, message contents and nonces. Function symbols are names for functions on the sorts of the signature. For example, an encryption function might have sort

$$\mathsf{encrypt} : \mathsf{key} \times \mathsf{msg} \rightarrow \mathsf{cipher}$$

where cipher is the sort for ciphertexts (encrypted text). In any signature, each function symbol must have a fixed set of parameter sorts (one for each function argument) and a result sort. A function with no arguments is called a *constant symbol.* Finally, a multi-sorted first-order signature has a set of predicate symbols, each with a fixed set of parameter sorts. It's also possible to consider order-sorted signatures [Gog78] and in fact we will find this convenient for specifying some example protocol theories in our formalism in Section 2.3.

Terms The terms over a signature are the set of expressions formed by applying functions to arguments. In each case, a function must be applied to arguments of the correct sort. For example, if $\mathsf{f} : s \rightarrow t$ and $x : s$, then $\mathsf{f}(x)$ is a well-formed term since the argument sort of f matches the sort of x. As suggested by this example, terms may contain variables, but each variable must have an associated sort. A variable is not allowed to be used with different sorts in different expressions associated with the same system. (All of this can be formalized using an inductive definition of the well-formed terms and their sorts, but we assume that most readers will be familiar with these standard concepts from logic.)

Facts A *fact* is a ground (i.e. variable-free) first-order atomic formula. This means that a fact is the result of applying a predicate symbol to ground terms of the correct sorts.

States A *state* is a multiset of facts (all over the same signature). In this paper we are only concerned with finite multisets.

Rules State transitions are written using two multisets of atomic formulas, in the following syntactic form:

$$F_1, \ldots, F_k \longrightarrow \exists x_1 \ldots \exists x_j . G_1, \ldots, G_n$$

The meaning of this rule is that if state S contains facts $\sigma F_1, \ldots \sigma F_k$ for some ground substitution σ, then one possible next state is the state S' that is similar to S, but with:

- facts $\sigma F_1, \ldots, \sigma F_k$ removed,
- $\sigma G_1, \ldots \sigma G_n$ added, where substitution σ replaces $x_1 \ldots x_j$ by new constant symbols.

While existential quantification does not semantically imply there exist "new" values with certain properties, standard proof rules for manipulating existential quantifiers require introduction of fresh symbols (sometimes called Skolem constants), as described in Section 2.2.3.

If there are free variables in the rule $F_1, \ldots, F_k \longrightarrow \exists x_1 \ldots \exists x_j . G_1, \ldots, G_n$, these are treated as universally quantified. In an application of a rule, these free variables may be replaced by any terms.

For example, consider the state

$$S = \{\mathsf{P}(\mathsf{f}(\mathsf{a})), \mathsf{P}(\mathsf{b})\}$$

and rule

$$\mathsf{P}(x) \longrightarrow \mathsf{P}(\mathsf{f}(x)).$$

Then one possible next state is obtained by using the substition $\sigma = [x \mapsto \mathsf{f}(\mathsf{a})]$, instantiating the rule to

$$\mathsf{P}(\mathsf{f}(\mathsf{a})) \longrightarrow \mathsf{P}(\mathsf{f}(\mathsf{f}(\mathsf{a}))).$$

With this substitution, we can remove $\mathsf{P}(\mathsf{f}(\mathsf{a}))$ from S and obtain the next state $S' = \{\mathsf{P}(\mathsf{f}(\mathsf{f}(\mathsf{a}))), \mathsf{P}(\mathsf{b})\}$.

We can then use a different instance of the rule, with substitution $\sigma' = [x \mapsto \mathsf{b}]$,

$$\mathsf{P}(\mathsf{b}) \longrightarrow \mathsf{P}(\mathsf{f}(\mathsf{b}))$$

to reach state $S'' = \{\mathsf{P}(\mathsf{f}(\mathsf{f}(\mathsf{a}))), \mathsf{P}(\mathsf{f}(\mathsf{b}))\}$. It is also possible to reach S'' from S by performing these replacements in the opposite order.

If a function is invertible, then this can also be expressed as a rule. For example, the rule

$$\mathsf{P}(\mathsf{f}(x)) \longrightarrow \mathsf{Q}(x)$$

involves recovering the data x from $\mathsf{f}(x)$. We will use rules of this form to describe decryption of encrypted messages.

An *MSR Theory* is defined by a signature and a set of rules. Given an MSR Theory and a state there is a set of *traces*, with each state reached from the previous one by applying one of the rules from the theory.

2.2.2 Example: Finite Automata

As a first example, without existential quantification, we describe a method for presenting finite-state automata in this notation. Assuming we have some specific automaton A, we choose a vocabulary for describing the input tape and the states of the automaton, and we have a rule corresponding to each state transition of the automaton. Each rule consumes an input and moves to the next state. The rules will depend on the specific automaton A, but the basic method can be applied to any automaton.

Sorts Given an automaton A, the signature for the theory $Th(A)$ has three sorts: st for automaton states, symb for input symbols and string for lists of input symbols.

Predicates We use predicates to represent the automaton state, and its current input string.

$\mathsf{State} : \mathsf{st}$	current state
$\mathsf{Input} : \mathsf{string}$	current input string

Functions We use the cons function to represent concatenation of strings,

$$\mathsf{cons} : \mathsf{symb} \times \mathsf{string} \rightarrow \mathsf{string}$$

For simplicity, we will write $\mathsf{a}{\cdot}x$ for $\mathsf{cons}(\mathsf{a}, x)$.

Constants We need names for the states of automaton A, and names for the symbols of the input alphabet, as well as a name for the empty string.

$\mathsf{q}_0, \mathsf{q}_1, \mathsf{q}_2, \ldots : \mathsf{st}$	finite set of states
$\mathsf{a}, \mathsf{b} : \mathsf{symb}$	input alphabet
$\mathsf{nil} : \mathsf{string}$	empty string

Rules There is one rule for each state transition of A. For example, here are some rules describing possible transitions between states q_0 and q_1:

$$\mathsf{State}(\mathsf{q}_0), \mathsf{Input}(\mathsf{a}{\cdot}x) \longrightarrow \mathsf{State}(\mathsf{q}_1), \mathsf{Input}(x)$$

$$\mathsf{State}(\mathsf{q}_0), \mathsf{Input}(\mathsf{b}{\cdot}x) \longrightarrow \mathsf{State}(\mathsf{q}_2), \mathsf{Input}(x)$$

$$\mathsf{State}(\mathsf{q}_1), \mathsf{Input}(\mathsf{a}{\cdot}x) \longrightarrow \mathsf{State}(\mathsf{q}_3), \mathsf{Input}(x)$$

$$\mathsf{State}(\mathsf{q}_1), \mathsf{Input}(\mathsf{b}{\cdot}x) \longrightarrow \mathsf{State}(\mathsf{q}_0), \mathsf{Input}(x)$$

A sample derivation gives us automata state transitions from q_0 to q_1 and back on input $\mathsf{a} \cdot \mathsf{b} \cdot \mathsf{nil}$. We will write this out as a sequence of states, starting with the multiset

$\{\mathsf{State}(\mathsf{q}_0), \mathsf{Input}(\mathsf{a} \cdot \mathsf{b} \cdot \mathsf{nil})\}$ that represents the automaton in state q_0 with input string $\mathsf{a} \cdot \mathsf{b} \cdot \mathsf{nil}$.

$$\begin{array}{rcl}\{\mathsf{State}(\mathsf{q}_0), \mathsf{Input}(\mathsf{a} \cdot \mathsf{b} \cdot \mathsf{nil})\} & \longrightarrow & \{\mathsf{State}(\mathsf{q}_1), \mathsf{Input}(\mathsf{b} \cdot \mathsf{nil})\} \\ & \longrightarrow & \{\mathsf{State}(\mathsf{q}_0), \mathsf{Input}(\mathsf{nil})\}\end{array}$$

It should be easy to see that if we begin with a system state consisting of one fact about the state of the automaton, and one fact about the input string, then we can only reach other system states of this form. In particular, we can never reach a system state where the automaton is in two states.

2.2.3 Existential Quantification

It is possible to give finite descriptions of infinite-state systems using function symbols. For example, if we have $\mathbf{0} : \mathsf{nat}$ and $\mathsf{suc} : \mathsf{nat} \to \mathsf{nat}$, then we can write expressions for arbitrarily many natural numbers. If each system state has a natural number, then we will have infinitely many possible system states.

Existential quantification provides an alternate way of expressing infinitely many possible states. As we will see in Section 2.3, the importance of existential quantification, for security protocols, is that it provides a direct mechanism for choosing a new value that is different from other values used in the execution of a system. Since many protocols involve choosing fresh nonces, fresh encryption keys, and so on, existential quantification seems like a useful primitive for describing security protocols.

The way that existential quantification is used in our formalism is based on the existential elimination rule from natural deduction. This proof rule is commonly written as follows.

$$(\exists\ \mathbf{elim}) \qquad \frac{\exists x.\phi \qquad \begin{array}{c}[c/x]\phi \\ \vdots \\ \psi\end{array}}{\psi} \quad \begin{array}{l}c \text{ does not occur in any} \\ \text{other hypothesis}\end{array}$$

If we have an existentially quantified axiom, $\exists x.\phi$, then this rule says that if we wish to prove some formula ψ, we can choose a new constant symbol c for the "x that is presumed to exist" and proceed to derive ψ from $[c/x]\phi$. The side condition "c does not occur in any other hypothesis in the proof of ψ" means that the only hypothesis in the proof of ψ that can contain c is the hypothesis $[c/x]\phi$.

2.2.4 Example: Turing Machine

We can see how existential quantification allows us to describe infinite-state systems by axiomatizing a Turing machine. This construction shows that MSR theories with existentials are undecidable. Later (in Appendix A.1), we will use other encodings of Turing machines to prove the undecidability and complexity lower bounds for security protocols in the presence of an attacker. Because of the attacker and the details of the protocols, the encoding used in those examples will be different.

Let us assume we have some specific Turing machine M. We choose a vocabulary for describing states of the machine and its input, and write rules to decribe transitions according to machine state and input. The rules will depend on the specific machine M, but the basic method can be applied to any Turing machine.

Sorts The signature for this theory has three sorts: state for the Turing machines states, cell for the cell names, and symbol for the cell contents.

Predicates The first predicate used in this example is used to describe the current machine state and tape position. The other two predicates describe the contents of a tape cell and the order (adjacency) between cells.

$\mathsf{Curr} : \mathsf{state} \times \mathsf{cell}$	current state, tape pos.
$\mathsf{Cont} : \mathsf{cell} \times \mathsf{symbol}$	contents of cell is symbol
$\mathsf{Adj} : \mathsf{cell} \times \mathsf{cell}$	keep cells in order

Constants We also need names for the states of the machine M, names for the cells at the beginning and end of the tape, and names for the symbols that may appear on the tape (0,1, and blank). The reason we have an end-of-tape cell, $\mathsf{c}_{\mathsf{eot}}$, is that we will represent an unbounded tape by including rules that will allow us to allocate as many tape cells as needed. In other words, we will represent the Turing machine tape by explicitly constructing the finite list of cells that the machine has looked at, one at a time.

$\mathsf{q}_0, \mathsf{q}_1, \mathsf{q}_2, \ldots : \mathsf{state}$	finite set of states
$\mathsf{c}_0, \mathsf{c}_1, \ldots, \mathsf{c}_{\mathsf{eot}} : \mathsf{cell}$	initial tape cells
$\mathbf{0}, \mathbf{1}, \square : \mathsf{symbol}$	tape symbols

Rules There are three classes of transition rules: tape maintenance rules, transition rules that correspond to Turing machine moves that move the head to the right, and transition rules that correspond to moving the head left. Initially, we will start the machine with two tape cells, the leftmost cell c_0 and the rightmost end-of-tape cell $\mathsf{c}_{\mathsf{eot}}$. This is expressed by the fact

$$\mathsf{Adj}(\mathsf{c}_0, \mathsf{c}_{\mathsf{eot}})$$

At any step in the computation, we can apply the tape maintenance rule

$$\mathsf{Adj}(c, \mathsf{c}_{\mathsf{eot}}) \longrightarrow \exists c'.\mathsf{Adj}(c, c'), \mathsf{Cont}(c', \square), \mathsf{Adj}(c', \mathsf{c}_{\mathsf{eot}})$$

Informally, this rule "says" that if cell c is adjacent to the end-of-tape cell, then we can allocate a new cell c' and place c' between c and the end-of-tape cell. The new cell will be blank. An example computation below shows how this rule can be used.

The rules for the actual moves of the Turing machine will depend on the structure of the specific machine M we wish to represent. Suppose that Turing machine M moves to the right, if it is in state q_i with symbol $\mathbf{0}$ on the tape cell currently under the tape head. If the move of M, in this case, is to state q_j, writing $\mathbf{1}$ into the tape cell, we will have a rule of the following form:

$$\mathsf{Curr}(q_i, c), \mathsf{Cont}(c, \mathbf{0}), \mathsf{Adj}(c, c') \longrightarrow \mathsf{Curr}(q_j, c'), \mathsf{Cont}(c, \mathbf{1}), \mathsf{Adj}(c, c')$$

If, instead of moving right, the machine would move left in this case, we would instead have the transition rule

$$\mathsf{Curr}(q_i, c), \mathsf{Cont}(c, \mathbf{0}), \mathsf{Adj}(c', c) \longrightarrow \mathsf{Curr}(q_j, c'), \mathsf{Cont}(c, \mathbf{1}), \mathsf{Adj}(c', c)$$

Note that moving to the right, we assume that there is a tape cell to the right of the tape head. This assumption can be satisfied by using the tape maintenance rule if needed before executing this Turing machine move.

Sample computation Although the exact moves will depend on the specific Turing machine that is represented by this method, we can illustrate the use of existential quantification by showing some example moves of a sample machine. Let us consider a Turing machine that reads the input tape until two consecutive **0**'s are read, then inserts a **1** after the both of them and moves to the left over the second **0**. Suppose that the machine starts in state q_0 and remains in this state until it reaches a cell containing **0**. At that point, the machine changes to state q_1 to "remember that it has seen a **0**" and moves right. Then the first few moves of the machine on input **100** might appear as follows:

$$\begin{aligned}
&\mathsf{Curr}(\mathsf{q}_0, \mathsf{c}_0), \mathsf{Cont}(\mathsf{c}_0, \mathbf{1}), \mathsf{Cont}(\mathsf{c}_1, \mathbf{0}), \mathsf{Cont}(\mathsf{c}_2, \mathbf{0}),\\
&\qquad\qquad \mathsf{Adj}(\mathsf{c}_0, \mathsf{c}_1), \mathsf{Adj}(\mathsf{c}_1, \mathsf{c}_2), \mathsf{Adj}(\mathsf{c}_2, \mathsf{c}_{\mathsf{eot}})\\
&\longrightarrow \mathsf{Curr}(\mathsf{q}_0, \mathsf{c}_1), \mathsf{Cont}(\mathsf{c}_0, \mathbf{1}), \mathsf{Cont}(\mathsf{c}_1, \mathbf{0}), \mathsf{Cont}(\mathsf{c}_2, \mathbf{0}),\\
&\qquad\qquad \mathsf{Adj}(\mathsf{c}_0, \mathsf{c}_1), \mathsf{Adj}(\mathsf{c}_1, \mathsf{c}_2), \mathsf{Adj}(\mathsf{c}_2, \mathsf{c}_{\mathsf{eot}})\\
&\longrightarrow \mathsf{Curr}(\mathsf{q}_1, \mathsf{c}_2), \mathsf{Cont}(\mathsf{c}_0, \mathbf{1}), \mathsf{Cont}(\mathsf{c}_1, \mathbf{0}), \mathsf{Cont}(\mathsf{c}_2, \mathbf{0}),\\
&\qquad\qquad \mathsf{Adj}(\mathsf{c}_0, \mathsf{c}_1), \mathsf{Adj}(\mathsf{c}_1, \mathsf{c}_2), \mathsf{Adj}(\mathsf{c}_2, \mathsf{c}_{\mathsf{eot}})
\end{aligned}$$

At this point, the appropriate transition will be a move to the right on to the next cell, where the machine will write a **1**. However, this would place the tape head over the special "end-of-tape" marker. Since we would like the machine to proceed as if the tape were infinite, we must use the tape maintenance rule

$$\mathsf{Adj}(c, \mathsf{c}_{\mathsf{eot}}) \longrightarrow \exists c'.\mathsf{Adj}(c, c'), \mathsf{Cont}(c', \Box), \mathsf{Adj}(c', \mathsf{c}_{\mathsf{eot}})$$

to insert a new cell in front of the end-of-tape cell. This gives us the transition

$$\begin{array}{l}\mathsf{Curr}(\mathsf{q}_1,\mathsf{c}_2),\mathsf{Cont}(\mathsf{c}_0,\mathbf{1}),\mathsf{Cont}(\mathsf{c}_1,\mathbf{0}),\mathsf{Cont}(\mathsf{c}_2,\mathbf{0}),\\ \qquad\qquad \mathsf{Adj}(\mathsf{c}_0,\mathsf{c}_1),\mathsf{Adj}(\mathsf{c}_1,\mathsf{c}_2),\mathsf{Adj}(\mathsf{c}_2,\mathsf{c}_{\mathsf{eot}})\\ \longrightarrow \mathsf{Curr}(\mathsf{q}_1,\mathsf{c}_2),\mathsf{Cont}(\mathsf{c}_0,\mathbf{1}),\mathsf{Cont}(\mathsf{c}_1,\mathbf{0}),\mathsf{Cont}(\mathsf{c}_2,\mathbf{0}),\mathsf{Cont}(\mathsf{c}_3,\Box),\\ \qquad\qquad \mathsf{Adj}(\mathsf{c}_0,\mathsf{c}_1),\mathsf{Adj}(\mathsf{c}_1,\mathsf{c}_2),\mathsf{Adj}(\mathsf{c}_2,\mathsf{c}_3),\mathsf{Adj}(\mathsf{c}_3,\mathsf{c}_{\mathsf{eot}})\end{array}$$

which inserts a new blank cell, c_3 in front of the end-of-tape marker. Then the machine head may be moved right over the new blank square and write a **1**. Although it would be possible to apply a transition rule moving the machine head over the end-of-tape marker, the end-of-tape marker has no contents. Since each machine move requires a tape cell with some contents (possibly including the blank $\Box$), a derivation that places the Turing machine head over the end-of-tape marker will "hang" the machine and have no effect on the set of accepting computations.

In this example we have used existential quantification to avoid the unbounded use of function applications. It would be possible to implement a Turing machine without existentials by using skolem functions.

2.2.5 Creation, Consumption, Persistence

Some preliminary definitions involve the ways that a fact may be created, preserved, or consumed by a rule. While multiple copies of some facts may be needed in some derivations, we are able to eliminate the need for multiple copies of certain facts.

Definition 2.2.1. Assume $\mathcal{T}$ is a theory and P is a predicate. Any rule has the form $l \longrightarrow r$, where l is the facts $F_1, \ldots, F_k$ on the left hand side, and r is the facts $G_1, \ldots, G_n$, possibly with one or more existential quantifiers, on the right hand side. A rule in a theory $\mathcal{T}$ *creates P facts* if some $P(\vec{t})$ occurs more times in r than in l. A rule in a theory $\mathcal{T}$ *preserves P facts* if every $P(\vec{t})$ occurs the same number of times in r and l. A rule in a theory $\mathcal{T}$ *consumes P facts* if some fact $P(\vec{t})$ occurs more times in l than in r. A predicate P in a theory $\mathcal{T}$ is *persistent* if every rule in $\mathcal{T}$ which contains P either creates or preserves P facts.

As an example, a rule of form

$$Q(\vec{x}) \longrightarrow Q(\vec{y})$$

does not preserve Q facts, since it can be used to create a fact $Q(\vec{t})$ and consume a fact $Q(\vec{s})$.

Since a persistent fact is never consumed by any rule, there is no need to generate more than one copy of a particular fact – as long as that fact is never needed more than once by a single rule. However, by simple transformation, it is possible to eliminate the need for more than one copy of any persistent fact.

For example, a rule of form:

$$P(\vec{x}), P(\vec{y}), \ldots \longrightarrow Q(\vec{x}, \vec{y}), P(\vec{x}), P(\vec{y}), \ldots$$

(with P a persistent predicate) can be replaced by rules of form:

$$\begin{array}{lcl} P(\vec{x}) & \longrightarrow & P_1(\vec{x}), P(\vec{x}) \\ P(\vec{x}) & \longrightarrow & P_2(\vec{x}), P(\vec{x}) \\ P_1(\vec{x}), P_2(\vec{y}), \ldots & \longrightarrow & Q(\vec{x}, \vec{y}), P_1(\vec{x}), P_2(\vec{y}), \ldots \end{array}$$

(where P_1 and P_2 are persistent predicates).

Definition 2.2.2. A rule $l \longrightarrow r$ in a theory $\mathcal{T}$ is a *single-persistent rule* if all predicates that are persistent in theory $\mathcal{T}$ appear at most once in l. A theory $\mathcal{T}$ is a *uniform theory* if all rules in $\mathcal{T}$ are single-persistent rules.

Since any theory can be rewritten as a uniform theory, we will assume that all theories discussed from this point forward are uniform theories.

Definition 2.2.3. Let **P** be a set of predicates, each persistent in a uniform theory $\mathcal{T}$. Two states S and S' are ***P-similar*** (denoted $S \simeq_P S'$) if, after removing all duplicate persistent P facts from each state, they are equal multisets.

Lemma 2.2.1. *If $S \simeq_P S'$ and $S \stackrel{\mathcal{T}}{\twoheadrightarrow} T$, then $\exists T'.T \simeq_P T'$ with $S' \stackrel{\mathcal{T}}{\twoheadrightarrow} T'$.*

Proof We construct the derivation $S' \xrightarrow{\mathcal{T}} T'$ as follows: We use the same rules and substitutions as the derivation $S \xrightarrow{\mathcal{T}} T$. This derivation is valid because all rules are single-persistent, so any rules and substitution used in the original derivation will also work in the second derivation (all necessary facts are available to enable the rules). $\square$

2.2.6 Equality and Disequality

The basic MSR framework defined in Section 2.2.1 can be extended with tests for disequality of terms using $\neq$ conditions in rules. In the extension *MSR*$_{\neq}$, these conditions are allowed only on the left hand sides of rules, and are not considered to be facts.

We illustrate this by example. Given a rule of form

$$P_1(t_1, t_2, \ldots), P_2(u_1, u_2, \ldots), t_2 \neq u_2 \longrightarrow Q_1(\ldots), Q_2(\ldots)$$

If a state S has facts P_1 and P_2 for terms $t_1, t_2, \ldots$ and $u_1, u_2, \ldots$ where t_2 is different from u_2, then a possible next state is S' with facts P_1 and P_2 replaced by facts Q_1 and Q_2.

To summarize, MSR$_{\neq}$ is the extension of MSR with these extended rules, keeping the same signature, terms and facts as defined in Section 2.2.1.

We do not need to add a condition to test for equality, because it is expressible by matching the names of the variables in the terms. For example, the set of facts $\{\mathsf{P}(\mathsf{a}, \mathsf{b}, \mathsf{c}), \mathsf{Q}(\mathsf{a}, \mathsf{d}, \mathsf{e})\}$ matches the left hand side of the rule

$$\mathsf{P}(x, y, z), \mathsf{Q}(x, v, w), y \neq v \longrightarrow \mathsf{R}(\ldots), \mathsf{S}(\ldots)$$

While the set $\{\mathsf{P}(\mathsf{a}, \mathsf{b}, \mathsf{c}), \mathsf{Q}(\mathsf{a}, \mathsf{b}, \mathsf{e})\}$ does not. The rule requires that the first two arguments of the facts for predicates P and Q be the same, and the second two arguments be different.

Note that since in MSR each nonce generated by an existential is unique, with disequality test the correspondence mentioned in Section 2.2.3 breaks down; the existentials in MSR no longer correspond to those in standard logical deduction.

$$
\begin{array}{lcl}
A & \longrightarrow & B : \{\!|A, n_a|\!\}_{K_b} \\
B & \longrightarrow & A : \{\!|n_a, n_b|\!\}_{K_a} \\
A & \longrightarrow & B : \{\!|n_b|\!\}_{K_b}
\end{array}
$$

Table 2.1: Needham-Schroeder Public Key Protocol

2.3 Multiset Rewriting for Protocol Theories

In this section we will explain the form of an MSR theory for a class of security protocols that use Public Key encryption. We will make an incremental presentation, starting with some simple protocol roles, then introducing the Dolev-Yao intruder model and our encryption model, and finally defining the requirements for a two-phase intruder theory. An example of a full theory for a a public key protocol is presented in Section 2.4.

Needham-Schroeder Public Key Protocol We will use the Needham-Schroeder Public Key protocol [NS78] as a running example throughout this paper. The complete core protocol, which omits the steps that use a trusted server to distribute the public keys, is shown in Table 2.1, using a common informal notation.

In the first step, the initiator A (commonly referred to as "Alice") sends a message to the responder B (commonly referred to as "Bob"). The message contains Alice's name, and a freshly chosen nonce, n_a (typically a large random number). The message is encrypted with Bob's public key, which means only somebody with Bob's private key can decrypt it and understand it's contents.

In the second step, Bob replies with a nonce of his own, n_b, along with Alice's nonce, both encrypted with Alice's public key.

In the final step, Alice replies by returning Bob's nonce, encrypted with his public key.

2.3.1 Protocol Theories

During a network transaction involving an implemented security protocol, several activities take place, possibly simultaneously. These include key generation, key distribution, and initiation of a protocol session by a specific participant. These activites can be arbitrarily interleaved. For example, some public key protocol sessions can take place between Alice and Bob, and then later a new participant Carol might join them by obtaining a public key certificate so that she can also converse with Alice and Bob in future sessions.

Here we introduce the notion of a protocol *role* – specific steps of the protocol meant to be carried out by a single principal. These are refered to as *local protocols* by Woo and Lam [WL93]. A typical protocol includes at least an initiator and a responder role, and often includes a trusted third party or a server. Protocol analysis concerns the interaction of an arbitrary number of instances of arbitrary assignments of principals to roles, in the presence of an intruder who can replay messages and parts of messages (i.e. a Dolev-Yao intruder).

In our model we separate the protocol execution into stages. There is an implicit or explicit initialization phase that distributes keys or establishes other shared information. Following this initialization phase, each agent may choose to carry out the protocol any number of times, in any combination of roles. For example a principal A may play the role of initiator twice, and responder once, during the course of a single attack. We incorporate these ideas into our formal definitions by letting a protocol theory consist of an initialization theory, a role generation theory, and the disjoint union of bounded subtheories that each characterize a possible role. We identify the syntactic form of a class of *well-founded protocol theories*.

It is relatively straightforward to use the multiset rewriting framework summarized in the preceding section to describe finite-state and infinite-state systems. Using function symbols, it is possible to describe computation over unbounded data types. In particular, it is easy to encode counter machines or Turing machines (as we did in Section 2.2.4), implying that secrecy is undecidable. However, the principal authentication and secrecy protocols of interest are all of bounded length, and most use data of bounded complexity (see [CJ97] for a relevant survey). So, it seems reasonable for our model to represent protocols in a way that reflects their bounded nature. Thus we assume that the initialization steps are bounded, and that initialization can be completed prior to the execution of the protocol steps proper. We also formally define protocol role theories as *bounded role theories*.

Definition 2.3.1. A rule $R = l \longrightarrow r$ *enables* a rule $l' \longrightarrow r'$ if there exist substitutions σ, σ' such that some fact $P(\vec{t}) \in \sigma r$, is also in $\sigma' l'$. A theory $\mathcal{T}$ *precedes* a theory $\mathcal{S}$ if no rule in $\mathcal{S}$ enables a rule in $\mathcal{T}$.

Intuitively, if a theory $\mathcal{T}$ precedes a theory $\mathcal{S}$, then no facts that appear in the left hand side of rules in $\mathcal{T}$ are created by rules that are in $\mathcal{S}$.

Definition 2.3.2. A theory $\mathcal{A}$ is a *bounded role theory* if there is a finite list of predicates called the *role states* and written $S_0, S_1, \ldots, S_k$ for some k, such that for each rule $l \longrightarrow r$ there is exactly one occurence of a state predicate in l, say S_i, and there is exactly one occurrence of a state predicate in r, say S_j. Furthermore, it must be the case that $i < j$. We call the first role state, S_0, an *initial role state*.

By defining roles in this way, we ensure that each application of a rule in $\mathcal{A}$ advances the state forward. Each instance of a role can only result in a finite number of steps in the derivation.

Definition 2.3.3. If $\mathcal{A}_1, \ldots, \mathcal{A}_k$ is a set of bounded role theories, a *role generation theory* is a set of rules of the form

$$P(\vec{s}), Q(\vec{t}), \ldots \longrightarrow S_i(\vec{r}), P(\vec{s}), Q(\vec{t}), \ldots$$

where $P(\vec{s}), Q(\vec{t}), \ldots$ is a finite list of persistent facts not involving any role states, and S_i is the initial role state for one of $\mathcal{A}_1, \ldots, \mathcal{A}_k$.

Definition 2.3.4. A theory $\mathcal{S} \subset \mathcal{T}$ is a *bounded sub-theory* of $\mathcal{T}$ if all formulas on the right hand side of the rules R in $\mathcal{S}$ either contain existentials or are persistent in $\mathcal{T}$.

Definition 2.3.5. A theory $\mathcal{P}$ is a *well-founded protocol theory* if $\mathcal{P} = \mathcal{I} \uplus \mathcal{R} \uplus \mathcal{A}_1 \uplus \ldots \uplus \mathcal{A}_n$ where $\mathcal{I}$ is a bounded sub-theory (called the *initialization theory*) not involving any role states, $\mathcal{R}$ is a role generation theory involving only facts created by $\mathcal{I}$ and the initial roles states of $\mathcal{A}_1, \ldots, \mathcal{A}_n$, and $\mathcal{A}_1, \ldots, \mathcal{A}_n$ are bounded role theories, with $\mathcal{I}$ preceding $\mathcal{R}$ and $\mathcal{R}$ preceding $\mathcal{A}_1, \ldots, \mathcal{A}_n$. For role theories $\mathcal{A}_i$ and $\mathcal{A}_j$, with $i \neq j$, no role state predicate that occurs in $\mathcal{A}_i$ can occur in $\mathcal{A}_j$, and vice-versa.

This form allows derivations in a protocol theory to be broken down into three stages – the initialization stage, the role generation stage, and the protocol execution stage. Tables 2.5 and 2.6 show examples of these theories for the Needham-Schroeder Protocol.

Lemma 2.3.1. *Given a well-founded protocol theory $\mathcal{P} = \mathcal{I} \uplus \mathcal{R} \uplus \mathbf{A}$, where $\mathcal{I}$ is an initialization theory, $\mathcal{R}$ is a role generation theory, and $\mathbf{A}$ is the union of one or more bounded role theories, if $S \xrightarrow{\mathcal{P}} T$ is a derivation over $\mathcal{P}$, then there exists a derivation $S \xrightarrow{\mathcal{I}} S'$, $S' \xrightarrow{\mathcal{R}} S''$ and $S'' \xrightarrow{\mathbf{A}} T$, where all rules from $\mathcal{I}$ are applied before any rules from $\mathcal{R}$, and all rules from $\mathcal{I}$ and $\mathcal{R}$ are applied before any rules from $\mathbf{A}$.*

Proof. Since $\mathcal{P}$ is a well-founded protocol theory, we know that $\mathcal{I}$ precedes $\mathcal{R}$ and $\mathcal{I}$ and $\mathcal{R}$ precede all of the theories in $\mathbf{A}$. Since no rules in $\mathcal{R}$ can enable rules in $\mathcal{I}$, all rules in $\mathcal{I}$ can be applied before any rules in $\mathcal{R}$. Similarly, since no rules in $\mathbf{A}$ can enable rules in $\mathcal{I}$ or $\mathcal{R}$, all rules from $\mathcal{I}$ and $\mathcal{R}$ can be applied before any rules from $\mathbf{A}$. $\square$

2.3.2 Encryption-Free Needham-Schroeder

As a means of explaining the Dolev-Yao intruder and encryption models using our notation, we begin with an overly simplified form of the Needham-Schroeder protocol. Without encryption, the Needham-Schroeder protocol proceeds as follows:

$$\begin{array}{ccccl} A & \longrightarrow & B & : & n_a \\ B & \longrightarrow & A & : & n_a, n_b \\ A & \longrightarrow & B & : & n_b \end{array}$$

where n_a and n_b are fresh nonces, chosen by Alice (A) and Bob (B), respectively.

Sorts The full Needham-Schroeder protocol uses several sorts, but here the data is all nonces, so we need only one sort, nonce. Later when we add more sorts, we will find it convenient to define nonce as a subsort of msg.

Predicates We can describe this simplified protocol in our notation using the predicates $\mathsf{A_i}$, $\mathsf{B_i}$, $\mathsf{N_i}$ for $0 \leq i \leq 3$, with the following intuitive meaning:

$\mathsf{A_0}(\ )$	Alice in state 0 (initial role state)
$\mathsf{A_1}(\mathsf{nonce})$	Alice in state 1, with her nonce
$\mathsf{A_2}(\mathsf{nonce}, \mathsf{nonce})$	Alice in state 2, with two nonces
$\mathsf{B_0}(\ )$	Bob in state 0 (initial role state)
$\mathsf{B_1}(\mathsf{nonce}, \mathsf{nonce})$	Bob in state 1, with two nonces
$\mathsf{B_2}(\mathsf{nonce}, \mathsf{nonce})$	Bob in state 2, with two nonces
$\mathsf{N_1}(\mathsf{nonce})$	Network has message 1, with indicated data
$\mathsf{N_2}(\mathsf{nonce}, \mathsf{nonce})$	Network has message 2, with indicated data
$\mathsf{N_3}(\mathsf{nonce})$	Network has message 3, with indicated data

The data associated with the state of some principal, or a network message, will depend on the particular state or message. Each principal begins in local state 0, with no data. Therefore, predicates $\mathsf{A_0}$ and $\mathsf{B_0}$ are predicates with no arguments. When Alice chooses a nonce, she moves into local state 1. Therefore, predicate $\mathsf{A_1}$ is a unary predicate of type nonce, intended to be the nonce chosen by Alice. Similarly, predicate $\mathsf{B_1}$ is a binary predicate of type $\mathsf{nonce} \times \mathsf{nonce}$, the data received from Alice in message one of the protocol and the nonce chosen by Bob for his response.

The subscripts on the message predicates $\mathsf{N_i}$ indicate which message of the protocol is being sent, which implicitly indicates the signature of the message. This format allows participants to distinguish the messages of a protocol. Since we will be considering an environment which includes an intruder (introduced in Section 2.3.3) that can transform any message from one type to another, this notation will not limit the analysis in any way.

Rules Using these predicates, we can state the protocol using four transition rules:

$$\begin{array}{rcl}
\mathsf{A_0}() & \longrightarrow & \exists x.\mathsf{A_1}(x), \mathsf{N_1}(x) \\
\mathsf{B_0}(), \mathsf{N_1}(x) & \longrightarrow & \exists y.\mathsf{B_1}(x, y), \mathsf{N_2}(x, y) \\
\mathsf{A_1}(x), \mathsf{N_2}(x, y) & \longrightarrow & \mathsf{A_2}(x, y), \mathsf{N_3}(y) \\
\mathsf{B_1}(x, y), \mathsf{N_3}(y) & \longrightarrow & \mathsf{B_2}(x, y)
\end{array}$$

$$\begin{array}{rrcll}
\mathsf{B}_0(), & \mathsf{A}_0() & \longrightarrow & \mathsf{A}_1(\mathsf{n_a}), \mathsf{N}_1(\mathsf{n_a}), & \mathsf{B}_0() \\
\mathsf{A}_1(\mathsf{n_a}), & \mathsf{B}_0(), \mathsf{N}_1(\mathsf{n_a}) & \longrightarrow & \mathsf{B}_1(\mathsf{n_a}, \mathsf{n_b}), \mathsf{N}_2(\mathsf{n_a}, \mathsf{n_b}), & \mathsf{A}_1(\mathsf{n_a}) \\
\mathsf{B}_1(\mathsf{n_a}, \mathsf{n_b}), & \mathsf{A}_1(\mathsf{n_a}), \mathsf{N}_2(\mathsf{n_a}, \mathsf{n_b}) & \longrightarrow & \mathsf{A}_2(\mathsf{n_a}, \mathsf{n_b}), \mathsf{N}_3(\mathsf{n_b}), & \mathsf{B}_1(\mathsf{n_a}, \mathsf{n_b}) \\
\mathsf{A}_2(\mathsf{n_a}, \mathsf{n_b}), & \mathsf{B}_1(\mathsf{n_a}, \mathsf{n_b}), \mathsf{N}_3(\mathsf{n_b}) & \longrightarrow & \mathsf{B}_2(\mathsf{n_a}, \mathsf{n_b}), & \mathsf{A}_2(\mathsf{n_a}, \mathsf{n_b})
\end{array}$$

Table 2.2: Sample Trace of Encryption-Free Needham-Schroeder

Each rule corresponds to an action by a principal. In the first rule, Alice chooses a nonce, sends it on the network, and remembers the nonce by moving into a local state that retains the nonce value. In the second step, Bob receives a message on the network, chooses his own nonce, transmits it and saves both nonces in his local state. In the third step, Alice receives Bob's message and replies, while in the fourth step Bob receives Alice's final message and changes state.

If we group the transition rules into roles,

$$\begin{aligned}
\mathcal{A} &= \{\ \mathsf{A}_0() \longrightarrow \exists x.\mathsf{A}_1(x), \mathsf{N}_1(x), \quad \mathsf{A}_1(x), \mathsf{N}_2(x,y) \longrightarrow \mathsf{A}_2(x,y), \mathsf{N}_3(y)\ \} \\
\mathcal{B} &= \{\ \mathsf{B}_0(), \mathsf{N}_1(x) \longrightarrow \exists y.\mathsf{B}_1(x,y), \mathsf{N}_2(x,y), \quad \mathsf{B}_1(x,y), \mathsf{N}_3(y) \longrightarrow \mathsf{B}_2(x,y)\ \}
\end{aligned}$$

we see that $\mathcal{A}$ and $\mathcal{B}$ are bounded role theories, where A_0 and B_0 are initial role state, and A_1, A_2, B_1, and B_2 are role states.

Sample Computation In Table 2.2 is a sample trace generated from these rules, beginning from state A_0, B_0. Spacing is used to separate the facts that participate in each step from those that do not.

2.3.3 Formalizing the Intruder

One of the original motivations for using multiset rewriting for protocol analysis was that this framework allows us to use essentially the same theory for all adversaries that follow the Dolev-Yao model, for all protocols. The precise formulation of the intruder will depend on the message format of the protocol being attacked, and on the type of encryption used, but the basic form of the *standard intruder* will be the same under the Dolev-Yao model.

The Dolev-Yao protocol adversary or "intruder" may nondeterministically choose among the following actions at each step:

- Read any message and block further transmission
- Decompose a message into parts and remember them, including decrypting any message for which the adversary has obtained the key
- Generate fresh data as needed
- Compose a new message from known data and send

By combining a read with resend, we can easily obtain the effect of passively reading a message without preventing another party from also receiving it.

There are two main parts of the Dolev-Yao model as commonly used in protocol analysis. The first is the set of possible intruder actions, applied nondeterministically throughout execution of the protocol. The second is a "black-box" model of encryption and decryption. We explain the intruder actions here, along with specifying some formal properties that are used to bound the number of intruder steps needed to produce a given message. The encryption model is presented in Section 2.3.4.

Sorts We still have the sort nonce, but as we will see in Section 2.3.4, it is convenient for the intruder model to use the sort msg, with nonce a subsort of msg.

Functions We introduce a new function for pairing, which is abbreviated as

$$\langle \_, \_ \rangle : \mathsf{msg} \times \mathsf{msg} \rightarrow \mathsf{msg}$$

Predicates We introduce the basic predicates D, M, and C for the intruder, with the following intuitive meaning:

$\mathsf{D}(\mathsf{msg})$	Decomposable messages known to intruder
$\mathsf{M}(\mathsf{msg})$	Information stored in intruder "memory"
$\mathsf{C}(\mathsf{msg})$	Composable messages known to intruder

Later, when we include encryption in our model and more sortnames are added, these predicates will become more complex, eventually reaching the form shown in Table 2.7.

Rules In our model, the intruder processes data in two phases. The first stage is to read and decompose data into parts and remember the parts, and the second stage is to compose a message from the parts it remembers. We will discuss the two-phase intruder more formally in Section 2.3.5. We illustrate the basic form of the intruder actions for an encryption-free protocol using two of the network-message predicates from the previous example, $\mathsf{N}_1(\mathsf{nonce})$ and $\mathsf{N}_2(\mathsf{nonce},\mathsf{nonce})$. Using predicates D for decomposable messages and M for the intruder "memory", the basic rules for intercepting, decomposing and remembering messages are

$$\begin{array}{lcl} \mathsf{N}_1(x) & \longrightarrow & \mathsf{D}(x) \\ \mathsf{N}_2(x,y) & \longrightarrow & \mathsf{D}(\langle x,y\rangle) \\ \mathsf{D}(\langle x,y\rangle) & \longrightarrow & \mathsf{D}(x),\mathsf{D}(y) \\ \mathsf{D}(z) & \longrightarrow & \mathsf{M}(z) \end{array}$$

The rules for composing messages from parts are written using the C, for "composable", predicate as follows:

$$\begin{array}{lcl} \mathsf{M}(x) & \longrightarrow & \mathsf{C}(x),\mathsf{M}(x) \\ \mathsf{C}(x) & \longrightarrow & \mathsf{N}_1(x) \\ \mathsf{C}(x),\mathsf{C}(y) & \longrightarrow & \mathsf{C}(\langle x,y\rangle) \\ \mathsf{C}(\langle x,y\rangle) & \longrightarrow & \mathsf{N}_2(x,y) \end{array}$$

The rule for generating new data is

$$\longrightarrow \exists x.\mathsf{M}(x)$$

The reason we need the last transition rule (which can be applied any time without any hypothesis) is that the intruder may need to choose new data in order to trick an honest participant in a protocol. This is illustrated in the sample computation shown below.

Sample Computation An attack on the encryption-free (and obviously insecure) portion of the Needham-Schroeder protocol is shown in Table 2.3. Here we have the actions of the honest participants in the left column and the actions of the intruder indented. For simplicity, duplicate copies of $\mathsf{M}(\;)$ facts are not shown, since these have no effect on the execution of the protocol or intruder.

In this attack, the intruder intercepts messages between A and B, replacing data so that the two principals have a different view of the messages that have been exchanged. Specifically, the intruder replaces Alice's nonce $\mathsf{n_a}$ by a value n chosen by the intruder. When Bob responds to the altered message, the intruder intercepts the result and replaces n by $\mathsf{n_a}$ so that Alice receives the message she expects. Introducing encryption eliminates this attack.

2.3.4 Modeling Perfect Encryption

The commonly used "black-box" model of encryption may be written in our multiset notation using the following vocabulary. For concreteness, we discuss public-key encryption. Symmetric or private-key encryption can be characterized similarly. For simplicity we will identify principal identities with their public keys.

Sorts We introduce several new sorts, including cipher for ciphertext, $\mathsf{d\_key}$ for decryption keys and $\mathsf{e\_key}$ for encryption keys. Since data can be transformed by encryption into a different type, and the type of an encrypted message can't be known until the message is decrypted, we choose to introduce an order-sorted algebra [Gog78]. This approach also serves to keep the signatures of our functions and predicates reasonably simple.

$\mathsf{B}_0(), \mathsf{A}_0()$	Initial configuration
$\longrightarrow \mathsf{A}_1(\mathsf{n_a}), \mathsf{N}_1(\mathsf{n_a}), \mathsf{B}_0()$	Alice chooses nonce and sends
$\longrightarrow \mathsf{A}_1(\mathsf{n_a}), \mathsf{B}_0(), \mathsf{D}(\mathsf{n_a})$	Intruder intercepts message $\mathsf{n_a}$
$\longrightarrow \mathsf{A}_1(\mathsf{n_a}), \mathsf{B}_0(), \mathsf{M}(\mathsf{n_a})$	Intruder learns value $\mathsf{n_a}$
$\longrightarrow \mathsf{A}_1(\mathsf{n_a}), \mathsf{B}_0(), \mathsf{M}(\mathsf{n_a}), \mathsf{M}(n)$	Intruder generates fresh value n
$\longrightarrow \mathsf{A}_1(\mathsf{n_a}), \mathsf{B}_0(), \mathsf{M}(\mathsf{n_a}), \mathsf{M}(n), \mathsf{C}(n)$	Intruder begins composing message
$\longrightarrow \mathsf{A}_1(\mathsf{n_a}), \mathsf{N}_1(n), \mathsf{B}_0(), \mathsf{M}(\mathsf{n_a}), \mathsf{M}(n)$	Intruder sends n to Bob
$\longrightarrow \mathsf{B}_1(n, \mathsf{n_b}), \mathsf{N}_2(n, \mathsf{n_b}), \mathsf{A}_1(\mathsf{n_a}), \mathsf{M}(\mathsf{n_a}), \mathsf{M}(n)$	Bob receives, generates nonce, replies
$\longrightarrow \mathsf{A}_1(\mathsf{n_a}), \mathsf{B}_1(n, \mathsf{n_b}), \mathsf{M}(\mathsf{n_a}), \mathsf{M}(n),$ $\mathsf{D}(\langle n, \mathsf{n_b}\rangle)$	Intruder intercepts message with n
$\longrightarrow \mathsf{A}_1(\mathsf{n_a}), \mathsf{B}_1(n, \mathsf{n_b}), \mathsf{M}(\mathsf{n_a}), \mathsf{M}(n),$ $\mathsf{D}(n), \mathsf{D}(\mathsf{n_b})$	Intruder decomposes message
$\longrightarrow \mathsf{A}_1(\mathsf{n_a}), \mathsf{B}_1(n, \mathsf{n_b}), \mathsf{M}(\mathsf{n_a}), \mathsf{M}(n), \mathsf{M}(\mathsf{n_b})$	Intruder learns value $\mathsf{n_b}$
$\longrightarrow \mathsf{A}_1(\mathsf{n_a}), \mathsf{B}_1(n, \mathsf{n_b}), \mathsf{M}(\mathsf{n_a}), \mathsf{M}(n), \mathsf{M}(\mathsf{n_b}),$ $\mathsf{C}(\mathsf{n_a}), \mathsf{C}(\mathsf{n_b})$	Intruder starts composing message
$\longrightarrow \mathsf{A}_1(\mathsf{n_a}), \mathsf{B}_1(n, \mathsf{n_b}), \mathsf{M}(\mathsf{n_a}), \mathsf{M}(n), \mathsf{M}(\mathsf{n_b}),$ $\mathsf{C}(\langle \mathsf{n_a}, \mathsf{n_b}\rangle)$	Intruder composes message
$\longrightarrow \mathsf{A}_1(\mathsf{n_a}), \mathsf{N}_2(\mathsf{n_a}, \mathsf{n_b}), \mathsf{B}_1(n, \mathsf{n_b}),$ $\mathsf{M}(\mathsf{n_a}), \mathsf{M}(n), \mathsf{M}(\mathsf{n_b})$	Intruder sends message with $\mathsf{n_a}$
$\longrightarrow \mathsf{A}_2(\mathsf{n_a}, \mathsf{n_b}), \mathsf{N}_3(\mathsf{n_b}), \mathsf{B}_1(n, \mathsf{n_b}),$ $\mathsf{M}(\mathsf{n_a}), \mathsf{M}(n), \mathsf{M}(\mathsf{n_b})$	Alice receives and responds
$\longrightarrow \mathsf{B}_2(n, \mathsf{n_b}), \mathsf{A}_2(\mathsf{n_a}, \mathsf{n_b}), \mathsf{M}(\mathsf{n_a}), \mathsf{M}(n), \mathsf{M}(\mathsf{n_b})$	Bob changes to final state, indicating successful completion of protocol

Table 2.3: Sample Attack on Encryption-Free Needham-Schroeder

We introduce msg as the super-sort, with the following relations: $\mathsf{nonce} < \mathsf{msg}$, $\mathsf{cipher} < \mathsf{msg}$, $\mathsf{d\_key} < \mathsf{msg}$, $\mathsf{e\_key} < \mathsf{msg}$.

Predicates The predicates from the previous simpler example remain, though with their sort types modified appropriately to account for encryption. For example the role states must now remember information about principals' public keys, and the network messages are now encrypted. These predicates are described in detail in Section 2.4.

We introduce new predicates related to management of encryption keys. $\mathsf{KP}(\mathsf{e\_key}, \mathsf{d\_key})$ is used for associating public/private key pairs, and $\mathsf{AnnK}(\mathsf{e\_key})$ indicates that a public key has been published. We also introduce the predicate $\mathsf{GoodGuy}(\mathsf{e\_key}, \mathsf{d\_key})$, which is used to identify the honest participants of the protocol, and their keys. The list of predicates for the full theory is shown in Tables 2.5, 2.6, and 2.7.

Functions In addition to the pairing function from the previous section,

$$\langle \_, \_ \rangle : \mathsf{msg} \times \mathsf{msg} \to \mathsf{msg}$$

we introduce a function for encryption,

$$\mathsf{enc} : \mathsf{e\_key} \times \mathsf{msg} \to \mathsf{cipher}$$

It is not necessary to include a decryption function $\mathsf{dec} : \mathsf{d\_key} \times \mathsf{cipher} \to \mathsf{msg}$, since we write protocols using pattern-matching (encryption on the left-hand-side of a rule) to express decryption.

Rules The core Needham-Schroeder protocol with encryption assumes that each principal has a previously generated keypair with a published public key. We simulate this in the initialization theory, with rules of the following form.

$$\begin{aligned} &\longrightarrow \exists k_e.k_d.\mathsf{GoodGuy}(k_e, k_d), \mathsf{KP}(k_e, k_d) \\ \mathsf{GoodGuy}(k_e, k_d) &\longrightarrow \mathsf{AnnK}(k_e), \mathsf{GoodGuy}(k_e, k_d) \end{aligned}$$

The first rule (without hypothesis) generates a keypair for an honest principal. The second rule announces the public key so it is accessible to other roles and to the intruder.

New roles are generated in the role generation theory, which generates the initial role states for each instance of a protocol role. These rules are of the following form:

$$\begin{aligned} \mathsf{GoodGuy}(k_e, k_d) &\longrightarrow \mathsf{GoodGuy}(k_e, k_d), \mathsf{A}_0(k_e) \\ \mathsf{GoodGuy}(k_e, k_d) &\longrightarrow \mathsf{GoodGuy}(k_e, k_d), \mathsf{B}_0(k_e) \end{aligned}$$

Here any honest participant can choose to participate as either the initiator or the responder, by generating the appropriate initial role state.

Finally, the first step of the protocol is changed to include encryption and the use of the published public keys. Alice's first step becomes the following:

$$\mathsf{AnnK}(k'_e), \mathsf{A}_0(k_e) \longrightarrow \exists x. \mathsf{A}_1(k_e, k'_e, x), \mathsf{N}_1(\mathsf{enc}(k'_e, \langle x, k_e \rangle)), \mathsf{AnnK}(k'_e)$$

Here Alice chooses to talk to somebody whose public key has been announced. She generates a nonce as before, and then sends out the first message encrypted by the public key she has selected.

The following transition rule then allows Bob to decrypt the message from Alice and send a reply.

$$\begin{aligned} &\mathsf{B}_0(k_e), \mathsf{N}_1(\mathsf{enc}(k_e, \langle x, k'_e \rangle)), \mathsf{AnnK}(k'_e) \longrightarrow \\ &\qquad \exists y. \mathsf{B}_1(k_e, k'_e, x, y), \mathsf{N}_2(\mathsf{enc}(k'_e, \langle x, y \rangle)), \mathsf{AnnK}(k'_e) \end{aligned}$$

The complete initialization theory, role generation theory and protocol theories for Needham-Schroeder are shown in Section 2.4.

Intruder To model the encryption capabilities of the intruder, we add a decomposition and a composition rules of the following basic form to the intruder model. The decomposition rule allows the intruder to decrypt a message (or part of a message) when the decryption key is known.

$$\begin{aligned} &\mathsf{D}(\mathsf{enc}(k,x)),\ \mathsf{KP}(k,k'),\ \mathsf{M}(k') \\ &\quad\longrightarrow\ \mathsf{D}(x),\ \mathsf{KP}(k,k'),\ \mathsf{M}(k'), \mathsf{M}(\mathsf{enc}(k,x)) \end{aligned}$$

The composition rule allows the intruder to encrypt a message with any encryption key known to the intruder.

$$\mathsf{M}(k),\ \mathsf{C}(x) \longrightarrow \mathsf{C}(\mathsf{enc}(k,x)),\ \mathsf{M}(k)$$

A complete example of the rules defining an intruder for Needham-Schroeder (including more complex sorts and some other changes explained in Section 2.3.5), is shown in Table 2.7.

2.3.5 Two-Phase Intruder Theory

In this subsection, we formally specify the properties of intruder theories that are required in order to bound the number of intruder steps needed to produce a given message. We make use of a standard notion from proof theory based on the normalization of proofs for natural deduction. This strategy was first applied to security protocols by [CJM98], who explain that the actions of the standard intruder can be syntactically separated into two phases – a decomposition phase in which messages are decomposed into smaller parts, and a composition phase in which these parts are (re)assembled into a new message. This *two-phase intruder* provides us with a proof search strategy that is the basis for the decidability of the intruder actions.

First we will need to define some new terms.

Definition 2.3.6. The *size* of an atomic formula is the number of symbols it contains. We count one for the predicate name, one for each function name, and one for each variable or constant symbol. We introduce the notation $|f|$ to indicate "size of atomic formula f".

For example, $|P(x,y)| = 3$, and $|P(f(x,y),z)| = 5$.

Definition 2.3.7. A *weighting function* is a function $f \to \mathbb{N}$ that maps atomic formulas to numeric weights. We introduce the notation $\mathcal{W}(P(x))$ to indicate the "weight of atomic formula $P(x)$". The relative weight of formulas must be preserved under substitution, *i.e.*. if A and B are atomic formulas and σ is a substitution, then

$$\mathcal{W}(A) > \mathcal{W}(B) \implies \mathcal{W}(\sigma A) > \mathcal{W}(\sigma B).$$

We will use weighting functions to guarantee termination.

Many weighting functions are possible, but we are interested in a class of functions that calculate the weight based on a pair $\langle n, P\rangle$, where n is a number indicating the size of the atomic formula, and P is the predicate name. We define a strict (non-reflexive) partial-ordering on the predicates of a theory. A particular theory has a particular ordering. For example in the standard intruder model, $\mathsf{D} > \mathsf{M}$, and $\mathsf{C} > \mathsf{M}$. An example ordering of the predicates for the intruder theory is shown in Table 2.7. The ordering is defined for formulas $P(x)$ and $P'(x')$, as follows: $\langle |P(x)|, P\rangle < \langle |P'(x')|, P'\rangle$ if $|P(x)| < |P'(x')|$ or $|P(x)| = |P'(x')|$ and $P < P'$.

For example, if N, D, and M are predicates with $\mathsf{N} > \mathsf{D} > \mathsf{M}$, then

$$\begin{array}{rcl} \mathcal{W}(\mathsf{D}(\langle x, y\rangle)) & > & \mathcal{W}(\mathsf{D}(x)) \\ \mathcal{W}(\mathsf{M}(\langle x, y\rangle)) & > & \mathcal{W}(\mathsf{D}(x)) \\ \mathcal{W}(\mathsf{N}(x)) & > & \mathcal{W}(\mathsf{D}(x)) \end{array}$$

An example of a weighting function that conforms to these constraints is as follows:

$$\mathcal{W}_1(P(x)) := 10 * |P(x)| + val(P)$$

where val is a function mapping predicate names to numbers, represented as a set of ordered pairs as follows:

$$val := \{(\mathsf{N}, 4), (\mathsf{D}, 3), (\mathsf{M}, 1), (\mathsf{C}, 3)\}$$

Here the value "10" is arbitrarily chosen to be larger than any of the values appearing in the val function.

Definition 2.3.8. A rule $R = \ell \longrightarrow r$ is a *decomposition rule* with respect to weighting function $\mathcal{W}$ if the total weight of the terms in r is less than the total weight of the terms in ℓ. A rule $R = \ell \longrightarrow r$ is a *composition rule* with respect to weighting function $\mathcal{W}$ if the total weight of the terms in r is greater than the total weight of the terms in ℓ.

For example,

$$\mathsf{D}(\langle x, y\rangle) \longrightarrow \mathsf{D}(x), \mathsf{D}(y)$$

is a decomposition rule with respect to weighting function $\mathcal{W}_1$, and

$$\mathsf{C}(x), \mathsf{C}(y) \longrightarrow \mathsf{C}(\langle x, y\rangle)$$

is a composition rule with respect to weighting function $\mathcal{W}_1$.

For the intruder theories we will consider, we allow persistent facts to appear in both the left and right hand sides. So, in general a decomposition rule is of form:

$$D(\langle A, B\rangle), \vec{P}(\dots) \longrightarrow D(A), D(B), \vec{P'}(\dots)$$

where $\vec{P}$ and $\vec{P'}$ are sets of persistent predicates, with $\vec{P} \subseteq \vec{P'}$ (and similarly for composition rules).

We also need to introduce more complicated decomposition rules, which we call "Decomposition rules with Auxiliary facts". These are pairs of rules of form:

$$D(t), \vec{P}(\dots) \longrightarrow \vec{P'}(\dots), A(t)$$

and

$$A(t), \vec{Q}(\dots) \longrightarrow \vec{Q'}(\dots), D(t')$$

where $\vec{P} \subseteq \vec{P'}$, $\vec{Q} \subseteq \vec{Q'}$, $A < D$, and $|t'| < |t|$. Here, A represents an Auxiliary fact (which can appear only in a pair of rules of this form) which is used to amortize the decomposition of $D(t)$ into $D(t')$ across the two rules. Section 2.4.4 shows an example of this type of decomposition rule, used to allow decrypting an old fact with a newly learned encryption key.

Definition 2.3.9. A theory $\mathcal{T}$ is a *two-phase* theory if its rules can be divided into three theories that share no non-persistent predicates, $\mathcal{T} = \mathcal{I} \uplus \mathcal{C} \uplus \mathcal{D}$, where $\mathcal{I}$ is a bounded sub-theory preceding $\mathcal{C}$ and $\mathcal{D}$, $\mathcal{C}$ contains only composition rules, $\mathcal{D}$ contains only decomposition rules, and no rules in $\mathcal{C}$ precede any rules in $\mathcal{D}$.

Definition 2.3.10. A *normalized derivation* is a derivation where all rules from the initialization theory are applied first, then all rules from the decomposition theory are applied before any rules from the composition theory.

It is shown in [CJM98] in a slightly different context that, with the restriction that keys must be atomic, all derivations in a two-phase theory can be transformed into normalized derivations.

Definition 2.3.11. A protocol theory is *limited to atomic keys* if its signature does not contain any functions that return data of type $\mathsf{e}_{\mathsf{key}}$ or $\mathsf{d}_{\mathsf{key}}$.

Lemma 2.3.2. *Given a state S, a two-phase intruder theory $\mathcal{M}$ with a signature that is limited to atomic keys, and a target message X: it is decidable in polynomial time whether the message X is derivable from the state S using the theory $\mathcal{M}$.*

Proof. We construct a polynomial-time decision procedure for testing whether message X is derivable from state S using the theory $\mathcal{M}$. We use the decomposition rules of $\mathcal{M}$ to decompose the state S into a set of base facts B. Then we use the composition rules of $\mathcal{M}$ "backwards" to decompose the goal message X into a set of base facts B'. The message X is derivable if $B' \subseteq B$.

The algorithm is as follows: Let $\mathcal{D}$ be the decomposition theory of $\mathcal{M}$, and let $\mathcal{C}$ be the composition theory. We write $S - T$ for multiset difference and $S \uplus T$ for multiset union. If $\ell \longrightarrow r$ is a rule and σ a subsitution, then $\sigma\ell$ and σr are multisets.

1. Decompose state S into base facts B using rules from $\mathcal{D}$.

   (a) $S_0 = S$

   (b) Repeat until no more rules in $\mathcal{D}$ can be applied:

i. Find a rule $\ell \longrightarrow r$ in $\mathcal{D}$ with $\sigma\ell \in S_i$, for some substitution σ.

ii. $S_{i+1} = (S_i - \sigma\ell) \uplus \sigma r$

(c) $B = S_i$

2. Let P be the persistent facts in B (i.e. for our Standard Intruder, P contains all the $\mathsf{M}(\,)$ facts from B).

3. Decompose goal message X into base facts B' using rules from $\mathcal{C}$ and the persistent facts from B.

   (a) $S_0 = \{X\} \uplus P$

   (b) Repeat until no more rules in $\mathcal{C}$ can be applied:

      i. Find a rule $\ell \longrightarrow r$ in $\mathcal{C}$ with $\sigma r \in S_i$, for some substitution σ.

      ii. $S_{i+1} = (S_i - \sigma r) \uplus \sigma\ell$

   (c) $B' = S_i$

4. If $B' \subseteq B$ then ACCEPT else REJECT.

This procedure terminates because of the properties of the composition and decomposition rules, which guarantee that the total size of the multiset gets smaller for each application of a decomposition rule, and larger for each application of a composition rule (or in this case smaller, since we are applying them in reverse). In this comparison, the size of the multiset is the sum of the sizes of the formulas it contains.

Note that the order of the choice of rule in Step 1b doesn't matter. Each rule selects a particular term and decomposes it, but this doesn't effect other terms not mentioned in that rule. So the state B that results in Step 1c is the same no matter what order the rules were applied. Similarly for Steps 3b and 3c.

To show correctness of the algorithm, we need to prove the two directions. If message X is derivable by theory $\mathcal{M}$ from state S, then the procedure ACCEPTs. Since the message is derivable, that means it can be obtained from S using a normalized derivation that applies rules from the decomposition theory $\mathcal{D}$ first, followed by rules from the composition theory $\mathcal{C}$. Let d be the set of decomposition rules used in the derivation, and let d' be the decomposition rules used in Step 1. Let D be the facts derived by applying the rules in d. Since we apply all possible decomposition rules in Step 1, we know that $d \subseteq d'$, so $D \subseteq B$. The derivation uses composition rules c to construct fact X, possibly along with other facts, from D. Meanwhile, Step 3 decomposes fact X into its component facts, set B', so $B' \subseteq D$. So we have $D \subseteq B \wedge B' \subseteq D$, which means $B' \subseteq B$, and the procedure ACCEPTs.

For the reverse direction, if the procedure ACCEPTs, then the message X is derivable by theory $\mathcal{M}$ from state S. If the procedure accepts, we can construct a derivation that applies all the rules from Step 1, then all the rules from Step 3 in the forward direction. Since $B' \subseteq B$, we know that X can be derived from the facts in B, so a valid derivation of X can be constructed. □

Note that because $\mathcal{M}$ is a two-phase theory, we only need to apply the above procedure once. No term produced by a rule in $\mathcal{C}$ can appear on the left-hand side of a rule in $\mathcal{D}$, so applying the composition theory does not enable any new rules to be applied in the decomposition theory. Without the restriction to atomic keys, the forward direction of our proof would fail, because the application of rules in $\mathcal{C}$ might result in the creation of a new key that would allow new messages to be decrypted using rules in $\mathcal{D}$. Rusinowitch and Turuani present a proof in a slightly different setting, representing message terms as Directed Acyclic Graphs, which removes this restriction to atomic keys [RT01].

2.3.6 Protocol and Intruder

The primary goal of security protocol analysis is to try to find flaws in a protocol – to find attack scenarios that result in the failure of properties such as secrecy or authentication, or ultimately to prove that a protocol is correct (*i.e.* that no attacks are possible). In the MSR framework, we consider the interaction of a well-founded protocol theory with a two-phase intruder intruder theory, by analyzing *standard traces* of the protocol.

Definition 2.3.12. Given a well-founded protocol theory $\mathcal{P} = \mathcal{I} \uplus \mathcal{R} \uplus \mathbf{A}$ and a two-phase intruder theory $\mathcal{M}$, a *standard trace* is a derivation that has all steps from the $\mathcal{I}$ and $\mathcal{R}$ first, then interleaves steps from the principal theories $\mathbf{A}$ with normalized derivations from the intruder theory $\mathcal{M}$.

Sorts:

e_key	:	encryption key (and principal name)
d_key	:	decryption key
cipher	:	cipher text (encrypted)
nonce	:	nonces
msg	:	data of any type

Subsorts: $\mathsf{nonce} < \mathsf{msg}$, $\mathsf{cipher} < \mathsf{msg}$, $\mathsf{e\_key} < \mathsf{msg}$, $\mathsf{d\_key} < \mathsf{msg}$

Functions:

$\mathsf{enc} : \mathsf{e\_key} \times \mathsf{msg} \to \mathsf{cipher}$	:	encryption
$\langle \_, \_ \rangle : \mathsf{msg} \times \mathsf{msg} \to \mathsf{msg}$	:	pairing

Table 2.4: Needham-Schroeder Theory Signature

The notion of a standard trace is a useful one for reasoning about the complexity of security protocols, as we will see in Section 2.5.

The intruder is easily formalized as a set of rewrite rules. While the basic intruder steps remain the same from one protocol to the next, the exact formalization depends on the form of messages used in the protocol. A specific instance of the *standard intruder* is described in some detail in Section 2.4.4.

2.4 Example: Needham-Schroeder Public Key Protocol

In this section, we present the full theory of the three-step core of the Needham-Schroeder public-key protocol, which is shown in Table 2.1. The sorts and functions for the signature are shown in Table 2.4, with the predicates introduced as needed in the table for each subtheory. An example of a two-phase intruder is shown in Table 2.7.

2.4.1 Initialization Theory

The Initialization Theory $\mathcal{I}$, for a public key system without a trusted server, is shown in Table 2.5. Here, the predicate $\mathsf{GoodGuy}$ indicates an uncompromised principal, parameterized by its encryption and decryption (public and private) keys. For simplicity, we identify the principal with its public key (i.e. where "A" appears in the protocol, we use the public key "K_a"). The GOODGUY rule allows for the creation of an unlimited number of principals, each with a unique key pair, denoted by the predicate KP.

The BADKEY rule provides a mechanism for specifying an unlimited number of compromised key pairs, which appear to belong to valid principals, but whose private keys are known to the intruder. The predicate BadKey denotes these compromised key pairs. There is no need to distinguish between the case of an honest participant who does follow the protocol but has had his keys compromised, and the case where the intruder himself is simply posing as an honest participant, but is not constrained to follow the protocol. This is because the intruder can simply simulate the first case, by performing the steps of the protocol, if he wants. *I.e.* there is no need to have both $\mathsf{GoodGuy}(k, k')$ and $\mathsf{BadKey}(k, k')$ facts generated for the same keys.

We accomplish key distribution by having the principals announce their public keys. The ANNK rule accomplishes this for the $\mathsf{GoodGuy}$ participants, while the ANNKB rule does the same for the BadKey pairs. Note that both rules generate a predicate AnnK indicating a public key that is available for communication, so from this point the honest participants can not distinguish the good guys from the bad guys.

Note that in order for the Initialization theory to be a bounded-sub theory, as defined in Definition 2.3.4, all the predicates that appear on the left-hand side of the rules (i.e. $\mathsf{GoodGuy}$, KP, BadKey and AnnK) must be persistent in the protocol theory. This can be verified by examining how these predicates are used in each of the rules of the Role Generation and Role theories in Table 2.6.

2.4.2 Role Generation Theory

The Role Generation Theory $\mathcal{R}$ is shown in Table 2.6. Rules ROLA and ROLB allow an unlimited number of sessions to be started for any principal to act in the role of either

Predicates:

$\mathsf{GoodGuy}(\mathsf{e\_key}, \mathsf{d\_key})$	:	identity of an honest participant
$\mathsf{BadKey}(\mathsf{e\_key}, \mathsf{d\_key})$	:	keys of a dishonest participant
$\mathsf{KP}(\mathsf{e\_key}, \mathsf{d\_key})$	:	encryption key pair
$\mathsf{AnnK}(\mathsf{e\_key})$	:	published public key

Initialization Theory $\mathcal{I}$:

GOODGUY:		$\longrightarrow$	$\exists k_e.k_d.\mathsf{GoodGuy}(k_e, k_d), \mathsf{KP}(k_e, k_d)$
BADKEY:		$\longrightarrow$	$\exists k_e.k_d.\mathsf{BadKey}(k_e, k_d), \mathsf{KP}(k_e, k_d)$
ANNK:	$\mathsf{GoodGuy}(k_e, k_d)$	$\longrightarrow$	$\mathsf{AnnK}(k_e), \mathsf{GoodGuy}(k_e, k_d)$
ANNKB:	$\mathsf{BadKey}(k_e, k_d)$	$\longrightarrow$	$\mathsf{AnnK}(k_e), \mathsf{BadKey}(k_e, k_d)$

Table 2.5: Public Key Initialization Theory

“Alice” (the initiator) or “Bob” (the responder). A_0 and B_0 denote the initial role states for the A and B roles, respectively, parameterized by the public key (principal) acting in that role.

Note that this theory satisfies Definition 2.3.3, since the rules involve only the persistant predicate $\mathsf{GoodGuy}$, and the initial role states A_0 and B_0, which appear only on the right hand side of the rules.

2.4.3 Protocol Role Theories

The Role Theories, shown in Table 2.6, are derived directly from the specification of the Needham-Schroeder protocol. Theory $\mathcal{A}$ corresponds to the role of “Alice”, and theory $\mathcal{B}$ corresponds to “Bob”.

In rule A1, which corresponds to the first line of the protocol, a principal k_e, in its initial state A_0, decides to talk to another principal k'_e, whose key has been announced. A new nonce x is generated, along with a network message N_{S1} corresponding to the first message sent in the protocol, and the principal moves to the new state A1, remembering the values of x and k'_e. Note that since AnnK is persistent, it must also appear on the right hand side of the rule.

In step B1, corresponding to the second step of the protocol, a principal k_e, in the initial state B_0, responds to a message on the network which is of the expected format (i.e. encrypted with k_e's public key, and with the identity of a participant whose key has been announced, embedded inside). k_e generates another nonce, and replies to the message, moving to a new state B_1 where all the information (the two nonces and the two principals) is remembered.

Similarly, A2 corresponds to the third line of the protocol, and B2 corresponds to the implicit step where the responder actually receives the final message.

Note that sent messages are denoted by $\mathsf{N_{Si}}$ and received messages are denoted by a corresponding predicate $\mathsf{N_{Ri}}$. The minimal intruder theory can be thought of as providing a network that transforms $\mathsf{N_{Si}}$'s to $\mathsf{N_{Ri}}$'s, so the protocol can execute. There are several ways to encode protocol theories using our MSR formalism. For example an alternate encoding could use a single N predicate for all network messages. In the presence of an intruder, these alternate encodings are all logically equivalent, because the intruder can transform from one network predicate to another, so we have chosen the one that seems most convenient for our purposes.

2.4.4 Intruder Theory

The Intruder Theory, which is an example of a Standard Two-Phase Intruder, is shown in Table 2.7. Here the M_* predicates denotes persistent facts known to the intruder, while D, A and C represent non-persistent facts which can be decomposed and composed into other facts.

LRNKB and LRNK are initialization rules that allows the intruder to learn keys. Since the BadKey and AnnK predicates are generated only by the initialization theory, we know from Lemma 2.2.1 that this rule only needs to be applied once per derivation, per BadKey and AnnK fact.

The REC and SND rules are used to connect the intruder to the network being used by the participants. The REC rule intercepts a message from the network and saves it as a decomposable fact. The SND rule sends composed facts onto the network.

Predicates:

$\mathsf{A}_0(\mathsf{e\_key})$	:	Role state 0 for initiator
$\mathsf{A}_1(\mathsf{e\_key}, \mathsf{e\_key}, \mathsf{nonce})$	:	Role state 1 for initiator
$\mathsf{A}_2(\mathsf{e\_key}, \mathsf{e\_key}, \mathsf{nonce}, \mathsf{nonce})$	:	Role state 2 for initiator
$\mathsf{B}_0(\mathsf{e\_key})$	:	Role state 0 for responder
$\mathsf{B}_1(\mathsf{e\_key}, \mathsf{e\_key}, \mathsf{nonce}, \mathsf{nonce})$	:	Role state 1 for responder
$\mathsf{B}_2(\mathsf{e\_key}, \mathsf{e\_key}, \mathsf{nonce}, \mathsf{nonce})$	:	Role state 2 for responder
$\mathsf{N}_{\mathsf{Si}}(\mathsf{cipher})$	:	$(i = 1, 2, 3)$ encrypted message (sent)
$\mathsf{N}_{\mathsf{Ri}}(\mathsf{cipher})$	:	$(i = 1, 2, 3)$ encrypted message (received)

Role Generation Theory $\mathcal{R}$:

$$\begin{array}{lrcl} \text{ROLA:} & \mathsf{GoodGuy}(k_e, k_d) & \longrightarrow & \mathsf{GoodGuy}(k_e, k_d), \mathsf{A}_0(k_e) \\ \text{ROLB:} & \mathsf{GoodGuy}(k_e, k_d) & \longrightarrow & \mathsf{GoodGuy}(k_e, k_d), \mathsf{B}_0(k_e) \end{array}$$

Protocol Theories $\mathcal{A}$ and $\mathcal{B}$:

$$\begin{array}{llcl} \text{A1:} & \mathsf{AnnK}(k'_e), \mathsf{A}_0(k_e) & \longrightarrow & \\ & & \multicolumn{2}{r}{\exists x.\mathsf{A}_1(k_e, k'_e, x), \mathsf{N}_{\mathsf{S1}}(\mathsf{enc}(k'_e, \langle x, k_e\rangle)), \mathsf{AnnK}(k'_e)} \\ \text{A2:} & \mathsf{A}_1(k_e, k'_e, x), \mathsf{N}_{\mathsf{R2}}(\mathsf{enc}(k_e, \langle x, y\rangle)) & \longrightarrow & \mathsf{A}_2(k_e, k'_e, x, y), \mathsf{N}_{\mathsf{S3}}(\mathsf{enc}(k'_e, y)) \\ \text{B1:} & \mathsf{B}_0(k_e), \mathsf{N}_{\mathsf{R1}}(\mathsf{enc}(k_e, \langle x, k'_e\rangle)), \mathsf{AnnK}(k'_e) & \longrightarrow & \\ & & \multicolumn{2}{r}{\exists y.\mathsf{B}_1(k_e, k'_e, x, y), \mathsf{N}_{\mathsf{S2}}(\mathsf{enc}(k'_e, \langle x, y\rangle)), \mathsf{AnnK}(k'_e)} \\ \text{B2:} & \mathsf{B}_1(k_e, k'_e, x, y), \mathsf{N}_{\mathsf{R3}}(\mathsf{enc}(k_e, y)) & \longrightarrow & \mathsf{B}_2(k_e, k'_e, x, y) \end{array}$$

Table 2.6: Needham-Schroeder Theory

The COMP rule allow the user to compose small terms into larger ones, while the DCMP rule allows for decomposition of large terms into smaller ones..

LRN converts a decomposable fact into intruder knowledge, and USE converts intruder knowledge into a composable fact.

The ENC and DEC rules allow the intruder to decrypt a message if it knows the private key, and to generate encrypted message from known public keys.

Note that LRNA and DECA are decomposition rules with auxiliary facts that handle a special case for encrypted messages. If the message can't be decrypted because the key isn't currently known, LRNA remembers the decrypted message with the special "Auxiliary" predicate, A. The DECA rule allows Auxiliary messages to be decrypted at a later time, if the decryption key becomes known.

Finally, GEN allows the intruder to generate new facts (i.e. nonces) as needed. We could also include rules to generate other new data types, such as dynamic keys, but we omit those here because they are not relevant to our Needham-Schroeder example.

We expand the weighting function described in Section 2.3.5 to include the predicates used here, *i.e.*

$$\mathcal{W}_2(P(x)) := 10 * |P(x)| + val(P)$$

where

$$val := \{(\mathsf{N}_{\mathsf{Si}}, 4), (\mathsf{D}, 3), (\mathsf{A}, 2), (\mathsf{M}_*, 1), (\mathsf{N}_{\mathsf{Ri}}, 4), (\mathsf{C}, 3)\}$$

This Intruder Theory can be divided into Composition and Decomposition rules, as shown in Table 2.7. So, this is a Two-Phase Intruder Theory with respect to weighting function $\mathcal{W}_2$, as described in Section 2.3.1.

2.5 Complexity Results

In this section, we investigate the complexity of secrecy for bounded protocols of a restricted form. More specifically, we give upper bounds that are as general as possible, and

Variables: $x : \mathsf{msg}, y : \mathsf{msg}, n : \mathsf{nonce}, c : \mathsf{cipher}, k_e : \mathsf{e\_key}, k_d : \mathsf{d\_key}$

Predicates:

$\mathsf{M_{ek}}(\mathsf{e\_key})$	:	fact in intruder memory (encryption key)
$\mathsf{M_{dk}}(\mathsf{d\_key})$	:	fact in intruder memory (decryption key)
$\mathsf{M_{n}}(\mathsf{nonce})$	:	fact in intruder memory (nonce)
$\mathsf{M_{c}}(\mathsf{cipher})$	:	fact in intruder memory (ciphertext)
$\mathsf{D}(\mathsf{msg})$	:	decomposable fact
$\mathsf{C}(\mathsf{msg})$	:	composable fact
$\mathsf{A}(\mathsf{cipher})$	:	auxiliary fact (for deferred decryption)

Ordering of Predicates: $\mathsf{N_{Si}} > \mathsf{D} > \mathsf{A} > \mathsf{M}_*$ and $\mathsf{N_{Ri}} > \mathsf{C} > \mathsf{M}_*$
Weighting Function: $\mathcal{W}_2$

Initialization Rules:			
LRNKB:	$\mathsf{BadKey}(k_e, k_d)$	$\longrightarrow$	$\mathsf{M_{ek}}(k_e), \mathsf{M_{dk}}(k_d), \mathsf{BadKey}(k_e, k_d)$
LRNK:	$\mathsf{AnnK}(k_e)$	$\longrightarrow$	$\mathsf{M_{ek}}(k_e), \mathsf{AnnK}(k_e)$
I/O Rules:			
REC:	$\mathsf{N_{Si}}(x)$	$\longrightarrow$	$\mathsf{D}(x)$
SND:	$\mathsf{C}(x)$	$\longrightarrow$	$\mathsf{N_{Ri}}(x)$
Decomposition Rules:			
DCMP:	$\mathsf{D}(\langle x, y\rangle)$	$\longrightarrow$	$\mathsf{D}(x), \mathsf{D}(y)$
LRNEK:	$\mathsf{D}(k_e)$	$\longrightarrow$	$\mathsf{M_{ek}}(k_e)$
LRNDK:	$\mathsf{D}(k_d)$	$\longrightarrow$	$\mathsf{M_{dk}}(k_d)$
LRNN:	$\mathsf{D}(n)$	$\longrightarrow$	$\mathsf{M_{n}}(n)$
DEC:	$\mathsf{M_{dk}}(k_d), \mathsf{KP}(k_e, k_d),$ $\mathsf{D}(\mathsf{enc}(k_e, x))$	$\longrightarrow$	$\mathsf{M_{dk}}(k_d), \mathsf{KP}(k_e, k_d), \mathsf{D}(x), \mathsf{M_c}(\mathsf{enc}(k_e, x))$
LRNA:	$\mathsf{D}(\mathsf{enc}(k_e, x))$	$\longrightarrow$	$\mathsf{M_c}(\mathsf{enc}(k_e, x)), \mathsf{A}(\mathsf{enc}(k_e, x))$
DECA:	$\mathsf{M_{dk}}(k_d), \mathsf{KP}(k_e, k_d),$ $\mathsf{A}(\mathsf{enc}(k_e, x))$	$\longrightarrow$	$\mathsf{M_{dk}}(k_d), \mathsf{KP}(k_e, k_d), \mathsf{D}(x)$
Composition Rules:			
COMP:	$\mathsf{C}(x), \mathsf{C}(y)$	$\longrightarrow$	$\mathsf{C}(\langle x, y\rangle)$
USEEK:	$\mathsf{M_{ek}}(k_e)$	$\longrightarrow$	$\mathsf{C}(k_e), \mathsf{M_{ek}}(k_e)$
USEDK:	$\mathsf{M_{dk}}(k_d)$	$\longrightarrow$	$\mathsf{C}(k_d), \mathsf{M_{dk}}(k_d)$
USEN:	$\mathsf{M_n}(n)$	$\longrightarrow$	$\mathsf{C}(n), \mathsf{M_n}(n)$
USEC:	$\mathsf{M_c}(c)$	$\longrightarrow$	$\mathsf{C}(c), \mathsf{M_c}(c)$
ENC:	$\mathsf{M_{ek}}(k_e), \mathsf{C}(x)$	$\longrightarrow$	$\mathsf{C}(\mathsf{enc}(k_e, x)), \mathsf{M_{ek}}(k_e)$
GEN:		$\longrightarrow$	$\exists n. \mathsf{M_n}(n)$

Table 2.7: Two-Phase Intruder Theory

lower bounds that apply to as restricted a subset of the protocol as possible. In general, a secrecy specification stipulates that certain "secret" data must not fall into the hands of the intruder. This is a derivability or reachability problem in our framework: given an initial secret such as $\mathsf{A_0}(S)$, indicating that a secret S is known to principal A at the beginning of a protocol run, is there a run of the protocol and intruder in which the adversary learns S, *i.e.* $\mathsf{M}(S)$ appears in the state of the system?

2.5.1 Restricted Protocol Form

The purpose of the initialization theory, in our framework, is to formalize the choice of initial conditions, such as shared public or private keys. However, as we have defined protocol theories, there are few restrictions on the form of initialization theories. Since derivability in multiset rewriting is undecidable, we can prove trivial lower bounds by encoding complex problems into the initialization theory. However, this kind of lower bound would not shed any light on protocol analysis. In order to avoid this essentially degenerate case, we will analyze decidability and complexity for protocol theories with initialization theories that consist only of a finite set of ground facts, and no rewrite rules. Intuitively, this means that we analyze decidability and complexity of the role generation and protocol execution phases, under the assumption that initialization has already been completed.

In addition, there are undecidability results for models that allow either an unbounded number of tuplings [HT96], or unbounded nesting of encryption and decryption [EG83]. So we will consider derivations that limit both the length of messages and the depth of encryption, by bounding the size of the ground facts that can appear in a derivation.

We also restrict the form of the protocol roles. In our previous examples, a step of a protocol role A has the form

$$\begin{array}{l} \mathsf{A_i}(\ldots), \mathsf{N_{Rj}}(\ldots), \mathsf{P}(\ldots), \mathsf{Q}(\ldots), \ldots \\ \to \vec{\exists} \ldots .\mathsf{A_k}(\ldots), \mathsf{N_{S\ell}}(\ldots), \mathsf{P}(\ldots), \mathsf{Q}(\ldots), \ldots \end{array}$$

where $\mathsf{A_i}(\ldots)$ and $\mathsf{A_k}(\ldots)$ are role states, $\mathsf{N_{Rj}}(\ldots)$ and $\mathsf{N_{S\ell}}(\ldots)$ are network messages, and $\mathsf{P}(\ldots), \mathsf{Q}(\ldots), \ldots$ are persistent facts appearing on the left and right of the rule. However, for the purpose of proving a stronger negative result, we restrict our attention to a simpler form of protocol step in this section, by omitting the persistent facts.

With these restrictions in mind, we come to the following definitions:

Definition 2.5.1. A role $\mathcal{P}$ of agent A is a *restricted role* if

- Its role states are drawn from a finite list of predicates, $\mathsf{A_1}, \ldots, \mathsf{A_a}$.
- The network predicates $\mathsf{N_{Rj}}$ and $\mathsf{N_{S\ell}}$ are drawn from a finite list of predicates $\mathsf{N_{R1}}, \ldots, \mathsf{N_{Rn}}$ and $\mathsf{N_{S1}}, \ldots, \mathsf{N_{Sn}}$.
- It contains only rules of form

$$\mathsf{A_i}(\ldots), \mathsf{N_{Rj}}(\ldots) \rightarrow \vec{\exists} \ldots .\mathsf{A_k}(\ldots), \mathsf{N_{S\ell}}(\ldots)$$

  where $\mathsf{A_i}$ and $\mathsf{A_k}$ are role states, and $\mathsf{N_{Rj}}$ and $\mathsf{N_{S\ell}}$ are network predicates, with $i < k \leq a$ and $j < \ell \leq n$, in each rule.

Definition 2.5.2. A protocol theory $\mathcal{T} = \mathcal{I} \uplus \mathcal{R} \uplus \mathbf{A}$ is in *restricted form* if

- The initialization theory $\mathcal{I}$ is a set of ground facts.
- $\mathcal{R}$ is a role generation theory
- $\mathbf{A}$ is a finite set of roles, each a restricted role.

Note that the role generation theory $\mathcal{R}$ is not bounded: a protocol theory in restricted form may have a standard role generation phase that allows each initialized principal to participate in an arbitrary number of bounded roles.

Definition 2.5.3. A derivation has *bound* k if no ground fact has size more than k in any state of the derivation.

We will consider bounded standard traces of protocols of restricted form, in conjuction with a standard intruder.

2.5.2 Protocol Complexity Matrix

Even with a protocol in restricted form, there are several interesting cases to consider, depending on whether the number of existentials and roles is bounded, and on whether tests for disequality are allowed. Table 2.8 shows the combinations we will examine.

The two main columns consider the case of whether the number of roles is bounded or unbounded. This refers to the number of instances of each role that are allowed to participate in a protocol run. In the left column $\mathcal{R}$ is bounded, meaning we fix in advance the maximum number r of protocol role instances, and then consider all runs with r or fewer roles. In the right two sub-columns, $\mathcal{R}$ is not bounded, meaning runs with an arbitrary number of roles need to be considered.

The second column is further sub-divided according to whether the number of existentials instantiated during execution in these roles is bounded or not. If the number of existentials is bounded, then we fix in advance the maximum number n of protocol nonces, and consider all runs with n or fewer nonces. Because the number is fixed, the nonces can be assumed to have been produced during initialization, and not within the roles themselves.

The two main rows consider whether the intruder is allowed to generate fresh values or not. These rows are further subdivided into the cases where the roles can perform disequality tests which would allow them to determine whether two fresh values are different from each other. The $\neq$ row allows both equality and disequality tests, while the $=$ row allows only equality tests. In a protocol, a test for disequality on a nonce would mean the protocol compares a supposedly fresh nonce it receives against all the other nonces it has received, to make sure it is actually fresh. If disequality is not allowed, then this test is not performed.

Table 2.8 shows the complexity results for these cases. The numeric references indicate the propositions about specific lower or upper bounds which we discuss in the following sections. With the exception of the top case of column two (bounded roles with existentials and disequality test, and intruder with unbounded existentials), we will see that the lower bounds apply to all cases above them in the table, and the upper bounds apply to all cases below them.

		Bounded # Roles	Unbounded # Roles	
			bounded $\exists$	Unbounded $\exists$
I with $\exists$	$\neq$	(2.5.3) NPC	???	(2.5.1) Undec.
	$=$	NPC	(2.5.2) DEXPC	Undec.
I no $\exists$	$\neq$	NPC	DEXPC	Undec.
	$=$	(2.5.6) NPC	(2.5.5) DEXPC	(2.5.4) Undec.

Table 2.8: Protocol Theory Complexity Matrix

k-Bounded Secrecy Decision Problem Given a ground formula S called the secret, a protocol theory $\mathcal{T}$, and a standard intruder $\mathcal{M}$, decide whether there exists a run of $\mathcal{T}$ and $\mathcal{M}$, with derivation bound k, where the goal formula $\mathsf{M}(S)$ occurs in any state of the run. Variants of the problem include possible bounds r on the number of role instances in $\mathcal{T}$, bounds n on the number of existentials instantiated in $\mathcal{T}$, an optional test for disequality in $\mathcal{T}$, and optional existentials instantiated in $\mathcal{M}$.

Theorem 1. *There is some small number k such that k-bounded secrecy is undecidable for protocol theories in restricted form. More specifically, there is no algorithm for deciding whether a given protocol, run in combination with the standard intruder with derivation bound k, allows the intruder to gain access to a given initial secret.*

Theorem 2. *Failure of secrecy is* DEXP*-complete for finite-length protocols of restricted form that have no new data, and no disequality test. More specifically, the following decision problem is* DEXP*-complete: Given a finite-length protocol which cannot generate new data or test for disequality, a derivation bound k, and a standard intruder, does the protocol allow the intruder to gain access to a given initial secret?*

Theorem 3. *Failure of secrecy is* NP*-complete for finite-length protocols of restricted form with a bounded Role Generation theory. More specifically, the following decision problem is* NP*-complete: Given a finite-length protocol which can test for disequality, a derivation bound k, a bounded number of instances of each role, and a standard intruder, does the protocol allow the intruder to gain access to a given initial secret?*

Proof. Theorem 1 follows from the upper bound (Proposition 2.5.1) in Section 2.5.3.1 and the lower bound (Proposition 2.5.4) in Section 2.5.4.2.

Theorem 2 follows from the upper bound (Proposition 2.5.2) in Section 2.5.3.2 and the lower bound (Proposition 2.5.5) in Section 2.5.4.3.

Theorem 3 follows from the upper bound (Proposition 2.5.3) in Section 2.5.3.3 and the lower bound (Proposition 2.5.6) in Section 2.5.4.4. □

Open Problem: The series of ??? in the box at the top of column two indicates an unresolved question for the upper bound in the case of unbounded roles, bounded protocol existentials, and unbounded intruder existentials, when disequality tests are allowed.

2.5.3 Upper Bounds

2.5.3.1 Reachability for protocols is r.e. (Thm 1)

Proposition 2.5.1. *Given a protocol theory $\mathcal{T}$, a secret S, and a standard intruder $\mathcal{M}$, it is recursively enumerable to determine whether there exists a run in which the secret $\mathsf{M}(S)$ appears in any state.*

Proof. This is immediate because we can enumerate all the execution sequences, *i.e.* all the computations of the protocol. □

2.5.3.2 Bounded Existentials without disequality is in DEXP (Thm 2)

Proposition 2.5.2. *The following decision problem is in* DEXP*:*

$$\mathcal{P} = \{\langle \mathcal{T}, \mathcal{M}, S, n, k\rangle \mid \mathcal{T} \textit{ has an attack with } \leq n \textit{ protocol nonces and} \leq k \textit{ message size}\}$$

where $\mathcal{T}$ is a protocol theory, $\mathcal{M}$ is a standard intruder, and S is a ground formula we call the secret. "Has an attack" means there exists a run with derivation bound k, with at most n nonces, where $\mathsf{M}(S)$ appears in some state.

Proof. We prove that $\mathcal{P}$ has a deterministic exponential time decision procedure. Given $\langle \mathcal{T}, \mathcal{M}, S, n, k\rangle$, the algorithm runs in time exponential in $|\mathcal{T}| + |\mathcal{M}| + n + k$, where $|\mathcal{T}|$ and $|\mathcal{M}|$ are the sizes of the protocol role theories and the intruder theory, respectively.

We restrict the protocol theory by placing a bound on the number of existentials that can be generated during protocol execution. First, we consider the case without disequality tests, where the intruder cannot generate any existentials, the number of roles is unbounded, but the number of existentials generated by the protocol theory is bounded. This corresponds to the bottom box in the second column of Table 2.8. Then we show that the upper bound also holds for the two boxes above this one in the table, when we introduce the disequality test, and when the intruder can generate existentials (but with no disequality test).

First we observe that bounding the number of protocol existentials means that we can generate all the existentials used by all runs of the protocol during the initialization phase. No new data can be generated during protocol execution, and k gives a limit on the size of any ground fact that can appear in a run. Therefore there is an exponential bound on the number of distinct ground facts that may appear in any possible protocol execution.

Next we observe that because role-generation is unlimited and no role creates new data, each role can be re-run as many times as desired. More specifically, if $\mathcal{I}$ is the initialization theory, and fact $P(\vec{t})$ is in a state S derivable from $\mathcal{I}$ by the role generation and protocol rules, then there is another state $S' \supseteq S$ containing an additional occurrence of $P(\vec{t})$ that can also be derived from $\mathcal{I}$ by the role generation and protocol rules. Specifically this means that our algorithm can freely apply any rule from any role, as many times as needed, because it is always possible to apply the earlier rules from that role in order to create the required role state for the rule we want to apply.

In short, our algorithm can treat all facts as if they were persistent facts, since role-states can always be regenerated, and network messages can always be replayed by the intruder.

Therefore, we can decide whether a fact $\mathsf{M}(S)$ is derivable by the following decision procedure:

1. set F := a set containing the ground facts from $\mathcal{I}$
2. set R := a set containing all ground instances of the rules from $\mathcal{T} + \mathcal{M}$
3. repeat:
    (a) Select a rule $\ell \longrightarrow r$ from R.

   (b) Apply the rule if $\ell \subseteq F$.

   (c) $N := r$, *i.e.* $N :=$ the facts generated by applying the rule to F.

   (d) $F := F + N$

4. until $F + N = F$ for all rules in R.

5. if fact $\mathsf{M}(S) \in F$, return YES, else return NO.

Since there is an exponential bound on the number of bounded-length facts that can be written over the signature, this process will terminate in exponential time. This decision procedure resembles the DEXP upper bound for Datalog, once we observe that all role steps can be repeated as many times as needed [DEGV97, Imm86, Var82].

In the case where the protocol roles can test for disequality, as long as no new nonces can be produced during execution (*i.e.* the intruder can't create any nonces), the disequality tests just further restrict which rules are applicable in a state. This might decrease the number of reachable facts, but the above algorithm still works, and it does not affect the upper bound.

Similarly, although we have presented this upper bound in terms of a protocol in restricted form, in fact the upper bound still hold for protocols whose roles contain persistent facts, *i.e.* it holds for roles that are not in restricted form. This is because the presence of additional persistent predicates in the LHS of rules just further restricts which rules are applicable in a state. As with disequality tests, this might decrease the number of reachable facts, but does not affect the upper bound. The above decision procedure would still work.

The above argument assumes that the number of intruder existentials is bounded. However, in the case where the roles cannot test for disequality, any attack with more than one nonce provided by the intruder can be reduced to an attack with only one new nonce provided by the intruder. Therefore, in the absence of a disequality test, the intruder with one new nonce is equivalent to the intruder with unlimited new nonces. So this case is also in DEXP. $\square$

2.5.3.3 Bounded Roles is in NP (Thm 3)

In the following section we prove our result for the case of theories whose signature uses only atomic keys, *i.e.* theories in which the signature cannot contain any functions that return messages of type $\mathsf{e_{key}}$ or $\mathsf{d_{key}}$. Although our result holds only for atomic keys, Rusinowitch and Turuani have proved that the secrecy decision problem for bounded roles, with non-atomic keys and without disequality test is also in NP [RT01].

Proposition 2.5.3. *The following decision problem is in* NP*:*

$$\mathcal{P} = \{\langle \mathcal{T}, \mathcal{M}, S, r, k\rangle \mid \mathcal{T} \textit{ has attack with} \leq r \textit{ role instances and} \leq k \textit{ message size}\}$$

where $\mathcal{T}$ is a protocol theory (recall from Definition 2.5.2 that $\mathcal{T} = \mathcal{I} \uplus \mathcal{R} \uplus \mathbf{A}$), $\mathcal{M}$ is a standard intruder, and S is a ground formula we call the secret. The signature for $\mathcal{T}$ and $\mathcal{M}$ is restricted to non-atomic keys. "Has an attack" means there exists a run with derivation bound k, with at most r protocol role instances, where $\mathsf{M}(S)$ appears in some state.

Proof. We prove that $\mathcal{P}$ has a non-deterministic polynomial-time decision procedure.

We restrict the protocol theory by placing a bound on the number of role instances that can be generated by the role generation theory $\mathcal{R}$, *i.e.* we place a limit r on the number of initial role states that can appear in any run, by limiting how many times the rules in $\mathcal{R}$ can be used. The number of existentials generated by the intruder is not bounded, and tests for disequality (as well as equality) are allowed. This corresponds to the top box in the first column of Table 2.8.

To prove this upper bound we present a decision procedure that takes as a witness a polynomial length input representing an instantiation of a sequence of protocol rules used in an attack run. The decision procedure verifies that the intruder is capable of generating the messages necessary to make the run valid, and it verifies whether the run is actually an attack. We show that this decision procedure has a polynomial running time.

First we show that a candidate attack run is of polynomial length. Let R be the maximum number of steps in the longest role of the protocol. Since we limit the number of roles to r, any attack contains at most rR protocol steps.

Next we observe that, although we have allowed the intruder to use an unlimited number of nonces in his attack, in fact the limited size of the protocol run means that the number of nonces that are relevant to the attack has a polynomial bound. Since each message is limited to size k, there can be at most k different values in a given message, or a maximum of kR values for a given role. Thus, the maximum number of distinct values that can occur in an attack run of at most r roles is bounded by $B = krR$. For a given attack, only at most B of the nonce's generated by the intruder can actually appear in the protocol steps of the run.

Note that the number of nonces generated by the protocol is also bounded by the value B. As in Section 2.5.3.2, the protocol nonces can be generated during the role generation phase, instead of during protocol execution. So in our analysis we can treat protocol nonces as constants that are included in the initial role state, and only consider the intruder-generated nonces during the protocol execution.

Using these observations, we can construct an a candidate attack run of polynomial length, as follows:

1. Guess a set of at most $B = krR$ nonces to be used by the intruder in the attack. These are included in the intruder's initial knowledge, *i.e.* for each nonce $n_i \in \{n_0, \ldots, n_B\}$, $\mathsf{M}(n_i) \in \mathcal{I}$.

2. Guess a selection of up to r roles to be generated from the role generation theory $\mathcal{R}$. Note that these role state facts include as arguments any nonces that would be generated by the protocol roles. Call this set of initial role states I_r.

3. Guess a candidate sequence of up to rR protocol steps for the attack. The rules used in the candidate attack are fully instantiated, using the values from $\mathcal{I}$, I_r, and the B intruder nonces in a specific way. Let $N \leq rR$ be the number of steps in the candidate sequence, and label each step in the sequence s_i, for $1 \leq i \leq N$.

Since the protocol is in restricted form, and no nonces are generated by the protocol roles during the run, we know from Definition 2.5.1 that all roles must be of the form:

$$\mathsf{A_i}(\ldots), \mathsf{N_{Rj}}(\ldots) \to \mathsf{A_k}(\ldots), \mathsf{N_{S\ell}}(\ldots)$$

where $\mathsf{A_i}$ and $\mathsf{A_k}$ are role states, and $\mathsf{N_{Rj}}$ and $\mathsf{N_{S\ell}}$ are network predicates. Note that if the $\mathsf{N_{Rj}}$ is missing from the rule (as in the case of the first step of an initiator role), then a null message body can be used, so all rules can be assumed to be of this form.

To verify that the candidate attack sequence is a valid protocol run, we must confirm that each step in the sequence is feasible. This involves verifying that the protocol role sequence is valid (*i.e.* each role state is used only once, and they are not used in a step until after they have been generated), and that the network messages needed at each step can be generated by the intruder from knowledge it has available at that step.

Let S_i be the current multiset of facts after step s_i. Let $S_0 = I + I_r$, where I is the multiset of facts from the initialization theory $\mathcal{I}$ and I_r is the set of initial role states generated in step 2 above. For each rule

$$s_i : \mathsf{A_i}(\vec{m}), \mathsf{N_R}(\vec{n}) \to \mathsf{A_j}(\vec{m'}), \mathsf{N_S}(\vec{n'})$$

we do the following:

1. Check that $\mathsf{A_i}(\vec{m}) \in S_{i-1}$. If not, REJECT.
2. Decompose the target message $\mathsf{N_R}(\vec{n})$ into its smallest components by applying rules from the intruder composition theory in reverse, until a set of persistent M predicates remain. Call this set of M terms M.
3. Check that $M \subseteq A_{i-1}$. If not, REJECT.
4. Fully decompose the term $\mathsf{N_S}(\vec{n'})$ by applying all rules from the intruder decomposition theory until only the persistent M predicates remain. Call this set of M terms M'.
5. Update $S_i := (S_{i-1} \uplus \{\mathsf{A_j}(\vec{m'})\} \uplus M') - \{\mathsf{A_i}(\vec{m})\}$
6. Increment i.

After repeating the above procedure for all N rules, check if $\mathsf{M}(S) \in S_N$. If yes, then ACCEPT (meaning this is an attack sequence), if not REJECT.

Since the size of each message is bounded by k, the above steps can each be done in polynomial time. We do not go into detail here about steps 2 and 4, but for our standard two-phase intruder theory, we know from Lemma 2.3.2 that they can be accomplished in polynomial time. Similar algorithms are presented in [CJM98, RT01].

Since the number of protocol steps is polynomial, that means a candidate attack sequence can be verified in polynomial time. Therefore, $\mathcal{P}$ is in NP.

This upper bound also holds for the case where the intruder is not allowed to generate nonces, or when disequality tests are not allowed, since the decision procedure still works on these more restricted protocols. So the upper bound holds for all cases in the first column of Table 2.8. $\square$

2.5.4 Lower Bounds

In this section we examine the lower bounds. The proofs make use of a reduction from Horn clauses to protocols in restricted form, so we examine that reduction first. Horn clauses without function symbols are undecidable if existentials are allowed, and DEXP-hard without existentials. Our proof is by reduction from existential Horn clauses without function symbols to protocol theories in restricted form. The reduction introduces function symbols in the protocol theory, but not in the Horn clauses.

Also note that these proofs do not rely on any use of existentials from the intruder, nor on the use of disequality tests. This means that one lower bound suffices for each column in the complexity matrix.

2.5.4.1 Representing Horn Clauses as Protocol Theories

An *existential Horn clause* is a closed first-order formula of the form

$$\begin{aligned}\forall x_1 \ldots \forall x_i[(\alpha_1 \wedge \ldots \wedge \alpha_k) \\ \Longrightarrow \exists y_1 \ldots \exists y_j(\beta_1 \wedge \ldots \wedge \beta_\ell)]\end{aligned}$$

where $\alpha_1, \ldots, \alpha_k, \beta_1, \ldots, \beta_\ell$ are first-order atomic formulas.

We will show that, given a Horn theory that consists of a set of existential Horn formulas, we can construct a protocol so that, when combined with the standard intruder theory, the intruder may learn a representation of a formula iff it is a consequence of the Horn theory.

Our encoding uses the intruder to replicate formulas. Since each protocol role can only execute a finite sequence of steps, we use a separate role for each Horn clause. The function of the intruder is to convert the final message sent by one role to an initial message received by another role. As a result of intruder actions, a datum may pass through an unbounded number of protocol steps.

In order to represent the Horn theory faithfully, we cannot give the intruder complete access to the atomic formulas used in a Horn clause. In particular, we cannot let the intruder combine data from different messages. For example, if one role sends a message representing $P(a, b)$, we cannot allow the intruder to intercept this message and replace it with $P(b, a)$. We prevent this form of interference by encrypting atomic formulas with a shared private key.

We define an encoding of a conjunction of atomic formulas into a single term of type msg. We use the notation $\lceil \phi \rceil$ to indicate the encoding of formula ϕ, where $\phi = P_1(t_{1,1}, \ldots t_{1,i_1}) \wedge \ldots \wedge P_k(t_{k,1}, \ldots , t_{k,i_k})$

For this encoding we use a secret key K which is not known by the intruder, and we assume that for each sequence of predicates $P_1, \ldots , P_k$ that occurs together in the left or right hand side of a given Horn clause, we have a constant symbol $P_1.P_2.\ldots.P_k$. (Although we have used a sequence of letters, numbers and subscripts to write out our name for this constant symbol, we assume it is an atomic constant symbol of the language.)

Given these assumptions, we let

$$\begin{aligned}&\lceil P_1(t_{1,1}, \ldots t_{1,i_1}) \wedge \ldots \wedge P_k(t_{k,1}, \ldots , t_{k,i_k})\rceil \\ &= \mathsf{enc}(K, \langle P_1.P_2.\ldots.P_k, t_{1,1}, \ldots , t_{1,i_1}, t_{k,1}, \ldots , t_{k,i_k}\rangle)\end{aligned}$$

We encode a given existential Horn clause C into a set of protocol roles. One role represents the clause itself, and in addition, for each conjunction of atomic formulas that appears in the Horn theory, we need a way to create that conjunction from atomic formulas, and to decompose it into atomic formulas. We define $Role(C) = \mathcal{R}(C) + \mathcal{C}(C) + \mathcal{D}(C)$, where $\mathcal{R}(C)$ is the role corresponding to clause C, and $\mathcal{C}(C)$ and $\mathcal{D}(C)$ are the composition and decomposition roles for clause C.

The role $\mathcal{R}(C)$ for a clause C

$$\begin{aligned}&\forall x_1 \ldots \forall x_i[(\alpha_1 \wedge \ldots \wedge \alpha_k)\\&\implies \exists y_1 \ldots \exists y_j(\beta_1 \wedge \ldots \wedge \beta_\ell)]\end{aligned}$$

is

$$\begin{aligned}&\mathsf{A}_0, \mathsf{N}_{\mathsf{Ra}_0}(\lceil \alpha_1 \wedge \ldots \wedge \alpha_k \rceil)\\&\longrightarrow \exists y_1 \ldots \exists y_j.\, \mathsf{A}_1, \mathsf{N}_{\mathsf{Sa}_1}(\lceil \beta_1 \wedge \ldots \wedge \beta_\ell \rceil)\end{aligned}$$

where $i, j \geq 0$, $k, \ell \geq 1$. For example, the role for a pure Horn clause

$$\forall x_1 \ldots \forall x_i[(\alpha_1 \wedge \ldots \wedge \alpha_k) \implies \beta]$$

is

$$\mathsf{A}_0, \mathsf{N}_{\mathsf{Ra}_0}(\lceil \alpha_1 \wedge \ldots \wedge \alpha_k \rceil) \longrightarrow \mathsf{A}_1, \mathsf{N}_{\mathsf{Sa}_1}(\lceil \beta \rceil).$$

The representation of the composition role $\mathcal{C}(C)$ and the decomposition role $\mathcal{D}(C)$ is perhaps best illustrated by an example. Suppose that the existential Horn clause

$$\forall x \forall y[(P(x) \wedge Q(x, y) \wedge R(y)) \implies \exists z(P(z) \wedge Q(y, z)]$$

is part of the Horn theory we wish to represent by a protocol. In order to use this implication, the protocol must produce a message containing $\lceil (P(a) \wedge Q(a, b) \wedge R(b) \rceil$ for some a and b. However, the protocol roles that represent Horn clauses produce encodings of conjunctions of atomic formulas, and the atomic formulas here may come from different rules. Therefore, we need additional protocol roles that select atomic formulas out of conjunctions and combine them.

The process is very similar to the encoding of the two-phase intruder in Section 2.3.5, except that protocol roles can manipulate encrypted values. For each conjunction form (including variables) that appears on the right-hand side of a Horn clause, such as $P(z) \wedge$

$Q(y, z)$, we include *decomposition roles* of the form

$$\mathsf{A}_0, \mathsf{N}_{\mathsf{R0}}(\lceil P(z) \wedge Q(y,z) \rceil) \longrightarrow \mathsf{A}_1, \mathsf{N}_{\mathsf{S1}}(\lceil P(z) \rceil)$$

and

$$\mathsf{B}_0, \mathsf{N}_{\mathsf{R0}}(\lceil P(z) \wedge Q(y,z) \rceil) \longrightarrow \mathsf{B}_1, \mathsf{N}_{\mathsf{S1}}(\lceil Q(y,z) \rceil)$$

for predicates $\mathsf{A}_0, \mathsf{A}_1, \mathsf{B}_0, \mathsf{B}_1$ not used for other roles. We also need a *composition role* for the left-hand-side of each original Horn clause. For the clause above, the role will have states $\mathsf{A}_0, \mathsf{A}_1, \mathsf{A}_2$ and A_3. At each step, the role reads one of the atomic formulas in its target conjunction, sending out either a dummy message or, at the last step, a message containing the conjunction of atomic formulas needed. In order to assemble $\lceil (P(x) \wedge Q(x,y) \wedge R(y) \rceil$, we can use a role A with the following steps:

$$\begin{array}{l}
\mathsf{A}_0, \mathsf{N}_{\mathsf{R0}}(\lceil P(x) \rceil) \longrightarrow \mathsf{A}_1(\lceil P(x) \rceil), \mathsf{N}_{\mathsf{S1}}() \\
\mathsf{A}_1(\lceil P(x) \rceil), \mathsf{N}_{\mathsf{R2}}(\lceil Q(x,y) \rceil) \longrightarrow \\
\qquad\qquad \mathsf{A}_2(\lceil P(x) \wedge Q(x,y) \rceil), \mathsf{N}_{\mathsf{S3}}() \\
\mathsf{A}_2(\lceil P(x) \wedge Q(x,y) \rceil), \mathsf{N}_{\mathsf{R4}}(\lceil R(y) \rceil) \longrightarrow \\
\qquad \mathsf{A}_3(), \mathsf{N}_{\mathsf{S5}}(\lceil P(x) \wedge Q(x,y) \wedge R(y) \rceil)
\end{array}$$

After this role sends message N_{S5}, the intruder can read the data $\lceil P(x) \wedge Q(x,y) \wedge R(y) \rceil$ contained in this message and forward it to the role representing the Horn clause with hypothesis $P(x) \wedge Q(x,y) \wedge R(y)$.

Given a set of existential Horn clauses H, we define an encoding into a protocol theory in restricted form, $\mathcal{P}(H) = \bigcup_{\phi \in H} Role(\phi)$.

Lemma 2.5.1. *The construction of $\mathcal{P}(H)$ from H is computable in polynomial time. Furthermore, if H has no existential quantifiers, then $\mathcal{P}(H)$ has no existential quantifiers.*

Proof. If a Horn clause theory H consists of n clauses $\{C_1, C_2, \ldots, C_n\}$ with a maximum conjunction size of m atomic formulas in any clause, then the corresponding protocol theory $\mathcal{P}(H)$ contains n 1-step roles $\mathcal{R}(C_i)$, one corresponding to each clause, plus a set of up to m 1-step decomposition roles in each $\mathcal{D}(C_i)$ for each conjunction on the right hand side of a clause C_i, and a (at most) m-step composition role $\mathcal{C}(C_i)$ for the conjunction on the left hand side of the clause C_i. Thus the encoding is polynomial and $O(mn)$. □

Lemma 2.5.2. *Let $\mathcal{P}(H)$ be the encoding of a set of existential Horn clauses H into a restricted protocol theory, let ϕ be a formula, and let $\mathcal{M}$ be a standard two-phase intruder theory. A run of $\mathcal{P}(H) + \mathcal{M}$ can lead to a state containing $\mathsf{M}(\lceil\phi\rceil)$ iff ϕ is derivable from H.*

Proof. The proof is by induction on the length of derivations. We need to prove both directions. First we show that if $H \vdash \phi$ then a run of $\mathcal{P}(H) + \mathcal{M}$ can lead to a state containing $\mathsf{M}(\lceil\phi\rceil)$.

First consider the base case. If ϕ is initially true, that is equivalent to a Horn clause of form

$$\mathsf{true} \implies \phi$$

From our construction, the clause C_i yields a protocol role

$$\mathsf{A}_0, \mathsf{N}_{\mathsf{RA0}}(\,) \longrightarrow \mathsf{A}_1, \mathsf{N}_{\mathsf{SA1}}(\lceil\phi\rceil)$$

The intruder can obtain $\mathsf{M}(\lceil\phi\rceil)$ by the following sequence:

$$\begin{array}{rcl}
\mathsf{M}(\,) & \longrightarrow & \mathsf{C}(\,) \\
\mathsf{C}(\,) & \longrightarrow & \mathsf{N}_{\mathsf{RA0}}(\,) \\
\mathsf{A}_0, \mathsf{N}_{\mathsf{RA0}}(\,) & \longrightarrow & \mathsf{A}_1, \mathsf{N}_{\mathsf{SA1}}(\lceil\phi\rceil) \\
\mathsf{N}_{\mathsf{SA1}}(\lceil\phi\rceil) & \longrightarrow & \mathsf{D}(\lceil\phi\rceil) \\
\mathsf{D}(\lceil\phi\rceil) & \longrightarrow & \mathsf{M}(\lceil\phi\rceil)
\end{array}$$

This proves the base case.

For the induction step, assume from our derivation up to clause C_{i-1}, we have proven $H \vdash \{\phi_0, \phi_1, \ldots, \phi_{n-1}\}$ and the intruder knows $\{\mathsf{M}(\lceil\phi_0\rceil), \ldots, \mathsf{M}(\lceil\phi_{n-1}\rceil)\}$. Suppose the next clause C_i in the derivation is

$$\phi_0 \wedge \phi_1 \wedge \ldots \wedge \phi_{n-1} \implies \exists\vec{z}.\phi_n(\vec{z})$$

From our construction this yields a protocol theory $Role(C_i) = \mathcal{R}(C_i) + \mathcal{C}(C_i) + \mathcal{D}(C_i)$, where $\mathcal{R}(C_i)$ is

$$\mathsf{A}^{\mathsf{i}}_{0}, \mathsf{N}^{\mathsf{i}}_{\mathsf{Ra0}}(\lceil\phi_0 \wedge \phi_1 \wedge \ldots \wedge \phi_{n-1}\rceil) \longrightarrow \exists\vec{z}.\mathsf{A}^{\mathsf{i}}_{1}, \mathsf{N}^{\mathsf{i}}_{\mathsf{Sa1}}(\lceil\phi_n(\vec{z})\rceil)$$

And $\mathcal{C}(C_i)$ is

$$\begin{array}{rcl}
\mathsf{C}^{\mathsf{i}}_{0}, \mathsf{N}^{\mathsf{i}}_{\mathsf{Rc0}}(\lceil\phi_0\rceil) & \longrightarrow & \mathsf{C}^{\mathsf{i}}_{1}(\lceil\phi_0\rceil), \mathsf{N}^{\mathsf{i}}_{\mathsf{Sc1}}() \\
\mathsf{C}^{\mathsf{i}}_{1}(\lceil\phi_0\rceil), \mathsf{N}^{\mathsf{i}}_{\mathsf{Rc1}}(\lceil\phi_1\rceil) & \longrightarrow & \mathsf{C}^{\mathsf{i}}_{2}(\lceil\phi_0 \wedge \phi_1\rceil), \mathsf{N}^{\mathsf{i}}_{\mathsf{Sc2}}() \\
& \ldots & \\
\mathsf{C}^{\mathsf{i}}_{\mathsf{n}-2}(\lceil\phi_0 \wedge \ldots \wedge \phi_{n-2}\rceil), \mathsf{N}^{\mathsf{i}}_{\mathsf{Rcn}-2}(\lceil\phi_{n-1}\rceil) & \longrightarrow & \mathsf{C}^{\mathsf{i}}_{\mathsf{n}-1}(), \mathsf{N}^{\mathsf{i}}_{\mathsf{Scn}-1}(\lceil\phi_0 \wedge \ldots \wedge \phi_{n-1}\rceil)
\end{array}$$

Using these roles ($\mathcal{D}(C_i)$ is not needed here), it is possible for the intruder to construct a run that results in $\mathsf{M}(\lceil\phi_n\rceil)$ in a manner similar to the base case above.

Next we show that if a run of $\mathcal{P}(H) + \mathcal{M}$ can lead to a state containing $\mathsf{M}(\lceil\phi\rceil)$ then $H \vdash \phi$.

Initially if the intruder memory contains $\mathsf{M}(\lceil\phi\rceil)$, this corresponds to a clause $\mathtt{true} \implies \phi$.

Suppose the intruder knows $\{\mathsf{M}(\lceil\phi_0\rceil), \mathsf{M}(\lceil\phi_1\rceil), \ldots, \mathsf{M}(\lceil\phi_{n-1}\rceil)\}$ after executing role R_{i-1}, and our corresponding derivation has proven $H \vdash \{\phi_0, \phi_1, \ldots \phi_{n-1}\}$. From our construction, the next role R_i used by the intruder to obtain $\mathsf{M}(\lceil\phi_n\rceil)$ will be of type $\mathcal{R}(C)$, $\mathcal{D}(C)$, or $\mathcal{C}(C)$, for some clause $C \in H$.

If the role is of form $\mathcal{R}(C)$, then this corresponds directly to the clause C.

If the role is of form $\mathcal{D}(C)$, then this corresponds to a use of the logical axiom

$$A \wedge B \wedge C \implies A$$

If the role is of form $\mathcal{C}(C)$, then this corresponds to a use of the logical axiom

$$A \wedge B \implies (A \wedge B)$$

So in each case, after the intruder executes the next role in the derivation to obtain $\mathsf{M}(\lceil \phi_n \rceil)$, it is possible to prove ϕ_n from H by using a clause C corresponding to that role. □

2.5.4.2 Unbounded existentials is Undecidable (Thm 1)

For the case of unbounded roles and unbounded existentials (the rightmost column in Table 2.8), we turn to results from Database theory, where the complexity results for Embedded Implicational Dependencies (EIDs) [CLM81] and Datalog [DEGV97] can be applied. Embedded Implication Dependencies are exactly Horn clauses with existentials and equality, as defined in Section 2.5.4.1. In [CLM81], this problem is proved to be undecidable, by a reduction from the halting problem for a two-counter machine. A reduction can also be made from the halting problem for a Turing machine, as we show in Appendix A.1.1.

Without restriction on the form of the atomic formulas, undecidability of the implication problem for existential Horn clauses follows immediately from the undecidability of Horn clauses without existential quantifiers. The problem of interest to us, however, is implication when the atomic formulas contain no function symbols.

Proposition 2.5.4. *Given a protocol theory $\mathcal{T}$, a secret S, and a standard intruder $\mathcal{M}$, there is a derivation bound k for which it is undecidable to determine whether there exists a run in which the secret $\mathsf{M}(S)$ appears in any state.*

Proof. Our proof is a reduction from existential Horn clauses. Given any set H of existential Horn clauses with no function symbols, and a formula ϕ, we can construct a protocol theory in restricted form $\mathcal{P}(H)$. We know from Lemma 2.5.1 that the number of roles and the size of the messages in $\mathcal{P}(H)$ is polynomial in the number and size of the Horn clauses. If $\mathcal{M}$ is a standard two-phase intruder, then a run of $\mathcal{P}(H) + \mathcal{M}$ can lead to a state containing $\mathsf{M}(\phi)$ iff ϕ is derivable from H. This follows from Lemma 2.5.2. Since the derivability problem is undecidable by reduction from the halting problem (Lemma A.1.2), the secrecy problem for protocol theories is also undecidable. □

Note that we can see that the derivation bound k is small and bounded, since the maximum message size in the run comes from the maximum conjunction size in the Horn theory H. The number of conjunctions appearing in the Turing machine construction in Appendix A.1.1 is quite small.

2.5.4.3 No existentials is DEXP-hard (Thm 2)

In the case of no existentials, both [DEGV97] and [CLM81] show a DEXP lower bound. In fact, DEXP-hardness follows by the same encoding of Horn formulas (Datalog programs, or full embedded implicational dependencies) as in our undecidability proof, applied to Horn clauses without function symbols and without existential quantification. For these Horn theories, DEXP-hardness of the implication problem (measured as a function of the size of the theory) is implicit in [Imm86, Var82], as explained in [DEGV97]. The lower bound for Horn theories is similar to the machine representation for the unbounded case, using a form of "symbolic counter" instead of Skolem symbols to name the cells in a bounded section of the Turing machine tape.

Proposition 2.5.5. *The following decision problem is* DEXP*-hard:*

$$\mathcal{P} = \{\langle \mathcal{T}, \mathcal{M}, S, n, k\rangle \mid \mathcal{T} \textit{ has an attack with } \leq n \textit{ protocol nonces and} \leq k \textit{ message size}\}$$

where $\mathcal{T}$ is a protocol theory, $\mathcal{M}$ is a standard intruder, and S is a ground formula we call the secret. "Has an attack" means there exists a run with derivation bound k, with at most n nonces, where $\mathsf{M}(S)$ appears in some state.

Proof. Our proof is a reduction from Horn clauses without existential quantifiers. Given any set H of Horn clauses without existential quantifiers or function symbols, and a formula ϕ, we can construct a protocol theory in restricted form $\mathcal{P}(H)$. We know from Lemma 2.5.1 that this construction is polynomial in size. If $\mathcal{M}$ is a standard two-phase intruder, then a run of $\mathcal{P}(H) + \mathcal{M}$ can lead to a state containing $\mathsf{M}(\phi)$ iff ϕ is derivable from H. This follows from Lemma 2.5.2, since the construction doesn't require any protocol or intruder nonces. Since the derivability problem for non-existential Horn clauses is DEXP-hard by reduction from Deterministic Turing Machines (DTM) with exponential running time

$$
\begin{array}{lll}
\text{A:} & \mathsf{A}_0, \mathsf{N}_{\mathsf{R0}}(v_1, v_2, \ldots, v_n) & \longrightarrow \\
& \multicolumn{2}{r}{\mathsf{A}_1, \mathsf{N}_{\mathsf{S1}}(\mathsf{enc}(P, \langle \mathsf{f}_1(V), \mathsf{f}_2(V), \ldots, \mathsf{f}_{\mathsf{m}}(V), \mathsf{end}\rangle))} \\
\text{B:} & \mathsf{B}_0, \mathsf{N}_{\mathsf{R1}}(\mathsf{enc}(P, \langle \top, x, y, z\rangle)) & \longrightarrow \\
& & \mathsf{B}_1, \mathsf{N}_{\mathsf{S2}}(\mathsf{enc}(P, z)) \\
\text{B':} & \mathsf{B}'_0, \mathsf{N}_{\mathsf{R1}}(\mathsf{enc}(P, \langle \mathsf{enc}(K, \neg\top), x, y, z\rangle)) & \longrightarrow \\
& & \mathsf{B}'_1, \mathsf{N}_{\mathsf{S2}}(\mathsf{enc}(P, z)) \\
\text{C:} & \mathsf{C}_0, \mathsf{N}_{\mathsf{R1}}(\mathsf{enc}(P, \langle x, \top, y, z\rangle)) & \longrightarrow \\
& & \mathsf{C}_1, \mathsf{N}_{\mathsf{S2}}(\mathsf{enc}(P, z)) \\
\text{C':} & \mathsf{C}'_0, \mathsf{N}_{\mathsf{R1}}(\mathsf{enc}(P, \langle x, \mathsf{enc}(K, \neg\top), y, z\rangle)) & \longrightarrow \\
& & \mathsf{C}'_1, \mathsf{N}_{\mathsf{S2}}(\mathsf{enc}(P, z)) \\
\text{D:} & \mathsf{D}_0, \mathsf{N}_{\mathsf{R1}}(\mathsf{enc}(P, \langle x, y, \top, z\rangle)) & \longrightarrow \\
& & \mathsf{D}_1, \mathsf{N}_{\mathsf{S2}}(\mathsf{enc}(P, z)) \\
\text{D':} & \mathsf{D}'_0, \mathsf{N}_{\mathsf{R1}}(\mathsf{enc}(P, \langle x, y, \mathsf{enc}(K, \neg\top), z\rangle)) & \longrightarrow \\
& & \mathsf{D}'_1, \mathsf{N}_{\mathsf{S2}}(\mathsf{enc}(P, z)) \\
\text{E:} & \mathsf{E}_0, \mathsf{N}_{\mathsf{R2}}(\mathsf{enc}(P, \mathsf{end})) & \longrightarrow \;\; \mathsf{E}_1, \mathsf{N}_{\mathsf{S3}}(S)
\end{array}
$$

Table 2.9: 3-SAT Theory $\mathcal{P}_{3SAT}$

(Lemma A.1.4), the secrecy problem for protocol theories with bounded nonces is also DEXP-hard. □

2.5.4.4 Bounded Roles is NP-hard (Thm 3)

Here we consider a restricted protocol theory where the Intruder can not generate existentials, the number of instances of each role is bounded, and no disequality tests are allowed. This corresponds to the bottom box in the first column of Table 2.8.

We can prove this problem is NP-hard by reducing Turing machines to Horn clauses, similar to the proofs for Theorems 1 and 2. A direct Turing machine proof was used for the result reported in the workshop for [DLMS99]. Subsequently, several authors have shown a direct reduction from 3-SAT to prove NP-hardness for this case [AL00, RT01]. Since that proof is straightforward, we reproduce here the reduction from [RT01], modified to use our restricted protocol theory notation.

3-SAT is a version of the satisfiability problem in conjunctive normal form, with exactly three literals per clause [Sip97, page 249]. We define an instance of 3-SAT, and some notation as follows

- Propositional Variables $V = \{v_1, v_2, \ldots, v_n\}$.
- Literals $L = v$ or $L = \neg v$.
- Clause $C = L \vee L' \vee L''$.
- Formula $F = C \wedge C' \wedge \ldots \wedge C''$.

Given a formula F, let C_i be the i-th conjunct, $L_{i,j}$ be the j-th literal of the i-th conjunct, and $x_{i,j} \in V$ be the variable appearing in the literal $L_{i,j}$. We can write $L_{i,j} = x_{i,j}^{\epsilon_{i,j}}$, where $\epsilon_{i,j} \in \{0, 1\}$ and $x^0 = x$ and $x^1 = \neg x$.

Thus an instance of 3-SAT with variables V and clauses I can be written $F(V) = \bigwedge_{i \in I} C_i$

Table 2.9 shows a protocol theory $\mathcal{P}_{3SAT}$ in restricted form that corresponds to an encoding of an instance of the 3-SAT problem. This is a general construction for all instances of 3-SAT, with the particular instance encoded into the details of the initialization role, role A. The input to role A is a message containing assignments of $\top$ and $\neg\top$ to each of the propositional variables v_i. It outputs a message representing the 3-SAT encoding with a special token end appended, encrypted with the secret key P which is shared by the protocol roles, but not known to the intruder. This message encodes each clause C_i using the function $f_i(\ )$ which is described below. The roles B, B', C, C', D, D' each examine a 3-tuple of the encoding, and if it contains either $\top$ or $\{\!|\neg\top|\!\}_K$ (both representations of "true"), then it strips off the 3-tuple and returns the rest of the encoding. The final role E broadcasts the secret S if it receives a message containing the special token $\{\!|\mathsf{end}|\!\}_P$.

In the encoding of the literals, we use encryption to represent logical negation, *i.e.* $\neg x \mapsto \{\!|x|\!\}_K$, and $x \mapsto \{\!|\neg x|\!\}_K$. We introduce the function $g(\ )$ to formalize this:

- $g(0, x) = x$
- $g(1, x) = \{\!|x|\!\}_K$

Finally, each clause C_i is encoded by a function $f_i(\ )$ in role A as follows:

- $\forall i \in I, f_i(V) = \langle g(\epsilon_{i,1}, x_{i,1}), g(\epsilon_{i,2}, x_{i,2}), g(\epsilon_{i,3}, x_{i,3})\rangle$

For example, if the clause $C_1 = v_1^0 \vee v_2^1 \vee v_4^0$, then $f_1(V) = \langle g(0, v_1), g(1, v_2), g(0, v_4)\rangle = \langle v_1, \{\!|v_2|\!\}_K, v_4\rangle$.

Proposition 2.5.6. *The following decision problem is* NP*-hard:*

$$\mathcal{P} = \{\langle \mathcal{T}, \mathcal{M}, S, r, k\rangle \mid \mathcal{T} \textit{ has attack with } \leq r \textit{ role instances and } \leq k \textit{ message size}\}$$

where $\mathcal{T}$ is a protocol theory, $\mathcal{M}$ is a standard intruder, and S is a ground formula we call the secret. "Has an attack" means there exists a run with derivation bound k, with at most r protocol role instances, where $\mathsf{M}(S)$ appears in some state.

Proof. By reduction from 3-SAT. By construction, the intruder can guess a solution and run to completion iff the instance of 3-SAT is satisfiable.

Given a set of propositional variables V, a set of clauses I, and an instance of 3-SAT $F(V) = \bigwedge_{i\in I} C_i$, construct $\langle \mathcal{T}, \mathcal{M}, S, r, k\rangle$, where S is a secret, $\mathcal{M}$ is a two-phase intruder, and $\mathcal{T}$ is the 3-SAT theory constructed as described above. Specifically, $r = |I|$, i.e. the number of role instances needed is equal to the number of 3-SAT conjuncts. The maximum term size, k, in the attack is also proportional to the number of clauses in I, since the largest term appearing in the attack will be the output term $\mathsf{N}_{\mathsf{S1}}(\,)$ from role A of the protocol.

Given $\langle \mathcal{T}, \mathcal{M}, S, r, k\rangle$, with $|I|$ instances of each of the roles, and an initial intruder knowledge of $\{\mathsf{M}(\top), \mathsf{M}(\neg\top)\}$, the intruder can learn the secret $\mathsf{M}(S)$ by broadcasting a message containing a solution of the 3-SAT problem on the network and then transforming the various network message formats to make the protocol run. Thus, there is an attack on the protocol $\mathcal{P}_{3SAT}$ iff the corresponding 3-SAT problem has a solution. $\square$

2.6 Examples: Lower Bounds as Protocols

The previous section examined the complexity of security protocols in terms of the Multiset rewriting formalism, but it may be useful to examine the phenomena that cause protocols to be difficult to analyze, using a more common and less formal notation.

2.6.1 An Exponential Attack Without Nonces

Here we present a simple protocol construction that gives some intuition for the exponential lower bound. Even without generating new data, determining a security property may require exponentially-many runs of a protocol. This example shows that there are protocols without nonces where the shortest insecure run is exponential. This is illustrated by the following family of protocols, one for each integer k. The protocol for integer k assumes that a private symmetric key K is shared between principals A, $B_1, ..., B_k$ and C. (The same effect can be achieved in a public key protocol, by first running secure key exchange steps.) In Table 2.10 we show the protocol for $k = 4$. Here the "Counter" roles corresponding to the B_i's are used to increment the value in a message, and the "Checker" roles are used to broadcast certain messages when a particular encrypted message is received.

In this artificial protocol, A sends C a message containing $(0, 0, 0, 0)$, expecting the response OK. But if C receives the message $(1, 1, 1, 1)$, C will broadcast SECRET. Principals $B_1, \ldots, B_k$ implement a k-bit increment on encrypted tuples. Therefore, if an intruder routes the initial message from A through $2^k - 1$ B principals in repeated runs of the protocol, C will broadcast the SECRET. It is easy to see that unless an exponential number of messages are sent, the SECRET remains secret.

An interesting aspect of the protocol above is that it shows that a protocol can be secure against polynomial-time attack, but considered insecure under Dolev-Yao assumptions.

2.6.2 A Class of Undecidable Protocols

We will generalize the exponential protocol used in the previous example by adding nonces to construct an undecidable protocol that uses small message size. First, we briefly review the Post correspondence problem.

A well known example of an undecidable problem is the *Post correspondence problem* (PCP) [Pos46], which concerns simple manipulation of strings. This problem can be formally described (as in [Sip97, page 184]) as follows:

An instance of the PCP is a collection P of tiles:

Keys: K - symmetric key shared by A, B_i, C
Counter Roles:

$$\begin{array}{rl}
A \longrightarrow B_1: & \{\!|x_1, x_2, x_3, 0|\!\}_K \\
B_1 \longrightarrow A: & \{\!|x_1, x_2, x_3, 1|\!\}_K \\
\\
A \longrightarrow B_2: & \{\!|x_1, x_2, 0, 1|\!\}_K \\
B_2 \longrightarrow A: & \{\!|x_1, x_2, 1, 0|\!\}_K \\
\\
A \longrightarrow B_3: & \{\!|x_1, 0, 1, 1|\!\}_K \\
B_3 \longrightarrow A: & \{\!|x_1, 1, 0, 0|\!\}_K \\
\\
A \longrightarrow B_4: & \{\!|0, 1, 1, 1|\!\}_K \\
B_4 \longrightarrow A: & \{\!|1, 0, 0, 0|\!\}_K
\end{array}$$

Checker Roles:

$$\begin{array}{rl}
A \longrightarrow C: & \{\!|0, 0, 0, 0|\!\}_K \\
C \longrightarrow A: & \mathsf{OK} \\
\\
A \longrightarrow C: & \{\!|1, 1, 1, 1|\!\}_K \\
C \longrightarrow A: & \mathsf{SECRET}
\end{array}$$

Table 2.10: Rules for Exponential Protocol, $k = 4$

$$P = \left\{ \left[\frac{t_1}{b_1}\right], \left[\frac{t_2}{b_2}\right], \ldots, \left[\frac{t_k}{b_k}\right], \right\},$$

and a match is a sequence $i_1, i_2, \ldots, i_\ell$, where $t_{i_1} t_{i_2} \ldots t_{i_\ell} = b_{i_1} b_{i_2} \ldots b_{i_\ell}$. The problem is to determine whether P has a match. Let

$$\text{PCP} = \{\langle P \rangle | P \text{ is an instance of the Post correspondence problem with a match}\}.$$

We define the *width* of a PCP instance as the length of the longest string in a tile. The *size* of a PCP instance is the number of tiles in P. PCP is undecidable for relatively small problem sizes. In particular, it has been proven to be undecidable for size 7 [MS96].

We propose a protocol that makes it possible to construct a "route" consisting of two strings, a "path" and a "return path", by selecting from a set of sub-routes. Each sub-route (which we call a "tile") will actually contain two parts, one for the forward path (which we call the "top"), and one for the return path (which we call the "bottom"). The intent is that the set of sub-route pieces (the set of tiles) has been chosen in such a way as to make it impossible, or at least extremely difficult, to construct a sequence of tiles such that the top path and the bottom path are the same.

Table 2.11 shows an example of part of our proposed path protocol, for a particular set of tiles. The "Tiles" roles are used to build a complete route out of sub-route tiles. Each role adds its tile, one node at a time, to the front of the current route, building a linked list for the top and bottom paths. For an arbitrary set of tiles, the number of "Tiles" roles is equal to the number of tiles in the set, and the number of messages broadcast by each role corresponds to the width of the tiles.

The "Checker" roles are included to allow a designer to determine if they have selected a "good" set of sub-route tiles, by testing with a protocol analyzer to see if it is possible to create a route with identical forward and backward paths, using these tiles. The "Checker" protocol steps through the links in a route and broadcasts the message SECRET if all nodes on the top and bottom paths match (indicating an error).

In this example, we assume the values n_i are nonces, that K is a session key, and T and B are long term secrets used to encode the top and bottom paths, respectively (actually

Keys: K - A's session key; T, B - long term secrets shared by C,T_i
Constants: $\mathsf{EOF_T}, \mathsf{EOF_B}$ - indicate the end of a path
Tiles: $\left[\frac{\mathsf{abc}}{\mathsf{bc}}\right]$, $\left[\frac{\mathsf{ac}}{\mathsf{cab}}\right]$, $\left[\frac{\mathsf{b}}{\mathsf{bca}}\right]$, . . .

$$
\begin{array}{lrl}
\left[\frac{\mathsf{abc}}{\mathsf{bc}}\right] & A \longrightarrow T_1 : & \{\!|t_1, b_1|\!\}_K \\
 & T_1 \longrightarrow A : & \{\!|n_1, \mathsf{a}, n_2|\!\}_T, \{\!|n_2, \mathsf{b}, n_3|\!\}_T, \{\!|n_3, \mathsf{c}, t_1|\!\}_T, \\
 & & \{\!|n_4, \mathsf{b}, n_5|\!\}_B, \{\!|n_5, \mathsf{c}, b_1|\!\}_B, \{\!|n_1, n_4|\!\}_K \\
\left[\frac{\mathsf{ac}}{\mathsf{cab}}\right] & A \longrightarrow T_2 : & \{\!|t_1, b_1|\!\}_K \\
 & T_2 \longrightarrow A : & \{\!|n_1, \mathsf{a}, n_2|\!\}_T, \{\!|n_2, \mathsf{c}, t_1|\!\}_T, \{\!|n_3, \mathsf{c}, n_4|\!\}_B, \\
 & & \{\!|n_4, \mathsf{a}, n_5|\!\}_B, \{\!|n_5, \mathsf{b}, b_1|\!\}_B, \{\!|n_1, n_3|\!\}_K \\
\left[\frac{\mathsf{b}}{\mathsf{bca}}\right] & A \longrightarrow T_3 : & \{\!|t_1, b_1|\!\}_K \\
 & T_3 \longrightarrow A : & \{\!|n_1, \mathsf{b}, t_1|\!\}_T, \{\!|n_2, \mathsf{b}, n_3|\!\}_B, \{\!|n_3, \mathsf{c}, n_4|\!\}_B, \\
 & & \{\!|n_4, \mathsf{a}, b_1|\!\}_B, \{\!|n_1, n_2|\!\}_K \\
\ldots & &
\end{array}
$$

Checker:

$$
\begin{array}{rl}
A \longrightarrow C : & \{\!|t_1, b_1|\!\}_K, \{\!|t_1, X, t_2|\!\}_T, \{\!|b_1, X, b_2|\!\}_B \\
C \longrightarrow A : & \{\!|t_2, b_2|\!\}_K \\
\\
A \longrightarrow C : & \{\!|t_1, b_1|\!\}_K, \{\!|t_1, X, \mathsf{EOF_T}|\!\}_T, \{\!|b_1, X, \mathsf{EOF_B}|\!\}_B \\
C \longrightarrow A : & \mathsf{SECRET}
\end{array}
$$

Table 2.11: Rules for Path Protocol

we use two different keys to serve the additional purpose of providing typing information to distinguish messages that are part of the top path from messages that are part of the bottom path – this typing information could be part of the message payload, and then one key would suffice). The constants $\mathsf{EOF_T}$ and $\mathsf{EOF_B}$ are used to mark the end of the path chain.

To build a path using the "Tiles" protocol, a client establishes a session key K, and then builds a path by contacting various T_i roles to add a tile to the path. An example of session using the tiles shown in Table 2.11 is:

$$\begin{array}{rl}
A \longrightarrow T_1: & \{\!|\mathsf{EOF_T}, \mathsf{EOF_B}|\!\}_K \\
T_1 \longrightarrow A: & \{\!|n_1, \mathsf{a}, n_2|\!\}_T, \{\!|n_2, \mathsf{b}, n_3|\!\}_T, \{\!|n_3, \mathsf{c}, \mathsf{EOF_T}|\!\}_T, \\
& \{\!|n_4, \mathsf{b}, n_5|\!\}_B, \{\!|n_5, \mathsf{c}, \mathsf{EOF_B}|\!\}_B, \{\!|n_1, n_4|\!\}_K \\
A \longrightarrow T_2: & \{\!|n_1, n_4|\!\}_K \\
T_2 \longrightarrow A: & \{\!|n_6, \mathsf{a}, n_7|\!\}_T, \{\!|n_7, \mathsf{c}, n_1|\!\}_T, \{\!|n_8, \mathsf{c}, n_9|\!\}_B, \\
& \{\!|n_9, \mathsf{a}, n_{10}|\!\}_B, \{\!|n_{10}, \mathsf{b}, n_4|\!\}_B, \{\!|n_6, n_9|\!\}_K \\
A \longrightarrow T_1: & \{\!|n_6, n_9|\!\}_K \\
T_1 \longrightarrow A: & \{\!|n_{11}, \mathsf{a}, n_{12}|\!\}_T, \{\!|n_{12}, \mathsf{b}, n_{13}|\!\}_T, \{\!|n_{13}, \mathsf{c}, n_6|\!\}_T, \\
& \{\!|n_{14}, \mathsf{b}, n_{15}|\!\}_B, \{\!|n_{15}, \mathsf{c}, n_9|\!\}_B, \{\!|n_{11}, n_{14}|\!\}_K
\end{array}$$

This session, which uses the tiles in the order T_1, T_2, T_1, builds a top path $\mathsf{abcacabc}$, and a bottom path $\mathsf{bccabbc}$.

We assume that the values $\mathsf{EOF_T}$ and $\mathsf{EOF_B}$, as well as the nodes in the tile set, are public information. An intruder is able to attack the protocol and learn SECRET if he can construct a route where the top and bottom paths are identical, by selecting the tiles in the appropriate sequence, feeding messages into the protocol to build a route, and then feeding the route through the Checker protocol to cause it to broadcast SECRET. Note that selecting the set of tiles is equivalent to solving PCP, so the protocol is insecure if the intruder can solve PCP. In other words, since PCP is undecidable, the secrecy problem for this protocol class is undecidable.

Note that is also possible to construct a simple protocol using MSR that solves PCP, but uses arbitrary length messages. The point of our example here is that by using nonces we can construct a PCP solution using small messages, where the size of the messages depends on the size of the problem (*i.e.* the size of the tiles), not on the size of the problem solution.

2.7 Comparison to Other Work

The MSR formalism is based on earlier work first presented in [Mit98, DM99b, CDL$^+$99]. The complexity results for undecidability and DEXP-completeness without a disequality test were first published in [DLMS99], with the NP-completeness results presented at the FMSP workshop talk in 1999. The complexity results with disequality test are presented here for the first time.

A complexity case that has been studied fairly extensively is the one with a bounded number of roles and an unbounded message size. This class is of interest because it seems to be practical to apply model-checking and exhaustive search techniques. This case was shown to be decidable in [Hui99] and NP-complete (without restriction to atomic keys) in [RT01]. This work basically shows that the bounded nature of the protocols imposes its own natural limit on the message space that can be productively exploited by the attacker.

Additional work has also been done making use of the MSR formalism in areas other than complexity analysis. This includes relating strands [FHG98] to MSR [CDM$^+$00], and using MSR as a common intermediate language for CAPSL [DM99a]. A typing infrastructure has been added to MSR, based on the theory of dependent types with subsorting [Cer01b], and this typed MSR was used to prove that the Dolev-Yao intruder can emulate the actions of an arbitrary adversary [Cer01a]. Recent work used MSR to formally analyze the Kerberos 5 protocol, discovering several anomalies [BCJS02].

2.8 Conclusion

In this chapter we have defined the formalism for Multiset Rewriting with existentials, and shown how to use this formalism to describe security protocols and the Dolev-Yao attacker model. We use this formalism to analyze the complexity of the secrecy problem in protocol analysis, under various restrictions to message size, number of protocol roles, and number of nonces.

Protocol analysis is theoretically hard, but many automated tools do exist that can provide useful insight into the problem. These tools usually are limited to some approximation of the protocol secrecy problem as we have defined it in this chapter. For instance, tools such as model-checkers [Low96, Mea96a, MMS97, Ros95, Sch96] limit the number of roles and nonces, other tools such as TAPS [Coh00] ignore the linear nature (states) of the protocol roles. These tools can prove quite useful in identifying protocol bugs and and possible attacks scenarios, though a model-checker can only prove there are no attacks within the limits of its search, and a non-linear model might discover spurious attacks that need to be examined and eliminated by hand. Symbolic tools like Athena [Son99] don't limit the number of roles, but also can't be guaranteed to terminate. In general, finding attacks in a limited case is easier than proving that there aren't any attacks in the general case.

We have identified an open problem for the complexity of the secrecy case with disequality, unbounded roles, and bounded nonces (the ??? box in Table 2.8). We conjecture that the additional power of the disequality test makes this case undecidable. Other future work in this area could include solving this open problem, as well as relating MSR to other protocol analysis formalisms such as spi calculus, and applying MSR to the analysis of specific protocols.

Chapter 3

Compositional Logic

3.1 Introduction

There has been considerable research on formal analysis of security protocols, ranging from BAN logic and related approaches [BAN89, GNY90, Sv94] to finite-state analysis [Ros95, MMS97] and proof methods based on higher-order logic [Pau97b]. Most approaches in current use are based on enumeration or reasoning about a set of protocol traces, each trace obtained by combining protocol actions with actions of a malicious intruder. Automated trace-based tools can be used to find protocol errors after a few weeks of human effort, but it remains significantly more time-consuming to prove protocols correct using logics that reason about traces. While it is difficult to give specific numbers, since efforts depend on the complexity of the protocol and the experience of those involved, it seems that most formal proofs require months of effort, even with assistance from powerful automated tools. We have therefore developed a formal logic capable of relatively abstract reasoning about protocol traces. In this logic, we are able to prove properties of common authentication and secrecy protocols by derivations of twenty to sixty lines of proof. The reason for this succinctness is that the proof rules of the logic state general properties of protocol traces that can be reused for many different protocols.

The logic we present includes modal operators naming sequences of actions from a process calculus. This logic provides a method for attaching assertions to protocol actions, in a manner resembling dynamic logic for sequential imperative programs – applying Floyd-Hoare style annotations [Flo67, Hoa69] so that the composition of the assertions associated with each action can provide the basis for a protocol correctness proof. The underlying logic is different from previous "belief" logics such as BAN and its descendants [BAN89, GNY90, Sv94] and from explicit reasoning about protocol and intruder as in Paulson's inductive method [Pau97b]. The central idea is that assertions associated with an action will hold in any protocol execution that contains this action. This gives us the power to reason about all possible runs of a protocol, without explicitly reasoning about steps that might be carried out by an attacker. At the same time, the semantics of our logic is based on sets of traces of protocol execution (possibly including an attacker), not the kind of abstract idealization found in some previous logics.

Our logic uses five predicates: Sent, Decrypts, Knows, Source, and Honest. The first two make relatively simple statements about what has happened. For example, $\mathsf{Sent}(X, m)$ holds at some state in the execution of a protocol if principal X has sent the message m. The interpretation of Knows is also very mechanical, and much more elementary than in logics of knowledge. Specifically, a principal "knows" a datum if the principal either generated this datum or received it in a message in a form that is not encrypted under a key that is not known to the principal. The last two predicates are more novel. The central predicate for reasoning about secrecy, and authentication based on secrecy, is Source. Intuitively, Source is used to identify the "source" of some datum, i.e., the way that a principal might come to know the contents of some message. The predicate Honest is used primarily to assume that one party follows the prescribed steps of the protocol correctly. For example, if Alice initiates a transaction with Bob, and wishes to conclude that only Bob knows the data she sends, she must explicitly assume that Bob is honest. If Bob is not honest, meaning that Bob does not follow the protocol (or, equivalently, Bob's key is known to the attacker), then any data Alice sends to Bob could be read by the attacker and the attacker could forge all of the messages Alice receives from Bob. Therefore, many correctness assertions involve an assumption that one or more principals are honest.

Most of the axioms and inference rules of our logic are ways of attaching assertions to actions, and rules for combining these assertions when actions are combined in a role of a protocol. The main inference rule that is not of this form is a rule we refer to as the "honesty rule." This is a form of invariance rule, used to reason about all possible actions of honest principals. Suppose that in some protocol, whenever a principal receives a message of the form $\{|a, b|\}_K$, meaning the encryption of a pair (a, b) under key K, the principal then responds with $\{|b|\}_K$. Assume further that this is the only situation in which the protocol specifies that a message consisting of a single encrypted datum is sent. Using the honesty rule, it is possible to prove that if a principal A is honest, and A sends a message of the form $\{|b|\}_K$, then A must have previously received a message of the form $\{|a, b|\}_K$. For certain authentication protocols, this form of reasoning allows us to prove that if one protocol participant completes the prescribed sequence of actions, and another principal named in one of the messages is honest, then some secret is shared between the two principals.

Section 3.2 describes the process calculus and Section 3.3 shows how we use the process calculus to express steps of a protocol. Section 3.4 describes the formulas and semantics of our logic. The proof system is presented in Section 3.5. A sample proof is given in Section 3.6, with discussion of related work appearing in Section 3.7 and concluding remarks in Section 3.8.

The example derivation given in Section 3.6 shows how to prove a significant property of Lowe's variant [Low96] of the Needham-Schroeder public key protocol [NS78]. A brief discussion in that section also shows how an attempt to prove the same property for the original Needham-Schroeder protocol fails in an insightful way. Specifically, since the main axioms and inference rules of our logic are each tied to a specific action, the outline of any possible proof is determined by the steps of the protocol. Therefore, we can reduce the problem of proving a protocol property to the problem of finding instances of axioms and rules that match in specific ways. For the well-studied Needham-Schroeder protocol, under the model of attacker capabilities used in this paper, proof of an authentication property fails precisely because initiator Alice cannot correctly establish the identity of the responder from the data she receives. In effect, the protocol logic presented in this paper leads directly to rediscovery of Lowe's observation [Low96].

The process calculus and logic presented in this paper support only public key encryption and cannot be used to reason about the source of an encrypted nonce if the principal who generated it sends more than one message containing the nonce. While we believe that the approach can be extended fairly easily to handle symmetric-key encryption and more general patterns of messages containing encrypted nonces, the more restricted form of logic used in this paper is simpler and easier to understand. We hope to explore extensions of the system in future work.

3.2 Communicating Cords

Cords are the formalism we use to represent protocols and their parts. They form an action calculus [Mil92, Mil93a, Pav97], based on π-calculus [MPW92], and related to spi-calculus [AG99]. The cords formalism is also similar to the approach of the Chemical Abstract Machine formalism [BB92], in that the communication actions can be viewed as reactions between "molecules". The basic idea of π-calculus is to represent communication by term reduction, so that the communication links can be created dynamically [Mil99]. The idea of spi is to add to π the suitable constructors for encryption and decryption, and analyze secure communication using process equivalence; some similar ideas also appear in [LM92]. We treat the encryption in a manner similar to spi-calculus, but decryption is reduced to term reduction. The idea of cord calculus is not so much to capture security within the meta-theory of processes, but rather to serve as a simple "protocol programming language", intuitive enough to support our Floyd-Hoare style logical annotations, and verifications in an axiomatic semantics. The formalism is designed to support protocol composition and synthesis, in addition to reasoning about protocol correctness. Although we do not explore protocol composition in this paper, the static interfaces and associated composition operators in 3.2.4 suggest certain forms of protocol composition that preserve logical reasoning about sequences of actions.

In fact, cords are first of all based on the informal language of arrows and messages, widely used in the security community. For instance, an arrows-and-messages picture of Lowe's variant [Low96] of the Needham-Schroeder public key protocol [NS78], which we will refer to as NSL, might look something like Figure 3.1.

Strand spaces [FHG98] have been developed in an effort towards formalizing this language. The messages are captured in a term calculus, and decorated by $+$ and $-$, respectively denoting the send and the receive actions. The roles are then presented as sequences

$$\begin{tikzcd}
\mathbf{A} & \bullet \arrow[d, "\{\!|A, m|\!\}_B"] & \bullet \arrow[r, Rightarrow] & \bullet \arrow[d, "\{\!|n|\!\}_B"] \\
\mathbf{B} & \bullet \arrow[r, Rightarrow] & \bullet \arrow[u, "\{\!|m, B, n|\!\}_A"] & \bullet
\end{tikzcd}$$

Figure 3.1: NSL as Arrows and Messages

$$\begin{tikzcd}
\mathbf{A} \; = & +\{\!|A, m|\!\}_B \arrow[d] & -\{\!|m, B, n|\!\}_A \arrow[r, Rightarrow] & +\{\!|n|\!\}_B \arrow[d] \\
\mathbf{B} \; = & -\{\!|A, m|\!\}_B \arrow[r, Rightarrow] & +\{\!|m, B, n|\!\}_A \arrow[u] & -\{\!|n|\!\}_B
\end{tikzcd}$$

Figure 3.2: NSL as a Strand Space

of such actions, called *strands*. Viewed as a strand space, the above protocol run is shown in Figure 3.2.

The fact that an agent only sees his or her own actions, *viz* sending and receiving messages, is reflected in the strand formalism. However, communication, the fact that by receiving a message, an agent may learn something new, is not reflected. E.g., in the above example, the strand B already contains the term $\{\!|A, m|\!\}_B$, and appears to know the exact form of the message that he is about to receive, including A's fresh nonce, before the communication is ever initiated. Indeed, the formalism is set up so that the particles $+\{\!|A, m|\!\}_B$ and $-\{\!|A, m|\!\}_B$ can react only when the terms involved exactly coincide.

In strand spaces, which provide the basis for a series of interesting results and applications, roles are treated as families of strands, parameterized by all the possible values that can be received. However, we found this approach not only somewhat artificial, but also technically insufficient for our form of reasoning about secure communication. For instance, it is difficult to identify the data "known" to a strand when an agent in a protocol is parameterized by values unknown to them. Such parameters come about, e.g., when a

$$\begin{array}{lcl}
\mathbf{A} & = & [\ (\nu m)\quad \langle \{\!|A, m|\!\}_B \rangle \qquad\qquad (u)\ (u/\{\!|m, B, v|\!\}_{\overline{A}})\ \ \langle \{\!|v|\!\}_B \rangle\ \ ] \\
\\
\mathbf{B} & = & [\quad (x)\ \ (x/\{\!|Y, z|\!\}_{\overline{B}})\ \ (\nu n)\ \ \langle \{\!|z, B, n|\!\}_Y \rangle \qquad (w) \quad (w/\{\!|n|\!\}_{\overline{B}})\ \ ]
\end{array}$$

Figure 3.3: NSL as a Cord Space

role is sent a term encrypted by someone else's key, which it should forward, rather than attempt to decrypt. More generally, a formal occurrence of a subterm in a strand is unrelated to the knowledge of the agent to whom that strand belongs.

Cord spaces are a result of our effort to overcome such shortcomings. In comparison with strands, we add variables to the term calculus. Of course, just like a parameter, a variable is just a placeholder for a family of values; but variables come with a formal binding and substitution mechanism. The action of receiving a value into a variable x is expressed by the operator (x), which binds the occurrences of x to the right of it. The action of sending a term t is now written $\langle t \rangle$, rather than $+t$. When the term t is closed, i.e. reducible to a value, the particles (x) and $\langle t \rangle$ can *react*: they are eliminated, and t is substituted for all occurrences of x that were bound to (x). The value propagation resulting from the communication is modelled by the substitution.

The cord space, corresponding to the above protocol is shown in Figure 3.3. Here we introduce the notation (νm) which is a binding operation denoting the generation of a new nonce, m. A generates m and sends the term $\{\!|A, m|\!\}_B$ which B now receives into the variable x, and substitutes for it on the right. In particular, the pattern-matching operator $(x/\{\!|Y, z|\!\}_{\overline{B}})$, which represents asymmetric or public key encryption, is now instantiated to $(\{\!|A, m|\!\}_B/\{\!|Y, z|\!\}_{\overline{B}})$. The matching succeeds, and the values A and m get substituted for Y and z. The term $\{\!|z, B, n|\!\}_Y$ is thus instantiated to $\{\!|m, B, n|\!\}_A$, which contains no variables any more, and can be sent. Now A receives this term into the variable u, and substitutes it into $(u/\{\!|m, B, v|\!\}_{\overline{A}})$. The encryption is matched against the expected encryption, the first two encrypted components against the nonce m and the name B, whereas the third component, the nonce n is substituted for the variable v. The term $\{\!|v|\!\}_B$ now becomes a

(names)	N	$::=$	X	variable name
			A	constant name
(basic keys)	K_0	$::=$	y	variable key
			k	constant key
			N	name
(keys)	K	$::=$	K_0	basic key
			$\overline{K_0}$	inverse key
(terms)	t	$::=$	x	variable term
			c	constant term
			N	name
			K	key
			t, t	tuple of terms
			$\{\|t\|\}_K$	term encrypted with key K
(actions)	a	$::=$	ϵ	the null action
			$\langle t \rangle$	send a term t
			(x)	receive term into variable x
			(νx)	generate new term x
			$(t/q(x_1, \ldots, x_n))$	match term t to pattern q

Table 3.1: Terms and Actions

value, which is sent, received into w and pattern matched, *viz* decrypted, and tested to be equal to the nonce n.

A formal definition of cords requires several syntactical steps.

3.2.1 Terms and Actions

The terms t are built starting from the variables x and the constants c. Moreover, the set of basic terms also contains the names N, which can be variables X, Y, Z, or constants A, B, C, and keys K which can be variables y and constants k.

Upon these basic sets, the term language is then generated by some given constructors p, which always include tupling, and the public key encryption $\{|t|\}_K$, of the term t by the key K.

(basic terms)	b	$::=$	$x \mid c \mid N \mid K$	basic terms allowed in patterns
(basic patterns)	p	$::=$	$b, \ldots, b$	tuple pattern
(patterns)	q	$::=$	p	basic pattern
			$\{\|p\|\}_{\overline{K}}$	decryption pattern

Table 3.2: Patterns

The language of actions is then built upon the terms by further constructors. They include sending a term $\langle t\rangle$, receiving into a variable (x), matching a term against a pattern $(t/q(x))$, and creating a new value (νx). The extensions may allow other actions, such as reading time, or point-to-point communication. The language of terms and actions is summarized in Table 3.1.

Patterns act as terms with holes into which we substitute other terms. The syntax of patterns is summarized in Table 3.2. We write $p(x_1, \ldots, x_n)$ if the list $x_1, \ldots, x_n$ contains all the variables in p. If $p(x_1, \ldots, x_n)$ is a pattern and $t = t_1, \ldots, t_n$ is a term, then $p(t) = [t_1, \ldots, t_n/x_1, \ldots, x_n]p$ is the term obtained by substituting $t_1, \ldots, t_n$ for $x_1, \ldots, x_n$ in p.

Some examples of patterns are:

$$\begin{aligned}
p_1(x_1, x_2) &= x_1, x_2 \\
p_2(x_1, x_2) &= x_1, A, x_2 \\
q_1(x_1, x_2) &= \{\|p_1(x_1, x_2)\|\}_{\overline{K}} \\
&= \{\|x_1, x_2\|\}_{\overline{K}} \\
q_2(x_1, x_2) &= \{\|p_2(x_1, x_2)\|\}_{\overline{K}} \\
&= \{\|x_1, A, x_2\|\}_{\overline{K}}
\end{aligned}$$

Here, the patterns p_1 and p_2 are basic tuple patterns, each taking two arguments, with p_2 containing the constant A. The patterns q_1 and q_2 are decryption patterns, again taking two arguments and containing the value K as the key.

We introduce decryption patterns in order to characterize asymmetric decryption. The decryption key $\overline{K}$ is required to decrypt a message $\{|m|\}_K$, where the notation $\{|\_|\}_\_$ represents asymmetric encryption (with the encryption key different from the decryption key). As we shall see in Section 3.2.3, the action of matching a pattern $(\{|x|\}_A/\{|z|\}_{\overline{A}})$ binds x to z in the strand to the right, thus decrypting the message $\{|x|\}_A$ to reveal the plaintext x.

Strands, defined by the following grammar, are lists of actions.

$$\text{(strands)} \quad S \quad ::= \quad aS \mid a$$

Strands include operations that are not efficiently computable. For example, the strand $(x)(y)(x/\{|z|\}_{\overline{y}})\langle z\rangle$ receives a message x and encryption key y, then decrypts x with the *decryption* key $\overline{y}$ associated with y. There is no known way to compute $\overline{y}$ efficiently from y. However, since we wish to give principal A access to a key pair K_A and $\overline{K_A}$, which we write A and $\overline{A}$ for notational convenience, it is useful to have syntax for inverse keys. In section Section 3.3, additional restrictions will be imposed on strands that represent roles in a protocol (or attacks against a protocol) so that all actions performed in a protocol run are efficiently computable.

3.2.2 Strand Order and Cords

Although a strand lists actions in a specific order, one order may not be distinguishable from another. This occurs when a variable bound in one action does not occur free in another. For example, consider the strand

$$(x)(y)\langle x, y\rangle$$

which receives two inputs and then sends a pair comprised of the two inputs. Since we choose not to assume that network communication preserves message order, this strand is equivalent to

$$(y)(x)\langle x, y\rangle$$

To see that these are equivalent, imagine running each in parallel with a strand that outputs numbers 2 and 3. Since the outputs 2 and 3 can be received in either order by the first strand, the first strand could output $\langle 2, 3\rangle$ or $\langle 3, 2\rangle$, which is clearly the set of possible outputs of the second strand.

In general, the strands S and T will be considered independent if no values can be passed between them. Formally, we capture this by defining an equivalence relation $\approx$ that includes the relation $\sim$ defined by

$$ST \sim TS \quad \Longleftrightarrow \quad \begin{cases} FV(S) \cap BV(T) & = & \emptyset \\ FV(T) \cap BV(S) & = & \emptyset \end{cases}$$

where the operators FV and BV, giving the sets of the free and the bound variables (respectively), are inductively defined as follows:

$$\begin{array}{lcl} FV\ (\langle t\rangle S) & = & FV(t) \cup FV(S) \\ FV\ ((x)S) & = & FV(S) \setminus \{x\} \\ FV\ ((t/q(x))S) & = & FV(t) \cup FV(S) \setminus \{x\} \\ FV\ ((\nu x)S) & = & FV(S) \setminus \{x\} \\ \\ BV\ (\langle t\rangle S) & = & BV(S) \\ BV\ ((x)S) & = & BV(S) \cup \{x\} \\ BV\ ((t/q(x))S) & = & BV(S) \cup \{x\} \\ BV\ ((\nu x)S) & = & BV(S) \cup \{x\} \end{array}$$

The set $FV(t)$ of the free variables occurring in a term t is defined as usual: whenever a variable x is used in the formation of a term t, it is added to $FV(t)$.

The actions "receive" (x), "match" (or "test") $(t/q(x))$ and "new" (νx) thus bind the variable x. The scope is always to the right, and name clashes are avoided by renaming bound variables. A value is propagated through a strand by substituting it for x everywhere within the scope of a binding operator. In this way, the condition $FV(S) \cap BV(T) = \emptyset$ indeed ensures that S cannot depend on T.

The equivalence relation $\approx$ is the least relation containing the transitive, reflexive closure of $\sim$,and closed under α-conversion (renaming the bound variables). The strands S and T will thus be $\approx$-equivalent if and only if one can be obtained from the other by renaming the bound variables, and permuting the actions *within the scopes of the bound variables*, i.e. in such a way that no free variable becomes bound, or *vice versa*. Note that $\approx$ preserves the free variables of a strand.

Using calligraphic $\mathcal{S}$ for the set of all possible strands, we let *cords* be equivalence classes of strands, modulo the relation $\approx$,

$$\mathcal{C} = \mathcal{S}/_{\approx}$$

To indicate a specific cord, we enclose the strand in brackets [ ], which serves to indicate both the scope of binding and that the cord encompasses all equivalent strands modulo $\approx$. That is

$$[S]_{\approx} = \{S' \mid S' \approx S\}$$

We omit the relation $\approx$ in the rest of the paper.

The equivalence relation $\approx$ allows renaming of bound variables and reordering of non-conflicting actions. For example, $\approx$ identifies strands like $(x)\langle c\rangle$ and $\langle c\rangle(x)$, with the independent actions permuted. In contrast, the actions (x) and $\langle\{\!|x|\!\}_A\rangle$ are not independent, because a value must be received into x *before* it can be sent out in a message. Sending a pair $\langle a, b\rangle$ preserves the order of terms a and b, while sending the two values one at a time, $\langle a\rangle\langle b\rangle$, does not.

3.2.3 Cord Spaces and Runs

A cord space is a multiset of cords that may interact via communication. We use $\otimes$ for multiset union and [ ] for the empty multiset. In the terminology associated with the Chemical Abstract Machine [BB92], a cord space is a "soup" in which the particles (cords) may react. We use a cord space consisting of a set of protocol roles (each represented by a cord) to represent a state and set of remaining actions of a protocol. For instance, one possible run

$$
\begin{aligned}
[S(x)S'] \otimes [T\langle t\rangle T'] \otimes C &\quad \rhd\!\rhd \quad [SS'(t/x)] \otimes [TT'] \otimes C && (1)\\
[S\,(p(t)/p(x))\,S'] \otimes C &\quad \rhd\!\rhd \quad [SS'(t/x)] \otimes C && (2)\\
[S\,(\{\!|p(t)|\!\}_y/\{\!|p(x)|\!\}_{\overline{y}})\,S'] \otimes C &\quad \rhd\!\rhd \quad [SS'(t/x)] \otimes C && (3)\\
[S(\nu x)S'] \otimes C &\quad \rhd\!\rhd \quad [SS'(m/x)] \otimes C && (4)
\end{aligned}
$$

Where the following conditions must be satisfied:

(1) $FV(t) = \emptyset$
(2) $FV(t) = \emptyset$
(3) $FV(t) = \emptyset$ and y bound
(4) $x \notin FV(S)$ and $m \notin FV(C) \cup FV(S) \cup FV(S')$

Table 3.3: Basic Reaction Steps

of the NSL protocol, with Alice initiating a conversation with Bob and no other protocol roles (or intruder cords) involved, arises from the cord space

$$\mathbf{A} \otimes \mathbf{B} = [(\nu x)\langle\{\!|A, x|\!\}_B\rangle \ldots] \ \otimes \ [(x)(x/\{\!|Y, z|\!\}_{\overline{B}}) \ldots]$$

Mathematically speaking, cord spaces form a free commutative monoid generated by cords, with $\otimes$ as the binary operation and [ ] as the unit. We sometimes write ellipses "..." inside a cord to indicate the presence of arbitrary additional actions.

Reactions

The basic reactions within a cord space are shown in Table 3.3, with the required side conditions for each reaction shown below them. The substitution (t/x) is assumed to act on the strand left of it, *viz* S'. As usual, it is assumed that no free variable becomes bound after substitution, which can always be achieved by renaming of the bound variables.

Reaction (1) is a send and receive interaction, showing the simultaneous sending of term t by the first cord, with the receiving of t into variable x by the second cord. We call this an *external action* because it involves an interaction between two cords.

The other reactions all take place within a single cord. We call these *internal actions*. Reaction (2) is a basic pattern match action, where the cord matches the pattern $p(t)$ with the expected pattern $p(x)$, and substitutes t for x. Note that as mentioned in Section 3.2.1, the pattern $p(x)$ can contain multiple holes, in which case the term t would contain multiple sub-terms.

Reaction (3) is a decryption pattern match action, where the cord matches the pattern $\{|p(t)|\}_y$ with the decryption pattern $\{|p(x)|\}_{\overline{y}}$ and substitutes t for x. Again, the pattern $p(x)$ can contain multiple holes.

Finally, reaction (4) shows the binding action where the cord creates a new value that doesn't appear elsewhere in the cordspace, and substitutes that value for x in the cord to the right.

The intuitive motive for the condition $FV(t) = \emptyset$ should be clear: a term cannot be sent, or tested, until all of its free variables are instantiated. In addition in a decryption action, the key y used in the pattern must be bound. In other words, our variables are not public names, or references that could be passed around, but strictly private stores. This security-specific feature distinguishes cord calculus from the closely related, but general-purpose process and action calculi, to which it otherwise owes the basic ideas and notations.

Runs

The *runs* of a protocol arise as reaction sequences of cord spaces. The run of the NSL protocol displayed in Figures 3.1-3.3 can now be completely formalized as a sequence of syntactic reaction steps, which are shown in Table 3.4.

Steps (a) and (d) in Table 3.4 use rule (4), steps (b), (e) and (g) are based on rule (1), and steps (c), (f) and (h) on rule (3). In a sense, these eight reduction steps correspond to the five arrows in Figures 3.1 and 3.2, plus the explicit actions to create a new value and the final test of n that B performs, which were omitted in the diagrams. On the other hand, the three arrows that appear in Figure 3.3 correspond to the applications of rule (1). The actions based on rules (3) and (4) were represented in Figure 3.3 not by arrows, but by displaying the corresponding actions.

$$
\begin{array}{rcll}
\mathbf{A} \otimes \mathbf{B} & = & [(\nu x)\langle\{|A, x|\}_B\rangle(u)\ldots] \otimes [(x)(x/\{|Y, z|\}_{\overline{B}})\ldots] & \text{(a)} \\
& \rhd\rhd & [\langle\{|A, m|\}_B\rangle(u)\ldots] \otimes [(x)(x/\{|Y, z|\}_{\overline{B}})(\nu y)\,\langle\{|z, B, y|\}_Y\rangle\ldots] & \text{(b)} \\
& \rhd\rhd & [(u)\ldots] \otimes [(\{|A, m|\}_B/\{|Y, z|\}_{\overline{B}})(\nu y)\,\langle\{|z, B, y|\}_Y\rangle\ldots] & \text{(c)} \\
& \rhd\rhd & [(u)(u/\{|m, B, v|\}_{\overline{A}})\ldots] \otimes [(\nu y)\,\langle\{|m, B, y|\}_A\rangle\,(w)\ldots] & \text{(d)} \\
& \rhd\rhd & [(u)(u/\{|m, B, v|\}_{\overline{A}})\ldots] \otimes [\langle\{|m, B, n|\}_A\rangle\,(w)\ldots] & \text{(e)} \\
& \rhd\rhd & [(\{|m, B, n|\}_A/\{|m, B, v|\}_{\overline{A}})\,\langle\{|v|\}_B\rangle] \otimes [(w)\ldots] & \text{(f)} \\
& \rhd\rhd & [\langle\{|n|\}_B\rangle] \otimes [(w)(w/\{|n|\}_{\overline{B}})] & \text{(g)} \\
& \rhd\rhd & [\,] \otimes [(\{|n|\}_B/\{|n|\}_{\overline{B}})] & \text{(h)} \\
& \rhd\rhd & [\,] &
\end{array}
$$

Table 3.4: NSL Example Reaction

3.2.4 Static Binding and Cord Category

Many protocols of interest consist of a set of roles. For example, the Needham-Schroeder public-key protocol, mentioned earlier, has an initiator role and a responder role. An initiator starts an exchange by sending a message to a responder. In a local area network, for example, with principals (users or machines) named Alice, Bob, Charley, and Dolores, the protocol could be used simultaneously or in succession by several principals for several purposes. Alice may initiate an exchange with Bob and, before completing the three messages associated with this exchange, respond to an exchange initiated by Charlie. Concurrently, Charlie may initiate an exchange with Bob, and respond to a request from Dolores. We consider the set of messages exchanged by these four parties, together with any actions performed by an attacker, a possible run of the protocol.

Since a run may consist of several instances of the protocol roles, executed concurrently, it is useful to have a notation for a role that has not been assigned to any specific principal. In this section, we summarize the basic idea. However, since the mechanisms described here do not play a central role in this paper, the presentation is informal, proceeding by example rather than focussing on the general definitions. The main concepts that are needed in the remainder of the paper are that roles contain variables indicating principals, these variables are bound in the static interface of the role, and the variables are replaced by specific principals in constructing a run of the protocol.

Two cords of the NSL protocol, given in Figure 3.3, are:

$$\begin{aligned}\mathbf{A} &= [(\nu x)\langle\{\!|A, x|\!\}_B\rangle(u)(u/\{\!|x, B, v|\!\}_{\overline{A}})\langle\{\!|v|\!\}_B\rangle] \\ \mathbf{B} &= [(x)(x/\{\!|Y, z|\!\}_{\overline{B}})(\nu y)\langle\{\!|z, B, y|\!\}_Y\rangle(w)(w/\{\!|y|\!\}_{\overline{B}})]\end{aligned}$$

These are the initiator and responder roles of the NSL protocol, assigned to principals A and B, respectively. Replacing the names A and B with variables X and Y, and listing the "instantiable" variables outside the square brackets of the cord, we have:

$$\begin{aligned}\mathbf{Init} &= (X\ Y)[(\nu x)\langle\{\!|X, x|\!\}_Y\rangle(u)(u/\{\!|x, Y, v|\!\}_{\overline{X}})\langle\{\!|v|\!\}_Y\rangle]\langle\rangle \\ \mathbf{Resp} &= (X)[(x)(x/\{\!|Y, z|\!\}_{\overline{X}})(\nu y)\langle\{\!|z, X, y|\!\}_Y\rangle(w)(w/\{\!|y|\!\}_{\overline{X}})]\langle\rangle\end{aligned}$$

We consider the lists $(X\ Y)$ and (X) binding operators, so that principal variables X and Y are bound in these cords. The list are called the static interfaces of the cords, since we model replacing X and Y with *Alice* and *Bob* as a static operation that occurs before the dynamic execution of a run of the protocol.

In the cords $\mathbf{Init}$ and $\mathbf{Resp}$ above, empty angle brackets, $\langle\rangle$ appear at the end of each cord. Generally, angle brackets may contain an export list which is used to compose cords. Composition, written using ";", is achieved by substitution. For example,

$$(X\ Y)[(\nu x)\langle x\rangle]\langle X\rangle\ ;\ (Z)[(y)\langle\{\!|y|\!\}_Z\rangle]\langle\rangle = (X\ Y)[(\nu x)\langle x\rangle(y)\langle\{\!|y|\!\}_X\rangle]\langle\rangle$$

Note that in performing the composition, the exported X from the first cord is substituted for imported Z in the second. Since composition provides substitution, we introduce an abbreviation

$$\mathbf{Init}(A\ B) = (\ )[\ ]\langle A\ B\rangle\ ;\ \mathbf{Init}$$

to indicate the cord with its static interface instantiated to the values A and B. This gives us $\mathbf{A} = \mathbf{Init}(A\ B)$ and $\mathbf{B} = \mathbf{Resp}(B)$.

For the reader familiar with action calculus, we observe that cords, taken as particles, generate an action category [Mil93b, Pav97]. This is the source of the equivalence relation $\approx$. The idea is that a cord space C, displayed in the form

$$\gamma \quad = \quad (x_0 \ldots x_{i-1}) C \langle t_0 \ldots t_{j-1} \rangle$$

can be viewed as an arrow $\gamma : i \longrightarrow j$, where arities i,j are the objects of the category. The variables $x_\bullet$, assumed mutually different, form the input interface: the operator $(x_0, \ldots, x_i)$ *binds* their occurrences to the right. The terms $t_\bullet$ in the output interface may not be mutually different, nor different from $x_\bullet$. Of course, all expressions are up to variable renaming (α-conversion).

Given a morphism $\delta : j \longrightarrow k$, in the form

$$\delta \quad = \quad (u_0 \ldots u_{j-1}) D \langle v_0 \ldots v_{k-1} \rangle$$

the composite $\gamma\ ;\ \delta : i \longrightarrow k$ will be the cord morphism

$$\gamma\ ;\ \delta \quad = \quad (x_0 \ldots x_{i-1})\ CD(\vec{t}/\vec{u})\ \langle v_0 \ldots v_{k-1} \rangle$$

where it is assumed that the names in the interfaces of γ and δ have been chosen so that no clashes occur when $\vec{t}$ is substituted for $\vec{u}$.

The idea is that the dynamic binding by (x) and (νx) captures value propagation by communication, the *static binding* of the input interface is now used to compose agents at design time. The static interfaces are thus *not* used for passing any actual messages, but for propagating the public keys, connecting the various roles of the same principal, and for static links in general, independent of and prior to the execution. The cord $\mathbf{Init}$ above can be designed as the composite $\mathbf{Init} = \mathbf{I_0}\ ;\ \mathbf{I_1}$, where

$$\begin{array}{rcl}
\mathbf{I_0} & = & (X\ Y)[(\nu x)\langle \{|X, x|\}_Y \rangle]\langle X\ Y\ x \rangle \\
\mathbf{I_1} & = & (\tilde{X}\ \tilde{Y}\ z)[(u)(u/\{|z, \tilde{Y}, v|\}_{\overline{\tilde{X}}})\langle \{|v|\}_{\tilde{Y}} \rangle]\langle\rangle
\end{array}$$

The constant values of A and B can be passed to it, at design time, by precomposing it with the morphism $(\ )[\ ]\langle A\ B\rangle$, i.e., $\mathbf{A} = (\ )[\ ]\langle A\ B\rangle; \mathbf{Init}$.

Cord morphisms represent *processes*:

$$\text{(processes)} \quad r \quad ::= \quad (x \ldots x)C\langle x \ldots x\rangle$$

In a cord space, the interfaces of processes combine in the natural way, that is

$$(\vec{x})C\langle\vec{t}\rangle \otimes (\vec{u})D\langle\vec{v}\rangle = (\vec{x}\vec{u})C \otimes D\langle\vec{t}\vec{v}\rangle$$

where $\vec{x}$ is disjoint from $\vec{u}$ and the variables in $\vec{t}$ are disjoint from the variables in $\vec{v}$, to avoid naming conflicts. Since these disjointness conditions may be satisfied by suitable renaming, it is always possible to move the static interfaces to the outside of the cord space, and carry out all reactions within their scope.

Returning to the example NSL cords, if we define a substitution process $\sigma = (\ )[\ ]\langle ABB\rangle$, then

$$\mathbf{A} \otimes \mathbf{B} = \sigma\ ;\ (\mathbf{Init} \otimes \mathbf{Resp}).$$

Static binding plays a limited role in the present paper. We will display the interface only when it is relevant, and in other cases assume that all variables are bound.

3.3 Protocols

3.3.1 Protocol Roles

A *protocol* is defined by a finite set of roles, such as initiator, responder and server, each carried out by one or more participants in any protocol execution. In representing protocol roles by cords, it is useful to identify the principal who carries out the role; we do this by writing the name of the principal as a subscript on the square brackets. Formally, we distinguish roles from cords by calling a cord with identified principal, subject to the key condition given below, a *role*. Using variables from their static interface, the initiator and responder roles are now written precisely as

$$\begin{array}{rcl}\mathbf{Init} & = & (X\ Y)\,[(\nu x)\langle\{\!|X, x|\!\}_Y\rangle(u)(u/\{\!|x, Y, v|\!\}_{\overline{X}})\langle\{\!|v|\!\}_Y\rangle]_X \\ \mathbf{Resp} & = & (X)\,[(x)(x/\{\!|Y, z|\!\}_{\overline{X}})(\nu y)\langle\{\!|z, X, y|\!\}_Y\rangle(w)(w/\{\!|y|\!\}_{\overline{X}})]_X\end{array}$$

with the static output interfaces omitted since they are empty.

Formally, we define NSL protocol to be the set whose members are precisely these two roles,

$$\mathit{NSL} \quad = \quad \{\mathbf{Init}, \mathbf{Resp}\} \tag{5}$$

Roles are composed using substitution, in exactly the same way that cords are composed, with substitution performed on the subscript indicating the acting principal if it is a variable from the static interface.

The technical reason for identifying the principal carrying out a role is to specify the correct use of private keys. A *private key* is a key of form $\overline{X}$, which represents the decryption key in a public key cryptosystem. In a protocol role, the only place $\overline{X}$ is allowed to occur is in the decryption position of a decryption pattern of a role belonging to principal X. For example, the following cord

$$(X\ Y)[(x)(x/\{\!|y|\!\}_{\overline{Y}})]_X$$

is not a well-formed role, because this role allows the principal X to decrypt a message using Y's private key. The syntactic key restriction on roles prevents a role from "computing" the value of a private key from the public key. The following cord, which applies $\overline{(\ )}$ to a key it receives,

$$(X)[(x)(y)(x/\{\!|z|\!\}_{\overline{y}})]_X$$

is not a well-formed protocol role because the private key $\overline{y}$ used in the role is not the private key of the acting principal.

Since private keys can only occur in the decryption key position of a decryption pattern match, the key restriction prevents private keys from being sent in a message. While some useful protocols might send private keys, we prevent roles from sending their private keys (in this paper) since this allows us to take secrecy of private keys as an axiom, shortening proofs of protocol properties. (Otherwise, we would have to prove that in any protocol of interest, each role maintains the secrecy of its private key.) The key restriction does not prevent a decryption key from being received dynamically and used in a decryption action. For example, the following cord

$$(X)[(x)(y)(y/\{\!|z|\!\}_x)]_X$$

does not violate the syntactic key restriction. We expect to consider roles that may generate key pairs and send decryption keys in later versions of the protocol logic.

3.3.2 Intruder Role

The protocol intruder is capable of taking any of several possible actions, including receiving a message, decomposing it into parts, decrypting the parts if the key is known, remembering parts of messages, and generating and sending new messages. This is the standard "Dolev-Yao model", which appears to have developed from positions taken by Needham and Schroeder [NS78] and a model presented by Dolev and Yao [DY83]; see also [CDL$^+$99]. Although we use cords to represent the actions of a protocol intruder, the intruder may have several private keys available to it – by referring to "the" intruder, we do not mean to suggest that an attack must be carried out by a single principal.

In each run of a protocol, the intruder will be represented by a cord with an arbitrary collection of cord calculus actions from Section 3.2.1, subject only to a restriction on decryption keys defined in this section. We specify the set of decryption actions available to the intruder using a subsidiary definition of the set of decryption keys used in a cord. If C is a cord, then $DKeys(C)$ is the set of names that appear as private keys in the cord, that is:

$$DKeys(C) = \{X \mid \overline{X} \text{ appears in } C\}$$

Formally, an *intruder role with keys* $K_1, \ldots, K_n$ is a multiset I of cords such that $Dkeys(I)$

contains only $K_1, \ldots, K_n$ and the private key function $\overline{(\ )}$ only occurs in the decryption position of a decryption pattern match action. This restriction will prevent an intruder from using any key that has not been corrupted, and prevents the intruder from computing the private key from a public key.

3.3.3 Protocol Configurations and Runs

A *run of a protocol* is a sequence of reaction steps from an *initial configuration*. An *initial configuration* is determined by a set of principals, a subset of which are designated as honest, a cord space constructed by assigning one or more roles to each honest principal, and an intruder cord that may use only the secret keys of dishonest principals. We give an example without an intruder cord and a more complicated example with an intruder cord. Although we could assign protocol roles to corrupted keys, there is no need to do so. The reason is simply that when a key is available to the intruder, the intruder can simulate any number of protocol roles using that key.

Here is an example initial configuration allowing A to initiate a conversation with B, C a conversation with A, and allowing both A and B to respond using the responder role:

$$\begin{aligned}
&\mathbf{Init}(A\ B) \otimes \mathbf{Init}(C\ A) \otimes \mathbf{Resp}(A) \otimes \mathbf{Resp}(B) = \\
&\qquad [(\nu x)\langle\{\!|A, x|\!\}_B\rangle \ldots]_A \otimes\ [(\nu x)\langle\{\!|C, x|\!\}_A\rangle \ldots]_C \otimes \\
&\qquad [(x)(x/\{\!|Y, z|\!\}_{\overline{A}}) \ldots]_A \otimes [(x)(x/\{\!|Y, z|\!\}_{\overline{B}}) \ldots]_B
\end{aligned}$$

Using a substitution cord, $\sigma = (\ )[\ ]\langle A\ B\ C\ A\ A\ B\rangle$, the initial configuration can also be written as a composition of cords

$$\sigma\ ;\ (\mathbf{Init} \otimes \mathbf{Init} \otimes \mathbf{Resp} \otimes \mathbf{Resp}).$$

More generally, any initial configuration can be expressed using a selection of protocol roles, an intruder role, and a substitution cord morphism that connects the interfaces of all the roles. Each initial configuration has only one intruder role, but this role may be selected arbitrarily from all possible intruder roles.

For example, let $P_C = \{U_1, U_2, \ldots, U_\ell\}$ be a set of principals, and let subset

$HONEST(\mathbf{C}) = \{U_1, \ldots, U_k\}$ be a set of honest principals. Let $\mathbf{I_C}$ be any intruder role with decryption keys among $U_{k+1}, \ldots, U_\ell$. To define an initial configuration with n **Init** roles, m **Resp** roles, and intruder $\mathbf{I_C}$, we define the substitution

$$\sigma_{\mathbf{C}} = (\ )[\ ]\langle X_1\ Z_1\ X_2\ Z_2 \ldots X_n\ Z_n\ Y_1\ Y_2 \ldots Y_m\ U_1\ U_2 \ldots U_l\rangle.$$

Then an initial configuration of the *NSL* protocol is given by the cord space

$$\begin{aligned}\mathbf{C} \quad = \quad & \mathbf{Init}(X_1\ Z_1) \otimes \mathbf{Init}(X_2\ Z_2) \otimes \ldots \otimes \mathbf{Init}(X_n\ Z_n) \otimes \\ & \mathbf{Resp}(Y_1) \otimes \mathbf{Resp}(Y_2) \otimes \ldots \otimes \mathbf{Resp}(Y_m) \otimes \\ & \mathbf{I_C}(U_1\ U_2 \ldots U_l)\end{aligned}$$

where $X_1, X_2, \ldots X_n, Y_1, Y_2, \ldots, Y_n$ are honest keys in P_C. The intruder role may use any encryption keys among $U_1\ \ U_2 \ldots U_\ell$, but only the decryption keys that are not in $HONEST(\mathbf{C})$.

Example: Lowe's run of the original Needham-Schroeder protocol As an example, consider Lowe's anomaly in the original Needham-Schroeder protocol [Low95]. The original Needham-Schroeder protocol is represented by the following cords

$$\begin{aligned}\mathbf{Init_{NS}} &= (X\ Y)[(\nu x)\langle\{\!|X, x|\!\}_Y\rangle(u)(u/\{\!|x, v|\!\}_{\overline{X}})\langle\{\!|v|\!\}_Y\rangle]_X \\ \mathbf{Resp_{NS}} &= (X)[(x)(x/\{\!|Y, z|\!\}_{\overline{X}})(\nu y)\langle\{\!|z, y|\!\}_Y\rangle(w)(w/\{\!|y|\!\}_{\overline{X}})]_X\end{aligned}$$

It differs from *NSL* only in the second message, where the identity of the responder is omitted.

An initial configuration which will exhibit Lowe's anomaly is shown in Table 3.5. It has four principals: A and B who are honest, E whose private key is known to the intruder, and I who is the intruder. It is also possible for I and E to be the same principal, but we choose different entities to emphasize that the intruder can be unknown to the honest participants, but able to masquerade as any dishonest principal, in particular one whom A mistakenly believes to be honest. The cord space contains one initiator cord, in which A chooses to talk to E, and a responder cord belonging to B. The intruder cord, $\mathbf{I_C}$, contains the actions

$$
\begin{array}{rcl}
P_C & = & \{A, B, E, I\} \\
HONEST(\mathbf{C}) & = & \{A, B\} \\
\mathbf{I_C} & = & (X\ Y\ Z)[(x)(x/\{\!|X, z|\!\}_{\overline{Z}})\langle\{\!|X, z|\!\}_Y\rangle(w)(w/\{\!|x|\!\}_{\overline{Z}})\langle\{\!|x|\!\}_Y\rangle]_I \\
\sigma_{\mathbf{C}} & = & (\ )[\ ]\langle A\ E\ B\ A\ B\ E\rangle \\
\mathbf{C} & = & \sigma_{\mathbf{C}}; (\mathbf{Init_{NS}} \otimes \mathbf{Resp_{NS}} \otimes \mathbf{I_C}) \\
& = & \mathbf{Init_{NS}}(A\ E) \otimes \mathbf{Resp_{NS}}(B) \otimes \mathbf{I_C}(A\ B\ E) \\
& = & [(\nu x)\langle\{\!|A, x|\!\}_E\rangle(u)(u/\{\!|x, v|\!\}_{\overline{A}})\langle\{\!|v|\!\}_E\rangle]_A \otimes \\
& & [(x)(x/\{\!|Y, z|\!\}_{\overline{B}})(\nu y)\langle\{\!|z, y|\!\}_Y\rangle(w)(w/\{\!|y|\!\}_{\overline{B}})]_B \otimes \\
& & [(x)(x/\{\!|A, z|\!\}_{\overline{E}})\langle\{\!|A, z|\!\}_B\rangle(w)(w/\{\!|x|\!\}_{\overline{E}})\langle\{\!|x|\!\}_B\rangle]_I
\end{array}
$$

Table 3.5: Initial Configuration of Needham-Schroeder Protocol

necessary to intercept the messages from A and make it look like they were intended for B. Note that in this configuration, $\mathbf{I_C}(A\ B\ E)$ is a legal intruder cord, because it does not contain any decryption actions that require the private keys of the honest principals A and B, *i.e.,* $DKeys(\mathbf{I_C}(A\ B\ E)) = \{E\}$.

3.3.4 Events and Traces

A particular initial configuration may give rise to many possible runs. One possible run of the Needham-Schroeder configuration in Table 3.5 is shown in Table 3.6. In this run, all the actions shown in each cord take place. Another possible run of this configuration is one where A's first message is sent to B instead of I, and since B can't decrypt it, no further reactions are possible. In a more complex configuration, there would be many more possible runs.

Since the protocol logic we introduce in Section 3.4 reasons about protocol runs, we need to introduce some additional notation for them.

An *event* is a ground substitution instance of an action, i.e., an action in which all variables have been replaced by terms containing only constants. An event represents the

$$
\begin{array}{rcl}
\mathbf{C} & = & \mathbf{Init_{NS}}(A\ E) \otimes \mathbf{Resp_{NS}}(B) \otimes \mathbf{I_C}(A\ B\ E) \\
& = & [(\nu x)\langle\{\!|A,x|\!\}_E\rangle(u)\ldots]_A \otimes [\ldots]_B \otimes [(x)(x/\{\!|A,z|\!\}_{\overline{E}})\ldots]_I \\
& \rhd\rhd & [\langle\{\!|A,m|\!\}_E\rangle(u)\ldots]_A \otimes [\ldots]_B \otimes [(x)(x/\{\!|A,z|\!\}_{\overline{E}})\ldots]_I \\
& \rhd\rhd & [(u)\ldots]_A \otimes [\ldots]_B \otimes [(\{\!|A,m|\!\}_E/\{\!|A,z|\!\}_{\overline{E}})\langle\{\!|A,z|\!\}_B\rangle\ldots]_I \\
& \rhd\rhd & [\ldots]_A \otimes [(x)(x/\{\!|Y,z|\!\}_{\overline{B}})\ldots]_B \otimes [\langle\{\!|A,m|\!\}_B\rangle(w)\ldots]_I \\
& \rhd\rhd & [\ldots]_A \otimes [(\{\!|A,m|\!\}_B/\{\!|Y,z|\!\}_{\overline{B}})(\nu n)\langle\{\!|z,n|\!\}_Y\rangle\ldots]_B \otimes [(w)\ldots]_I \\
& \rhd\rhd & [(u)(u/\{\!|m,v|\!\}_{\overline{A}})\ldots]_A \otimes [(\nu y)\langle\{\!|m,y|\!\}_A\rangle(w)\ldots]_B \otimes [\ldots]_I \\
& \rhd\rhd & [(u)(u/\{\!|m,v|\!\}_{\overline{A}})\ldots]_A \otimes [\langle\{\!|m,n|\!\}_A\rangle(w)\ldots]_B \otimes [\ldots]_I \\
& \rhd\rhd & [(\{\!|m,n|\!\}_A/\{\!|m,v|\!\}_{\overline{A}})\langle\{\!|v|\!\}_E\rangle]_A \otimes [(w)\ldots]_B \otimes [\ldots]_I \\
& \rhd\rhd & [\langle\{\!|n|\!\}_E\rangle]_A \otimes [\ldots]_B \otimes [(w)(w/\{\!|x|\!\}_{\overline{E}})\ldots]_I \\
& \rhd\rhd & [\ ]_A \otimes [\ldots]_B \otimes [(\{\!|n|\!\}_E/\{\!|x|\!\}_{\overline{E}})\langle\{\!|x|\!\}_B\rangle]_I \\
& \rhd\rhd & [\ ]_A \otimes [(w)(w/\{\!|n|\!\}_{\overline{B}})]_B \otimes [\langle\{\!|n|\!\}_B\rangle]_I \\
& \rhd\rhd & [\ ]_A \otimes [(\{\!|n|\!\}_B/\{\!|n|\!\}_{\overline{B}})]_B \otimes [\ ]_I \\
& \rhd\rhd & [\ ]_A \otimes [\ ]_B \otimes [\ ]_I
\end{array}
$$

Table 3.6: Run Showing Lowe's Anomaly in Needham-Schroeder

result of a reaction step, viewed from the perspective of a single cord that participated in it. For example, if A sends message m to B, then the event $\langle m \rangle$ is a send event of A and (m) is a receive event of B. The initial cord of A may have contained the send action $\langle m \rangle$, or may have contained a send event $\langle x \rangle$, with some action before the send replacing x by m. The initial cord of B must have contained a receive action (y), with the reaction step replacing y by m. Since receive actions contain variables, (m) is not a well-formed action, but we consider this expression an event.

A *trace* is a list of the events by some principal that occur in a run. We use events as a bookkeeping mechanism and as the syntax for writing out traces.

Let $R|_X$ denote the events that occurred for principal X in run R. For example, if R is the run shown in Table 3.6, then we can write $R|_X$ as a trace (for $X = A, B, I$) showing the actions taken by X in run R, as follows

$$\begin{aligned}
R|_A &= [(\nu m)\langle\{\!|A, m|\!\}_E\rangle(\{\!|m, n|\!\}_A)(\{\!|m, n|\!\}_A/\{\!|m, v|\!\}_{\overline{A}})\langle\{\!|n|\!\}_E\rangle] \\
R|_B &= [(\{\!|A, m|\!\}_B)(\{\!|A, m|\!\}_B/\{\!|Y, z|\!\}_{\overline{B}})(\nu n)\langle\{\!|m, n|\!\}_A\rangle(\{\!|n|\!\}_B)(\{\!|n|\!\}_B/\{\!|n|\!\}_{\overline{B}})] \\
R|_I &= [(\{\!|A, m|\!\}_E)(\{\!|A, m|\!\}_E/\{\!|A, z|\!\}_{\overline{E}})\langle\{\!|A, m|\!\}_B\rangle(\{\!|n|\!\}_E)(\{\!|n|\!\}_E/\{\!|x|\!\}_{\overline{E}})\langle\{\!|n|\!\}_B\rangle]
\end{aligned}$$

Note that in this simple example, the list of actions performed by each principal corresponds exactly to the three traces from the run. In a more complex example where a principal acted in more than one role, actions from multiple roles could occur in the same run, and might be interleaved in an arbitrary fashion.

For actions P, protocol $\mathcal{Q}$, run R and principal X, we say "P *matches* $R|_X$ for $\mathcal{Q}$" precisely if $R|_X$ is the interleaving of a set of initial segments of traces of roles of $\mathcal{Q}$ carried out by X, and one of the segments of one of the roles ends in exactly σP, where σ is a substitution of values for variables. If P matches $R|_X$ using substitution σ, then σ is called the *matching substitution*.

Lemma 3.3.1. *For any configuration* $\mathbf{C}$ *of protocol* $\mathcal{Q}$*, and any run* R*, if principal* $X \in HONEST(\mathbf{C})$*, then* $R|_X$ *is an interleaving of traces of roles of* $\mathcal{Q}$ *executed by* X*.*

Proof. This follows from the definition of an initial configuration, which is constructed by assigning one or more roles from $\mathcal{Q}$ to each honest principal. □

A succinct notation for talking about specific events occurring in a run is

$$\begin{aligned} &EVENT(R, X, P, \vec{n}, \vec{x}) \equiv \\ &\quad (([SPS']_X \otimes C \ \ \triangleright\triangleright\ [SS'(\vec{n}/\vec{x})]_X \otimes C') \in R) \end{aligned}$$

In words, $EVENT(R, X, P, \vec{n}, \vec{x})$ means that in run R, principal X executes actions P, receiving data $\vec{n}$ into variables $\vec{x}$, where $\vec{n}$ and $\vec{x}$ are the same length. The predicate can be true for non-empty vectors $\vec{n}$ and $\vec{x}$ only if P is a receive or pattern match action. In the above example,

$EVENT(R, A, \langle\{\!|A, m|\!\}_E\rangle, \emptyset, \emptyset)$,
$EVENT(R, A, \langle\{\!|n|\!\}_E\rangle, \emptyset, \emptyset)$,
$EVENT(R, B, (\{\!|A, m|\!\}_B / \{\!|Y, z|\!\}_{\overline{B}}), \{A, m\}, \{Y, z\})$, and
$EVENT(R, A, (\nu x), m, x)$

are all examples of true facts about run R.

3.3.5 Protocol Properties

In this section, we collect some properties of the class of protocols we will reason about in the rest of the paper.

No Telepathy It is straightforward to check that if a role sends a message, then all the subterms of the message must be values that were either created by the role, received by the role, or that were known to the role from its static parameters.

Lemma 3.3.2 (No Telepathy). *Let Q be a protocol, R be an arbitrary run, X be a principal, and $R|_X$ consist of initial segments of traces $\rho_1, \rho_2, \ldots, \rho_k$, where each ρ_i is a role of Q. Let m be any message sent by X as part of role ρ_i. Then every symbol in the term m is either generated in ρ_i, received in ρ_i, or was in the static interface of ρ_i.*

Proof. This follows from the definition of the cords we use to represent roles. Each role is a closed cord, so all values must be bound. Symbols can be bound by the static interface, or by the ν, receive and pattern match actions. $\square$

Asynchronous communication Cord reactions, as we have defined them, require synchronous communication. That is, a message send action cannot happen in one cord unless a message receive action occurs simultaneously in another cord. Real network communication is asynchronous, though – the network itself effectively provides a buffer in which messages can be stored until somebody is ready to receive them. In order to model this with cords, we pad the intruder cord with an arbitrary number of "$(x)\langle x\rangle$" actions, which we call *forwarding actions*. We call this a *buffering intruder cord*. These forwarding actions model a message being received, and then eventually sent. Because these forwarding actions are independent from (share no variables with) the other actions in the cord, they can occur in any order, which effectively models the asynchronous nature of the network.

We define an *adequate buffering intruder cord* with respect to a configuration $\mathbf{C}$ as a buffering intruder cord that has enough forwarding actions to guarantee that every send action in the configuration can be taken immediately. For any configuration $\mathbf{C}$, we can construct an adequate buffering intruder cord by counting all the send actions occurring in the configuration, and including at least that many forwarding actions in the intruder cord.

Lemma 3.3.3 (Asynchronous Communication). *In a configuration with an adequate buffering intruder cord, any role that wishes to send a message can always send it.*

Proof. Since the buffered intruder cord provides a corresponding receive action for each send in the configuration, this action is always available for reaction with a send action. □

This lemma allows us to schedule reactions runs so that the only place where protocol roles are required to pause is at a receive action. All intermediate actions between receives (i.e. new, pattern match and send actions) can be assumed to all occur in the run.

Standard form traces One useful way of reasoning about protocols, as well as other kinds of computational systems, is by establishing invariance properties. The simplest form of invariance is a property that holds at every possible state of protocol execution. However, some useful properties might only hold at certain states. For example, a property that says that whenever a message of one form is received, another one is sent, depends on the ability of a protocol agent to progress. The property is true if we look at certain "good" states, but may fail at intermediate states where a protocol agent is in the process of responding to a message. For the purpose of strengthening the invariance rule in Section 3.4, we define a set of preferred states and a standard form for traces that progress from one preferred state to another.

We say a protocol role is *hung* if it is unable to complete its next internal action. The only internal action that can hang is a pattern match action. For example, a role such as

$$[(x)(x/\{\!|y, B|\!\}_{\overline{A}}) \ldots]_A$$

can become hung if the data received in the (x) action is $\{\!|B, n|\!\}_A$, which doesn't match the pattern $\{\!|y, B|\!\}_A$.

We say a protocol configuration (a cord space resulting from an initial configuration by a sequence of reaction steps) is a *good state* if no internal actions are pending, and no protocol role is hung. Recall that the reaction steps are divided into communication steps and internal actions. An internal action is pending in a state if there is some role that is able to execute an internal reaction.

A *standard trace* is one where all internal actions occur as early as possible. Starting from an initial configuration, each role takes all available internal actions, bringing the configuration to a state where it is ready to perform an external action.

If we consider traces to be equivalent if they have the same external actions in the same order, then every trace is equivalent to a standard trace. A trace that contains a hung protocol role is equivalent to a trace in which the last message received by the hung role was received by the intruder instead. We can therefore consider only standard traces in our analysis. In keeping with this, whenever we write a protocol role, we may write it in the standard order, with all internal actions listed as early as possible. For example, the $\mathbf{Resp}$ role may be written

$$\mathbf{Resp} = (X)\,[(\nu y)(x)(x/\{\!|Y, z|\!\}_{\overline{X}})\langle\{\!|z, X, y|\!\}_Y\rangle(w)(w/\{\!|y|\!\}_{\overline{X}})]_X$$

with the (νy) action moved all the way to the left.

Secrecy Restrictions For simplicity, our model places a secrecy restriction on runs. By design, it is impossible for any message encrypted with the public key of an honest principal to be decrypted by anybody other than that principal. This follows because 1) we do not allow the intruder cord to contain any static actions that decrypt using an honest principal's key (nor do we model any way for an intruder to "guess" a secret key), and 2) we reason only about protocols where the private keys can only appear in the decrypt pattern match action, meaning the roles can not send out their private keys in any message.

This leads to the following Lemma:

Lemma 3.3.4 (Secrecy). *For any configuration* $\mathbf{C}$ *of protocol* $\mathcal{Q}$*, and any run* R*,*

$$EVENT(R, X, (\{\!|t|\!\}_Y / \{\!|x|\!\}_{\overline{Y}}), t, x) \wedge Y \in HONEST(\mathbf{C}) \supset X = Y.$$

Proof. Let ρ be the role where the event occured in run R. Let σ_C be the substitution used in configuration $\mathbf{C}$. First, suppose $Y \in FV(\rho)$.

If ρ is the intruder role, then ρ cannot contain this action, because $(DKeys(\sigma_C\ ;\ \rho) \cap HONEST(\mathbf{C})) = \emptyset$.

If ρ is a protocol role, then $Dkeys(\sigma_C\ ;\ \rho) = \{X\}$, from the restriction on decryption keys in well-formed protocol roles, so Y can't be in $DKeys(\sigma_C\ ;\ \rho)$ unless $X = Y$.

Suppose $Y \notin FV(\rho)$. Then somehow $\overline{Y}$ (the decryption key) must have been received in a message by ρ. But since the restriction on private keys appearing only in decryption actions means that a well formed protocol role cannot contain any actions that send private keys in a message, this is not possible. □

A direct consequence of this is Axiom SEC in Table 3.8.

While this lemma holds for the system presented in this paper, we believe it is possible to relax the secrecy assumptions and add explicit rules to the logic that allow us to prove this property for appropriate protocols. For example, if the restriction on the use of private keys was removed, then it would still be possible to prove for a certain protocol that private keys were never sent in any message. So, Axiom SEC could still be proven to hold for a specific protocol.

3.4 A Protocol Logic

The protocol logic lets us reason about properties that are guaranteed to hold after a principal has performed a certain sequence of actions.

3.4.1 Syntax

The formulas of the logic are given by the following grammar, where ρ may be any role, written using the notation of the cord calculus (Section 3.3.3):

$$
\begin{array}{lrcl}
\text{(predicate formulas)} & \phi & ::= & \mathsf{Sent}(N,t) \\
 & & & \mathsf{Knows}(N,t) \\
 & & & \mathsf{Source}(t,N,t,N) \\
 & & & \mathsf{Decrypts}(N,t) \\
 & & & \mathsf{Honest}(N) \\
 & & & \phi \wedge \phi \\
 & & & \neg\phi \\
\text{(modal forms)} & \Psi & ::= & \rho\ \phi
\end{array}
$$

Here, t and N are terms and names, as defined in Section 3.2.1. We use ϕ and ψ to indicate predicate formulas, and m to indicate a generic term we call a "message".

Most protocol proofs use formulas of the form $[P]_X \phi$, which means that after X executes actions P, formula ϕ is true about the resulting state of X. Here are the informal interpretations of the predicates, with precise semantics given in the next section:

The formula $\mathsf{Knows}(X, x)$ means that principal X knows information x. This is "knows" in the limited sense of having either generated the data or received it in the clear or received it under encryption where the decryption key is known. If X knows $\{|m|\}_K$ and does not know $\overline{K}$, then X does *not* know m from this message.

The formula $\mathsf{Sent}(X, m)$ means that principal X sent message m. This implies that X knows the message m.

The formula $\mathsf{Decrypts}(X, t)$ means that principal X received a message t in a role that expected the message and was able to decrypt it. Note that $\mathsf{Decrypts}(X, t)$ can only be true if t is of form $\{|m|\}_K$. This implies that X knows the message $\{|m|\}_K$, and the decrypted text, m.

The formula $\mathsf{Honest}(X)$ means that the actions of principal X in the current run are precisely an interleaving of initial segments of traces of a set of roles of the protocol. In other words, X assumes some set of roles and does exactly the actions prescribed by them. A principal may be considered honest in a run if either its key is not corrupted, or the key is corrupted but the intruder only behaves (with this key) according to the roles of the protocol.

The Source predicate is used to reason about the source of a piece of information, such as a nonce. Intuitively, the formula

$$\mathsf{Source}(n, X, t, K)$$

means that the only way for a principal Y different from X to know n is if Y learned n from the message $\{|t|\}_K$, possibly by some indirect path. Note that $\mathsf{Source}(n, X, t, K)$ can only be true if n is a term generated by a (νx) action, and if n is never sent out unencrypted by X.

3.4.2 Semantics

A formula may be true or false at a run of a protocol. More precisely, the main semantic relation, $\mathcal{Q}, R \models \phi$, may be read, "formula ϕ holds for run R of protocol $\mathcal{Q}$." In this relation, R may be a complete run, with all roles that are started in the run completed, or an incomplete run with some principals waiting for additional messages to complete one or more roles.

As preliminaries to the inductive definition of $\mathcal{Q}, R \models \phi$, we make a few definitions regarding runs and the intruder. If $\mathcal{Q}$ is a protocol, then let $\bar{\mathcal{Q}}$ be the set of all initial configurations of protocol $\mathcal{Q}$, each including a possible intruder cord. Let $\mathsf{Runs}(\bar{\mathcal{Q}})$ be the set of all runs of protocol $\mathcal{Q}$ with intruder, as described in Section 3.3.3, each a sequence of reaction steps within a cord space.

We use $m \subseteq m'$ to indicate that m is a subterm of m'.

We define satisfaction $\mathcal{Q}, R \models \phi$, for ϕ without free variables, by induction on ϕ as follows:

- $\mathcal{Q}, R \models \mathsf{Sent}(A, m)$ if $EVENT(R, A, \langle m \rangle, \emptyset, \emptyset)$.

- $\mathcal{Q}, R \models \mathsf{Knows}(A, m)$ if there exists an i such that $\mathsf{Know}_i(A, m)$ where Know_j is defined by induction on j as follows:

  $(\mathsf{Know}_0(A, m)$ if $((m \in FV(R|_A))$
  $\vee\ EVENT(R, A, (\nu x), m, x)$
  $\vee\ EVENT(R, A, (x), m, x)$
  and $\mathsf{Know}_{i+1}(A, m)$ if $(\mathsf{Know}_i(A, m')$
  $\wedge\ ((m' = \{\!|p(t)|\!\}_K \wedge m \subseteq t$
  $\wedge\ EVENT(R, A, (m'/\{\!|p(y)|\!\}_{\overline{K}}), t, y))$
  $\vee\ (m' = p(t) \wedge m \subseteq t$
  $\wedge\ EVENT(R, A, (m'/p(y)), t, y))))$
  or $(\mathsf{Know}_i(A, m') \wedge \mathsf{Know}_i(A, m'')$
  $\wedge\ ((m = m', m'') \vee (m = m'', m')))$
  or $(\mathsf{Know}_i(A, m') \wedge \mathsf{Know}_i(A, K)$
  $\wedge\ m = \{\!|m'|\!\}_K)$

  Intuitively, Know_0 holds for terms that are known directly, either as a free variable of the role, or as the direct result of receiving or generating the term. Know_{i+1} holds for terms that are known by applying i operations (decomposing via pattern matching, or composing via encryption or tupling) to terms known directly.

- $\mathcal{Q}, R \models \mathsf{Decrypts}(A, \{\!|m|\!\}_K)$ if $\mathcal{Q}, R \models \mathsf{Knows}(A, \{\!|m|\!\}_K)$
  $\wedge\ EVENT(R, A, (\{\!|m|\!\}_K/\{\!|x|\!\}_{\overline{K}}), m, x)$

  Note: $\mathsf{Decrypts}(A, n)$ is *false* if $n \neq \{\!|m|\!\}_K$ for some m and K.

- $\mathcal{Q}, R \models \mathsf{Source}(m, A, t, K)$ if
  $EVENT(R, A, (\nu x), m, x) \wedge$
  $\mathcal{Q}, R \models \forall Z.((Z \neq A \wedge \mathsf{Knows}(Z, m)) \supset \mathsf{Decrypts}(Z, \{\!|t|\!\}_K) \vee$
  $(\exists Y.\mathsf{Decrypts}(Y, \{\!|t|\!\}_K) \wedge \mathsf{Sent}(Y, t')))$
  where $m \subseteq t'$

- $\mathcal{Q}, R \models \mathsf{Honest}(A)$ if $A \in HONEST(\mathbf{C})$ in initial configuration $\mathbf{C}$ for R.

- $\mathcal{Q}, R \models (\phi_1 \wedge \phi_2)$ if $\mathcal{Q}, R \models \phi_1$ and $\mathcal{Q}, R \models \phi_2$
- $\mathcal{Q}, R \models \neg\phi$ if $\mathcal{Q}, R \not\models \phi$
- $\mathcal{Q}, R \models [P]_A\phi$ if either P does not match $R|_A$ or P matches $R|_A$ and $\mathcal{Q}, R \models \sigma\phi$, where σ is the substitution matching P to $R|_A$.

If ϕ has free variables, then $Q, R \models \phi$ if we have $Q, R \models \sigma\phi$ for all substitutions σ that eliminate all the free variables in ϕ. We write $\mathcal{Q} \models \phi$ if $\mathcal{Q}, R \models \phi$ for all $R \in \mathsf{Runs}(\bar{\mathcal{Q}})$.

Note that the Source predicate only mentions a single message that contains a nonce created by principal X. We have investigated extensions which support reasoning about the situation where principal X may have sent more than one message containing a nonce he created, but we do not present the semantics here.

3.5 Proof System

3.5.1 Axioms and Rules About Protocol Actions

The axioms and inference rules about protocol actions are listed in Table 3.7. For the most part, these are relatively simple "translations" of actions into atomic formulas of the logic.

Intuitively, Axiom AN1 says that if X generates m, then the only principal that knows m is X. Axiom AN2 says that if principal X generates a new value m and does no further actions in this role, then X knows m, while Axiom AN3 is a somewhat vacuous statement (needed to provide a base case for Axiom S1), which states that if principal X generates a new value m and does no further actions in this role, then any encrypted message could be a possible source for another principal to have learned m.

Axioms AR1 and AR2 are about receiving messages. After X receives a message m, he knows that somebody must have sent it, and he also knows the message m.

Axiom AD1 says that after a principal performs a pattern match action containing a decryption pattern, then the principal has decrypted the message.

Axiom AM1 is about the binding of static variables of a protocol role. In this case the x_i in $(x_1 \ldots x_n)[\ ]_X$, is a value determined when the roles for each participant were

AN1	$[(\nu m)]_X$ $\mathsf{Knows}(Y, m) \supset (Y = X)$
AN2	$[(\nu m)]_X$ $\mathsf{Knows}(X, m)$
AN3	$[(\nu m)]_X$ $\forall t. \forall K.\mathsf{Source}(m, X, t, K)$
AS1	$[\langle m \rangle]_X$ $\mathsf{Sent}(X, m)$
AR1	$[(m)]_X$ $\exists Y.\mathsf{Sent}(Y, m)$
AR2	$[(m)]_X$ $\mathsf{Knows}(X, m)$
AD1	$[(\{\!|m|\!\}_K / \{\!|x|\!\}_{\overline{K}})]_X$ $\mathsf{Decrypts}(X, \{\!|m|\!\}_K)$
AM1	$(x_1 \ldots x_n)[\ ]_X$ $\mathsf{Knows}(X, x_1) \wedge \ldots \wedge \mathsf{Knows}(X, x_n)$
S1	$\dfrac{[P]_X \mathsf{Source}(m, X, t, K)}{[P\langle \{\!|t|\!\}_K \rangle]_X \mathsf{Source}(m, X, t, K)}\ {\scriptstyle m \subseteq t}$

Table 3.7: Axioms and Rules for Protocol Actions

assigned. Typically this will be the identity of the participant, and possibly the identity of other participants an initiator will try to talk to, along with shared keys, etc.

Perhaps the most subtle is the inference rule $\mathsf{S1}$, which says that if X sends a message containing a nonce m, then that message becomes the possible source from which another principal might learn m.

Here is a sample soundness proof, for Axiom $\mathsf{AN1}$. An important part of the soundness argument is that if $[(\nu m)]_X$ matches some run, then (νm) is the *last* action in some role carried out by X. In more detail,

$$\mathcal{Q}, R \models [(\nu m)]_X \ \mathsf{Knows}(Y, m) \supset (Y = X)$$

if $\mathcal{Q}, R \models \sigma\mathsf{Knows}(Y, m) \supset (Y = X)$ whenever (νm) matches $R|_X$, where σ is a matching substitution. By definition of *matches*, (νm) matches $R|_X$ with $\sigma(x) = m$ only if (νx) is the last action of one of the roles of X in R. But since this is the last action from that role in R, X cannot have sent m to any other principal, and from the side conditions for the ν reaction in Table 3.3 we know that m does not occur elsewhere in the cord space. Since no events of form $EVENT(R, X, \langle m' \rangle, \emptyset, \emptyset)$, where $m \subseteq m'$, have occurred, we know from Lemma C.1.1 that there can be no events $EVENT(R, Y, (x), m', x)$. So from the semantics of Knows, $\mathsf{Knows}(Y, m)$ can't be true for any principal besides X. Therefore, from the semantics of Knows, only X knows m. This shows that axiom AN1 is sound.

3.5.2 Axioms Relating Atomic Predicates

Table 3.8 lists axioms relating various propsitional properties, most of which follow naturally from the semantics of propositional formulas. For example, if X decrypts $\{|m|\}_K$, then X knows m because that is the result of the decryption, and if a principal knows a tuple x, y then he also knows x and y.

The SRC axiom is important for reasoning about the source of secret values. If a value m was created by X and has only been sent in the message $\{|t|\}_K$, then if somebody else has learned m, then somebody must have decrypted the message $\{|t|\}_K$.

Axiom SEC follows immediately from Lemma 3.3.4. This axiom says that the decryption keys of honest principals are secret.

DEC1	$\mathsf{Decrypts}(X, \{\|m\|\}_K) \supset \mathsf{Knows}(X, \{\|m\|\}_K)$
DEC2	$\mathsf{Decrypts}(X, \{\|m\|\}_K) \supset \mathsf{Knows}(X, m)$
K1	$\mathsf{Knows}(X, (x, y)) \supset \mathsf{Knows}(X, x) \wedge \mathsf{Knows}(X, y)$
SRC	$\mathsf{Source}(m, X, t, K) \wedge \mathsf{Knows}(Z, m) \wedge Z \neq X \supset$ $\exists Y.\mathsf{Decrypts}(Y, \{\|t\|\}_K)$
SEC	$\mathsf{Honest}(X) \wedge \mathsf{Decrypts}(Y, \{\|m\|\}_X) \supset (Y = X)$

$$\mathsf{CSent}(X, \{|m|\}_K) \equiv \mathsf{Knows}(X, m) \wedge \mathsf{Knows}(X, K) \wedge \mathsf{Sent}(X, \{|m|\}_K)$$

Table 3.8: Relationships Between Properties

Note that CSent is an abbreviation which will be useful in our proofs. It can be thought of as meaning "created and sent".

3.5.3 Preservation Rules

Most predicates are preserved by additional actions. For example, if X knows m before an action, then X will also know m after the action, regardless of what the action is. The reason is that we define the knowledge of X to include all data available to X at any step of the protocol execution. Inference rules showing preservation of properties are shown in Table 3.9.

The exception to preservation is Source. Since $\mathsf{Source}(m, X, t, K)$ means the only way for a principal other than X to know m is from the message $\{|t|\}_K$, this atomic formula may become false if X sends another message containing m.

In all of these formulas, the "principal" referred to is a single principal who may be participating in multiple roles. A consequence of our formulation of protocols is that, while a principal Alice may participate in many instances of many roles at the same time, there will be no communication (other than via message sends and receives) between the various instances if Alice is honest. This is the "No Telepathy Lemma," Lemma 3.3.2.

Generic Rules:

$$\frac{[P]\phi \quad [P]\psi}{[P]\phi \wedge \psi}\ \textsf{G1} \qquad \frac{[P]\phi \quad \phi \supset \psi}{[P]\psi}\ \textsf{G2} \qquad \frac{\phi}{[P]\phi}\ \textsf{G3}$$

Preservation Rules: (For $\mathsf{Persist} \in \{\mathsf{Knows}, \mathsf{Sent}, \mathsf{Decrypts}\}$)

$$\frac{[P]_A \mathsf{Persist}(X,t)}{[P\langle m\rangle]_A \mathsf{Persist}(X,t)}\ \textsf{P1} \qquad \frac{[P]_A \mathsf{Persist}(X,t)}{[P(x)]_A \mathsf{Persist}(X,t)}\ \textsf{P2}$$

$$\frac{[P]_A \mathsf{Persist}(X,t)}{[P(\nu x)]_A \mathsf{Persist}(X,t)}\ \textsf{P3} \qquad \frac{[P]_A \mathsf{Persist}(X,t)}{[P(u/q(x))]_A \mathsf{Persist}(X,t)}\ \textsf{P4}$$

$$\frac{[P]_A \mathsf{Source}(m,Y,t,K)}{[P\langle t'\rangle]_A \mathsf{Source}(m,Y,t,K)}\ (\mathbf{m} \not\subseteq \mathbf{t'})\ \textsf{P5} \qquad \frac{[P]_A \mathsf{Source}(m,Y,t,K)}{[P(x)]_A \mathsf{Source}(m,Y,t,K)}\ \textsf{P6}$$

$$\frac{[P]_A \mathsf{Source}(m,Y,t,K)}{[P(\nu x)]_A \mathsf{Source}(m,Y,t,K)}\ \textsf{P7} \qquad \frac{[P]_A \mathsf{Source}(m,Y,t,K)}{[P(u/q(x))]_A \mathsf{Source}(m,Y,t,K)}\ \textsf{P8}$$

Table 3.9: Inference Rules

$$\frac{\begin{array}{l} \textit{NSL} \vdash (X\ Y)[(\nu x)\langle\{\!|X, x|\!\}_Y\rangle]_X\ \phi(X) \\ \textit{NSL} \vdash (X\ Y)[(\nu x)\langle\{\!|X, x|\!\}_Y\rangle(u)(u/\{\!|x, Y, y|\!\}_{\overline{X}})\langle\{\!|y|\!\}_Y\rangle]_X\ \phi(X) \\ \textit{NSL} \vdash (X)[(\nu y)]_X\ \phi(X) \\ \textit{NSL} \vdash (X)[(\nu y)(u)(u/\{\!|Y, x|\!\}_{\overline{X}})\langle\{\!|x, X, y|\!\}_Y\rangle]_X\ \phi(X) \\ \textit{NSL} \vdash (X)[(\nu y)(u)(u/\{\!|Y, x|\!\}_{\overline{X}})\langle\{\!|x, X, y|\!\}_Y\rangle(w)(w/\{\!|y|\!\}_{\overline{X}})]_X\ \phi(X) \end{array}}{\textit{NSL} \vdash \mathsf{Honest}(X) \supset \phi(X)}\ \mathsf{HON}$$

Table 3.10: Honesty Rule for *NSL*

3.5.4 The Honesty Rule

Intuitively, the honesty rule is used to combine facts about one role with inferred actions of other roles. For example, suppose Alice receives a response from a message sent to Bob. Alice may wish to use properties of Bob's role to reason about how Bob generated his reply, for example. In order to do so, Alice may assume that Bob is honest and derive consequences from this assumption. Perhaps an analogy will help. In the game of bridge, one player may call out a series of bids. The player's partner may then draw conclusions about the cards in the player's hand. If we were to formalize the partner's reasoning, we might use implications such as, "if the player is following the Blackwood bidding convention, then she has three aces." The intuition is that a bid provides a signal to the other player, and the exact meaning of the signal is determined by the bidding conventions that the partners have established. In the same way, a message may imply something specific if the principal sending the message is following the protocol. But if the principal has revealed his private key to the attacker, for example, then receipt of that message does not provide the same information about the principal.

The honesty rule is essentially an invariance rule for proving properties of all roles of a protocol. Since honesty, by definition in our framework, means "following one or more roles of the protocol," honest principals must satisfy every property that is a provable invariant of the protocol roles.

Our notion of "honesty" is a generalization of the notion of "faithfulness" in [Syv93].

There, a participant is *faithful* if they only proceed when they receive the message they are expecting to receive, and if they always start the protocol at the beginning of their role (*i.e.,* a faithful participant will not respond to message 3 of a protocol unless they have already received and responded to messages 1 and 2).

Recall that a protocol $\mathcal{Q}$ is a set of roles, $\mathcal{Q} = \{\rho_1, \rho_2, \ldots, \rho_k\}$. If $\rho \in \mathcal{Q}$ is a role of protocol $\mathcal{Q}$, we write $P \subseteq \rho$ if P is an initial segment of the actions of role ρ such that the next action of ρ after P is a receive, or P is a complete execution of the role. The reason for only considering initial segments up to reads is that we know from the Asynchronous Communication Lemma (Lemma 3.3.3), that if a role contains a send, the send may be done asynchronously without waiting for another role to receive. Therefore, we can assume without loss of generality that the only "pausing" states of a principal are those where the role is waiting for input. If a role calls for a message to be sent, then we dictate that the principal following this role must complete the send before pausing.

Since the honesty rule depends on the protocol, we write $\mathcal{Q} \vdash [P]\phi$ if $[P]\phi$ is provable using the honesty rule for $\mathcal{Q}$ and the other axioms and proof rules.

Using the notation just introduced, the honesty rule may be written

$$\frac{\forall \rho \in \mathcal{Q}. \forall P \subseteq \rho. \mathcal{Q} \vdash (\vec{Z})[P]_X \ \phi}{\mathcal{Q} \vdash \mathsf{Honest}(X) \supset \phi}\ \mathsf{HON} \qquad \begin{array}{l}\text{no free variable in } \phi \\ \text{except } X \text{ bound in} \\ (\vec{Z})[P]_X\end{array}$$

Where $(\vec{Z})[P]_X$ is the initial steps P of role ρ with static variables $(\vec{Z})$, and no free variable in ϕ other than X is bound by $(\vec{Z})[P]_X$. In words, if every role of $\mathcal{Q}$, run either to completion or to a receiving state, satisfies ϕ, then every honest principal executing protocol $\mathcal{Q}$ must satisfy ϕ. The side condition prevents free variables in the conclusion $\mathsf{Honest}(X) \supset \phi$ from becoming bound in any hypothesis.

More concretely, the honesty rule for the *NSL* protocol defined in Equation 5 is shown in Table 3.10. Here, the antecedents of the rule enumerate the intermediate "waiting for input" states and final completed states of each of the roles of the protocol. If some formula ϕ expressible in our logic holds in these five local traces, then ϕ will hold for any honest principal executing this protocol.

The honesty rule is used in the proof of correctness of *NSL* that is given in full in Section 3.6. One place the honesty rule is used is to prove that if B completes the responder role, and the principal A sending the final message to B is honest, then A also sent the initial message to B. The key part of this deduction is to prove the formula

$$\begin{aligned}\phi(A) \quad = \quad & \forall m, n, B.\mathsf{Decrypts}(A, \{\!|m, B, n|\!\}_A) \supset \\ & (\mathsf{CSent}(A, \{\!|A, m|\!\}_B) \wedge \mathsf{CSent}(A, \{\!|n|\!\}_B))\end{aligned}$$

holds for honest participant A. This formula is proved using the honesty rule in line 10 of B's deduction in Table 3.13.

To give some feel for how the honesty rule works in this case, we give an informal, intuitive explanation of how the rule yields ϕ. Referring to the rule as instantiated for *NSL*, we can see that the antecedent $\mathsf{Decrypts}(A, \{\!|m, B, n|\!\}_A)$ is never true for A in the responder role, so ϕ trivially holds in those cases. To prove ϕ for the initiator role, we have two cases. For the case where the initiator never receives a reply to his message, $\mathsf{Decrypts}(A, \{\!|m, B, n|\!\}_A)$ is never true, so ϕ holds. For the case where the initiator receives a reply, we look at the $\mathcal{R}(\mathbf{Init})$ deduction shown in Table 3.11, which shows that $\mathsf{Decrypts}(A, \{\!|m, B, n|\!\}_A)$, $\mathsf{CSent}(A, \{\!|a, m|\!\}_B)$, and $\mathsf{CSent}(A, \{\!|n|\!\}_B)$ are all true, so ϕ holds. Since ϕ holds for any valid sequence of steps that an honest participant A would make, then $\mathsf{Honest}(A) \supset \phi(A)$.

3.5.5 Soundness

Theorem 4 (Soundness). *If* $\mathcal{Q} \vdash \phi$ *then* $\mathcal{Q} \models \phi$.

The proof is an induction on the structure of proofs. Several cases are sketched in Section 3.5.1. The details are included in Appendix C.1.

3.6 Sample Proof

We have constructed proofs for a few different protocols, including public-key and private key protocols (with appropriate modification of the decryption action). In addition, our attempt to prove an important property that holds for the Needham-Schroeder-Lowe protocol, fails for the original Needham-Schroeder protocol in an insightful way, due to an inability of the responder to identify the principal who decrypted the second message with the principal who sent the first and third.

Tables 3.11 through 3.13 contain a formal proof of the property ϕ for the *NSL* protocol.

In an informal sense, Table 3.11 shows the initiator's understanding of what has happened at the end of a successful run of the protocol. The final formula of Table 3.11 is a formula $[P]_X\, \psi$ where P is the trace of actions of the initiator's role in the protocol, and ψ is a formula collecting together a set of the initiator's observations.

In the same informal sense, Table 3.12 shows what the responder can observe by the end of a run of the responder's role of the protocol.

Informally, Table 3.13 shows how for a specific run where Bob thinks he has talked to Alice, Bob can combine reasoning about the protocol roles with his own observations to conclude that if Alice is honest (i.e., the decryption key $\overline{A}$ is not known to the attacker, and Alice has followed her role under *NSL*) then he has communicated with Alice.

The property ϕ is an important correspondence property, but it does not prove the full correctness of *NSL*. The goal here is to demonstrate a simple proof of a property that is provable under our system for *NSL* but is not provable for *NS*.

3.6.1 Failure of the Original Needham-Schroeder Protocol

Note that this proof fails for the original Needham-Schroeder protocol, which omitted the principal name from the second message.

For the original *NS*, we need to prove

$$\begin{aligned}\phi'(A) \quad = \quad & \forall m, n, B.\mathsf{Decrypts}(A, \{\!|m, n|\!\}_A) \supset \\ & (\mathsf{CSent}(A, \{\!|A, m|\!\}_B) \wedge \mathsf{CSent}(A, \{\!|n|\!\}_B))\end{aligned}$$

But when we try to apply the trying the honesty rule on ϕ', in the case where $\mathsf{Decrypts}(A, \{\!|m, n|\!\}_A)$ is true you can come up with a counterexample, for example: $(\mathsf{CSent}(A, \{\!|A, m|\!\}_E) \wedge \mathsf{CSent}(A, \{\!|n|\!\}_E))$.

It is possible to prove

$$\begin{aligned}\phi''(A) &= \forall m, n.\exists X.\mathsf{Decrypts}(A, \{\!|m, n|\!\}_A) \supset \\ &\quad (\mathsf{CSent}(A, \{\!|A, m|\!\}_X) \wedge \mathsf{CSent}(A, \{\!|n|\!\}_X))\end{aligned}$$

(i.e. that A's messages were sent to somebody, but not necessarily to B). This is exactly Lowe's observation about *NS* (where E is a principal whose private key is known to the intruder) [Low95]. B thinks he was talking to A, while A thinks she was talking to E.

3.7 Related Work

The cord notation and proof system presented here is strongly related to the Floyd-Hoare style logic used for making before-after assertions about imperative programs [Flo67, Hoa69]. As in Floyd-Hoare logic, we have axioms that correspond to each action in the calculus, though we use only post-conditions, not pre-conditions.

As explained in Section 3.2, our process calculus developed from an effort to refine the strand space formalism [FHG98] by variables and substitution. Strand spaces seemed attractive as a variant of the natural language of "arrows and messages"; unfortunately, they did not support the logical annotations that we wanted to add to protocols. In concrete analyses, each strand would actually represent a family of strands, parameterized by the possible values of all the data that occurred in it. Communicating a value from one agent to another was then modelled as spontaneous instantiation of the two parameters in the two corresponding strands to the same value. We started by replacing the parameter in the receiving strand by a variable, and modelling communication as substitution of the sent value for that variable. This allowed us to capture syntactically the data known to an agent.[1] Of course, this brought us to the well ploughed ground of process calculi.

Representing communication as reaction of a send-action and a receive-action, with the substitution as the effect of their reaction, is clearly very much in the spirit of π-calculus [MPW92] and chemical abstract machines [BB92]. There has been a substantial amount

[1] The parameter mechanism of strand spaces does not allow a formal distinction between two values in message sent to Bob, one visible to him, and another contained in a token encrypted by Alice's key, that he should forward to her.

$$
\begin{array}{rll}
\mathsf{AM1} & (X\ Y)[\ ]_X\ \mathsf{Knows}(X,X) \wedge \mathsf{Knows}(X,Y) & (1)\\
\mathsf{AN2} & [(\nu m)]_X\ \mathsf{Knows}(X,m) & (2)\\
\mathsf{AS1} & [\langle \{|X,m|\}_Y\rangle]_X\ \mathsf{Sent}(X,\{|X,m|\}_Y) & (3)\\
\mathsf{AD1} & [(\{|m,Y,n|\}_X/\{|m,Y,x|\}_{\overline{X}})]_X\ \mathsf{Decrypts}(X,\{|m,Y,n|\}_X) & (4)\\
\mathsf{AS1} & [\langle \{|n|\}_Y\rangle]_X \mathsf{Sent}(X,\{|n|\}_Y) & (5)\\
1,2,\mathsf{P3} & (X\ Y)[(\nu m)]_X \mathsf{Knows}(X,X) \wedge \mathsf{Knows}(X,Y) \wedge \mathsf{Knows}(X,m) & (6)\\
3,6,\mathsf{P1} & (X\ Y)[(\nu m)\langle \{|X,m|\}_Y\rangle]_X \mathsf{Knows}(X,X) \wedge \mathsf{Knows}(X,Y) \wedge & \\
 & \quad \mathsf{Knows}(X,m) \wedge \mathsf{Sent}(X,\{|X,m|\}_Y) & (7)\\
7,\mathsf{P2} & (X\ Y)[(\nu m)\langle \{|X,m|\}_Y\rangle(\{|m,Y,n|\}_X)]_X \mathsf{CSent}(X,\{|X,m|\}_Y) & (8)\\
4,8,\mathsf{P4} & (X\ Y)[(\nu m)\langle \{|X,m|\}_Y\rangle(\{|m,Y,n|\}_X)(\{|m,Y,n|\}_X/\{|m,Y,x|\}_{\overline{X}})]_X & \\
 & \quad \mathsf{CSent}(X,\{|X,m|\}_Y) \wedge \mathsf{Decrypts}(X,\{|m,Y,n|\}_X) & (9)\\
9,\mathsf{DEC2} & (X\ Y)[(\nu m)\langle \{|X,m|\}_Y\rangle(\{|m,Y,n|\}_X)(\{|m,Y,n|\}_X/\{|m,Y,x|\}_{\overline{X}})]_X & \\
 & \quad \mathsf{CSent}(X,\{|X,m|\}_Y) \wedge \mathsf{Decrypts}(X,\{|m,Y,n|\}_X) \wedge & \\
 & \quad \mathsf{Knows}(X,n) & (10)\\
5,10,\mathsf{P1} & (X\ Y)[(\nu m)\langle \{|X,m|\}_Y\rangle(\{|m,Y,n|\}_X) & \\
 & \qquad (\{|m,Y,n|\}_X/\{|m,Y,x|\}_{\overline{X}})\langle \{|n|\}_Y\rangle]_X & \\
 & \quad \mathsf{CSent}(X,\{|X,m|\}_Y) \wedge \mathsf{Decrypts}(X,\{|m,Y,n|\}_X) \wedge & \\
 & \quad \mathsf{Knows}(X,n) \wedge \mathsf{Sent}(X,\{|n|\}_Y) & (11)\\
11 & (X\ Y)[(\nu m)\langle \{|X,m|\}_Y\rangle(\{|m,Y,n|\}_X) & \\
 & \qquad (\{|m,Y,n|\}_X/\{|m,Y,x|\}_{\overline{X}})\langle \{|n|\}_Y\rangle]_X & \\
 & \quad \mathsf{CSent}(X,\{|X,m|\}_Y) \wedge \mathsf{Decrypts}(X,\{|m,Y,n|\}_X) \wedge & \\
 & \quad \mathsf{CSent}(X,\{|n|\}_Y) & (12)
\end{array}
$$

Table 3.11: $\mathcal{R}(\mathbf{Init})$ – Deductions From $\mathbf{Init}$ Role

AR1	$[(\{\!\|Y,m\|\!\}_X)]_X\ \exists Z.\mathsf{Sent}(Z,\{\!\|Y,m\|\!\}_X)$	(1)
AN3	$[(\nu n)]_X\ \mathsf{Source}(n,X,(m,X,n),Y)$	(2)
AR1	$[(\{\!\|n\|\!\}_X)]_X \exists Z.\mathsf{Sent}(Z,\{\!\|n\|\!\}_X)$	(3)
$1,2,$ P6	$(X)[(\nu n)(\{\!\|Y,m\|\!\}_X)]_X$ $\mathsf{Source}(n,X,(m,X,n),Y) \wedge \exists Z.\mathsf{Sent}(Z,\{\!\|Y,m\|\!\}_X)$	(4)
$4,$ P4,P8	$(X)[(\nu n)(\{\!\|Y,m\|\!\}_X)(\{\!\|Y,m\|\!\}_X/\{\!\|Y',x\|\!\}_{\overline{X}})]_X$ $\mathsf{Source}(n,X,(m,X,n),Y) \wedge \exists Z.\mathsf{Sent}(Z,\{\!\|Y,m\|\!\}_X)$	(5)
$5,$ P1,S1	$(X)[(\nu n)(\{\!\|Y,m\|\!\}_X)(\{\!\|Y,m\|\!\}_X/\{\!\|Y',x\|\!\}_{\overline{X}})\langle\{\!\|m,X,n\|\!\}_Y\rangle]_X$ $\mathsf{Source}(n,X,(m,X,n),Y) \wedge \exists Z.\mathsf{Sent}(Z,\{\!\|Y,m\|\!\}_X)$	(6)
$3,6,$ P2,P6	$(X)[(\nu n)(\{\!\|Y,m\|\!\}_X)(\{\!\|Y,m\|\!\}_X/\{\!\|Y',x\|\!\}_{\overline{X}})\langle\{\!\|m,X,n\|\!\}_Y\rangle(\{\!\|n\|\!\}_X)]_X$ $\mathsf{Source}(n,X,(m,X,n),Y) \wedge \exists Z.\mathsf{Sent}(Z,\{\!\|Y,m\|\!\}_X) \wedge$ $\exists Z'.\mathsf{Sent}(Z',\{\!\|n\|\!\}_X)$	(7)
$7,$ P4,P8	$(X)[(\nu n)(\{\!\|Y,m\|\!\}_X)(\{\!\|Y,m\|\!\}_X/\{\!\|Y',x\|\!\}_{\overline{X}})$ $\langle\{\!\|m,X,n\|\!\}_Y\rangle(\{\!\|n\|\!\}_X)(\{\!\|n\|\!\}_X/\{\!\|n\|\!\}_{\overline{X}})]_X$ $\mathsf{Source}(n,X,(m,X,n),Y) \wedge \exists Z.\mathsf{Sent}(Z,\{\!\|Y,m\|\!\}_X) \wedge$ $\exists Z'.\mathsf{Sent}(Z',\{\!\|n\|\!\}_X)$	(8)

Table 3.12: $\mathcal{R}(\mathbf{Resp})$ – Deductions From $\mathbf{Resp}$ Role

$\mathcal{R}(\mathbf{Resp}), \mathsf{G2}$	$(B)[(\nu n)(\{\mid A, m\mid\}_B)(\{\mid A, m\mid\}_B/\{\mid Y, x\mid\}_{\overline{B}})$ $\langle\{\mid m, B, n\mid\}_A\rangle(\{\mid n\mid\}_B)(\{\mid n\mid\}_B/\{\mid n\mid\}_{\overline{B}})]_B$ $\mathsf{Source}(n, B, (m, B, n), A) \wedge$ $\exists X.(\mathsf{Sent}(X, \{\mid n\mid\}_B)$	(1)
$\mathcal{R}(\mathbf{Resp})$	$(B)[(\nu n)(\{\mid A, m\mid\}_B)\ldots]_B$ $\mathsf{Honest}(B) \supset \neg\mathsf{Sent}(B, \{\mid n\mid\}_B)$	(2)
$1, 2, \mathsf{G1}$	$(B)[(\nu n)(\{\mid A, m\mid\}_B)\ldots]_B\ \mathsf{Source}(n, B, (m, B, n), A) \wedge$ $\exists X.(\mathsf{Sent}(X, \{\mid n\mid\}_B) \wedge (X \neq B))$	(3)
$\mathcal{R}(\mathbf{Init}), \mathsf{HON}$	$\mathsf{Honest}(X) \supset (\mathsf{Sent}(X, \{\mid n\mid\}_B) \supset$ $\mathsf{Knows}(X, n) \wedge \mathsf{Knows}(X, B))$	(4)
$3, 4$	$(B)[(\nu n)(\{\mid A, m\mid\}_B)\ldots]_B\ \mathsf{Source}(n, B, (m, B, n), A) \wedge$ $\exists X.\mathsf{Honest}(X) \supset$ $(\mathsf{Sent}(X, \{\mid n\mid\}_B) \wedge (X \neq B) \wedge \mathsf{Knows}(X, n))$	(5)
SRC	$\mathsf{Source}(n, B, (m, B, n), A) \wedge \exists X.(X \neq B) \wedge$ $\mathsf{Knows}(X, n) \supset \exists Y.\mathsf{Decrypts}(Y, \{\mid m, B, n\mid\}_A)$	(6)
SEC	$\mathsf{Honest}(A) \wedge \mathsf{Decrypts}(Y, \{\mid m, B, n\mid\}_A) \supset (Y = A)$	(7)
$6, 7$	$\mathsf{Honest}(A) \wedge \mathsf{Source}(n, B, (m, B, n), A) \wedge$ $\exists X.(X \neq B) \wedge \mathsf{Knows}(X, n) \supset$ $\mathsf{Decrypts}(A, \{\mid m, B, n\mid\}_A)$	(8)
$5, 8, \mathsf{G1\text{-}3}$	$(B)[(\nu n)(\{\mid A, m\mid\}_B)\ldots]_B$ $\mathsf{Honest}(A) \supset \mathsf{Decrypts}(A, \{\mid m, B, n\mid\}_A)$	(9)
$\mathcal{R}(\mathbf{Init}), \mathsf{HON}$	$\mathsf{Honest}(A) \supset (\mathsf{Decrypts}(A, \{\mid m, B, n\mid\}_A) \supset$ $\mathsf{CSent}(A, \{\mid A, m\mid\}_B) \wedge \mathsf{CSent}(A, \{\mid n\mid\}_B))$	(10)
$9, 10, \mathsf{G2}$	$(B)[(nun)(\{\mid A, m\mid\}_B)\ldots]_B\ \mathsf{Honest}(A) \supset$ $(\mathsf{CSent}(A, \{\mid A, m\mid\}_B) \wedge \mathsf{CSent}(A, \{\mid n\mid\}_B))$	(11)

Table 3.13: B's Completed Deduction With Honesty Rule

of work on developing process calculi of this kind for specific applications in security. In particular, the Spi-calculus [AG99] was developed as an extension of the π-calculus specifically designed for analyzing security by coinductive methods of process calculus, i.e. in terms of bisimilarity of processes.

Our idea was, however, to proceed in a different direction: towards logical semantics of protocols. We needed a process calculus as a rudimentary programming language, to support logical annotations, rather than to directly analyze protocols as processes. The result is the cord calculus. Its core, presented here, is simplified wherever possible: we dropped channels, reducing all communication to broadcasting; pattern-matching was elevated to a generic destructor operator, capturing both value comparison, and decryption. The main purpose of such reductions are the resulting simplifications, that more clearly display the essential logical issues. More or less straightforward extensions of the cord calculus allow representing point-to-point communication, as well as the various cryptographic concepts not considered here .

Several of the concepts presented here have appeared in other work on security protocols. Our use of the ν operator to represent new data is borrowed from process calculus, but is also reminiscent of the use of the existential quantifier used for the same purpose in Multiset Rewriting [CDL$^+$99]. Events are a common idea from process calculus and Hoare logic. In protocol analysis, they have also been used in the NRL Protocol Analyzer [Mea96b], though our notion of event here is more basic and corresponds exactly to the actions in the cord calculus. The Source predicate is used to capture the notion of data origination and casuality. This is similar to the notions of originating and uniquely originating data in strands [THG99].

In [Syv93], a faithfulness assumption and causality criterion are given for protocols, and it was shown that protocols failing to satisfy them were flawed. The "honesty rule" is a generalization and formalization of the faithfulness assumption. By combining the honesty rule with our proof system, we can reason about a protocol based on assumptions about the honesty of its participants.

3.8 Conclusion

We propose a specialized protocol logic that is built around a process language for communicating cords describing the actions of a protocol. The logic contains axioms and inference rules for each of the main protocol actions and proofs are protocol-directed, meaning that the outline of a proof of correctness follows the sequence of actions in the protocol.

A central idea is that assertions associated with an action will hold in any protocol execution that contains this action. This gives us the power to reason about all possible runs of a protocol, without explicitly reasoning about steps that might be carried out by an attacker. At the same time, the semantics of our logic is based on sets of traces of protocol execution (possibly including an attacker), not the kind of abstract idealization found in some previous logics. This approach lets us prove properties of protocols that hold in all runs, without any explicit reasoning about the potential actions of an intruder.

Chapter 4

Conclusion

In this thesis we have formalized the Dolev-Yao model of security protocols by defining two notations, one based on multiset rewriting with existentials, and the other based on a process calculus combined with Hoare logic-style logical assertions. We used the multiset rewriting formalism to analyze the complexity of the secrecy problem in protocol analysis, under various restrictions to message size, number of protocol roles, and number of nonces. We also showed how to use the compositional logic formalism to prove security properties of a simple protocol.

The two notations described here should not be viewed as competing formalisms – both are useful tools for understanding the complexities of designing correct security protocols. Both notations can be used to analyze a specific protocol for design flaws. The compositional logic formalism can be used to find simple proofs of security properties, while the multiset rewriting notation can serve as the core notation that is the basis of automated tools that find protocol flaws. In addition, multiset rewriting can be used for meta-analysis of the security protocol problem, for example to explore the complexity of the secrecy problem, or to prove that optimizations made in a model checker program are correct.

Possible future work includes extending the formalisms to allow them to be applied to more complex protocols with features like timestamps or signatures, relating the MSR formalism to other protocol analysis formalisms such as spi calculus, and extending the compositional features of the logical formalism to allow for the design of complex protocols by composing them from protocol fragments with known provable properties.

Bibliography

[AG99] M. Abadi and A. Gordon. A calculus for cryptographic protocols: the spi calculus. *Information and Computation*, 148(1):1–70, 1999.

[AL00] R. Amadio and D. Lugiez. On the reachability problem in cryptographic protocols. In *International Conference on Concurrency Theory*, pages 380–394, 2000.

[Asp87] A. Asperti. A logic for concurrency. Technical report, Department of Computer Science, University of Pisa, 1987.

[BAN89] M. Burrows, M. Abadi, and R. Needham. A logic of authentication. *Proceedings of the Royal Society, Series A*, 426(1871):233–271, 1989. Also appeared as SRC Research Report 39 and, in a shortened form, in ACM Transactions on Computer Systems 8, 1 (February 1990), 18-36.

[BB90] G. Berry and G. Boudol. The Chemical Abstract Machine. In *Seventeenth Annual ACM Symposium on Principles of Programming Languages*, pages 81–94, San Francisco, California, January 17–19, 1990. ACM Press, New York.

[BB92] G. Berry and G. Boudol. The chemical abstract machine. *Theoretical Computer Science*, 96:217–248, 1992.

[BCJS02] F. Butler, I. Cervesato, A. Jaggard, and A. Scedrov. A formal analysis of some properties of Kerberos 5 using *MSR*. In *15th IEEE Computer Security Foundations Workshop — CSFW-02*, pages 175–190, Cape Breton, NS, Canada, 24–26 June 2002. IEEE Computer Society Press.

[BM93] J.-P. Banâtre and D. Le Métayer. Computing by multiset transformation. *Communcations of the ACM*, 36:98–111, 1993.

[CD98] K. Compton and S. Dexter. Proving authentication protocols in a fragment of linear logic. Manuscript, 1998.

[CDKS00] I. Cervesato, N. Durgin, M. Kanovich, and A. Scedrov. Interpreting strands in linear logic, 2000. In Proc. of FMCS'00.

[CDL+99] I. Cervesato, N. Durgin, P. Lincoln, J. Mitchell, and A. Scedrov. A meta-notation for protocol analysis. In P. Syverson, editor, *12-th IEEE Computer Security Foundations Workshop*. IEEE Computer Society Press, 1999.

[CDM+00] I. Cervesato, N. Durgin, J. Mitchell, P. Lincoln, and A. Scedrov. Relating strands and multiset rewriting for security protocol analysis. In *CSFW: Proceedings of The 13th Computer Security Foundations Workshop*. IEEE Computer Society Press, 2000.

[Cer95] I. Cervesato. Petri nets and linear logic: a case study for logic programming. In M. Alpuente and M. I. Sessa, editors, *Proceedings of the 1995 Joint Conference on Declarative Programming — GULP-PRODE'95*, pages 313–318, Marina di Vietri, Italy, 1995. Palladio Press.

[Cer01a] I. Cervesato. The Dolev-Yao intruder is the most powerful attacker. In J. Halpern, editor, *Proceedings of the Sixteenth Annual Symposium on Logic in Computer Science — LICS'01*, Boston, MA, 16–19 June 2001. IEEE Computer Society Press. Short paper.

[Cer01b] I. Cervesato. A specification language for crypto-protocols based on multiset rewriting, dependent types and subsorting. In G. Delzanno, S. Etalle, and M. Gabbrielli, editors, *Proceedings of the Workshop on Specification, Analysis and Validation for Emerging Technologies — SAVE'01*, pages 1–22, Paphos, Cyprus, 1 December 2001.

[Cer01c] I. Cervesato. Typed MSR: Syntax and examples. In V.I. Gorodetski, V.A. Skormin, and L.J. Popyack, editors, *Proceedings of the First International Workshop on Mathematical Methods, Models and Architectures for Computer Network Security — MMM'01*, pages 159–177, St. Petersburg, Russia, 21–23 May 2001. Springer-Verlag LNCS 2052.

[CJ97] J. Clark and J. Jacob. A survey of authentication protocol literature. Web Draft Version 1.0 available from `http://www.cs.york.ac.uk/~jac`, 1997.

[CJM98] E.M. Clarke, S. Jha, and W. Marrero. Using state space exploration and a natural deduction style message derivation engine to verify security protocols. In *Proc. IFIP Working Conference on Programming Concepts and Methods (PROCOMET)*, 1998.

[CLM81] A. K. Chandra, H. R. Lewis, and J.A. Makowsky. Embedded implicational dependencies and their inference problem. In *Conference Proceedings of the Thirteenth Annual ACM Symposium on Theory of Computation*, pages 342–354, Milwaukee, Wisconsin, 11–13 May 1981.

[Coh00] E. Cohen. Taps: A first-order verifier for cryptographic protocols. In *Proceedings of the Thirteenth IEEE Computer Security Foundations Workshop*, pages 144–158, Cambridge, England, July 2000. IEEE Computer Society Press.

[CP96] I. Cervesato and F. Pfenning. A linear logical framework. In E. Clarke, editor, *Proceedings of the Eleventh Annual Symposium on Logic in Computer Science — LICS'96*, pages 264–275, New Brunswick, NJ, July 1996. IEEE Computer Society Press.

[DEGV97] E. Dantsin, T. Eiter, G. Gottlob, and A. Voronkov. Complexity and expressive power of logic programming. In *Proc. 12th Annual IEEE Conference on Computational Complexity*, 1997.

[DLMS99] N. Durgin, P. Lincoln, J. Mitchell, and A. Scedrov. Undecidability of bounded security protocols, 1999. In N. Heintze and E. Clarke, editors, Proceedings of the Workshop on Formal Methods and Security Protocols — FMSP, Trento, Italy, July 1999.

[DM99a] G. Denker and J. Millen. Capsl intermediate language, 1999.

[DM99b] N. Durgin and J. Mitchell. Analysis of security protocols. In *Calculational System Design, Series F: Computer and Systems Sciences, Vol. 173*. IOS Press, 1999.

[DMT98] G. Denker, J. Meseguer, and C. Talcott. Protocol specification and analysis in Maude. In *Proc. of Workshop on Formal Methods and Security Protocols*, 1998.

[DY83] D. Dolev and A. Yao. On the security of public-key protocols. *IEEE Transactions on Information Theory*, 2(29), 1983.

[EG83] S. Even and O. Goldreich. On the security of multi-party ping–pong protocols. Technical Report 285, Technion – Israel Institute of Technology, 1983.

[End72] H.B. Enderton. *A Mathematical Introduction to Logic*. Academic Press, 1972.

[FHG98] F. Javier Thayer Fábrega, Jonathan C. Herzog, and Joshua D. Guttman. Strand spaces: Why is a security protocol correct? In *Proceedings of the 1998 IEEE Symposium on Security and Privacy*, pages 160–171, Oakland, CA, May 1998. IEEE Computer Society Press.

[FKK96] A. Freier, P. Karlton, and P. Kocher. The SSL protocol version 3.0. `draft-ietf-tls-ssl-version3-00.txt`, November 18 1996.

[Flo67] R. W. Floyd. Assigning meaning to programs. In J. T. Schwartz, editor, *Mathematical aspects of computer science: Proc. American Mathematics Soc. symposia*, volume 19, pages 19–31, Providence RI, 1967. American Mathematical Society.

[GG90a] V. Gehlot and C. Gunter. Normal process representatives. In *Proceedings of Fifth Symposium on Logic in Computer Science*, pages 200–207, Philadelphia, Pennsylvania, June 1990.

[GG90b] V. Gehlot and C. Gunther. Normal process representatives. In *Proceedings of the Fifth Annual IEEE Symposium on Logic in Computer Science — LICS'90*, pages 200–207, Philadelphia, PA, 4–7 June 1990. IEEE Computer Society Press.

[Gir87a] J.-Y. Girard. Linear logic. *Theoretical Computer Science*, 50:1–102, 1987.

[Gir87b] J.-Y. Girard. Linear logic. *Theoretical Computer Science*, 50:1–102, 1987.

[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.

[Gog78] J. Goguen. Order sorted algebra. Technical Report 14. Semantics and Theory of Computation Series, UCLA Computer Science Department, 1978.

[Hoa69] C. A. R. Hoare. An axiomatic basis for computer programming. *Communications of the ACM*, 12(10):576–580, 1969.

[HT96] N. Heintze and J. D. Tygar. A model for secure protocols and their compositions. *IEEE Transactions on Software Engineering*, 22:16–30, 1996. Special section from 1994 IEEE Symposium on Security and Privacy.

[Hui99] A. Huima. Efficient infinite-state analysis of security protocols, 1999.

[Imm86] N. Immerman. Relational queries computable in polynomial time. *Information and Control*, 68:86–104, 1986.

[Kan94] M.I. Kanovich. Linear logic as a logic of computation. *Annals of Pure and Applied Logic*, 67(1–3):183–212, May 1994.

[Klo87] J.W. Klop. Term rewriting: a tutorial. *EATCS Bulletin*, 32:143–182, 1987.

[KMM94] R. Kemmerer, C. Meadows, and J. Millen. Three systems for cryptographic protocol analysis. *J. Cryptology*, 7(2):79–130, 1994.

[KN93] J.T. Kohl and B.C. Neuman. The Kerberos network authentication service (version 5). Internet Request For Comment RFC-1510, September 1993.

[KNT94] J.T. Kohl, B.C. Neuman, and T.Y. Ts'o. *The evolution of the Kerberos authentication service*, pages 78–94. IEEE Computer Society Press, 1994.

[KOS98] M.I. Kanovich, M. Okada, and A. Scedrov. Specifying real-time finite-state systems in linear logic. In *COTIC '98: Second Workshop on Concurrent Constraint Programming for Time-Critical Applications and Multi-Agent Systems*, Nice, France, 1998. Electronic Notes in Theoretical Computer Science, Volume 16, Issue 1. http://www.elsevier.nl/locate/entcs.

[LM92] Kim G. Larsen and Robin Milner. A compositional protocol verification using relativized bisimulation. *Information and Computation*, 99, 1992.

[Low95] G. Lowe. An attack on the Needham-Schroeder public-key protocol. *Info. Proc. Letters*, 56:131–133, 1995.

[Low96] G. Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using CSP and FDR. In *2nd International Workshop on Tools and Algorithms for the Construction and Analysis of Systems*. Springer-Verlag, 1996.

[Mea96a] C. Meadows. Analyzing the Needham-Schroeder public-key protocol: a comparison of two approaches. In *Proc. European Symposium On Research In Computer Security*. Springer Verlag, 1996.

[Mea96b] C. Meadows. The NRL protocol analyzer: an overview. *J. Logic Programming*, 26(2):113–131, 1996.

[Mil92] Robin Milner. Action structures. LFCS report ECS-LFCS-92-249, Department of Computer Science, University of Edinburgh, JCMB, The Kings Buildings, Mayfield Road, Edinburgh, December 1992.

[Mil93a] Robin Milner. Action calculi and the pi-calculus. In *NATO Summer School on Logic and Computation*, Marktoberdorf, November 1993.

[Mil93b] Robin Milner. Action calculi, or syntactic action structures. In *Mathematical Foundations of Computer Science 1993, 18th International Symposium, MFCS'93*, pages 105–121. Springer, 1993.

[Mil99] Robin Milner. *Communicating and Mobile Systems: The π-Calculus*. Cambridge University Press, Cambridge, U.K, 1999.

[Mit96] J.C. Mitchell. *Foundations for Programming Languages*. MIT Press, 1996.

[Mit98] J.C. Mitchell. Analysis of security protocols. Slides for invited talk at CAV '98, available at `http://www.stanford.edu/~jcm`, July 1998.

[MMS97] J.C. Mitchell, M. Mitchell, and U. Stern. Automated analysis of cryptographic protocols using Murφ. In *Proc. IEEE Symp. Security and Privacy*, pages 141–151, 1997.

[MOM89] N. Martí-Oliet and J. Meseguer. From Petri nets to linear logic. In P. Dybjer, A. M. Pitts, D. H. Pitt, A. Poigné, and D. E. Rydeheard, editors, *Proceedings of the Conference on Category Theory and Computer Science*, Springer-Verlag LNCS 389, pages 313–340, Manchester, United Kingdom, September 1989.

[MOM91] N. Marti-Oliet and J. Meseguer. From Petri nets to linear logic. *Mathematical Structures in Computer Science*, 2(2):69–101, March 1991.

[MPW92] R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes, part i. *Information and Computation*, 100(1):1–40, 1992.

[MS96] Y. Matiyasevich and G. Senizergues. Decision problems for semi-thue systems with a few rules. In *Logic in Computer Science*, pages 523–531, 1996.

[NS78] R.M. Needham and M.D. Schroeder. Using encryption for authentication in large networks of computers. *Communications of the ACM*, 21(12):993–999, 1978.

[Pau97a] L.C. Paulson. Mechanized proofs for a recursive authentication protocol. In *10th IEEE Computer Security Foundations Workshop*, pages 84–95, 1997.

[Pau97b] L.C. Paulson. Proving properties of security protocols by induction. In *10th IEEE Computer Security Foundations Workshop*, pages 70–83, 1997.

[Pav97] D. Pavlovic. Categorical logic of names and abstraction in action calculi. *Math. Structures in Comp. Sci.*, 7(6):619–637, 1997.

[Pos46] E. L. Post. A variant of a recursively unsolvable problem. *Bulletion of the American Mathematical Society*, 52:264–268, 1946.

[RC01] A. Scedrov R. Chadha, M. Kanovich. Inductive methods and contract-signing protocols. In *Proceedings of the Eighth ACM Conference on Communications and Security*, pages 176–185, Philadelphia, PA, 2001. ACM Press.

[Ros95] A. W. Roscoe. Modelling and verifying key-exchange protocols using CSP and FDR. In *8th IEEE Computer Security Foundations Workshop*, pages 98–107. IEEE Computer Soc Press, 1995.

[RT01] M. Rusinowitch and M. Turuani. Protocol insecurity with finite number of sessions is np-complete. In *Proceedings of the 14th Computer Security Foundations Workshop*, pages 174–187, Cape Breton, Nova Scotia, Canada, 2001. IEEE Computer Society Press.

[Sch96] S. Schneider. Security properties and CSP. In *IEEE Symp. Security and Privacy*, 1996.

[Sip97] M. Sipser. *Introduction to the Theory of Computation*. PWS Publishing, 1997.

[Son99] D. Song. Athena: a new efficient automatic checker for security protocol analysis. In *Proceedings of the Twelfth IEEE Computer Security Foundations Workshop*, pages 192–202, Mordano, Italy, June 1999. IEEE Computer Society Press.

[SS98] V. Shmatikov and U. Stern. Efficient finite-state analysis for large security protocols. In *Proceedings of the 11th Computer Security Foundations Workshop*, pages 106–115, Rockport, MA, 1998. IEEE Computer Society Press.

[Sv94] P. Syverson and P.C. van Oorschot. On unifying some cryptographic protocol logics. In *Proc. 1994 IEEE Computer Security Foundations Workshop VII*, pages 14–29, 1994.

[Syv93] P. Syverson. Adding time to a logic of authentication. In *ACM Conference on Computer and Communications Security*, pages 97–101, 1993.

[THG99] J. Thayer, J. Herzog, and J. Guttman. Strand spaces: Proving security protocols correct, 1999. Journal of Computer Security, 1999. 15.

[Var82] M. Y. Vardi. The complexity of relational query languages (extended abstract). In *14-th Annual ACM Symposium on Theory of Computing*, pages 137–146, 1982.

[WL93] T. Woo and S. Lam. A semantic model for authentication protocols. In *RSP: IEEE Computer Society Symposium on Research in Security and Privacy*, 1993.

Appendix A

MSR Turing Machine Reductions

A.1 Horn Clause Turing Machine Reductions

A.1.1 Existential Horn Clauses is Undecidable

We use a construction based on axiomatizing a Cook's-theorem-style Turing machine tableau, to prove the undecidability of existential Horn clauses. The notation we use here to encode a Turing machine is similar to that used in Section 2.2.4, though here we are representing an entire Turing machine tableau, and in Section 2.2.4 we were representing the step-by-step computation. Because the tableau is non-linear in nature, all facts that appear in the Horn clauses must be true at all times, so there are necessarily differences between the two encodings. For example, we cannot use the Curr predicate to represent the current state of the machine, because the intruder could replay an out-of-date fact at any

#	$1.q_0$	1	...	0	□	□	#							
#	0	$1.q_1$	...	0	□	□	□	#						
#	$0.q_2$	0	...	0	□	□	□	□	#					
#	1	0	...	0	$1.q$	1	...	1	1	□	...	□	#	
#	1	0	...	0	0	$1.q'$	...	1	1	□	...	□	□	#

Table A.1: Example: Turing Machine Tableau time. Instead, the contents of the Curr predicate is included in the Cont predicate, which includes both the unique name of the cell in the tableau, and the cell's contents.

We construct a tableau describing the computation of a DTM M on a given input $w \in \Sigma^*$, where $|w| = n$, using a set of existential Horn Clauses. An example of the tableau we construct is shown in Table A.1. Also, a nice picture of a tableau similar to the one we use appears in [Sip97, page 255]. The atomic formula $A(b_1, \ldots, b_k)$ mentioned in the statement of the lemma can be an atomic formula that is derivable by a rule that requires, in its hypothesis, that the Turing machine is in a halting state.

Take any language $A \subseteq \Sigma^*$. Let $M = \langle \Sigma, Q, \delta, q_0, Q^+ \rangle$ be a Deterministic Turing Machine (DTM). Here, Σ is a finite alphabet of tape symbols, containing the special blank symbol $\Box$, Q is a finite set of states, $\delta : (Q \times \Sigma) \to \Sigma \times \{L, R\} \times Q$ is the transition relation, $q_0 \in Q$ is the initial state, and $Q^+ \subseteq Q$ is the set of accepting states. Without loss of generality, we assume a Turing machine with a semi-infinite tape that guarantees the head will not run off the left end of the tape, and we assume that every accepting state is a terminal state.

We construct a set of Horn Clauses $H(M, w)$ as follows:

Notation First we define three predicates that describe the contents of the cells in the tableau, and their relationship to each other.

$\mathsf{Cont}(x, a, q)$	Cell x has contents a. If present, q means the tape head is in cell x and the machine is in state q.
$\mathsf{Adj}(x, y)$	Cell x is adjacent to cell y.
$\mathsf{Below}(x, y)$	Cell y is below cell x.

We introduce some special constants. $\mathsf{c_{eot}}$ is a special cell name that labels the cell at the right end of the tape on each row of the tableau. The symbol $@ \notin Q$ is a placeholder for the machine state in those cells that don't contain the tape head. The symbol $\# \notin \Sigma$ is used as the contents for the cells at both ends of the tape.

Transition clauses For each transition relation in δ, we introduce a clause. For a transition that moves the tape head to the left, $\delta(q_i, s) = \{(q_j, s', L)\}$, we have the following:

$$\begin{aligned}
&\forall x, y, z, a, b.[(\mathsf{Adj}(x, y) \wedge \mathsf{Adj}(y, z) \wedge \\
&\mathsf{Cont}(x, a, @) \wedge \mathsf{Cont}(y, s, q_i) \wedge \mathsf{Cont}(z, b, @)) \\
&\implies \exists x', y', z'.((\mathsf{Adj}(x', y') \wedge \mathsf{Adj}(y', z') \wedge \\
&\mathsf{Below}(x, x') \wedge \mathsf{Below}(y, y') \wedge \mathsf{Below}(z, z')) \wedge \\
&\mathsf{Cont}(x', a, q_j) \wedge \mathsf{Cont}(y', s', @) \wedge \mathsf{Cont}(z', b, @))]
\end{aligned}$$

Similarly, for a transition that moves the tape head to the right, $\delta(q_i, s) = \{(q_j, s', R)\}$:

$$\begin{aligned}
&\forall x, y, z, a, b.[(\mathsf{Adj}(x, y) \wedge \mathsf{Adj}(y, z) \wedge \\
&\mathsf{Cont}(x, a, @) \wedge \mathsf{Cont}(y, s, q_i) \wedge \mathsf{Cont}(z, b, @)) \\
&\implies \exists x', y', z'.((\mathsf{Adj}(x', y') \wedge \mathsf{Adj}(y', z') \wedge \\
&\mathsf{Below}(x, x') \wedge \mathsf{Below}(y, y') \wedge \mathsf{Below}(z, z')) \wedge \\
&\mathsf{Cont}(x', a, @) \wedge \mathsf{Cont}(y', s', @) \wedge \mathsf{Cont}(z', b, q_j))]
\end{aligned}$$

Maintenance clauses In addition to the transition clauses, we need several other clauses that are used to construct the rest of the tableau that is not near the tape head.

This clause copies the contents of the tape to the next row in the tableau, creating a new cell below the old one, provided the cell and its neighbors do not contain the tape head.

$$\begin{aligned}
&\forall x, y, z, a, b, c.[(\mathsf{Adj}(x, y) \wedge \mathsf{Adj}(y, z) \wedge \\
&\mathsf{Cont}(x, a, @) \wedge \mathsf{Cont}(y, b, @) \wedge \mathsf{Cont}(z, c, @)) \\
&\implies \exists y'.(\mathsf{Below}(y, y') \wedge \mathsf{Cont}(y', b, @))]
\end{aligned}$$

There is a tape maintenance clause for adding a new cell at the right end of the tape at each step:

$$\forall x, y, a.[\mathsf{Adj}(x, \mathsf{c_{eot}}) \wedge \mathsf{Below}(x, y) \implies \exists z.\mathsf{Adj}(y, z) \wedge \mathsf{Cont}(z, \Box, @) \wedge \mathsf{Adj}(z, \mathsf{c_{eot}})]$$

This rule ensures that the computation can never run off the right end of the tape, since the tape head starts at the leftmost cell, each step can only move the tape head to the right by at most one, and this rule creates a new cell at each step. So at the m'th step of the computation, there are always at least $m + n$ cells in the tape. The right end of the actual tape is infinite, but our tableau only needs to represent a finite number of cells on each row, with the rest of the cells to the right assumed to contain $\square$.

Another special maintenance clause is needed to allow the left marker cell to be copied down to each successive row of the tableau:

$$\forall x, y, a.[\mathsf{Adj}(x, y) \wedge \mathsf{Cont}(x, \#, @) \implies \exists z.\mathsf{Below}(x, z) \wedge \mathsf{Cont}(z, \#, @)]$$

Finally, we need a way to connect together the cells created by the above rules. This clause generates the adjacency facts for cells that are below adjacent cells.

$$\forall x, y, x', y'.[\mathsf{Adj}(x, y) \wedge \mathsf{Below}(x, x') \wedge \mathsf{Below}(y, y') \implies \mathsf{Adj}(x', y')]$$

Initialization The initial state of the DTM has the input $w = w_1 w_2 \ldots w_n$ in the first n cells of the tape, with the rest of the infinite tape containing $\square$, the tape head in the first cell, and the machine state q_0. In our construction we assign the cells in the first row the names $\mathsf{c_0}, \mathsf{c_1}, \ldots, \mathsf{c_n}, \mathsf{c_{n+1}}, \mathsf{c_{n+2}}$, and we include some special marker cells at the ends of the active region of the tape.

We represent the top row of the tableau by describing the adjacency of the cells, and their contents, as follows:

$$\begin{aligned}
&\{\mathsf{Adj}(\mathsf{c_0}, \mathsf{c_1}), \mathsf{Adj}(\mathsf{c_1}, \mathsf{c_2}), \ldots, \\
&\quad \mathsf{Adj}(\mathsf{c_n}, \mathsf{c_{n+1}}), \mathsf{Adj}(\mathsf{c_{n+1}}, \mathsf{c_{n+2}}), \mathsf{Adj}(\mathsf{c_{n+2}}, \mathsf{c_{eot}}), \\
&\mathsf{Cont}(\mathsf{c_0}, \#, @), \mathsf{Cont}(\mathsf{c_1}, w_1, q_0), \mathsf{Cont}(\mathsf{c_2}, w_2, @), \ldots, \\
&\quad \mathsf{Cont}(\mathsf{c_n}, w_n, @), \mathsf{Cont}(\mathsf{c_{n+1}}, \square, @), \mathsf{Cont}(\mathsf{c_{n+2}}, \square, @), \mathsf{Cont}(\mathsf{c_{eot}}, \#, @)\}
\end{aligned}$$

Note: The 2 extra blank cells after the input are provided to ensure correct operation of the construction on an input of $w = \epsilon$.

Termination The acceptance condition is represented by a set of clauses that derive the ACCEPT fact:

$$\begin{aligned}&\forall x, a.[\mathsf{Cont}(x, a, q_{a1}) \implies \mathsf{ACCEPT}\\&\mathsf{Cont}(x, a, q_{a2}) \implies \mathsf{ACCEPT}\\&\ldots]\end{aligned}$$

for each $q_{ai} \in Q^+$.

Lemma A.1.1. $H(M, w) \vdash \mathsf{ACCEPT}$ *if and only if machine M halts in an accepting state on input string w.*

Proof. We first show that if machine M halts in an accepting state on input string w, then $H(M, w) \vdash \mathsf{ACCEPT}$. The accepting computation of M can be represented by an accepting tableau, as described earlier and illustrated in Table A.1, where each line of the tableau corresponds to a configuration of the accepting computation. If M halts in an accepting state on input string w after f steps, that means there is a sequence of configurations $c_0, c_1, \ldots c_f$, such that c_0 is the initial configuration (i.e. the top line of the tableau), c_f is a configuration with the state $q_f \in Q^+$, and for each consecutive configuration c_i, c_{i+i}, c_{i+1} can be obtained from c_i by applying some rule δ_i from δ.

By construction, the Initialization clauses ensure that initial Horn clauses in $H(M, w)$ correspond to configuration c_0. If a state corresponding to configuration c_i can be reached by applying clauses in $H(M, w)$, then the state corresponding to c_{i+1} can be reached by applying the Transition clause that corresponds to the rule δ_i, plus the Maintenance clauses. The Transition clause builds the cells near the tape head, and the Maintenance clauses build the other cells in the configuration, and ensure that they are connected together properly. These clauses together can be used to construction a state that corresponds to configuration c_{i+1}. Finally, when configuration c_f is reached in the Turing machine tableau, $\mathsf{Cont}(x, a, q_f)$ will be true for some cell x, so the Termination clause can be applied to derive ACCEPT from $H(M, w)$.

Now we show that if $H(M, w) \vdash \mathsf{ACCEPT}$, then machine M halts in an accepting state on input string w. If $H(M, w) \vdash \mathsf{ACCEPT}$, then $\mathsf{Cont}(x, a, q_f)$ must be derivable for some cell x and some state $q_f \in Q^+$. By construction, the initial Horn clauses in $H(M, w)$

$\langle 0,0\rangle$ $\#$	$\langle 1,0\rangle$ 1.q$_0$	$\langle 2,0\rangle$ 1	$\dots$	$\langle n,0\rangle$ 0	$\langle n+1,0\rangle$ $\square$	$\langle n+2,0\rangle$ $\square$	$\dots$	$\langle N-1,0\rangle$ $\square$	$\langle N,0\rangle$ $\#$
$\langle 0,1\rangle$ $\#$	$\langle 1,1\rangle$ 0	$\langle 2,1\rangle$ 1.q$_1$	$\dots$	$\langle n,1\rangle$ 0	$\langle n+1,1\rangle$ $\square$	$\langle n+2,1\rangle$ $\square$	$\dots$	$\langle N-1,1\rangle$ $\square$	$\langle N,1\rangle$ $\#$
$\langle 0,2\rangle$ $\#$	$\langle 1,2\rangle$ 0.q$_2$	$\langle 2,2\rangle$ 0	$\dots$	$\langle n,2\rangle$ 0	$\langle n+1,2\rangle$ $\square$	$\langle n+2,2\rangle$ $\square$	$\dots$	$\langle N-1,2\rangle$ $\square$	$\langle N,2\rangle$ $\#$
$\langle 0,N\rangle$ $\#$	$\langle 1,N\rangle$	$\langle 2,N\rangle$	$\dots$	$\langle n,N\rangle$	$\langle n+1,N\rangle$	$\langle n+2,N\rangle$	$\dots$	$\langle N-1,N\rangle$	$\langle N,N\rangle$ $\#$

Table A.2: Example: DEXP Turing Machine Tableau

correspond to the Turing machine's initial configuration, c_0. Only the Termination clauses can be used to derive ACCEPT, and only the Transition clauses can be used to create new cells whose contents contain the tape head (and thus the machine state). So the sequence of transition clauses used to derive ACCEPT corresponds exactly to the sequence of transition rules in δ that are used in the accepting computation of M. $\square$

Lemma A.1.2. *The implication problem for existential Horn clauses without function symbols is undecidable. In particular, there is no algorithm for deciding whether a set of existential Horn clauses without function symbols implies a single atomic formula $A(b_1, \dots, b_k)$ without function symbols or variables.*

Proof. This follows from Lemma A.1.1. Since we can reduce the halting problem to the implication problem for existential Horn clauses without function symbols, the implication problem is undecidable. $\square$

A.1.2 Non-existential Horn Clauses is DEXP-Hard

We show that if a DTM M halts in $N = 2^{n^\ell}$ steps on a given input w, where $|w| = n$, then M can be simulated by a set of existential-free Horn Clauses. As for the unbounded Turing Machine in the previous section, we will construct a tableau representing the computation. An example of such a tableau is in Table A.2.

Take any language A in DEXP. Let $M = \langle \Sigma, Q, \delta, q_0, Q^+ \rangle$ be a Deterministic Turing Machine (DTM) that decides A in $N = 2^{n^\ell}$ time for some constant ℓ, and input size n.

Here, Σ is a finite alphabet of tape symbols, containing the special blank symbol $\Box$, Q is a finite set of states, $\delta : (Q \times \Sigma) \to \Sigma \times \{L, R\} \times Q$ is the transition relation, $q_0 \in Q$ is the initial state, and $Q^+ \subseteq Q$ is the set of accepting states. Without loss of generality, we assume a one-tape Turing machine that guarantees that a program running in $N = 2^{n^\ell}$ time will not run off the end of the tape, and we assume that every accepting state is a terminal state.

We construct a set of Horn Clauses $H(M, w, 2^{n^\ell})$, where $w \in A$ is the input, with $|w| = n$, as follows:

Notation We use the same notation as for the unbounded Turing machine in Appendix A.1.1, though we represent the cell number symbolically, breaking it up into two parts, which can be viewed as representing the row and column of the cell. The cell number is actually composed of $n^\ell + 1$ binary digits for the row and column respectively, each bit a separate argument to the predicate.

A cell position is a pair of numbers, with each number represented by a sequence of bits. For conciseness, we use the following abbreviations:

$$\begin{aligned}
\mathsf{Cont}(p, a, q) &\equiv \mathsf{Cont}(\langle n, m\rangle, a, q) \\
\mathsf{Cont}(\langle n, m\rangle, a, q) &\equiv \mathsf{Cont}(\langle \vec{x}, \vec{y}\rangle, a, q) \\
\mathsf{Cont}(\langle \vec{x}, \vec{y}\rangle, a, q) &\equiv \mathsf{Cont}(x_0, \ldots, x_{n^\ell}, y_0, \ldots, y_{n^\ell}, a, q)
\end{aligned}$$

where p is any position $\langle n, m\rangle$ and n and m are the numbers represented by the bit vectors $\vec{x}$ and $\vec{y}$. We use similar abbreviations for the cell numbers in the Adj and Below predicates.

Transition Clauses For each transition relation in δ, we introduce a clause. For a transition that moves the tape head to the left, $\delta(q_i, s) = \{(q_j, s', L)\}$, we have the following:

$$\begin{aligned}
&\forall x, x', y, y', z, z', a, b.[(\mathsf{Adj}(x, y) \wedge \mathsf{Adj}(y, z) \wedge \\
&\mathsf{Cont}(x, a, @) \wedge \mathsf{Cont}(y, s, q_i) \wedge \mathsf{Cont}(z, b, @)) \wedge \\
&\mathsf{Below}(x, x') \wedge \mathsf{Below}(y, y') \wedge \mathsf{Below}(z, z') \\
&\implies \mathsf{Cont}(x', a, q_j) \wedge \mathsf{Cont}(y', s', @) \wedge \mathsf{Cont}(z', b, @))
\end{aligned}$$

And similarly for a transition that moves the tape head to the right.

Maintenance clauses This clause copies the contents of the tape to the next row in the tableau, provided the cell and its neighbors do not contain the tape head.

$$\begin{aligned}&\forall x, x', y, y', z, z', a, b, c.[(\mathsf{Adj}(x, y) \wedge \mathsf{Adj}(y, z) \wedge \\ &\mathsf{Cont}(x, a, @) \wedge \mathsf{Cont}(y, b, @) \wedge \mathsf{Cont}(z, c, @)) \wedge \mathsf{Below}(y, y') \\ &\implies \mathsf{Cont}(y', b, @))\end{aligned}$$

This clause copies the contents of the marker cells (which will be initialized to be the the left and right margins of the first row) to the cells below.

$$\forall x, y.[(\mathsf{Below}(x, y) \wedge \mathsf{Cont}(x, \#, @) \implies \mathsf{Cont}(y, \#, @))$$

Initialization As for the unbounded Turing machine, the initial state of the DTM has the input $w = w_1 w_2 \ldots w_n$ in the first n cells of the tape, with the rest of the infinite tape containing $\square$, the tape head in the first cell, and the machine state q_0. As described above, we use symbolic tape cell names of the form $\langle x, y \rangle$, to label the cells. We also include some special marker cells at the ends of the tape.

We represent the top row of the tableau by describing the initial contents of the tape, with the input word first, and the rest of the tape row containing blanks.

$$\{\mathsf{Cont}(\langle 1, 0\rangle, w_1, q_0), \mathsf{Cont}(\langle 2, 0\rangle, w_2, @), \ldots, \mathsf{Cont}(\langle n, 0\rangle, w_n, @)\}$$

and

$$\{\mathsf{Cont}(\langle n+1, 0\rangle, \square, @), \mathsf{Cont}(\langle n+2, 0\rangle, \square, @), \ldots, \mathsf{Cont}(\langle 2^{n^k} - 1, 0\rangle, \square, @)\}$$

We use a set of $n^\ell + 1$ adjacency facts to describe the horizontal connections between the cells in the tableau. Here we use the notation 1^n to indicate a string of n 1's, and 0^n to indicate a string of n 0's.

$$
\begin{array}{l}
\{ \\
\forall \vec{x}, \vec{y}.[\mathsf{Adj}(\langle \vec{x}0, \vec{y} \rangle, \langle \vec{x}1, \vec{y} \rangle] \\
\forall \vec{x}, \vec{y}.[\mathsf{Adj}(\langle \vec{x}01, \vec{y} \rangle, \langle \vec{x}10, \vec{y} \rangle)] \\
\forall \vec{x}, \vec{y}.[\mathsf{Adj}(\langle \vec{x}011, \vec{y} \rangle, \langle \vec{x}100, \vec{y} \rangle)] \\
\ldots \\
\forall \vec{y}.[\mathsf{Adj}(\langle 01^{n^\ell}, \vec{y} \rangle, \langle 10^{n^\ell}, \vec{y} \rangle)] \\
\}
\end{array}
$$

And we need a set of $n^\ell + 1$ belowness facts to describe the vertical connections between the cells.

$$
\begin{array}{l}
\{ \\
\forall \vec{x}, \vec{y}.[\mathsf{Below}(\langle \vec{x}, \vec{y}0 \rangle, \langle \vec{x}, \vec{y}1 \rangle] \\
\forall \vec{x}, \vec{y}.[\mathsf{Below}(\langle \vec{x}, \vec{y}01 \rangle, \langle \vec{x}, \vec{y}10 \rangle)] \\
\forall \vec{x}, \vec{y}.[\mathsf{Below}(\langle \vec{x}, \vec{y}011 \rangle, \langle \vec{x}, \vec{y}100 \rangle)] \\
\ldots \\
\forall \vec{x}.[\mathsf{Below}(\langle \vec{x}, 01^{n^\ell} \rangle, \langle \vec{x}, 10^{n^\ell} \rangle)] \\
\}
\end{array}
$$

Finally, we need to initialize the contents of the special marker cells on the two ends of the tape, the left margin and right margin of the first row:

$$\{\mathsf{Cont}(\langle 0, 0 \rangle, \#, @), \mathsf{Cont}(\langle 2^{n^\ell}, 0 \rangle, \#, @)\}$$

Termination The acceptance condition is represented by a set of clauses that derive the ACCEPT fact:

$$
\begin{array}{l}
\forall x, a.[\mathsf{Cont}(x, a, q_{a1}) \implies \mathsf{ACCEPT} \\
\mathsf{Cont}(x, a, q_{a2}) \implies \mathsf{ACCEPT} \\
\ldots]
\end{array}
$$

Lemma A.1.3. $H(M, w, N) \vdash \mathsf{ACCEPT}$ *if and only if machine* M *accepts the input string* w *of length* n *within* $N = 2^{n^\ell}$ *steps.*

Proof. This follows from an argument similar to Lemma A.1.1, though the cells are named symbolically using constants, and the Turing machine tableau is slightly different, as described above, and illustrated in Table A.2. □

Lemma A.1.4. *The implication problem for Horn clauses without function symbols or existentials is in* DEXP*-hard. In particular, an algorithm for deciding whether a set of existential Horn clauses without function symbols implies a single atomic formula* $A(b_1, \ldots, b_k)$ *without function symbols, variables or existentials runs in time exponential in the size of the input formula.*

Proof. This follows from Lemma A.1.3. Since we can reduce the decision problem for a DEXP-time Turing machine to the implication problem for Horn clauses without existentials or function symbols, the implication problem is DEXP-hard. □

Appendix B

Compositional Logic Syntax

B.1 Summary of Syntax

For convenience, we present a summary in Table B.1 of the BNF syntax for cords and the protocol logic, as presented throughout the paper.

(names)	N	$::=$	X	variable name
			A	constant name
(basic keys)	K_0	$::=$	k	constant key
			y	variable key
			N	name
(keys)	K	$::=$	K_0	basic key
			$inv(K_0)$	inverse key
(terms)	t	$::=$	x	variable term
			c	constant term
			N	name
			K	key
			t, t	tuple of terms
			$\{\|t\|\}_K$	term encrypted with key K
(actions)	a	$::=$	ϵ	the null action
			$\langle t \rangle$	send a term t
			(x)	receive term into variable x
			(νx)	generate new term x
			$(t/q(x_1, \ldots, x_n))$	match term t to pattern q
(basic terms)	b	$::=$	$x \mid c \mid N \mid K$	basic terms allowed in patterns
(basic patterns)	p	$::=$	$b, \ldots, b$	tuple pattern
(patterns)	q	$::=$	p	basic pattern
			$\{\|p\|\}_{\overline{K}}$	decryption pattern
(strands)	S	$::=$	$aS \mid a$	
(processes)	r	$::=$	$(x \ldots x)C\langle x \ldots x \rangle$	
(formulas)	ϕ	$::=$	$\mathsf{Sent}(N, t)$	
			$\mathsf{Knows}(N, t)$	
			$\mathsf{Source}(t, N, t, N)$	
			$\mathsf{Decrypts}(N, t)$	
			$\mathsf{Honest}(N)$	
			$\phi \wedge \phi$	
			$\neg\phi$	
(modal forms)	Ψ	$::=$	$\rho\ \phi$	

Table B.1: Summary of Syntax

Appendix C

Compositional Logic Proofs

C.1 Soundness of Axioms and Proof Rules

Here we prove Theorem 4, the soundness of the axioms and proof rules. Section C.1.1 proves the soundness of the axioms, Section C.1.2 proves the soundness of the relationships between the predicates, and Section C.1.3 proves the soundness of the proof rules.

First we prove some lemmas that will be useful for the proofs that follow. In the following lemmas, R indicates a run of any protocol.

Lemma C.1.1. $EVENT(R, X, \langle m\rangle, \emptyset, \emptyset) \supset \exists Y.EVENT(R, Y, (x), m, x)$ *and* $EVENT(R, X, (x), m, x) \supset \exists Y.EVENT(R, Y, \langle m\rangle, \emptyset, \emptyset)$

Proof. This follows from the definitions of the basic cord calculus reactions.

Given a procotol $\mathcal{Q}$ with run R, where S and S' are arbitrary actions, recall that $EVENT(R, X, \langle m\rangle, \emptyset, \emptyset)$ means that the reduction

$$([S\langle m\rangle S']_X \otimes C) \;\; \triangleright\triangleright \;\; ([SS']_X \otimes C')$$

occurs in run R. Refering to the list of basic reaction steps in Table 3.3, we see that this is a reaction of type (1). Plugging in that rule, we get

$$([S\langle m\rangle S']_X \otimes [T(x)T']_Y \otimes D) \;\; \triangleright\triangleright \;\; ([SS']_X \otimes [TT'(m/x)]_Y \otimes D)$$

with side condition $FV(m) = \emptyset$. Note that these are the same reduction, if $C = [T(x)T']_Y \otimes D$ and $C' = [TT'(m/x)]_Y \otimes D$. But by taking $E = ([S\langle m\rangle S']_X \otimes D$ and $E' = [SS']_X \otimes D$, this reduction can also be written as

$$([T(x)T']_Y \otimes E) \;\; \triangleright\triangleright \;\; ([TT'(m/x)]_Y \otimes E')$$

which from the definition of $EVENT$ means $EVENT(R, Y, (x), m, x)$.

Similarly for the other direction. □

Lemma C.1.2. *If $EVENT(R, X, \sigma\langle\{|x|\}_Y\rangle, \emptyset, \emptyset)$ for some protocol $\mathcal{Q}$, where $\sigma x = m$ and $\sigma Y = K$, then $\mathcal{Q}, R \models \mathsf{Knows}(X, K) \wedge \mathsf{Knows}(X, m)$.*

Proof. This also follows from the definitions of the cord calculus – no term containing a free variable can be sent.

Let R' be the run containing all the events of run R up to, but not including, the $\langle\{|m|\}_K\rangle$ event.

Given a procotol $\mathcal{Q}$ with run R, where S and S' are arbitrary actions, recall that $EVENT(R, X, \langle\{|m|\}_K\rangle, \emptyset, \emptyset)$ means that the reduction

$$([S\langle\{|m|\}_K\rangle S']_X \otimes C) \;\; \triangleright\triangleright \;\; ([SS']_X \otimes C')$$

occurs in run R. Refering to the list of basic reaction steps in Table 3.3, we see that this is a reaction of type (1). Plugging in that rule, we get

$$\begin{aligned} &([S\langle\{|m|\}_K\rangle S']_X \otimes [T(x)T']_Y \otimes D) \\ &\qquad\qquad \triangleright\triangleright \quad ([SS']_X \otimes [TT'(\{|m|\}_K/x)]_Y \otimes D) \end{aligned}$$

with side condition $FV(\{|m|\}_K) = \emptyset$. In order for this reaction to occur, we have the condition that $\{|m|\}_K$ can not contain free variables at the time of the reaction. That means one of the following must be true in the run R' for K

1. $(K) \in R'|_X$, i.e. $EVENT(R', X, (x), K, x)$

2. $(\nu K) \in R'|_X$, i.e. $EVENT(R', X, (\nu x), K, x)$

3. $K \in FV(R'|_X)$ (i.e. K appears in the static interface to the role)

4. $(p(t)/p(x)) \in R|_X$, for $K \subseteq t$ *i.e.,* $EVENT(R, X, (p(t)/p(x)), t, x)$

5. $(\{|p(t)|\}_{K'}/\{|p(x)|\}_{\overline{K'}}) \in R|_X$ for $K \subseteq t$,
   i.e. $EVENT(R, X, (\{|p(t)|\}_{K'}/\{|p(x)|\}_{\overline{K'}}), t, x)$

In the first three cases, the semantics of Knows gives us $\mathsf{Know}_0(X, K)$, so $\mathsf{Knows}(X, K)$. In the fourth case, K occurred in some pattern match action in R', which means $\mathsf{Know}_i(X, p(K))$, so $\mathsf{Know}_{i+1}(X, K)$ and therefore $\mathsf{Knows}(X, K)$. In the last case, K occurred in some decryption action in R', which means $\mathsf{Know}_i(X, \{|p(t)|\}_{K'})$, so $\mathsf{Know}_{i+1}(X, K)$ and therefore $\mathsf{Knows}(X, K)$. If m is atomic (contains no subterms), then the same argument holds for m.

If m is not atomic, then m must be built by some constructors in the cord calculus. The possibilities are as follows

1. $m = m', m''$

2. $m = \{|m'|\}_{m''}$

In both of the above cases, m' and m'' can contain no free variables, so if m' and m'' are atomic, we have $\mathsf{Know}_i(X, m')$ and $\mathsf{Know}_i(X, m'')$ from the argument above. If m' or m'' is not atomic, then they must have been constructed and again they can contain no free variables, so we repeat the argument for each component until all components are atomic.

Therefore, $\mathsf{Knows}(X, m)$ and $\mathsf{Knows}(X, K)$, so the lemma is proved. □

C.1.1 Axioms

In the following subsections we prove the soundness of the axioms from Table 3.7. Each section starts with an informal statement of the axiom, and then proves its soundness.

C.1.1.1 AN1 $[(\nu m)]_X$ $\mathsf{Knows}(Y, m) \supset (Y = X)$

Informally, Axiom AN1 says that if a principal X generates a new value m and takes no further actions, then the only principal who knows m is X.

By definition,

$$\mathcal{Q}, R \models [(\nu m)]_X \ \mathsf{Knows}(Y, m) \supset (Y = X)$$

if $\mathcal{Q}, R \models \sigma(\mathsf{Knows}(Y, m) \supset (Y = X))$ whenever (νm) matches $R|_X$, where σ is a matching substitution. By definition of *matches*, (νm) matches $R|_X$ with $\sigma(x) = m$ only if (νx) is the last action of one of the roles of X in R. But since this is the last action from that role in R, X cannot have sent m to any other principal, and from the side conditions for the ν reaction in Table 3.3 we know that m does not occur elsewhere in the cordspace. Since no events of form $EVENT(R, X, \langle m' \rangle, \emptyset, \emptyset)$, where $m \subseteq m'$, have occurred, we know from Lemma C.1.1 that there can be no events $EVENT(R, Y, (x), m', x)$. So from the semantics of Knows, $\mathsf{Knows}(Y, m)$ can't be true for any principal besides X. Therefore, from the semantics of Knows, only X knows m. This shows that axiom AN1 is sound.

C.1.1.2 AN2 $[(\nu m)]_X$ $\mathsf{Knows}(X, m)$

Informally, Axiom AN2 says that if a principal X generates a new value m and takes no further actions, then after that action the principal X knows m.

By definition,

$$\mathcal{Q}, R \models [(\nu m)]_X \ \mathsf{Knows}(X, m)$$

if $\mathcal{Q}, R \models \sigma(\mathsf{Knows}(X, m))$ whenever (νm) matches $R|_X$, where σ is a matching substitution. By definition of *matches*, (νm) matches $R|_X$ with $\sigma(x) = m$ only if (νx) is the last action of one of the roles of X in R.

In this case, we have $EVENT(R, X, (\nu x), m, x)$, which from the semantics for Knows, means $\mathsf{Know}_0(X, m)$.

C.1.1.3 AN3 $[(\nu m)]_X$ $\forall t.\forall K.\mathsf{Source}(m, X, t, K)$

Informally, Axiom AN3 says that if a principal X generates a new value m and takes no further actions, then after that action any encrypted message could be the source from which a principal other than X could have learned m. This is a vacuous statement, since

Axiom AN1 tells us that only X could know m at this point anyway.

By definition,

$$\mathcal{Q}, R \models [(\nu m)]_X \ \forall t.\forall K.\mathsf{Source}(m, X, t, K)$$

if $\mathcal{Q}, R \models \sigma(\forall t.\forall K.\mathsf{Source}(m, X, t, K))$ whenever (νm) matches $R|_X$, where σ is a matching substitution. By definition of *matches*, (νm) matches $R|_X$ with $\sigma(x) = m$ only if (νx) is the last action of one of the roles of X in R.

In this case we have $EVENT(R, X, (\nu x), m, x)$, and we know from Axiom AN1 that $\mathsf{Knows}(Y, m) \supset X = B$, so from the semantics of Source we have a false antecedent, so we can prove $\mathsf{Source}(m, X, t, K)$ for any term t and key K.

C.1.1.4 AS1 $[\langle m \rangle]_X$ $\mathsf{Sent}(X, m)$

Informally, Axiom AS1 says that if a principal X sends a message m, then the principal has sent m.

By definition,

$$\mathcal{Q}, R \models [\langle m \rangle]_X \ \mathsf{Sent}(X, m)$$

if $\mathcal{Q}, R \models \sigma(\mathsf{Sent}(X, m))$ whenever $\langle m \rangle$ matches $R|_X$, where σ is a matching substitution. By definition of *matches*, $\langle m \rangle$ matches $R|_X$ only if $\langle x \rangle$ with $\sigma(x) = m$ is the last action of one of the roles of X in R.

In that case we have $EVENT(R, X, \sigma\langle x \rangle, \emptyset, \emptyset) = EVENT(R, X, \langle m \rangle, \emptyset, \emptyset)$. So, from the semantics of Sent, this means $\mathsf{Sent}(X, m)$.

C.1.1.5 AR1 $[(m)]_X$ $\exists Y.\mathsf{Sent}(Y, m)$

Informally, Axiom AR1 says that if a principal X receives a message m, then that message was sent by some principal Y.

By definition,

$$\mathcal{Q}, R \models [(m)]_X \ \exists Y.\mathsf{Sent}(Y, m)$$

if $\mathcal{Q}, R \models \sigma(\exists Y.\mathsf{Sent}(Y, m))$ whenever (m) matches $R|_X$, where σ is a matching substitution. By definition of *matches*, (m) matches $R|_X$ only if (x) with $\sigma(x) = m$ is the last action of one of the roles of X in R.

In this case, we have $EVENT(R, X, (x), m, x)$. From Lemma C.1.1, we know that $EVENT(R, X, (x), m, x) \supset \exists Y.EVENT(R, Y, \langle m \rangle, \emptyset, \emptyset)$. From the semantics of Sent, this means $\exists Y.\mathsf{Sent}(Y, m)$.

C.1.1.6 AR2 $[(m)]_X$ $\mathsf{Knows}(X, m)$

Informally, Axiom AR2 says that if a principal X receives a message m, then the principal X knows m.

By definition,

$$\mathcal{Q}, R \models [(m)]_X \ \mathsf{Knows}(X, m)$$

if $\mathcal{Q}, R \models \sigma(\mathsf{Knows}(X, m))$ whenever (m) matches $R|_X$, where σ is a matching substitution. By definition of *matches*, (m) matches $R|_X$ only if (x) with $\sigma(x) = m$ is the last action of one of the roles of X in R.

In this case, we have $EVENT(R, X, (x), m, x)$, which from the semantics for Knows, means $\mathsf{Know}_0(X, m)$.

C.1.1.7 AM1 $(x_1 \ldots x_n)[\,]_X$ $\mathsf{Knows}(X, x_1) \wedge \ldots \wedge \mathsf{Knows}(X, x_n)$

Informally, Axiom AM1 says a principal X always knows its own static parameters.

By definition,

$$\mathcal{Q}, R \models (x_1 \ldots x_n)[\,]_X \ \mathsf{Knows}(X, x_1) \wedge \ldots \wedge \mathsf{Knows}(X, x_n)$$

if $\mathcal{Q}, R \models \sigma(\mathsf{Knows}(X, x_i))$ whenever $x_i \in FV(R|_X)$, where σ is a matching substitution. Since the x_i are static parameters of the cord, the substitution for x_i is the substitution cord σ_C of the initial configuration of the run. That is, if $\sigma_{\mathbf{C}}(x_i) = A$, then $\mathsf{Knows}(A, m)$.

From the semantics of Knows, this means $\mathsf{Know}_0(X, x_i)$.

C.1.1.8 AD1 $[(\{|m|\}_K/\{|y|\}_{\overline{K}})]_X$ $\mathsf{Decrypts}(X, \{|m|\}_K)$

Informally, Axiom AD1 says that a principal X has decrypted the message $\{|m|\}_K$, if it knows a message $\{|m|\}_K$ and successfully pattern matched it against $\{|y|\}_{\overline{K}}$.

By definition,

$$\mathcal{Q}, R \models [(\{|m|\}_K/\{|y|\}_{\overline{K}})]_X \mathsf{Decrypts}(X, \{|m|\}_K)$$

if $\mathcal{Q}, R \models \sigma(\mathsf{Decrypts}(X, \{|m|\}_K))$ whenever $(\{|m|\}_K/\{|y|\}_{\overline{K}})$ matches $R|_X$, where σ is a matching substitution.

By definition of *matches*, $(\{|m|\}_K/\{|y|\}_{\overline{K}})$ matches $R|_X$ only if $(\{|x|\}_K/\{|y|\}_{\overline{K}})$ with $\sigma(x) = m$ is the last action of one of the roles of X in R.

In that case we have $EVENT(R, X, (\{|m|\}_K/\{|y|\}_{\overline{K}}), m, y)$, and in order for the pattern match action to take place, $FV(\{|m|\}_K) = \emptyset$, which means $\mathsf{Knows}(X, \{|m|\}_K)$. From the semantics of Decrypts, this means $\mathsf{Decrypts}(X, \{|m|\}_K)$.

C.1.2 Relationships Between Predicates

C.1.2.1 DEC1 $\mathsf{Decrypts}(X, \{|m|\}_K) \supset \mathsf{Knows}(X, \{|m|\}_K)$

Informally, DEC1 says that if a principal X decrypts a message $\{|m|\}_K$, then X knows the message $\{|m|\}_K$.

From the semantics of Decrypts, we have $EVENT(R, X, (\{|m|\}_K/\{|x|\}_{\overline{K}}), m, x)$ and $\mathsf{Knows}(X, \{|m|\}_K)$, so this follows trivially.

C.1.2.2 DEC2 $\mathsf{Decrypts}(X, \{|m|\}_K) \supset \mathsf{Knows}(X, m)$

Informally, DEC2 says that if a principal X decrypts a message $\{|m|\}_K$, then X knows the message m.

From the semantics of Decrypts, we have $EVENT(R, X, (\{|m|\}_K/\{|x|\}_{\overline{K}}), m, x)$ and $\mathsf{Knows}(X, \{|m|\}_K)$. From the decryption pattern match case of the semantics of Knows, with $p(m) = m$, that means $\mathsf{Knows}(X, m)$.

C.1.2.3 SEC $\mathsf{Honest}(X) \wedge \mathsf{Decrypts}(Y, \{\{\!|m|\!\}_X\}) \supset (Y = X)$

Informally, SEC says that if a principal X is honest, and some principal Y has decrypted a message $\{\!|m|\!\}_X$ (i.e. a message encrypted with X's public key), then Y must be X. In other words, if X is honest, then nobody but X knows $\overline{X}$ (X's private key).

This axiom follows from Lemma 3.3.4, which says for any configuration $\mathbf{C}$ of protocol $\mathcal{Q}$, and any run R,

$$EVENT(R, X, (\{\!|t|\!\}_Y / \{\!|x|\!\}_{\overline{Y}}), t, x) \wedge X \in HONEST(\mathbf{C}) \supset X = Y.$$

Since $\mathsf{Honest}(X)$ means $X \in HONEST(\mathbf{C})$, and $EVENT(R, X, (\{\!|t|\!\}_Y / \{\!|x|\!\}_{\overline{Y}}), t, x)$ means $\mathsf{Decrypts}(X, \{\!|t|\!\}_Y)$, this follows directly.

C.1.2.4 SRC $\mathsf{Source}(m, X, t, K) \wedge \mathsf{Knows}(Z, m) \wedge Z \neq X \supset \exists Y.\mathsf{Decrypts}(Y, \{\!|t|\!\}_K)$

Informally, SRC says that if the only source of message m is the message $\{\!|t|\!\}_K$, then if anybody (besides principal X who created m) knows m, then somebody must have decrypted the message $\{\!|t|\!\}_K$.

From the semantics of $\mathsf{Source}(m, X, t, K)$ we have

$$\begin{aligned}&\forall Z.(Z \neq X \wedge \mathsf{Knows}(Z, m)) \supset \\ &\quad \mathsf{Decrypts}(Z, \{\!|t|\!\}_K) \vee (\exists Y.\mathsf{Decrypts}(Y, \{\!|t|\!\}_K) \wedge \mathsf{Sent}(Y, t'))\end{aligned}$$

And since $\mathsf{Knows}(Z, m) \wedge Z \neq X$, that means the antecedent is statisfied, so $\exists Y.\mathsf{Decrypts}(Y, \{\!|t|\!\}_K)$.

C.1.2.5 K1 $\mathsf{Knows}(X, x, y) \supset \mathsf{Knows}(X, y) \wedge \mathsf{Knows}(X, x)$

Informally, K1 says that if a principal X knows the tuple x, y, then the principal knows x and y.

This follows from the semantics of Knows. If $\mathsf{Knows}(X, x, y)$, then $\mathsf{Know}_i(X, x, y)$ for some i, and therefore $\mathsf{Know}_{i+1}(X, x)$ and $\mathsf{Know}_{i+1}(X, y)$.

C.1.3 Proof Rules

C.1.3.1 S1 – The Source rule

$$\mathsf{S1}\quad \frac{[P]_X \mathsf{Source}(m, X, t, K)}{[P\langle\{\!|t|\!\}_K\rangle]_X \mathsf{Source}(m, X, t, K)}\ \mathbf{m \subseteq t}$$

Informally, Axiom S1 says that when principal X sends a message containing m, that message becomes the possible source from which another principal might obtain m.

Let R' be the run in the premise, and R be the continuation of R' in the conclusion. First note that the premise and the semantics of Source give us $EVENT(R', B, (\nu x), m, x)$ for P matching $R'|_X$, so we know $EVENT(R, B, (\nu x), m, x)$ for $P\langle\{\!|t|\!\}_K\rangle$ matching $R|_X$.

Assume $\mathsf{Knows}(X, m) \wedge X \neq B$.

We know from the semantics of Knows that $\mathsf{Knows}(X, m)$ can be true in a run R in the following circumstances:

1. $(\nu m) \in R|_X$, i.e. $EVENT(R, X, (\nu x), m, x)$

2. $m \in FV(R|_X)$ (i.e. m appears in the static interface to the role)

3. $(m) \in R|_X$, i.e. $EVENT(R, X, (x), m, x)$

4. $(p(t')/p(x)) \in R|_X$, for $m \subseteq t'$ *i.e.,* $EVENT(R, X, (p(t')/p(x)), t', x)$

5. $(\{\!|p(t')|\!\}_{K'}/\{\!|p(x)|\!\}_{\overline{K'}}) \in R|_X$ for $m \subseteq t'$,
   i.e.$EVENT(R, X, (\{\!|p(t')|\!\}_{K'}/\{\!|p(x)|\!\}_{\overline{K'}}), t', x)$

If we assume $X \neq B$, then the first possibility is ruled out, because we know $EVENT(R, B, (\nu x), m, x)$, which means from reaction rule (4) that m couldn't appear anywhere else in the cordspace at the time of the (νx) action. If $EVENT(R, B, (\nu x), m, x)$ occured in run R first, then $EVENT(R, X, (\nu x), m, x)$ for $X \neq B$ could not occur later, and vice versa.

Similarly, the second possibility is ruled out, because again we know $EVENT(R, B, (\nu x), m, x)$, which means from reaction rule (4) that m couldn't appear anywhere else in the cordspace, in particular not in the static interface of any of the roles.

For the last case, if $t = t'$ and $K = K'$, then $\mathsf{Decrypts}(X, \{|t|\}_K)$, and we are done.

Otherwise, X received either m directly, or in some pattern containing m. But in order for $EVENT(R, X, (x), t', x)$ with $m \subseteq t'$, we know from Lemma C.1.1 that $\exists Y.EVENT(R, Y, \langle t' \rangle, \emptyset, \emptyset)$, and we know from Lemma C.1.2 that all values in the term t', in particular m, must have been known. So in order for $\mathsf{Knows}(Y, m)$, either $\mathsf{Decrypts}(Y, \{|t|\}_K)$ or $EVENT(R, Y, (x), t'', x)$ with $m \subseteq t''$. We repeat this argument until we find the Y that actually decrypted $\{|t|\}_K$.

Thus we have

$$
\begin{aligned}
&EVENT(R, B, (\nu x), m, x) \wedge \\
&\quad ((X \neq B \wedge \mathsf{Knows}(X, m)) \supset \mathsf{Decrypts}(X, \{|t|\}_K) \vee \\
&\quad (\exists Y.\mathsf{Decrypts}(Y, \{|t|\}_K) \wedge \mathsf{Sent}(Y, t')))\text{where } m \subseteq t'
\end{aligned}
$$

which from the semantics of Source means $\mathsf{Source}(m, B, m', K)$.

C.1.3.2 HON – The Honesty Rule

For protocol $\mathcal{Q}$, the honesty rule is

$$
\frac{\forall \rho \in \mathcal{Q}.\forall P \subseteq \rho.\mathcal{Q} \vdash (\vec{Z})[P]_X\ \phi}{\mathcal{Q} \vdash \mathsf{Honest}(X) \supset \phi}\ \mathsf{HON} \qquad \begin{array}{l}\text{no free variable in } \phi \\ \text{except } X \text{ bound in} \\ (\vec{Z})[P]_X\end{array}
$$

Where $(\vec{Z})[P]_X$ is the initial steps P of role ρ with static variables $(\vec{Z})$, and no free variable in ϕ other than X is bound by $(\vec{Z})[P]_X$. Assume $\mathcal{Q} \vdash (\vec{Z})[P]_X\ \phi$ for all $P \subseteq \rho, \rho \in Q$. By soundness for shorter proofs, we assume $\mathcal{Q}, R \models (\vec{Z})[P]_X\ \phi$ for all runs R of protocol $\mathcal{Q}$.

We must show that $\mathcal{Q}, R \models \mathsf{Honest}(X) \supset \phi$, for any run R. Let σ be any substitution such that $\sigma(\mathsf{Honest}(X) \supset \phi)$ has no free variables, and let $A = \sigma(X)$. Then we must show that if $\mathcal{Q}, R \models \mathsf{Honest}(A)$ then $\mathcal{Q}, R \models \sigma\phi$.

Assume $\mathcal{Q}, R \models \mathsf{Honest}(A)$. Then from the semantics of Honest and Lemma 3.3.1, $R|_A$ is an interleaving of traces of roles of $\mathcal{Q}$ carried out by A. Consider any standard trace in $R|_A$. By definition of honesty, this trace is matched by some prefix P of a role ρ in $\mathcal{Q}$.

Let τ be the matching substitution. Because $\mathcal{Q}, R \models (\vec{Z})[P]_A\ \sigma\phi$, and P matches $R|_A$ by τ, we conclude that $\mathcal{Q}, R \models \tau(\sigma\phi)$. But since $(\vec{Z})[P]_X$ does not bind any variables in ϕ except X, $\tau(\sigma\phi)$ is just $\sigma\phi$, and therefore $\mathcal{Q}, R \models \sigma\phi$.

C.1.3.3 Generic Rules

G1 follows from the semantics of "$\wedge$", i.e.

- $\mathcal{Q}, R \models (\phi_1 \wedge \phi_2)$ if $\mathcal{Q}, R \models \phi_1$ and $\mathcal{Q}, R \models \phi_2$
- $\mathcal{Q}, R \models [P]_X\phi$ if P matches $R|_X$ implies $\mathcal{Q}, R \models \phi$.

Similarly, G2 follows from the meaning of the logical connectives.

G3 is valid because if ϕ is true after any run, then ϕ is true after a specific run that contains actions P.

C.1.3.4 Preservation Rules

Since the semantics of Knows, Sent, and Decrypts are all based on the existence of a certain event in a run, adding additional events to the run cannot make these predicates false, so they are always preserved.

$\mathsf{Source}(m, X, t, k)$ is not preserved if additional messages other than $\{|t|\}_K$ are sent that contain the nonce m, because it is now possible for $\mathsf{Knows}(Y, m) \wedge X \neq Y$ to be true without the message $\{|t|\}_K$ ever being decrypted. So Source can become false if additional messages are sent that contain m.