Reasoning about Concurrency for Security Tunnels

Alwyn E. Goodloe
University of Pennsylvania

Carl A. Gunter
University of Illinois at Urbana-Champaign

Abstract

There has been excellent progress on languages for rigorously describing key exchange protocols and techniques for proving that the network security tunnels they establish preserve confidentiality and integrity. New problems arise in describing and analyzing establishment protocols and tunnels when they are used as building blocks to achieve high-level security goals for network administrative domains. We introduce a language called the tunnel calculus *and associated analysis techniques that can address functional problems arising in the concurrent establishment of tunnels. In particular, we use the tunnel calculus to explain and resolve cases where interleavings of establishment messages can lead to deadlock. Deadlock can be avoided by making unwelcome security compromises, but we prove that it can be eliminated systematically without such compromises using a concept of* session *to relate tunnels. Our main results are noninterference and progress theorems familiar to the concurrency community, but not previously applied to tunnel establishment protocols.*

1 Introduction

Security tunnels are a common networking technique in which a pair of nodes share state that enables them to apply transformations to messages to ensure their security. There has been a great deal of study of protocols that establish security tunnels to demonstrate that they ensure integrity and confidentiality. However, there is less study of how these protocols can serve as building blocks to achieve the goals of network administrative domains. In such applications, there is often the need to coordinate the use of multiple tunnels to enforce the policies of security gateways and endpoints involved in the communication. For instance, a mobile client may need to set up a tunnel to a Network Access Server (NAS) to gain access to the Internet, and then a tunnel to a Virtual Private Network (VPN) to gain access to a company Intranet, and then finally a tunnel to an company server to access a resource. Such tunnel complexes can occur in many forms. For instance, rather than having the client set up its own 'voluntary' tunnel to the VPN, the NAS may set up an 'involuntary' tunnel to the VPN on behalf of the client. In the current state-of-practice, such interactions are often undesirable: examples like the ones above often involve two or three levels of redundant encryption at the client node because of poor coordination between tunnels in diverse network layers and at security gateways protecting diverse administrative domains. A more subtle collection of problems arise when security gateways aim to assure that all of the messages they admit are fully authenticated and authorized, and nodes aim to find and traverse these gateways dynamically. In this case tunnels must be used to find gateways and establish more tunnels. These problems—coordinating tunnels, dynamic discovery, authenticated traversal, and nested tunnels—have proved to be a difficult challenge. We need a formal notation to describe and reason about them.

In this paper we introduce a way to describe and analyze functional concurrency properties for the establishment and use of collections of security tunnels between endpoints and gateways. Our formalism is called the *tunnel calculus*. It models tunnels using layers that describe forwarding, secure processing, establishment, and authorization. These layers can be used to describe discovery protocols that dynamically find and coordinate the traversal of security gateways in accordance with high-level policies. In this paper we focus on the secure processing and establishment layers and how the calculus allows us to describe and reason about functional properties arising from concurrency between tunnel establishment negotiations. In particular, we demonstrate how such problems can arise, how they cannot be trivially or painlessly avoided by straight-forward techniques, and how they can be addressed with a suitable form of identifier. In our approach, collections of related tunnels can be negotiated in a form of distributed 'session' distinguished by an identifier that plays a role similar to a port number.

Suitable variations on the tunnel calculus could be used to model existing IPsec network layer security tunnels and the Internet Key Exchange (IKE) protocol, but the aim of the tunnel calculus is to provide an abstract foundation for designing future tunnel protocols in light of their use in tun-

20th IEEE Computer Security Foundations Symposium (CSF'07)
0-7695-2819-8/07 $20.00

nel complexes. For example, IPsec has been plagued by complications related to nested tunnels and dynamic discovery of security gateways. These problems arise in significant part from functional complications that have not been modeled theoretically. The tunnel calculus supplies a formalism to address this gap. It is a multiset rewrite system that can be described with modular groups of rules (the layers) and models the state used with tunnels. It differs from other formalisms like the spi calculus [1] and MSR [4] in that it focuses on functional properties, discovery protocols, and relating high- and low-level authorization policies rather than the confidentiality and integrity guarantees implied by the key exchange protocols, which the tunnel calculus treats as primitives. Our focus in this paper is on functional properties, specifically deadlock conditions arising in concurrent runs of the establishment protocol.

We demonstrate three theorems for the tunnel calculus. *Observational Commutativity* asserts that the order of execution of commands in distinct sessions can be interchanged without essentially changing the semantics of the execution trace. *Noninterference* asserts if there is a trace that establishes a communication in a 'virginal' network where only one pair of parties communicate, then if this communication occurs in a network with other communicating parties as well, the result will be equivalent. *Progress* asserts that if a communication between two parties is possible in a given trace, then it is possible to extend any other trace to complete the communication as well. Although the Noninterference and Progress Theorems are asserted in terms of pairs of nodes, the complexity in the results arises from the way in which tunnels are established at security gateways between the pairs of nodes.

The paper is organized into eight sections and an appendix. The second section gives some general background on IPsec and the configuration of policies and tunnels in networks. The third section describes mathematical foundations and notation needed to understand the tunnel calculus. The fourth section describes the concurrency problems that interest us. In the fifth section, we describe the tunnel calculus precisely, including its concept of session identifier. The sixth section introduces the trace theory used in formulating our theorems. The seventh section provides the three core theorems with associated lemmas. The eighth section concludes. The appendix provides a complete listing of the grammar and rules for the first three layers of the tunnel calculus together with most of the semantic functions used in these layers.

2 Motivation

Security tunnel protocols are used commonly on the Internet. SSL/TLS [7] is a transport layer tunnel protocol that is ubiquitously used for web security and electronic commerce. Secure Shell (SSH) [22] is widely used as a secure remote login for Unix systems. Perhaps the most interesting and ambitious tunnel protocol is IPsec [16, 15], which was designed by IETF to provide network layer tunnels and is projected to be a fundamental part of IPv6, the next generation of the Internet Protocol (IP). In this section we motivate the tunnel calculus by taking an abstract look at tunnels in general and IPsec in particular.

From a high-level perspective, a tunnel protocol can be viewed 'type-theoretically' as follows. A node a communicates with a node g by wrapping each message m it sends to g within a constructor C. Node g holds a corresponding destructor C^-, which it applies to get the message m. The constructor C represents the bulk protocol between a and g. Node a may have a policy that all messages sent to b must be wrapped in C and node g may have a policy that messages from a must be wrapped in C. To set this up, there is an *establishment protocol* that causes a and g to obtain C and C^- respectively in such a way that they authenticate each other, authorize the use of the constructor, and assure that they are the only parties that have these operators.

In IPsec, the constructor and destructor are called a *Security Association (SA)* and are collected in an *SA Database (SAD)*. The messages are individual IP packets and the SAs are indexed by *Security Parameter Index (SPI)*. The rules that determine which SAs are used with which messages are called *IPsec Security Policies (SPs)* and are held in an *SP Database (SPD)*. Although it is not critical for this paper, which does not discuss high-level authorization policies, we would in general like to distinguish IPsec security policies in the SPD from higher-level policies that determine them, so we will hereafter refer to IPsec-style security policies as *security mechanisms*. IPsec establishes tunnels using the Internet Key Exchange (IKE) protocol. The most recent version IKEv2 [13] uses four messages between a and g to set up a tunnel. This establishment is triggered by a security mechanism that indicates that messages from a to g must be in a tunnel: if no such tunnel currently exists, a run of IKE creates C and C^- and enters them in the SAD. We can abstract the establishment protocol by viewing it as two messages: a request and a response. This provides enough detail to model the important issue for the tunnel calculus, which is when the state that determines the subsequent packet processing is written at each of the nodes.

The most common use of IPsec, often called the *road warrior scenario* because of its use by employees traveling outside their enterprise network administrative domain, is depicted in Figure 1. In this scenario, a wishes to communicate with a node b that is located within a protected administrative domain, while a is located outside of it. To do this, a must convince a security gateway g protecting the domain that it is authorized to access b, so it establishes a tunnel from a to g and dispatches messages to b by encapsu-

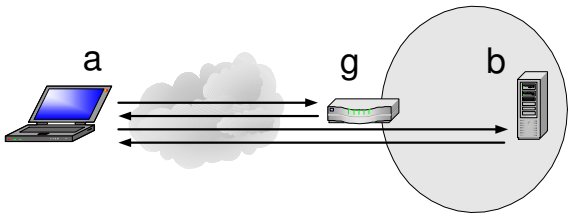

Figure 1. Road Warrior

lating them in messages to g, which are delivered within the tunnel from a to g so they can be efficiently authenticated and authorized for forwarding on to b. Where things start to get tricky is when the gateway g must be dynamically discovered or where a needs tunnels for other nodes besides g. Let us consider each of these cases in turn.

In something as simple as the road warrior scenario, it is plausible that a manually configures a mechanism for its home administrative domain that mentions g specifically. However, in a more general case, a is automatically given this mechanism or learns it by trying to communicate with b. An example of the first is given by Cisco's Dynamic Multipoint VPN (DM VPN) [5] system, which optimizes tunnels in hub and spoke configurations (not the road warrior scenario) by setting up tunnels to connect the nodes at the spokes directly, thereby relieving load on the hub. The second case is illustrated by Cisco's Tunnel Endpoint Discovery (TED) [8] protocol, which locates the gateway by sending a discovery packet from a to b which is intercepted by g because it is on the communication path between these nodes (guarding the administrative domain). Gateway g then informs a of its need for a tunnel. In these and more elaborate cases, we could benefit from notations to describe the exact messages that will propagate the mechanisms and the high-level policies that will determine them.

Special care must be taken when more than one security mechanism is need for a single destination. The early versions of IPsec included provisions for *nested* tunnels to accommodate this. For instance, suppose a wishes to protect its communications to both g and b. To do this it may establish a tunnel to both nodes. The resulting 'type theory' causes a message from a to b to have the form $C(D(m))$. Gateway g holds the destructor for the outer constructor C and node b has the one for D. This entails two IKE exchanges, as shown in Figure 1, the first setting up a tunnel between a and g, the second taking place within that tunnel. In particular, the request packet for the tunnel to b is sent to g in the form $C(\mathsf{Req})$ inside of the tunnel required to gain admission to the domain in which b resides. This sort of mechanism management has proved tricky, however, and many implementations of IPsec do not support it. A compromise is to enable IPsec establishment and bulk packets to pass through g without authenticating them. This breaks what we can call the *authenticated traversal* rule, which says that all packets passing g must be authenticated before they can be authorized for traversal. Although IPsec packets may be less harmful to the protected network than some other types of packets, it is certainly not desirable to admit them without knowing where they came from.

Our main focus in this paper is on the following problem, which arises in enforcing the authenticated traversal rule. Suppose a wishes to establish a tunnel to b at the same time that b wishes to establish one to a. At some point, each node aims to set up a constructor and invoke a rule saying that it must be used in their subsequent communications. However, the point at which this occurs can cause the nodes to mutually miss the requests from the other party, resulting in a deadlock in which neither party can form a message the other will accept. We introduce a solution and discuss it in four steps. First, we introduce a strategy for formalization. Second, we go into more detail to discuss the main focus of the paper, which is a deadlock problem that can arise in designs that support dynamic creation of security mechanisms, nested tunnels, and the authenticated traversal rule. Third, we introduce the tunnel calculus and a notion of session to address these concurrency problems. Fourth and finally, we prove that the new calculus and its sessions have a number of the desired properties, including freedom from deadlock.

3 Modeling Tunnels

A packet can be modeled as a term formed by applying the constructor P to a triple (a, b, y), where a is the source address, b is the destination address, and y is the message. This is written formally as $\mathsf{P}(a, b, y)$. We do not model the cryptographic transforms performed by a security association, but instead assume that any term encapsulated in an S constructor has undergone such a transformation. The S constructor is applied to each packet entering the tunnel and a destructor removes it at the other end. Each association has a security parameter index SPI that serves as a unique identifier for the association. Associations are assumed to act in 'tunnel' mode, meaning that a packet entering the association has the S constructor applied and becomes the payload of a packet traveling from the association's source to its endpoint. For example, suppose packet $\mathsf{P}(a, b, y)$ is to be placed in an association flowing from c to d with SPI ι_d. The constructor is applied and the result encapsulated in a packet represented by the term $\mathsf{P}(c, d, \mathsf{S}(\iota_d, \mathsf{P}(a, b, y)))$. The association flowing from node c to node d having SPI ι_d is represented at node c by the term $\mathsf{Out}(d, \iota_d)$ and at node d by the term $\mathsf{In}(c, \iota_d)$. The association database Σ contains the associations active at a node. The inbound and outbound security mechanism databases Π^i and Π^o contain entries of the form $\mathsf{Mech}(\psi : \beta)$, where ψ is a packet filter and β is a

list of security associations called a *bundle*. When an outbound packet matches a filter entry ψ, the packet is directed into the security associations listed in the bundle. That is, the constructor for each association in the bundle is applied to the packet. An inbound packet is checked against the entries in Π^i to ensure that the packet is traveling in the proper associations and the destructors are applied to the encapsulated packet. The details of packet processing are elaborated more fully in later sections as well as in Appendix A.

Tunnel establishment is the process of setting up a pair of associations between two nodes. Tunnel establishment has the following components: the authorization and authentication of the tunnel at both nodes, the updating of the association and mechanism databases, and the establishment of shared cryptographic keys by way of a key exchange protocol [18, 3]. The focus of our model is on the first two components, and, given that our model abstracts away the details of the cryptography, we do not model the key exchange process. Tunnel establishment is modeled using two messages that contain credentials for authorization, the SPI values identifying the associations, and filter entries for the mechanism database entry. In practice, establishment messages are distinguished by an identifier in the packet header. This is modeled by wrapping establishment messages in the X constructor.

Let us illustrate the basic ideas with the road warrior scenario shown in Figure 1, not including the end-to-end tunnel between a and b. Suppose Alice a is an employee away from the office and needs access to the Bob's server b. The corporate network is protected by a gateway g that requires all traffic to be authenticated and authorized with respect to a policy L enforced by g. So Alice must present a credential K to the gateway in order to demonstrate that she satisfies the policy. The gateway must also present credentials K' to Alice to prove that it belongs to a trusted administrative entity. If the polices at both nodes are satisfied, the establishment protocol will terminate after creating a pair of associations and updating the mechanism database. Here are the main steps of the protocol.

Req Sent: The initiator a generates a SPI value ι_a identifying the association flowing from the responder g to the initiator. The initiator then forms a message composed of the SPI, the credential K, and the filter selectors a and b. This message is formally expressed as a term $\mathsf{P}(a, g, \mathsf{X}(\mathsf{Req}(a, b, \iota_a, K)))$.

Req Received: Upon receiving a message of this from, the responder calls an oracle that verifies that the credential K satisfies the responder's policy L.

Rep Sent: If the oracle returns true, then the responder generates a SPI value ι_g identifying the association flowing from the initiator to the responder. The responder updates the state of its association database Σ by adding the association flowing from the initiator to responder $\Sigma \cup \mathsf{In}(a, \iota_g)$. An operation $\otimes$ adds a packet filter to responder's inbound mechanism database Π^i to indicate that all traffic from a to b should arrive at the responder in this association:

$$\mathsf{Mech}(a \rightarrow b : \mathsf{Bndl}[\mathsf{In}(a, \iota_g)]) \otimes \Pi^i.$$

The responder then forms a reply message containing the mechanism filters, both SPIs, and the responder's credentials K'. This message is formally expressed as a term $\mathsf{P}(g, a, \mathsf{X}(\mathsf{Rep}(a, b, \iota_a, \iota_g, K')))$.

Write State: After the reply message has been sent, the responder writes the state for the association flowing from the responder to the initiator.

$$\Sigma \cup \mathsf{Out}(a, \iota_a)$$
$$\mathsf{Mech}(b \rightarrow a : \mathsf{Bndl}[\mathsf{Out}(a, \iota_a)]) \otimes \Pi^o.$$

Rep Received: Upon receiving the reply message, the initiator calls upon an oracle to verify that K' satisfies its policy L' and if so writes entries to the association and mechanism databases for both associations.

Upon termination, a pair of associations is established between Alice and the gateway. When Alice sends a packet $\mathsf{P}(a, b, y)$ to the server, the filters in the mechanism database direct it into the association ι_g, and a constructor is applied yielding $\mathsf{P}(a, g, \mathsf{S}(\iota_g, \mathsf{P}(a, b, y)))$. When this packet arrives at g the destructor is applied and the encapsulated packet is sent on towards the server.

4 Interference

Using the model for packets, tunnels, and tunnel establishment given above, we demonstrate a situation where two different runs of the establishment protocol interleave to prevent messages from successfully being delivered, leaving both protocol instances in a deadlocked state. After considering several possible solutions, we introduce a new syntactic class called a 'session identifier' to prevent such harmful interactions.

The establishment initiator and responder may run concurrently at a node. Both processes operate on the association and mechanism databases. Given that both the initiator and responder add packet filters to the mechanism databases, there is the possibility that messages sent in one establishment session get captured by the filters installed by the other establishment session. The following scenario illustrates how this can lead to both establishment sessions becoming deadlocked. Suppose nodes a and b both initiate establishment with the other simultaneously. These nodes each act as both initiator and responder in these sessions of the establishment protocol. Table 2 demonstrates a particular interleaving of the execution of two sessions of the establishment protocol and illustrates how their interaction prevents either from terminating successfully. To conserve space, credentials in the messages are not included

	Node A	DB @ A	Node B	DB @ B
1	$\mathsf{P}(a, b, \mathsf{X}(\text{Req}(a, b, \iota_a)))$	$\Sigma = \emptyset$ $\Pi^i = \emptyset, \Pi^o = \emptyset$	$\mathsf{P}(b, a, \mathsf{X}(\text{Req}(b, a, \iota'_b)))$	$\Sigma = \emptyset$ $\Pi^i = \emptyset, \Pi^o = \emptyset$
2	$\mathsf{P}(b, a, \mathsf{X}(\text{Req}(b, a, \iota'_b)))$	$\cdots$	$\mathsf{P}(a, b, \mathsf{X}(\text{Req}(a, b, \iota_a)))$	$\cdots$
3		$\Sigma = \mathsf{In}(b, \iota'_a)$ $\Pi^i = b \longrightarrow a : [\mathsf{In}(b, \iota'_a)]$ $\Pi^o = \emptyset$		$\Sigma = \mathsf{In}(a, \iota_b)$ $\Pi^i = a \longrightarrow b : [\mathsf{In}(a, \iota_b)]$ $\Pi^o = \emptyset$
4	$\mathsf{P}(a, b, \mathsf{X}(\text{Rep}(b, a, \iota'_b, \iota'_a)))$	$\cdots$	$\mathsf{P}(b, a, \mathsf{X}(\text{Rep}(a, b, \iota_a, \iota_b)))$	$\cdots$
5	$\mathsf{P}(b, a, \mathsf{X}(\text{Rep}(a, b, \iota_a, \iota_b)))$	$\cdots$	$\mathsf{P}(a, b, \mathsf{X}(\text{Rep}(b, a, \iota'_b, \iota'_a)))$	$\cdots$
6	drop message		drop message	

Figure 2. Deadlock Scenario

in the table. In the first row of the table, the association and mechanism databases are empty and establishment request messages are sent by both principals. The messages arrive at their respective destinations in the second row, and the databases are updated in the third row. The filter at a now says all traffic flowing from b to a should be traveling in association ι'_a, and the filter at b now says that all traffic flowing from a to b should be traveling in association ι_b. The reply messages are formed in the fourth row of the table and arrive at their respective destinations in row five. These messages are not sent in associations, but the filters at their destinations indicate that they should have been. Hence both reply messages are dropped in the sixth line of the table. The two establishment sessions are in essence deadlocked. Consequently neither instance of the establishment protocol terminates successfully.

Is it necessary to eliminate this risk of deadlock? It is possible to detect it, tear down the partially set up tunnels, back off, and run the protocol again hoping it does not occur again. The overhead and complexity of this solution might be acceptable if the problem is a rare, and there are no stringent latency requirements. Yet history has shown that situations thought to be exceptional during design can become commonplace when systems are used in unexpected ways, and, in this case at least, one would rather avoid problems by design rather than attempt to recover from them. Here are a few ideas about how to do this.

- *Limit the establishment protocol to set up a series of unidirectional associations rather than the bidirectional ones in the given scheme.* A trace similar to that given above can be produced demonstrating the same deadlock.

- *Change the ordering of state changes and message sends and receives.* Having the responder write state for the association flowing from the initiator to the responder after the reply message is sent does not eliminate the problem.

- *Insist that the system obey a client/server assumption so nodes do not simultaneously act as both a initiator and responder.* This might solve the deadlock problem, but is overly constraining in a context where peer-to-peer communications are important.

- *Use locks to eliminate the problem by coordinating the activities of the establishment initiator and responder processes at the nodes.* This might prevent deadlock in the establishment protocol, but it has the effect of simply pushing the problem to the higher-layer protocols that invoked establishment.

- *Use a transaction protocol.* It is typical to avoid this type of complexity in protocols at the network layer. One hopes for a simpler solution.

- *Exempt tunnel establishment packets from processing by filters.* This indeed resolves the problem, but a blanket application of this approach violates authenticated traversal. A restricted variation engineers the packet filter processing mechanism so that it only exempts establishment traffic traveling between the initiator and responder from flowing in an association directly between them. This results in a complex packet processing mechanism.

Our proposed solution is to introduce a new syntactic class called a 'session identifier' that uniquely identifies a complex of tunnels set up during the execution of a protocol. This is similar to the idea of unique protocol identifiers employed in [14] to prevent messages from one protocol from being used in another. The session identifier is similar to a SPI, but rather than identifying a single association it identifies a complex of tunnels established during the session bearing that session identifier. The initiator of the session is assumed to generate the session identifier using the tunnel calculus *new* operator, which guarantees its uniqueness. The session identifier is incorporated into the mechanism database packet filters. An entry in the mechanism database at node a directing all traffic from s to d in session u into association ι flowing from a to b is written as $s \longrightarrow d : u : [\mathsf{Out}(b, \iota)]$. A packet matches a filter only if they both possess the same source, destination, and session identifier. A term representing a secure packet

now has the form $\mathsf{P}(a, b, \mathsf{S}(u, \iota, \mathsf{P}(s, d, y))$, where the secure header identifies both the session u and the association ι. The messages sent during establishment must contain the session identifier.

Suppose the proposed solution is applied in the above scenario. Alice initiates the establishment protocol for session u and Bob initiates the establishment protocol for session v. The first message sent by Alice is represented by the term $\mathsf{P}(a, b, \mathsf{X}(\mathsf{Req}(a, b, u, \iota_a)))$ and includes the session identifier. The filter installed at node b during session u would have the form $a \longrightarrow b : u : [\mathsf{In}(a, \iota_b)]$. When the establishment reply message $\mathsf{P}(a, b, \mathsf{X}(\mathsf{Rep}(b, a, v, \iota'_b, \iota'_a)))$ for session v arrives at node b, the packet will not match the filter installed in session u and the packet does not get dropped. The same logic applies to processing at node a.

Traffic belonging to a session will have the same session identifier as it travels in different associations belonging to that session's complex. Associations may be shared across sessions to improve efficiency. Before generating a new association our establishment protocol checks to see if there is an existing association that may be used. The SPI is bound to the association not the session so packets belonging to different sessions traveling in a single association will have the same SPI but different session identifiers. Yet there are scenarios where two concurrently executing sessions may create a pair of distinct associations, but this causes no harm as traffic for each session travels in its own association.

5 Tunnel Calculus

The tunnel calculus is intended as a formal framework for expressing and reasoning about protocols that set up a complex of security tunnels. The framework is structured in layers that roughly correspond to an abstraction of the network stack. Since we are interested in reasoning about tunnels, it is necessary to model the details of packet processing and persistent mutable structures such as the association and mechanism databases. This contrasts with requirements for reasoning about cryptographic protocols where one can abstract away such details because the focus is on properties such as message freshness and secrecy. In this section, an introduction to the formalism is followed by a brief description of each of the layers of the framework including examples of several of the rules.

The tunnel calculus is formally defined in terms of a tuple (D, S, T, N, E, R), where D is a set of types, S is a set of basic syntactic elements, T is a set of terms built from the elements and types, N is a set of node terms representing the terms located at a node, E is a set of equations over the elements and types, and R is a set of rules over N. Typically, (D, S, E) is an equational specification that makes precise the static aspects of the system. This includes the algebraic structure of the state space, which in our case is a multiset, *i.e.* a commutative monoid, of local state elements. The dynamics of the system is then given by the rewrite rules R, which operate modulo the equations E, and in our case correspond to multiset rewrite rules. Hence, we can visualize the state of the distributed system as a 'soup' of local state elements which are transformed by local state transitions represented by rewrite rules [2]. The tunnel calculus is obtained by instantiating the tuple with particular types, elements, terms, equations, and rules. The grammar of the tunnel calculus appears in Appendix A. The types of the calculus, such as node addresses $a \in \mathit{Node}$ and credentials $K \in \mathsf{Cred}$, are given in Table 1. Among the syntactic elements (Table 2) are $\sigma^i = \mathsf{In}(a, \iota)$ and $\sigma^o = \mathsf{Out}(a, \iota)$ representing associations and $p = \mathsf{P}(a, a, y)$ for packets. Terms, such as $\downarrow_{\mathrm{ip}(k)} \mathsf{P}(b, c, y)$, are formed from the syntactic elements, and are defined in Table 3. Node terms, such as $\downarrow_{\mathrm{ip}(k)} \mathsf{P}(b, c, y) \mathbin{@} a$, are formed from the terms by denoting the node at which the term is located.

A rewrite rule has the form

$$t_1 \mathbin{@} a_1, \ldots, t_n \mathbin{@} a_n \longrightarrow t'_1 \mathbin{@} a'_1, \ldots t'_m \mathbin{@} a'_m \quad \text{if } E$$

where E is an optional condition on the firing of the rule. If all the terms in a rule are located at node a, then we drop location annotation on each term and write the rule as

$$\vdash_a L \longrightarrow R.$$

Variables appearing on the right-hand side of a rule must also appear on the left-hand side of the rule or have its values randomly generated using the *new* operator.

The network state is represented by a multiset M of node terms written $t \mathbin{@} a$ (term t at node a). State is transformed by the application of a rewrite rule. A rule is executed at a node only if node terms matching the left side of the arrow are present at that node in the multiset. Given a multiset of node terms M and a rule of the form above, the left-hand side of the rule is matched (unified) against the node terms in M and rewritten to the pattern on the right-hand side of the rule. If all of the node terms in a rule are located at the same node, then its application can be viewed as a change of state at a single node. Communication between nodes is represented by a rule that moves a term from one node to another.

If more than one rule is ready for dispatch, then their order of execution is non-deterministic. This means that there is no natural ordering built into the model so, if we want a set of rules to be executed sequentially, then the rules themselves must enforce the ordering. Another feature of term rewriting is that state must be explicitly passed from one rule to the next when executing a sequence of rules. Both issues are resolved using the syntactic construct we call a *resumption term*. A resumption term is an n-tuple of elements $\langle ele_1, ele_2, \ldots, ele_n \rangle$ that represent the state of an execution. Such terms appear in most of the rules.

Each rule in the tunnel calculus is accompanied by a label given in bold face of the form **Rule X.Y.Z**, where **X** is a letter denoting the layer, **Y** is **1** if it is an initiator rule and **2** if it is a responder rule, **Z** is a numerical label for that rule. For instance, the first rule of the secure processing layer responder is labeled **S.2.1**.

The tunnel calculus is structured as four layers. The lowest layer of the tunnel calculus is the *forwarding layer* (ip), which models the forwarding of packets based on a forwarding table. The *secure processing layer* (sec) performs the processing associated with secure tunnels. The *authorization layer* (auth) acts as an oracle that, given a set of credentials K, returns true if they satisfy the given policy L. The *establishment layer* (estab) sets up a pair of unidirectional security associations. Historically, the focus at this layer has been on the key exchange. Instead, our focus is on the establishment of state at the nodes for the associations and the packet filters that direct traffic into the associations. These layers form the framework upon which discovery protocols are built.

The authorization layer acts as a function that is called by writing a $\downarrow_{\mathrm{auth}}$ term to the multiset and the result is returned via a $\uparrow_{\mathrm{auth}}$ term. The forwarding, secure processing, and establishment layers are structured as having initiator and responder processes expressed as a collection of rewrite rules. Both processes may run concurrently at each node in the system. The first rule of an initiator always has the form of a rewrite rule with $\downarrow_I$ on the left of the arrow, where $I \in \{\mathrm{ip}, \mathrm{sec}, \mathrm{est}\}$. The last rule of an initiator always has the form of a rewrite rule with $\uparrow_I$ on the right side of the arrow. The initiator will remove the $\downarrow_I$ term from the multiset when it begins executing and write an $\uparrow_I$ term when it terminates. So a layer is invoked by writing a $\downarrow_I$ term and then waiting for an $\uparrow_I$ term indicating that processing has terminated. The responder processes for the forwarding and secure processing layers run as daemons at each node. The establishment layer responder (eresp) is structured like the initiator processes using $\downarrow_{\mathrm{eresp}}$ and $\uparrow_{\mathrm{eresp}}$ terms. All responder processes await the arrival of a message from the corresponding initiator and, upon termination, pass information to a higher layer. If a responder is running as a daemon, information is passed to a higher level by writing a $\Uparrow_I$ term.

Each $\downarrow_I$ and $\uparrow_I$ term is annotated with a unique identifier k so that a rule with an $\uparrow_I$ on the left of the arrow can be assured that it matches the $\downarrow_I$ that was intended. Otherwise, there could be confusion as there may be many $\uparrow_I$ terms in the multiset. The identifiers are always generated using the tunnel calculus *new* operator to ensure uniqueness. In the forwarding layer, these terms have the form $\downarrow_{\mathrm{ip}(k)}$ and $\uparrow_{\mathrm{ip}(k)}$. The $\downarrow$ terms of the remaining layers are also annotated with the session identifier. For instance, the secure processing layer is invoked in session u with acknowledgment identifier k by writing the term $\downarrow_{\mathrm{sec}(u,k)}\ p$ to the multiset.

The layers are intended to model those of a network stack. It is assumed that only the secure processing layer makes use of the forwarding layer. All other messages are sent via the secure processing layer. To see how the layers interact, consider what happens when a packet $p \mathbin{@} a$ is sent to node b via the secure processing layer. The secure layer applies the appropriate constructors to the packet and sends the result p' to the forwarding layer; the forwarding layer forwards it to the next node where it is processed by the forwarding layer responder, which passes it up to be processed by the secure processing layer responder, which applies the appropriate destructors and passes the packet up for processing. At node a this sequence of operations will add the terms:

$$\downarrow_{\mathrm{sec}(v,k_1)}\ p \mathbin{@} a, \downarrow_{\mathrm{ip}(k_2)}\ p' \mathbin{@} a, \uparrow_{\mathrm{ip}(k_2)}\ \mathbin{@} a, \uparrow_{\mathrm{sec}(k_1)}\ \mathbin{@} a$$

and at node b they will add the terms:

$$\Uparrow_{\mathrm{ip}}\ p' \mathbin{@} b, \Uparrow_{\mathrm{sec}(v)}\ p \mathbin{@} b.$$

The send/acknowledgment structure of messages models the processing in the IP stack where a send does not return until the message has traversed the stack [9].

Having given an explanation of the structure of the tunnel calculus, a brief survey of the processing performed at each layer follows. A formal presentation of all the rules can be found in Appendix A.

The forwarding layer models the movement of packets based on a forwarding table. Packets can move from one node to another only via an application of the forwarding layer. We do not attempt to model packet fragmentation or routing.

The secure processing layer provides an abstract model of the processing performed by secure tunnels. This layer employs an abstract model of cryptographic protocols that suppresses explicit mention of keys, time stamps, and nonces. The processing associated with security associations is performed at this layer. Each node maintains an association database Σ as well as inbound (Π^i) and outbound (Π^o) mechanism databases. The entries in the mechanism database take the form $\mathsf{Mech}(\psi : v : \beta)$ where ψ and v act as a filter consisting of source and destination address and a session identifier, and β is a bundle of security associations that are applied to packets matching the filter.

There are two outbound processing rules which we describe below for illustration. The other rules of the calculus are given in the appendix.

Rule S.1.1

$$\begin{array}{lll} \Pi^o & \vdash_e & \downarrow_{\mathrm{sec}(v,k)}\ \mathsf{P}(b,c,y) \longrightarrow \\ & & \downarrow_{\mathrm{ip}(k')}\ \mathsf{Nest}(\mathsf{BndlSel}(b,c,v,\Pi^o), e, v, \mathsf{P}(b,c,y)), \\ & & \langle k, k', v\rangle \\ & & \text{where } k' \text{ is } \textit{new}. \end{array}$$

The outbound mechanism database Π^o appears to the left of the turnstile indicating it can be used in the rule, but not consumed. If a secure layer message is ready for dispatch, the semantic function BndlSel is invoked to determine the security association(s) to apply to the packet. The semantic function Nest applies the appropriate constructors and encapsulates the packets in the proper header creating a packet p'. The term $\downarrow_{\mathrm{ip}(k')}\ p'$ is written to the multiset indicating that the packet should be sent to the forwarding layer with the acknowledgment identifier k' generated by the *new* operator. A resumption term is written to the multiset containing the two acknowledgment identifiers k and k'. Here is the second rule for outbound processing.

Rule S.1.2

$$\vdash_e \quad \langle k, k', v\rangle,\ \uparrow_{\mathrm{ip}(k')} \longrightarrow \uparrow_{\mathrm{sec}(k)}$$

If a forwarding layer acknowledgment term is in the multiset and that term possesses the acknowledgment identifier k' (matching the resumption term), then this rule rewrites a secure layer acknowledgment indicating that the message has been sent to its destination.

Secure layer inbound processing works as follows. The forwarding layer relays all packets to the secure processing layer responder process for further processing. A packet arriving at a node must either be traveling in a valid association or be a distinguished packet, such as an establishment packet. In either case, the packet contains the session identifier. The responder strips off and verifies the secure headers for all associations terminating at that node. The inbound mechanism database is consulted to verify that the incoming message arrived in the proper associations. If the decapsulated packet is a distinguished packet it is passed to higher layers for further processing. If the decapsulated packet p is destined for this node, then a $\Uparrow_{\mathrm{sec}}\ p$ term is written to the multiset. If the decapsulated packet does not have this node as a destination, the packet is sent on its way.

The authorization layer takes as parameters a credential K and a policy L, and calls an oracle that returns true or false depending on whether K satisfies L. The choice of this oracle is determined by the security objectives of hosts and gateways. In other work we explored a specific authorization layer that provides high-level control over packet flows based on public key certificates.

The establishment layer is the highest layer of the tunnel calculus framework and was previously described in some detail. Intended for use in the design of protocols that discover security gateways and set up a complex of tunnels among them, the establishment layer is used in the following way. A distinguished discovery packet in session u is intercepted by a gateway a on the dataflow path. The discovery protocol invokes the establishment layer $\downarrow_{\mathrm{est}(u,k)}\ \mathsf{E}(b, s, d)$ to set up a tunnel with node b that is already known to session u, where the filter values are s and d. The establishment responder does not run as daemon, but must be invoked by the discovery protocol by writing the term $\downarrow_{\mathrm{eresp}(u,k)}$ to the multiset.

We are aware of only one other effort [12] to formally model the packet header processing associated with a tunnel complex. This uses an automata-based approach to model IPsec security association processing to prove confidentiality and authentication properties. We have attempted to model tunnel processing in the π-calculus, but it does not provide explicit help for representing and manipulating persistent mutable structures such as the association and mechanism databases and the operations for updating these structures. The resulting experiment left us with a system that looked like the present tunnel calculus with the π-calculus bolted on the side. Like automata and process algebras, multiset rewriting possess a rich theory and excellent tool support. Using this framework leads to a modular system in which it is rather straight-forward to prove correctness properties in terms of the trace semantics given in the next section.

6 Trace Theory

Our analysis requires a certain amount of trace theory, which we now describe. The application of **Rule X** : $L \longrightarrow R$ to the multiset M rewrites to the multiset $M' = (M - L') \cup R'$, where $L' = L\rho$ and $R' = R\rho$ and ρ is a unifier. We call L' the *redux* and R' the *contractum*. To indicate that $M \longrightarrow M'$ is an application of **Rule X** at node a executing in session u with redux L' and contractum R' we often write **Rule X**$(u)(L', R')(a)$ or

$$M \overset{\mathbf{X}(u)(L',R')(a)}{\longrightarrow} M'.$$

When the context is clear we drop the redux, contractum, and node. If all the rules belong to the same session or the session identifier is not needed in an argument, then we drop the session identifier as well and simply write $M \overset{\mathbf{X}}{\longrightarrow} M'$ or in some cases $\mathbf{X}$ to denote an application of **Rule X**. The sequential application of **Rules** $\mathbf{X}_1(u_1), \ldots, \mathbf{X}_n(u_n)$ to the multiset M_1 is written as

$$M_1 \overset{\mathbf{X}_1(u_1)}{\longrightarrow} M_2 \overset{\mathbf{X}_2(u_2)}{\longrightarrow} \cdots M_n \overset{\mathbf{X}_n(u_n)}{\longrightarrow} M_{n+1}.$$

The sequence of multisets $M_1, M_2, \ldots, M_{n+1}$ is called a *trace* of the sequential execution of rules $\mathbf{X}_1(u_1), \ldots, \mathbf{X}_n(u_n)$ and provides a view of the multiset representing the network state as the protocol executes. Each change to the network state results in a new multiset being added to the trace sequence.

The following lemma shows that there is a one-to-one relationship been the rules and a step in the trace. A consequence of this result is that it is possible to formulate many of the functional correctness properties that interest us in terms of a protocol's trace.

Lemma 1 *Consider the trace M, M' produced by the application of a rule in the tunnel calculus. There exists only one rule **X** whose application to M ($M \xrightarrow{\mathbf{X}} M'$) could have produced M'.* □

We have asserted that session identifiers are always generated by the tunnel calculus *new* operator ensuring their uniqueness. The following proposition asserts uniqueness for acknowledgment identifiers and follows from an inspection of the rules.

Proposition 2 (Uniqueness of Identifiers) *Let $M_1, \ldots, M_n$ be a trace. Suppose $I \in \{\mathrm{sec}, \mathrm{auth}, \mathrm{eresp}, \mathrm{est}\}$ and $I' = \mathrm{ip}$. Suppose t is a term that begins with $\downarrow_{I(v,k)}$ or $\downarrow_{I'(k)}$ and there is an i such that $t \notin M_{i-1}$ and $t \in M_i$. Suppose t' is another term having the form $\downarrow_{I(v,k')}$ or $\downarrow_{I'(k')}$, and there is a $j > i$ with $t' \in M_j$ and $t' \notin M_{j-1}$. Then $k' \neq k$.* □

Consider the execution of the tunnel calculus establishment protocol between two nodes. If one only observed the actions at a single node, there is only one possible trace for a successful execution of the protocol. Yet the protocol is executing on a distributed network of nodes. A trace of this protocol must record that the initiator has sent the request message before it records that the message has been received at the responder and it must record that the reply has been sent by the responder and received by the initiator before the initiator writes state. This is due to the causal ordering induced by the messages [17]. No such ordering exists between the writing of state for the two associations at the initiator and the writing of state at the responder for the association flowing from the responder to the initiator. Hence there is more than one possible trace for the execution of the establishment protocol. This has been formalized in Mazurkiewicz trace theory [6] via the concept of an independence relation between actions that captures possible concurrency. For instance, if **Rule X** and **Rule Y** are independent of each other, the trace may record $M_i \xrightarrow{\mathbf{X}} M_{i+1} \xrightarrow{\mathbf{Y}} M_{i+2}$ or $M'_i \xrightarrow{\mathbf{Y}} M'_{i+1} \xrightarrow{\mathbf{X}} M'_{i+2}$. The formalization of independence that follows is similar to that found in [19].

State shared among different sessions at a node is maintained in the forwarding table, association database, and mechanism databases at a node. Let $\mathcal{H}(a)$ be the infinite multiset of all terms representing shared state at node a, that is:

$$\mathcal{H}(a) = \{\!|\mathsf{F}(f) @ a, \Sigma @ a, \Pi^i @ a, \Pi^o @ a|\!\},$$

where $\mathsf{F}(f), \Sigma, \Pi^i$, and Π^o represent all possible terms of that form. Let

$$\mathcal{H} = \bigcup_a \mathcal{H}(a).$$

Consider an application of **Rule X** having redux L_1 and contractum R_1 and an application of **Rule Y** having redux L_2 and contractum R_2. Define an ordering on the application of rules as $\mathbf{X} \prec \mathbf{Y}$ if and only if

$$(R_1 - \mathcal{H}) \cap (L_2 - \mathcal{H}) \neq \emptyset.$$

Define the *principal ideal* of an application of **Rule X** as $\check{\mathbf{X}} = \{\mathbf{Y} \mid \mathbf{Y} \prec \mathbf{X}\}$. The application of rules **X** and **Y** are said to be *dependent* if $\mathbf{X} \in \check{\mathbf{Y}}$ or $\mathbf{Y} \in \check{\mathbf{X}}$ or $(L_1 - \mathcal{H}) \cap (L_2 - \mathcal{H}) \neq \emptyset$. If an application of rules **X** and **Y** are not dependent, then they are said to be *independent* and we write $\mathbf{X} \parallel \mathbf{Y}$.

7 Noninterference and Progress

This section is devoted to the formalization of a noninterference theorem that implies that the deadlock illustrated in section 4 cannot occur in the tunnel calculus. We also demonstrate a progress property for the tunnel calculus. Although the theorems presented in this section presume reliable delivery of messages, the properties hold in the case of the unreliable message delivery as well. The theorems in the unreliable case are more involved and the proofs are tedious, so, for ease of presentation, we limit ourselves to the reliable case in this paper.

7.1 Noninterference

The first step in developing a noninterference theorem is to prove that the application of any two rules executing in distinct sessions is independent in the sense that we defined in the previous section. The session matching property formalizes the idea that the filters in the tunnel calculus mechanism database only match packets belonging to a specified session. The main lemma asserts that the order of execution of rules in distinct sessions can be swapped without changing the semantics of the trace. With this machinery in place it is possible to formulate and prove our noninterference theorem.

Recall that the defect exposed in Section 4 arose because packets from one session match the packet filters installed during establishment for another session. The adaption of session identifiers purportedly prevented this from occurring. The session matching property demonstrates that this is indeed so. Inspection of the secure processing layer rules immediately reveals the following

Lemma 3 (Session Matching Property) *Let $T = M_1, \ldots, M_n$ be a trace and assume session v is active in T. Suppose $M_i \longrightarrow M_{i+1}$ is an application of **Rule S.1.1**(v), where the outbound message being processed is the term $\downarrow_{\mathrm{sec}(v,k)} \mathsf{P}(b, c, y) @ a$. If the semantic function*

BndlSel produces a match in the outbound mechanism database Π^o @ a, then the matching database entry must have the form $\mathsf{Mech}(\psi : v : \beta^o)$.

A similar property holds for inbound messages. □

The Session Matching property is critical in proving the results that follow.

The following two results demonstrate that the application of any two rules in distinct sessions are independent. Let **Rule X**(u) and **Rule Y**(v) denote any two rules in the tunnel calculus.

Lemma 4 *Let u and v be distinct session identifiers. Let $T = M_1, \ldots, M_n$ be a trace. Suppose $M_i \longrightarrow M_{i+1}$ is an application of **Rule X**$(u)(L_1, R_1)(a)$, and $M_j \longrightarrow M_{j+1}$ is an application of **Rule Y**$(v)(L_2, R_2)(b)$. Then*

$$(L_1 - \mathcal{H}(a)) \cap (L_2 - \mathcal{H}(b)) = \emptyset \text{ and} \quad (1)$$
$$(L_1 - \mathcal{H}(a)) \cap (R_2 - \mathcal{H}(b)) = \emptyset. \quad (2)$$

□

That is, neither the execution of **Rule X**(u) nor **Rule Y**(v) will consume terms that would otherwise have been consumed by the execution of the other unless those terms represent shared state. The following is implied by Lemma 4, the definition of $\prec$, and the definition of independence.

Corollary 5 (Independence Between Sessions) *Let u and v be distinct session identifiers. Let $T = M_1, \ldots, M_n$ be a trace. Let a and b be nodes. Suppose $M_i \longrightarrow M_{i+1}$ is an application of **Rule X**$(u)(a)$, and $M_j \longrightarrow M_{j+1}$ is an application of **Rule Y**$(v)(b)$. Then $\mathbf{X}(u)(a) \parallel \mathbf{Y}(v)(b)$.* □

To illustrate the use of this result, consider applications $\mathbf{X}_1(u) \prec \mathbf{X}_2(u) \prec \mathbf{X}_3(u)$ in session u and $\mathbf{Y}_1(v) \prec \mathbf{Y}_2(v) \prec \mathbf{Y}_3(v)$ in session v. Although $\mathbf{X}_i \parallel \mathbf{Y}_j$ for $i, j \in \{1, 2, 3\}$, any legal trace must respect the ordering within the session. So a trace may record these rules being applied in the order $\mathbf{X}_1, \mathbf{Y}_1, \mathbf{Y}_2, \mathbf{Y}_3, \mathbf{X}_2, \mathbf{X}_3$ or $\mathbf{Y}_1, \mathbf{X}_1, \mathbf{X}_2, \mathbf{Y}_2, \mathbf{Y}_3, \mathbf{X}_3$, but not $\mathbf{X}_2, \mathbf{X}_1, \mathbf{Y}_3, \mathbf{X}_3, \mathbf{Y}_2, \mathbf{Y}_1$.

Intuitively, our notion of noninterference says that the communication pattern engendered by a protocol in the tunnel calculus is the same regardless of the actions performed by other sessions. In order to illustrate this, consider the two traces

$$T = M_1 \xrightarrow{\mathbf{X}(u)} M_2 \xrightarrow{\mathbf{Y}(v)} M_3$$
$$T' = M_1' \xrightarrow{\mathbf{Y}(v)} M_2' \xrightarrow{\mathbf{X}(u)} M_3'.$$

It follows from our informal view of what constitutes noninterference that the messages sent and received by session v are the same in both T and T'. In order to formalize our noninterference theorem, it is necessary to develop some machinery and prove several critical properties.

Let $\overline{M}$ denote a sequence of multisets $M_1, \ldots, M_n$. Let intersection distribute over a sequence of multisets as in $\overline{M} \cap V = M_1 \cap V, \ldots, M_n \cap V$ for some multiset of terms V. Define $Q(u)$ to be the infinite multiset of terms of the form $\downarrow_{\mathrm{sec}(u,k)} \mathsf{P}(a, b, y)$ and $\Uparrow_{\mathrm{sec}(u)} \mathsf{P}(a, b, y)$ containing session identifier u, where the values of a, b, y, and k can take any legal value. So given a trace T, the sequence $T \cap Q(u)$ is the messages sent and received in session u. Hence it may seem that we can formulate noninterference in terms of $T \cap Q(u) = T' \cap Q(u)$, but more work yet remains.

Consider the situation where the order of application of two operations in different sessions is swapped

$$M_1 \xrightarrow{\mathbf{X}(u)} M_2 \xrightarrow{\mathbf{Y}(v)} M_3$$
$$M_1' \xrightarrow{\mathbf{Y}(v)} M_2' \xrightarrow{\mathbf{X}(u)} M_3'$$

The traces $T = M_1, M_2, M_3$ and $T' = M_1', M_2', M_3'$ both contain the same v messages. If a term $t \in Q(v)$ was produced by $\mathbf{Y}(v)$, then $T \cap Q(v) = \{\!|t|\!\}$ and $T \cap Q(v) = \{\!|t|\!\}, \{\!|t|\!\}$ because the t term must still be in the multiset M_3' since the application of a rule in session u will not remove a v term. Although this satisfies our notion of the two traces containing the same messages, we would like the traces to be the same. The introduction of an operator rectifies this problem by removing duplicate entries in a sequence of multisets. Given a sequence of multisets $\overline{M}$, the *filter operator* $(\!|\overline{M}|\!)$ removes empty sets from the sequence and removes duplicate subsequences of multisets from the sequence. For example, $(\!|\emptyset, \{\!|1, 2, 2|\!\}, \{\!|1, 2, 2|\!\}, \{\!|3, 4|\!\}|\!) = \{\!|1, 2, 2|\!\}, \{\!|3, 4|\!\}$.

Given terms t and t', we define the relation $t \sim t'$ if and only if there exists a substitution ρ of SPI and acknowledgment identifier values for those in t such that t and t' are syntactically identical. Given a multiset M, let $M\rho = \{\!|t\rho \mid t \in M|\!\}$. Write $M_1 \sim M_2$ if, and only if, there exists a substitution ρ such that $M_1\rho = M_2$. Traces T and T' are said to be *v-observationally equivalent* if $(\!|T \cap Q(v)|\!) \sim (\!|T' \cap Q(v)|\!)$.

With this machinery in place it is now possible to formulate the Observational Commutativity Theorem. It says that, if we consider two operations belonging to two distinct sessions u and v, then the messages sent and received in session v are the same regardless of the interleaving of operations.

Theorem 6 (Observational Commutativity) *Let T be a trace*

$$M_1 \xrightarrow{\mathbf{X}_1(u_1)} M_2 \xrightarrow{\mathbf{X}_2(u_2)} M_3 \xrightarrow{\mathbf{X}_3(u_3)} \cdots \xrightarrow{\mathbf{X}_{n-1}(u_{n-1})} M_n,$$

where $u_1 \neq u_2$. Then there is a trace T' that has the form

$$M_1' \xrightarrow{\mathbf{X}_2(u_2)} M_2' \xrightarrow{\mathbf{X}_1(u_1)} M_3' \xrightarrow{\mathbf{X}_3(u_3)} \cdots \xrightarrow{\mathbf{X}_{n-1}(u_{n-1})} M_n',$$

where $(\!|T \cap Q(u_1)|\!) \sim (\!|T' \cap Q(u_1)|\!)$. □

The next lemma gives us a useful tool to apply when proving two traces are semantically the same.

Lemma 7 (Simulation Lemma) *If* $M_1 \xrightarrow{X} M_2$ *and* $M_1 \sim M_1'$, *then there exists* M_2' *such that* $M_2 \sim M_2'$ *and* $M_1' \xrightarrow{X} M_2'$. □

A *virginal* network state is defined as a multiset where the only terms in the multiset are the forwarding tables that define the topology (terms of the form $\mathsf{F}(f)$).

Consider a run of the establishment protocol. The initiator invokes the establishment protocol by writing a $\downarrow_{\mathrm{est}}$ term to the multiset, and a $\uparrow_{\mathrm{est}}$ term is written when the establishment initiator terminates. The responder process is invoked by writing a $\downarrow_{\mathrm{eresp}}$ term to the multiset, and a $\uparrow_{\mathrm{eresp}}$ term is written when the establishment responder terminates. We say that an invocation of establishment terminates successfully when both the initiator and responder's $\uparrow$ terms are written to the multiset. The discovery protocols built on the framework of the tunnel calculus are presumed to to have the same $\downarrow, \uparrow$ structure. In addition, the discovery protocol is assumed to be invoked with a fresh session identifier. Given a trace $T = M_1, \ldots, M_n$ of the execution of a discovery protocol. The trace is said to record a *complete session* if it contains both the $\downarrow$ and $\uparrow$ terms for the protocol session and all invocations of establishment terminate successfully.

We are now in a position to formalize our noninterference theorem. Suppose trace T records the execution of session v beginning in the same virginal network state and trace T' records the concurrent execution of sessions v and u also beginning in a virginal network state, then the messages sent and received in session v are the same in both traces up to the afore-mentioned α-equivalence.

Theorem 8 (Noninterference) *Let* $T = M_1, \ldots, M_n$ *be a trace in which* M_1 *is assumed to be a virginal network and the only active session recorded in the trace is session* v. *Let* $T' = M_1', \ldots, M_l'$ *be a trace where* $M_1 \sim M_1'$ *and sessions* v *and* u *are active in the trace. If session* v *is complete in both* T *and* T', *then*

$$(\!|T \cap Q(v)|\!) \sim (\!|T' \cap Q(v)|\!). \quad \square$$

That is, with the hypotheses of the Theorem, the messages sent and delivered in the execution of the rules in session v are the same in both traces. It follows that the execution of the rules in session u does not 'interfere' with the messages sent and delivered in session v.

There is a large body of work in the existing literature on formal reasoning about noninterference. Most approaches to controlling interference in concurrent and distributed systems have used a more general definition of interference and are intended for use in a more general setting. Axiomatic approaches are usually rooted in the Gries-Owiki [10] proof technique. A more complete axiomatic methodology for reasoning about interference in concurrent programs can be found in [11]. An alternate approach has its origins in Reynolds' Syntactic Control of Interference [20]. The goal of this program is the design of a powerful Algol-like language in which interference is possible, but syntactically detectable. The tunnel calculus is more application-specific, yet our solution adopts a similar philosophy. Having proven noninterference between two sessions with distinct identifiers, we need only verify that the respective sessions use the *new* operator to generate the session identifier thus guaranteeing its uniqueness. This is an easy syntactic check.

7.2 Progress

We now characterize several properties relating the execution of rules in the same session. The first result demonstrates that the application of two rules at different nodes, but executing in the same session, can be swapped without essentially altering the correctness of the protocol. The second result is a progress theorem that states that if communication between two parties is possible, then it is possible to extend any other to complete the communication.

When introducing the concept of independence, we demonstrated a situation where operations within the same session are independent. In particular, we showed that in the execution of the establishment protocol, the writing of state at the initiator and the writing of state at the responder for the tunnel flowing from the responder to the initiator are independent. The next result says that different interleavings of independent tunnel calculus operations at different nodes within the same session has no effect on the communication pattern engendered by the protocol.

Lemma 9 (Independence) *Suppose*

$$\boldsymbol{X}(u)(L_1, R_1)(a) \parallel \boldsymbol{Y}(u)(L_2, R_2)(b),$$

where $a \neq b$ *and* $M_1 \xrightarrow{\boldsymbol{X}(u)} M_2 \xrightarrow{\boldsymbol{Y}(u)} M_3$ *where* $M_1 \sim M_1'$. *Then there exist* M_2', M_3' *such that* $M_1' \xrightarrow{\boldsymbol{Y}(u)} M_2' \xrightarrow{\boldsymbol{X}(u)} M_3'$ *and*

$$(\!|M_1, M_2, M_3 \cap Q(u)|\!) \sim (\!|M_1', M_2', M_3' \cap Q(u)|\!). \quad \square$$

It is now possible to formulate a progress theorem that says that if a trace T records a complete session starting in the initial state M_1, then given an incomplete trace of the protocol, there is a possible extension that completes the trace.

Theorem 10 (Progress) *Consider the trace* $T = M_1, \ldots, M_n$ *and the only active session in the trace is session* u*, where*

$$\overbrace{M_1 \longrightarrow^* M_n}^{u \text{ complete}}.$$

Suppose there exists $W = N_1 \longrightarrow^* N_l$*, where* $N_1 \sim M_1$ *and session* u *is the only active session, but session* u *is not complete. Then there exists* $N_l \longrightarrow^* N_q$ *such that*

$$\underbrace{N_1 \longrightarrow^* N_l \longrightarrow^* N_q}_{u \text{ complete}},$$

where $(\!|T \cap Q(u)|\!) \sim (\!|N_1, \ldots, N_l, \ldots, N_q \cap Q(u)|\!)$. □

7.3 Relevance of the Theory

Let us now return to a concrete example to illustrate the utility of the theorems developed in this section. Consider two nodes a and b and assume that both have their oracles set to allow any connection. Suppose a initiates an establishment session u with b and b initiates an establishment session v with a. Assume that the execution of these two protocols reaches a point where the global state records that the request message for session u has arrived at b and the request message for session v has arrived at a. Expressing this in terms of our formalism, we say that at M_i the trace $T = M_1, \ldots, M_i$ of this activity records

$$\Uparrow_{\mathrm{sec}(u)} \mathsf{P}(a, b, \mathsf{X}(\mathrm{Req}(a, b, u, \iota_a, K^a))) \,@\, b \in M_i$$

and

$$\Uparrow_{\mathrm{sec}(v)} \mathsf{P}(b, a, \mathsf{X}(\mathrm{Req}(b, a, v, \iota_{b'}, K^b))) \,@\, a \in M_i.$$

It follows from Independence Between Sessions (corollary 5) that operations performed in sessions u and v are independent. From the Observational Commutativity Theorem 6 it follows that neither an operation in session u nor an operation in session v will affect the messages sent in the other session. Finally the Noninterference Theorem 8 informs us that regardless of the interleaving of the operations of the two sessions, they will terminate with their respective tunnels set up.

8 Conclusion

It is often appreciated in practical operations, but perhaps less so in theoretical investigations, that security systems create 'friendly fire' risks in which systems are harmed by their own security protection mechanisms. An illustration of this appears in a recent NIST report [21] on SCADA security, which includes a list of documented security incidents for SCADA facilities (see Section 3.7). In particular, the list contains as many incidents arising from faults in security protection measures as ones arising from deliberate attacks by adversaries. This phenomenon is not at all surprising, as anyone who has locked their keys in their car will testify, but it underscores the importance of theory that can provide assurances that security mechanisms will not themselves cause failures.

In this paper we showed that tunnel establishment protocols risk non-trivial functional problems with deadlock when negotiation packets are themselves placed in tunnels, as they would be if authenticated traversal is enforced. We presented a language, the tunnel calculus, that can be used to express these issues precisely and reason about them. We included in this system a concept of session that can be used to separate tunnels to assure non-interference between them. We stated a series of theorems that assert some desired properties for this solution. We expect in future work to provide a stronger version of the Progress Theorem: one that is analogous to results on routing protocols, where it is typical to show that suitable convergence can be achieved from an arbitrary network state. In particular, a proper version of this result will imply that any failures in the state of the tunnel complex can be addressed by simply rerunning the discovery protocol with a new session identifier.

A full version of this paper will include proofs of the results. Related work on the tunnel calculus can be found at the project web site: `seclab.uiuc.edu/tunnelcalculus`.

Acknowledgements

We appreciated the assistance and encouragement from Cathy Meadows, Marc-Oliver Stehr, and Steve Zdancewic on this project and comments from anonymous referees on the paper. The research was partially supported by NSF CNS05-5170 CNS05-09268 CNS05-24695, ONR N00014-04-1-0562 N00014-02-1-0715, and a grant from the MacArthur Foundation. Views expressed here are those of the authors only.

References

[1] M. Abadi and A. Gordan. A Calculus for Cryptographic Protocols: The Spi Calculus. *Information and Computation*, 148(1):1–70, 1999.

[2] G. Berry and G. Boudol. The chemical abstract machine. In *Principals of Programming Languages*, pages 81–94. ACM Press, 1989.

[3] C. Boyd and A. Mathuria. *Protocols for Authentication and Key Establishment*. Springer-Verlag, 2003.

[4] I. Cervesato. A specification language for crypto-protocols based on multiset rewriting, dependent types and subsorting. In *Workshop on Specification, Analysis, and Validation for Emerging Technologies*, pages 1–22, 2001.

[5] Dynamic multipoint vpn (dm vpn). Cisco White Paper. http://www.cisco.com/.
[6] V. Diekert and G. Rozenberg, editors. *The Book of Traces*. World Scientific, 1994.
[7] T. Dierks and E. Rescorla. The TLS Protocol. RFC 4346, IETF, 2006. Obsoletes: 2246.
[8] S. Fluhrer. Tunnel endpoint discovery. Internet Draft draft-fluhrer-ted-00.txt, IETF, 2001.
[9] A. Goodloe, M.-O. Stehr, and C. A. Gunter. Formal prototyping in early stages of protocol design. In *Workshop on Issues in the Theory of Security (WITS '05)*, Long Beach, CA, January 2005. IFIP.
[10] D. Gries and S. Owicki. An Axiomatic Proof Technique for Parallel Programs. *Acta Informatica*, 6:319–340, 1976.
[11] C. B. Jones. Accommodating Interference in the Formal Design of Concurrent Object-Based Programs. *Formal Methods in System Design*, 8(2), 1996.
[12] Joshua D. Guttman and Amy L. Herzog and F. Javier Thayer. Authentication and confidentiality via ipsec. In F. Cuppens and Y. Deswarte and D. Gollmann and M. Waidner , editor, *European Sympsosium on Reseach in Computer Security (ESORICS)*, Lecture Notes in Computer Science 1895. Springer-Verlag, 2000.
[13] C. Kaufman. Internet Key Exchange (IKE V2) Protocol. RFC 4306, IETF, 2005. Obsoletes: 2407, 2408, 2409.
[14] J. Kelsey, B. Schneier, and D. Wagner. Protocol interactions and the chosen protocol attack. In *Proceedings of the 5th International Workshop on Security Protocols*, Lecture Notes in Computer Science 1361, pages 91–104. Springer-Verlag, 1998.
[15] S. Kent. IP Encapsulating Security Payload (ESP). RFC 2406, IETF, 2005. Obsoletes: 2406.
[16] S. Kent and K. Seo. Security architecture for the internet protocol. RFC 4301, IETF, 2005. Obsoletes: 2401.
[17] L. Lamport. Time, Clocks, and the Ordering of Events in a Distributed System. *Communications of the ACM*, 21(7):558–565, July 1978.
[18] A. J. Menezs, P. C. van Oorchot, and S. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1996.
[19] R. Morin. On regular message sequence chart languages and relationships to Mazurkiewicz trace theory. In F. Honsell and M. Miculan, editors, *Foundations of Software Science and Computation Structures (FOSSACS)*, Lecture Notes in Computer Science 2030. Springer-Verlag, 2001.
[20] J. Reynolds. Syntactic control of interference. In *Proceeding of Fifth Symposium ACM on Principle of Programming Languages*, pages 39–46, 1978.
[21] K. Stouffer, J. Falco, and K. Kent. Guide to Supervisory Control and Data Acquisition (SCADA) and industrial control systems security. Technical Report Special Publication 800-82, NIST, September 2006.
[22] T. Ylonen and C. Lonvick. The Secure Shell (SSH) Transport Layer Protocol. RFC 4253, IETF, 2006.

A The Tunnel Calculus

This appendix contains a concise definition of the grammar and rules of the tunnel calculus. The types of the calculus are given in Table 1. The basic types include the addresses of nodes in the network, an address range ($Domain$), generic messages, Booleans, and SPI values. The forwarding table is a partial function from the destination address to the address of the next hop. A variable denoting the session identifier v may be either of type $Session$ or have the value ∞ indicating that no session identifier has been assigned to that variable. Policies and credentials are treated abstractly in this paper where they are represented by the basic types Pol and $Cred$.

Node	$a \in Node$
Message	$m \in Msg$
Forwarding Table	$f \in Addr \rightsquigarrow Addr$
Request Identifier	$k \in Identifier$
Security Parameter Index	$\iota \in SPI$
Domains	$d \in Domain$
Session	$v \in Session + -\infty$
Policy	$L \in Pol$
Credential	$K \in Cred$
Booleans	$B \in Boolean$

Table 1. Tunnel Calculus Types

Fwd	F	::=	$\mathsf{F}(f)$
Sec	s	::=	$\mathsf{S}(v, \iota, p)$
Req/Rep	κ	::=	$Req(a, a, v, \iota, K) \mid$
			$Rep(a, a, v, \iota, \iota, K, L)$
Establishment	χ	::=	$\mathsf{X}(\kappa)$
Payload	y	::=	$m \mid p \mid s \mid \chi$
Packet	p	::=	$\mathsf{P}(a, a, y)$
Association In	σ^i	::=	$\mathsf{In}(a, \iota)$
Association Out	σ^o	::=	$\mathsf{Out}(a, \iota)$
Associations	Σ	::=	$\mathsf{Assoc}(\{\sigma^{\mathsf{i}}, \ldots \sigma^{\mathsf{i}}\}$
			$\cup\{\sigma^o_1, \ldots \sigma^o_m\})$
Bundle In	β^i	::=	$\mathsf{Bndl}[\sigma^i, \ldots, \sigma^i]$
Bundle Out	β^o	::=	$\mathsf{Bndl}[\sigma^o, \ldots, \sigma^o]$
Pattern	ω	::=	$a \mid d \mid *$
Selector	ψ	::=	$\omega \rightarrow \omega$
Security Mech In	π^i	::=	$\mathsf{Mech}(\psi : v : \beta^i)$
Security Mech Out	π^o	::=	$\mathsf{Mech}(\psi : v : \beta^o)$
Mechanisms In	Π^i	::=	$\mathsf{MechIn}[\pi^i_1, \ldots, \pi^i_n]$
Mechanisms Out	Π^o	::=	$\mathsf{MechOut}[\pi^o_1, \ldots, \pi^o_m]$
Communication	η	::=	$\omega \mapsto \omega \mid \omega \leftrightarrow \omega$
Resumption	z	::=	$\beta \mid a \mid p \mid \iota \mid k$
Resumption Term	Z	::=	$\langle\rangle \mid \langle z, \ldots, z\rangle$

Table 2. Tunnel Calculus Elements

The elements given in Table 2 are the basic structures used to model packets, messages, associations, and mechanisms. The establishment messages must undergo special processing and are distinguished by the X constructor. The association database Σ is the set of associations active at a node. There are distinguished mechanisms for inbound π^i and outbound π^o traffic. As discussed above, a mechanism is a triple consisting of a packet filter ψ, a session identifier v, and a bundle of associations. A bundle is a list of inbound β^i or outbound β^o associations. An inbound Π^i and an out-

bound Π^o mechanism database is maintained at each node. A mechanism database is a list of inbound or outbound security mechanisms. Resumption terms represent the state of a protocol execution and are used to control the order of execution of rewrite rules.

The terms of the tunnel calculus are specified in Table 3. Packets, resumption terms, the association database, and the

Term	t	::=	$F \mid p \mid Z \mid \Sigma \mid \Pi^i \mid \Pi^o \mid$
To IP			$\downarrow_{\mathrm{ip}(k)} p \mid$
Ack from IP			$\uparrow_{\mathrm{ip}(k)} \mid$
Receive from IP			$\Uparrow_{\mathrm{ip}} p \mid$
To Sec			$\downarrow_{\mathrm{sec}(v,k)} p \mid$
Ack from Sec			$\uparrow_{\mathrm{sec}(k)} \mid$
Receive from Sec			$\Uparrow_{\mathrm{sec}(v)} p \mid$
To Establish			$\downarrow_{\mathrm{est}(v,k)} \mathsf{E}(a,a,a) \mid$
Ack from Estab			$\uparrow_{\mathrm{est}(k)} \mid$
To Est Resp			$\downarrow_{\mathrm{eresp}(v,k)} \mid$
Ack from Est Resp			$\uparrow_{\mathrm{eresp}(k)} \mathsf{R}(a)$
To Authorization			$\downarrow_{\mathrm{auth}(v,k)} \mathsf{A}(L,K)$
Ack from Auth			$\uparrow_{\mathrm{auth}(k)} B \mid$
Node Term	*nt*	::=	$t \mathbin{@} a$

Table 3. Tunnel Calculus Terms

mechanism databases are terms. The other terms represent interfaces. For instance, a packet p is sent down the IP stack by writing a $\downarrow_{\mathrm{ip}(k)} p$ term; and a packet traveling up the stack from the IP layer is given by the term $\Uparrow_{\mathrm{ip}} p$. *Node terms* have a grammar $nt ::= t \mathbin{@} a$, where t is a term located at node a. Each node in the network will have a collection of node terms representing the state at that node. The state of the entire network is represented as a multiset of node terms.

The outbound processing rules were given earlier in terms of semantic functions BndlSel and Nest. Here are precise definitions

$$\mathrm{BndlSel} : \mathit{Addr} \times \mathit{Addr} \times \mathit{Session} \times \mathit{Policies} \rightsquigarrow \mathit{Bundle}$$

$$\begin{aligned} \mathrm{BndlSel}(b,c,v,\Pi^o) &= \beta \\ &\quad \text{if } \mathsf{Mech}(b \to c : v : \beta) \in \Pi^o \\ \mathrm{BndlSel}(b,c,v,\Pi^o) &= \mathsf{Bndl}[] \quad \text{otherwise} \end{aligned}$$

$$\mathrm{Nest} : \mathit{Bundle} \times \mathit{Addr} \times \mathit{Session} \times \mathit{Packet} \to \mathit{Packet}$$

$$\begin{aligned} \mathrm{Nest}(\mathsf{Bndl}[], e, v, p) &= p \\ \mathrm{Nest}((\mathsf{Out}(d,\iota) :: \beta), e, v, p) &= \mathrm{Nest}(\beta, e, v, \mathsf{P}(e, d, \mathsf{S}(v, \iota, p))) \end{aligned}$$

Secure layer inbound processing uses a semantic function Strip, which removes and verifies the headers of secure packets that are destined for the node performing the processing. When Strip is initially called the session number is unknown and the parameter is assumed to be set to $-\infty$. The type is

$$\begin{aligned} &\mathrm{Strip} : \mathit{Associations} \times \mathit{Addr} \times \mathit{Packet} \times \mathit{Bundle} \rightsquigarrow \\ &\mathit{Packet} \times \mathit{Bundle} \; + \; \mathsf{Exchange}(\mathit{Packet} \times \mathit{Bundle}). \end{aligned}$$

Due to space considerations we do not give the detailed definition here.

We now give the rules for each layer of the tunnel calculus accompanied by a brief explanation. The rule identifiers use the notation F for Forwarding, S for Secure processing, and E for Establishment. In the presentation that follows, the rule precedes a brief explanation.

We begin by examining the two rules for the forwarding layer that move packets from one node to another.

Rule F.1.1

$$\begin{aligned} \mathsf{F}(f) \mathbin{@} a \quad \vdash \quad & \downarrow_{\mathrm{ip}(k)} \mathsf{P}(b,c,y) \mathbin{@} a \longrightarrow \\ & \mathsf{P}(b,c,y) \mathbin{@} f(c),\ \uparrow_{\mathrm{ip}(k)} \mathbin{@} a \end{aligned}$$

In **Rule F.1.1**, the forwarding table appears to the left of the $\vdash$ indicating that it can be used in the rule, but is not removed from the multiset. If a packet from b to c is ready for dispatch at a, then it is sent to the node $f(c)$ obtained from the forwarding table at a. An acknowledgment of this dispatch is provided at a. This is not an acknowledgment of delivery at $f(c)$, however.

Rule F.2.1

$$\vdash_a \quad p \longrightarrow \Uparrow_{\mathrm{ip}} p$$

If a packet p has been received at a node, the forwarding layer responder **Rule F.2.1** rewrites to $\Uparrow_{\mathrm{ip}} p$, indicating that the message has been received.

The secure layer performs processing of secure packets. The rules for the secure layer initiator are given in the main body of this text and are not repeated here. The secure layer responder processes an incoming message that has been passed up by the forwarding layer responder. If the message is an establishment packet, then the packet is sent up to the higher layers for processing (**Rule S.2.1**). If a non-establishment packet arrives in a valid association and is destined for this node, then pass the packet up for further processing (**Rule S.2.3**). If a non-establishment packet arrives in a valid association and is destined for a different node, then send the packet on its way (**Rules S.2.4, S.2.5**). This processing is formalized by the following five rules.

Rule S.2.1

$$\begin{aligned} \Sigma \quad \vdash_e \quad & \Uparrow_{\mathrm{ip}} p \longrightarrow \\ & \Uparrow_{\mathrm{sec}(v)} \mathsf{P}(b,c,\mathsf{X}(\kappa)) \\ & \text{where } \mathsf{Exchange}(\mathsf{P}(b,c,\mathsf{X}(\kappa)),\beta) \\ & = \mathrm{Strip}(\Sigma, e, -\infty, p, \mathsf{Bndl}[]) \\ & \text{if } \mathsf{Mech}(b \to c : v : \beta) \in \Pi^i \text{ or } (\beta = [] \\ & \text{and not } (\mathsf{Mech}(b \to c : v : \beta') \in \Pi^i \\ & \text{and } \beta \neq \beta')). \end{aligned}$$

Rule S.2.1 only executes if the decapsulated message received from the forwarding layer is an establishment message and there is either a matching entry in the mechanism database, indicating that the message arrived in a valid tunnel, or there is no matching entry and an empty bundle value has been returned by Strip, indicating that the packet has arrived in the clear. The rule passes the message up for further processing.

Rule S.2.2

$$\begin{aligned} \Sigma \quad \vdash_e \quad & \Uparrow_{\mathrm{ip}} p \longrightarrow \langle p', \beta, v \rangle \\ & \text{where } (p', v, \beta) = \mathrm{Strip}(\Sigma, e, -\infty, p, \mathsf{Bndl}[]). \end{aligned}$$

The second rule of the secure layer responder decapsulates messages that are not establishment packets. The next two

rules decide what to do with the message.

Rule S.2.3

$$\Pi^i \quad \vdash_e \quad \langle \mathsf{P}(b,c,y), \beta, v\rangle \longrightarrow \Uparrow_{\mathrm{sec}(v)} \mathsf{P}(b,c,y)$$
$$\text{if } e = c \text{ and } \mathsf{Mech}(b \to c : v : \beta) \in \Pi^i.$$

If the packet was traveling in a valid association and it is destined for this node, then **Rule S.2.3** passes it up for further processing.

Rule S.2.4

$$\Pi^i \quad \vdash_e \quad \langle \mathsf{P}(b,c,y), \beta, v\rangle \longrightarrow \downarrow_{\mathrm{sec}(v,k_4)} \mathsf{P}(b,c,y),\ \langle v, k_4\rangle$$
$$\text{if } e \neq c \text{ and } \mathsf{Mech}(b \to c : v : \beta) \in \Pi^i$$
$$\text{where } k_4 \text{ is } \textit{new}.$$

Rule S.2.5

$$\vdash_e \quad \langle k_4\rangle,\ \uparrow_{\mathrm{sec}(k_4)} \ \to \cdot$$

If the packet was traveling in a valid association and it is not destined for this node, then **Rule S.2.4** invokes the secure layer to send the packet towards its destination. Upon receiving the acknowledgment that the message has been sent, the protocol terminates in **Rule S.2.5**.

We now present the rules for the establishment layer. The initiator a invokes the establishment layer by writing a $\downarrow_{\mathrm{est}(v,k)} \mathsf{E}(b,s,d)$ term, where b is the responder and s and d are the packet filters to be installed in the mechanism database. The establishment layer initiator is defined by the following three rules.

Rule E.1.1

$$K^a \quad \vdash_a \quad \downarrow_{\mathrm{est}(v,k_1)} \mathsf{E}(b,s,d) \longrightarrow$$
$$\downarrow_{\mathrm{sec}(v,k_2)} \mathsf{P}(a,b,\mathsf{X}(\mathsf{Req}(s,d,v,\iota_a,K^a))),$$
$$\langle v,a,b,s,d,k_1,k_2,\iota_a\rangle$$
$$\text{if } \exists \mathsf{In}(b,\iota_x) \in \Sigma \text{ then } \iota_a = \iota_x \text{ else } \iota_a \text{ is } \textit{new}$$
$$\text{where } k_2 \text{ is } \textit{new}.$$

Rule E.1.1 generates an establishment request message. If there is an existing association flowing from b to a, then use the existing association. Otherwise, generate a new SPI value ι_a. The initiator then sends the establishment request message to node b.

Rule E.1.2

$$L^a \quad \vdash_a \quad \langle v,a,b,s,d,k_1,k_2,\iota_a\rangle,$$
$$\uparrow_{\mathrm{sec}(k_2)},$$
$$\Uparrow_{\mathrm{sec}(v)} \mathsf{P}(b,a,\mathsf{X}(\mathsf{Rep}(s,d,v,\iota_a,\iota_b,K^b))) \longrightarrow$$
$$\downarrow_{\mathrm{auth}(v,k_3)} \mathsf{A}(L^a,K^b),$$
$$\langle v,a,b,s,d,k_1,k_2,k_3,\iota_a,\iota_b\rangle$$
$$\text{where } k3 \text{ is } \textit{new}.$$

Upon receiving the establishment response message, the second rule of the establishment initiator invokes the authorization layer to see if the credential K^b satisfies the policy L^a.

Rule E.1.3

$$\vdash_a \quad \langle v,a,b,s,d,k_1,k_2,k_3,\iota_a,\iota_b\rangle,$$
$$\Sigma, \Pi^i, \Pi^o, \uparrow_{\mathrm{auth}(k_3)} (\mathsf{true}) \longrightarrow$$
$$\Sigma \cup \{\mathsf{Out}(b,\iota_b)\},$$
$$\mathsf{Mech}(d \longrightarrow s : v : \mathsf{Bndl}[\mathsf{Out}(b,\iota_b)]) \otimes \Pi^o,$$
$$\Sigma \cup \{\mathsf{In}(b,\iota_a)\},$$
$$\mathsf{Mech}(s \longrightarrow d : v : \mathsf{Bndl}[\mathsf{In}(b,\iota_a)]) \otimes \Pi^i,\ \uparrow_{\mathrm{est}(k_1)}.$$

If the authorization layer returns true, then **Rule E.1.3** updates the association and mechanism databases for both associations and writes the establishment acknowledgment term.

The establishment layer responder is invoked by node b in expectation that an initiator node will be discovered and perform establishment with it. The establishment layer responder is defined by the following three rules.

Rule E.2.1

$$L^b \quad \vdash_b \quad \downarrow_{\mathrm{eresp}(v,k_1)},$$
$$\Uparrow_{\mathrm{sec}(v)} \mathsf{P}(a,b,\mathsf{X}(\mathsf{Req}(s,d,v,\iota_a,K^a))) \longrightarrow$$
$$\downarrow_{\mathrm{auth}(v,k_2)} \mathsf{A}(L^b,K^a),$$
$$\langle v,a,b,s,d,\iota_a,k_1,k_2\rangle$$
$$\text{where } k_2 \text{ is } \textit{new}.$$

Upon the arrival of an establishment request message, **Rule E.2.1** invokes the authorization layer to verify that the initiator's credential K^a satisfies the policy L^b.

Rule E.2.2

$$K^b \quad \vdash_b \quad \langle v,a,b,s,d,\iota_a,k_1,k_2\rangle,$$
$$\uparrow_{\mathrm{auth}(k_2)} (\mathsf{true}),\ \Sigma,\ \Pi^i, \longrightarrow$$
$$\Sigma \cup \mathsf{In}(a,\iota_b),$$
$$\mathsf{Mech}(d \to s : v : \mathsf{Bndl}[\mathsf{In}(a,\iota_b)]) \otimes \Pi^i,$$
$$\downarrow_{\mathrm{sec}(v,k_3)} \mathsf{P}(b,a,v,$$
$$\mathsf{X}(\mathsf{Rep}(s,d,v,\iota_a,\iota_s,K^b))),$$
$$\langle v,a,b,s,d,\iota_a,\iota_b,k_1,k_2,k_3\rangle$$
$$\text{where } k_3 \text{ is } \textit{new}.$$
$$\text{if } \exists \mathsf{In}(a,\iota_x) \in \Sigma \text{ then } \iota_b = i_x \text{ else } \iota_b \text{ is } \textit{new}.$$

If the policy has been satisfied, then **Rule E.2.2** generates an establishment response message. If there is an existing association flowing from the initiator to the responder, then it gets reused. Otherwise, a new association is generated. The establishment response message is sent to node a and entries are then added to the association and mechanism databases for the association flowing from a to b and the establishment reply message is sent.

Rule E.2.3

$$\vdash_b \quad \langle v,a,b,s,d,\iota_a,\iota_b,k_1,k_2,k_3\rangle,$$
$$\Sigma, \Pi^o, \uparrow_{\mathrm{sec}(k_3)} \longrightarrow$$
$$\uparrow_{\mathrm{eresp}(k_1)} \mathsf{R}(a)$$
$$\Sigma \cup \{\mathsf{Out}(a,\iota_a)\},$$
$$\mathsf{Mech}(s \to d : v : \mathsf{Bndl}[\mathsf{Out}(a,\iota_a)]) \otimes \Pi^o.$$

Upon acknowledgment that the reply has been sent, **Rule E.2.3** adds entries the association and mechanism databases for the association flowing from b to a.