A FOUNDATION FOR TUNNEL-COMPLEX PROTOCOLS

Alwyn E. Goodloe

A DISSERTATION

in

Computer and Information Science

Presented to the Faculties of the University of Pennsylvania in Partial Fulfillment of the Requirements for the Degree of Doctor of Philosophy

2008

Carl A. Gunter
Supervisor of Dissertation

Rajeev Alur
Graduate Group Chairperson

Acknowledgements

I would like to thank my parents who supported me with encouragement, food, and at times with money. Without them, the whole process would have been much rougher. I would like to also think my sister who paid for my cell phone and constantly fed me the latest stories from ‘the big house’.

I would like to acknowledge professors L. Brian Lawrence and Philippe Loustaunau who during my time as a masters student in the George Mason University mathematics department taught me many of the skills I needed to survive the Ph.D. program at Penn as well as encouraging me in my crazy idea to abandon a steady six-figure income and live in poverty in the pursuit of science.

I have encountered many amazing students in the CIS department at Penn. I was especially like to acknowledge the friendship of Andrew Schein; the Johnson to my Boswell [52]. Michael McDougall was a wonderful office-mate for many years and I enjoyed our joint research efforts. I would also like to acknowledge the support others working in my research group. In particular, Karthik Bhargavan, Michael J. May, Matt Jacobs, and Gaurav Shah. I could not imagine that I would have survived grad school if it were not for the friendship of Vladimir Gapeyev, and Swarat Chaudhuri. Kurt Heidelberg, Albert Montillo, T.J. Smith and Steven Weber also provided much needed support. At times Mark-Oliver Stehr served as a surrogate adviser and his friendship kept me from going stir-crazy during a prolonged stay in Urbana-Champaign. I would also like to thank Elsa, George, and Richard Gunter for hosting me on my many subsequent visits to Urbana.

I must also acknowledge Prof. Benjamin C. Pierce for teaching me many things about programming languages and more importantly helping me to become a better writer. I would also like to acknowledge professors Jean Gallier, Rajeev Alur, Insup Lee, Andre, Schedrov, Peter Freyd, and Stephanie Werich for the many lessons learned in their classes and seminars. I would like the thank the Penn PL Club for including me in their seminars and activities during my time at Penn.

All of my committee members have supported me in this effort. I would especially like to thank Catherine Meadows for encouraging my investigations into tunnel-complex protocols and inviting me to visit NRL. I would also like to give special thanks to Steve Zdancewic who took time to listen to my ideas and provided funding without which I may never have finished. I would also like to acknowledge Joshua Guttman, and other the members of the protocol eXchange group that encouraged my investigations into tunnel-complex protocols.

Above all, I would like to acknowledge the support of Carl A. Gunter for acting as my adviser. The roots of this work began with experiments we carried out on authenticated SYN traversal almost six years ago, morphed into interests in security tunnels, and finally we hit upon the idea of studying tunnel-complex protocols. Without Carl's guidance and insight this dissertation would not have been possible.

ABSTRACT

A FOUNDATION FOR TUNNEL-COMPLEX PROTOCOLS

Alwyn E. Goodloe

Carl A. Gunter

Tunnel-complex protocols construct different tunnel topologies by directing tunnel-establishment protocols to set up pair-wise tunnels between different nodes, where the resulting tunnel complex satisfies some security requirement such as negotiating a defense in depth. Such protocols ease the burden on network managers deploying innovative solutions involving tunnel complexes to secure communication and protect networks. Tunnel-complex protocols exhibit subtleties relating to functional correctness and Denial of Service (DoS) that can benefit from formal analysis. We introduce a formalism called the *tunnel calculus*, which provides an operational semantics for a protocol stack incorporating the structures that maintain tunnel state as well the packet header transformations carried out by security tunnels. All subsequent analysis is based on this formalism. The tunnel calculus is applied to analyzing functional properties of both tunnel-establishment protocols and tunnel-complex protocols. The formalism is used to exhibit a situation where establishment protocol execution interacts with the state being installed so as to cause a deadlock. *Non-interference* and *progress* properties are formulated and proved in our framework showing the absence of this deadlock in a revised protocol. The utility of the tunnel calculus is illustrated in a number of case studies of *discovery protocols* that discover security gateways and set up tunnels to negotiate their traversal. For each protocol, we prove a functional *completeness* property that characterizes how the protocol delivers credentials to gateways as part of the negotiation process. We consider the the effectiveness of specific DoS protections for discovery protocols using a cost model for the tunnel calculus. In addition, we formulate and prove a theorem that says a particular class of attackers cannot induce the DoS-resistant protocol to perform high-cost activities.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Security tunnels protect information traveling between two hosts over an insecure network. A collection of security tunnels can be composed to provide more sophisticated security guarantees. Yet the difficulty of configuring tunnel complexes has been a limiting factor in practice. Support protocols can be employed to set up sophisticated tunnel complexes, but designing such protocols is a difficult and error-prone task that is poorly understood. If the underlying network topology is unknown, the design issues become even more formidable. This dissertation is the first formal study of protocols that coordinate the construction of a tunnel complex, which we designate as *tunnel-complex protocols*. Our focus is primarily on two issues confronting the designer of tunnel-complex protocols. The first issue is functional correctness, in particular, deadlocks, which we have found can arise in even seemingly simple protocols. The second issue is Denial-of-Service (DoS) threats against both the protocols and the tunnel configuration itself. A formalism called the *tunnel calculus* is introduced that has been engineered for expressing and reasoning about tunnel protocols. A particular strength of the tunnel calculus is the ability to reason about the interactions between executing protocols and the state being installed at the nodes as the protocol sets up tunnels, which can give rise to deadlocks unless care is taken. In addition, we use the formalism to study protocols that discover security gateways and negotiate their traversal by setting up security tunnels. The effectiveness of countermeasures used to protect tunnel-complex protocols against DoS attacks are also analyzed using the tunnel calculus.

The remainder of this chapter is organized as follows: we first provide background material on security tunnels and security gateways; a discussion motivating the need for tunnel-complex protocols follows; we then summarize the functional correctness and availability issues that are our primary focus; a survey of related work places the dissertation in relation to existing research; finally, we highlight the contributions made in this dissertation.

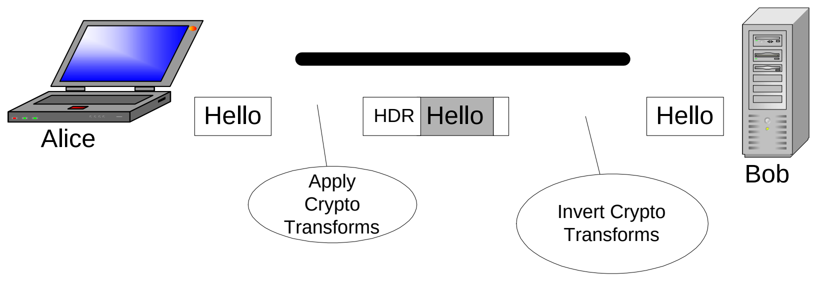

Figure 1.1: Security Tunnel Acting on a Message

1.1 Background

As Internet use has become ubiquitous, its public networks have become the conduit for exchanging extremely sensitive information, for instance, credit card transactions. Unless secured by additional means, traffic flowing over the Internet is subject to eavesdropping. Adversaries can also assume someone else's identity since authentication and authorization are not part of the basic suite of protocols that carry the majority of Internet traffic. The science of cryptography provides the means to secure messages from prying eyes via encryption and to ensure that a message actually does originate from the purported sender via digital signatures. These functions could be built into applications, but given the widespread need to protect sensitive information, a standardized device for protecting selected traffic is preferred. *Security tunnels* provide a solution to protecting private data traveling over a public network. Security tunnels are a mechanism where two nodes share state that enable them to apply cryptographic transformations to messages to ensure that they are secure and authenticated, resulting in a secure virtual communication channel. As each message enters a tunnel, a cryptographic operation is performed to encrypt and/or digitally sign the message and a special header is added. At the other end of the tunnel the digital signature is verified to ensure that the message actually originated at the purported origin and the message is decrypted. This processing is illustrated in Figure 1.1. Among the many standardized tunneling products in use today are the Point-to-Point Tunnel Protocol (PPTP) [63], which works at the link layer, Transport Layer Security (TLS) [36], which works at the transport layer, Secure Shell (SSH) [117, 116], which works at the application layer, and IP security (IPsec) [80, 79, 78], which works at the network layer. In this dissertation, we chose to model tunnels at the network layer because of the flexibility it affords and borrow terminology and basic structures from the IPsec, but are not wedded to a specific standard.

The primary components of our model for security tunnels are as follows. *Security associations* (SA) define the shared parameters between the tunnel end-points

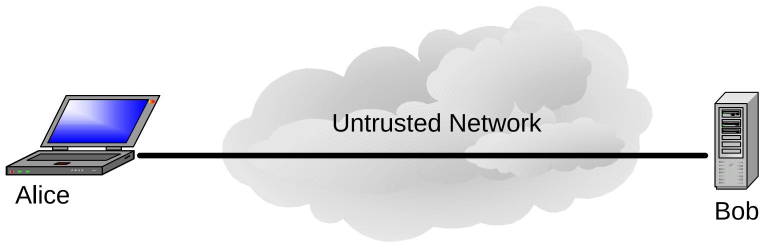

Figure 1.2: Basic Road Warrior Scenario

such as the cryptoprotocol parameters and secret keys that define the cryptographic transforms applied to each message traveling in a tunnel. Both nodes maintain this information in their *Security Association Database* (SADB). Each association possesses a unique identifier called the *Security Parameter Index*. *Security mechanisms* filter and direct traffic into the proper security associations. Such filters are sometimes referred to as security policies, but we prefer to distinguish these from high-level security policies. Security mechanisms are maintained in a *Mechanism Database* (MDB) at both nodes. Security tunnels are typically composed of a pair of unidirectional associations securing communication flowing in each direction. Differing technologies may or may not be structured in this fashion and may use different terminology.

Probably the most common application of security tunnels is when someone acting as a client employs a tunnel to secure communication to a server. This may be to secure a financial transaction such as buying a book over the Internet or to secure sensitive information, such as an employee communicating with a company server while away from the office. This scenario is often called the *Road-Warrior* scenario and is depicted in Figure 1.2.

The state at each tunnel endpoint can be manually configured. This requires that state either be preinstalled when the hardware is delivered or be distributed to the administrators who then install the keys and other information manually. Both choices are error prone making the secret keys vulnerable to an attacker. *Tunnel-establishment* protocols automate the most burdensome aspects of this task by creating the shared cryptographic keys used by the security association. The cryptographic keys are typically created using a Diffie-Hellman key-exchange [37, 93, 19]. The shared keys along with other information is installed in the SADB and the MDB may also be updated. Given that organizations may not want to allow just anyone to form a connection to their computers, tunnel-establishment protocols typically perform some form of authentication and authorization before creating a tunnel. This process may be as simple as requiring a password or as sophisticated as using digitally signed credentials such as SPKI/SDSI [44, 46] certificates. IKEv2 [76] is an example of a standardized tunnel-establishment protocol developed to set up IPsec tunnels.

Consider a corporate network providing no security. It resembles a public highway in that any packet may travel over the corporate network, where the sole burden for defense lies at the host. Virus scanners, spyware detectors, and similar software are necessary layers of defense that must reside at the host. Software based firewalls can be deployed at the hosts to block hostile traffic. Given that firewalls can be inconvenient, the administrators at each host are tempted to alter to firewall rules. Users may even disable the firewall unaware of the consequences to the security of their system. Hence host-based security complicates an administrator's ability to centrally formulate and enforce a corporate security policy. Another drawback to placing all the security at the hosts is that it offers no protection to the network infrastructure. If no mechanism is in place to regulate traffic flowing into an organization's network, attackers are given the freedom to probe every host and router for vulnerabilities as well as to flood a network with packets in a DoS attack. A solution is to push some of the burden for security into the network itself so as to control who is allowed to perform specified actions on a corporate network.

An integral part of incorporating security into the network is the placement of a security gateway at the perimeter of the corporate network, dividing the network into an untrusted network outside of the gateway and a trusted network behind the gateway. The gateway enforces policies governing the flow of data into and out of its administrative domain. It also centralizes security policy management providing protection for all elements within its protective zone. Packet filtering firewalls provide some protection by blocking access to all but specified ports and services, but do not provide authentication services. Since the responsibility for authentication and authorization remains at the hosts, the primary burden for preventing unauthorized activity remains at the hosts. A defense-in-depth philosophy advocates pushing more of the responsibility for security into the network. For instance, security gateways that perform authentication and authorization transform the network from a public highway into a gated community by only allowing authorized traffic to pass. A principal must present valid credentials in order to authenticate itself to the gateway before being allowed to pass. Since we want all traffic authorized and authenticated, but do not want to have to repeat the credential verification and authorization procedure for each packet, the process is integrated with the establishment of a security tunnel. Hence, gateways and tunnels naturally complement each other as gateways can enforce their policy during tunnel establishment. Once the tunnel is set up, all traffic traversing the gateway via the tunnel is ensured to be authenticated and authorized as satisfying the gateway's policies.

A wide range of attacks can be thwarted by a single gateway placed at an outer perimeter of a corporate network as illustrated in Figure 1.3. Yet this configuration is often referred to as a brittle outer shell defense because once it has been penetrated, only the defenses of the host remain. Thirty-nine percent of the responders to the 2006 FBI/CSI survey [56] indicate that more than twenty percent of their losses

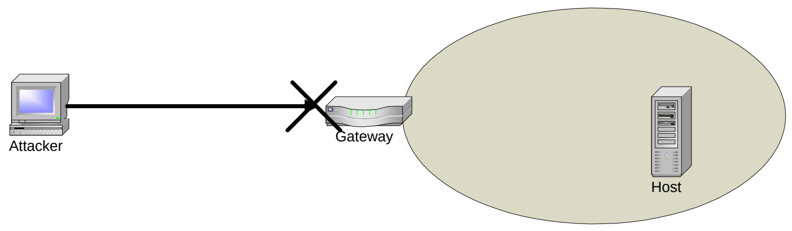

Figure 1.3: Single Gateway

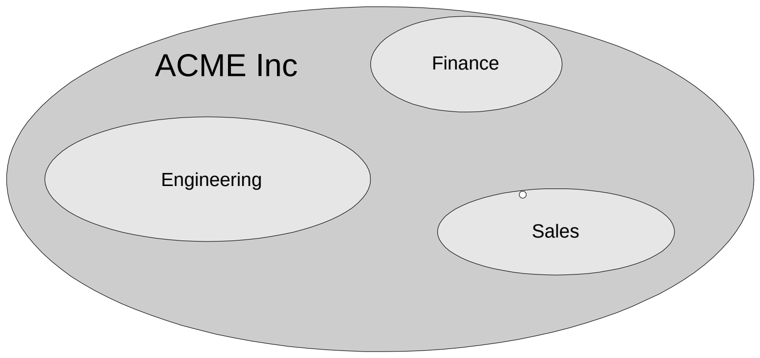

Figure 1.4: Nested Security Domains

were due to insiders. For example, an unscrupulous company salesperson attempting to gain access to financial records or a visitor who has been allowed network access by one department need not worry about the gateway and is given access to attack every host on the network. A more subtle attack could be carried out by a principal that is authorized to communicate with a host in one department, but takes over that machine and uses it to attack machines containing more sensitive data. These threats are increasingly being recognized in regulations that protect patient medical records or corporate financial data. For instance, the Health Insurance Portability and Accountability Act (HIPPA) (164.308(a)(4)(i), 164.308(a)(4)(ii)(A)) [115] requires polices and procedures that separate patient records from other operations as well as procedures to ensure that all access is authorized. Sensitive information is rarely contained on a single computer today, but distributed among the severs and workstations within an organization. In many cases, auditors insist a particular department be isolated from the rest of the organization. Failure to do so would subject every machine in the corporation to stringent and costly regulations. Partitioning the corporate network into domains protected by a security gateway as illustrated in Figure 1.4 allows for fine-grained access control. Figure 1.5 depicts the networks of two organizations who have adapted a defense-in-depth philosophy.

Consider the case of an organization composed of a collection of separate campuses. Each campus network is typically protected by a gateway and tunnels are used to connect the campuses to form a virtual private network (VPN) as seen in

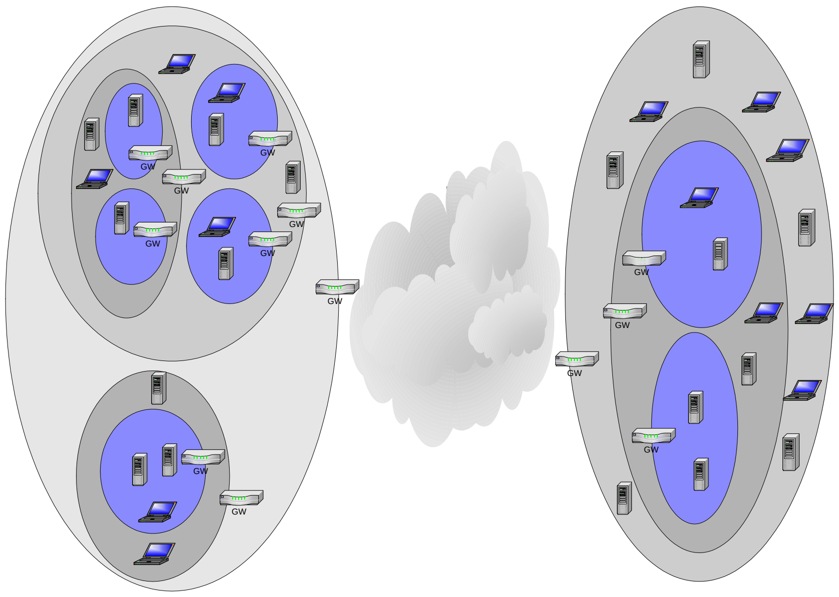

Figure 1.5: Defense in Depth

Figure 1.6. This is one of the more commonly deployed tunnel topologies and has a relatively straightforward configuration. Yet configuring tunnels to each gateway on the communication path in a network with many levels of nesting such as the one illustrated in Figure 1.5 is nontrivial. Nesting tunnels offers many benefits, but can complicate configuration. In the next section, we shall discuss how a special class of protocols can mitigate the difficulties in setting up such a collection of tunnels.

1.2 Motivation

Security tunnels can be composed to create a complex of tunnels that collectively provide a specific security guarantee. To illustrate this, consider the variant of the road warrior scenario presented in Figure 1.7. In order to communicate with Bob's server located back at Coyote corporation, Alice must authenticate to a wireless access port to gain access to the Internet, authenticate to the Coyote Corp corporate gateway to gain entrance to the company network, and finally authenticate to Bob. The end-to-end communication between Alice and Bob should be secured using encryption. Alice can satisfy this requirement by setting up tunnels to each of these nodes. A common solution is to set up the tunnel complex seen in Figure 1.7, where the end-to-end tunnel labeled C is nested inside of the tunnel labeled B flowing between Alice and the gateway, which is nested inside of the tunnel labeled A flowing between Alice and the wireless access point. Typically, this is set up in an ad hoc fashion with each tunnel using a different technology. For instance, tunnel A may be a PPTP link-layer tunnel, tunnel B a network-layer IPsec tunnel, and tunnel

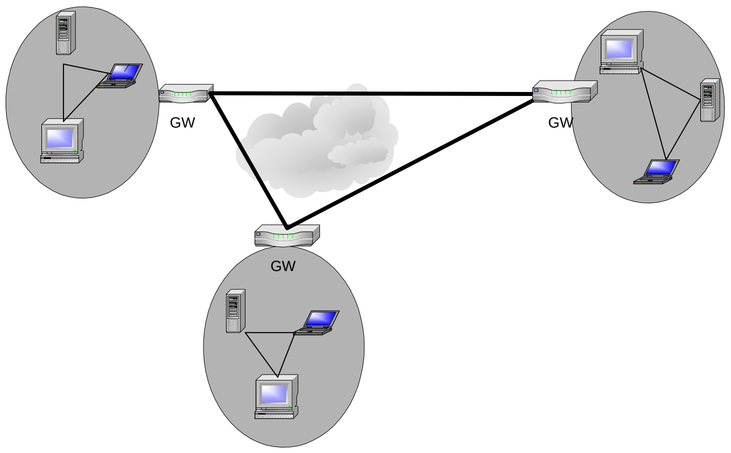

Figure 1.6: Virtual Private Network

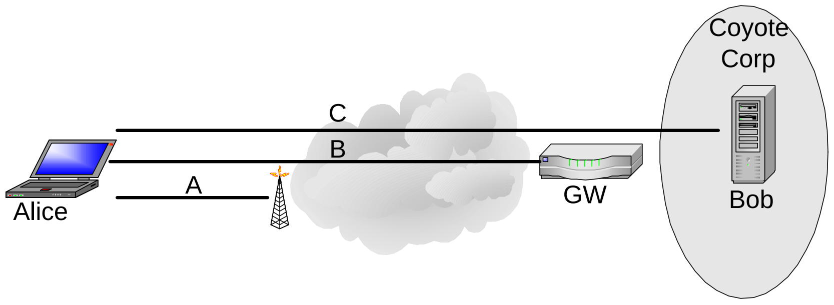

Figure 1.7: Road Warrior Scenario Variant

C a transport-layer TLS tunnel, where each tunnel performs both authentication and encryption even though only the end-to-end tunnel need encrypt. Given that data is being encrypted three times, users may experience significant performance degradation. A more coordinated solution would be to employ a support protocol to coordinate the configuration of the tunnels. In the case of the road-warrior example in Figure 1.7, the support protocol would configure three nested tunnels with an authentication tunnel to the access port, an authentication tunnel to the gateway, and an end-to-end tunnel that performs both encryption and authentication. Performing all of this at the network layer eases the burden of coordination. We designate this class of protocols as *tunnel-complex protocols*. These protocols use tunnel-establishment protocols as basic components and direct them to set up a tunnel between two nodes. Tunnel-complex protocols create different topologies by invoking establishment at different nodes with varying parameters.

As we have seen, tunnels and gateways are used to create secure virtual networks. The configuration of these networks is commonly done using a router command line interface. Tools such as Solsoft's Policy Manager [112] facilitate centralized management by allowing the network manager to specify and distribute policies to network devices throughout the organization. A more sophisticated approach to centralized VPN management generates tunnel mechanisms from a high-level specification and includes sophisticated algorithms to check the consistency of the generated database entries [50, 51]. Given that centralized administration of VPNs is often error prone and not practical in situations where the topology can change, dynamic VPN configuration is a desirable feature, and one that has not been ignored by industry.

The VPN depicted in Figure 1.6 forms a complete graph. Another popular VPN configuration in use today is the hub and spoke model where a collection of branch offices connect to a central 'hub' office as illustrated in Figure 1.8. In this topology, all traffic between different spokes must travel through the headquarters hub. Cisco's Dynamic Multipoint VPN (DMVPN) [28] protocol can be viewed as a basic tunnel-complex protocol aimed at this topology. DMVPN alleviates the burden on the hub by setting up direct spoke-to-spoke IPsec tunnels. The spokes typically have dynamic addresses, but maintain a tunnel to the hub. The hub acts as a next hop resolution protocol server (NHRP) [86] with a static IP address. A spoke router learns the address of another spoke from the NHRP hub. If Alice is a host located in spoke A and she wishes to communicate with host Bob located in spoke B, Alice will notify the DMVPN spoke router, which in turn sends a NHRP resolution request to the hub. The hub then sends a resolution request on to spoke B. Spoke B then initiates IKE tunnel establishment with spoke A. When DMVPN process terminates, an IPsec tunnel is set up directly between spokes A and B through which Alice and Bob communicate.

We have established that hierarchically arranged security gateways enforcing polices to protect their administrative domain can provide defense in depth and that

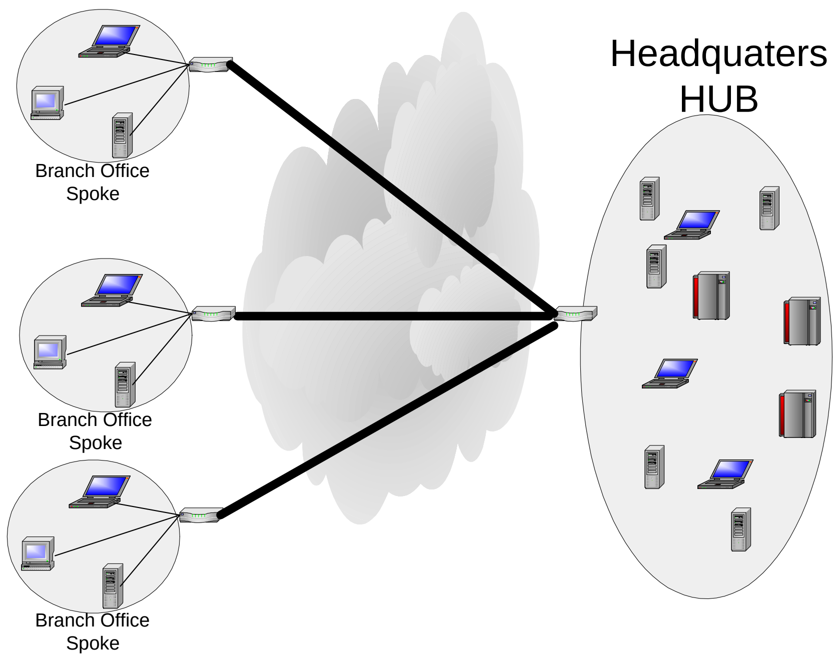

Figure 1.8: Hub and Spoke

security tunnels can be used to ensure that all the traffic that traverses the gateways is authenticated and authorized by the gateway. Tunnel-complex protocols help to automate this process. Any solution is unlikely to scale if the gateways on the dataflow path need to be known beforehand since two different organizations are not likely to communicate each time they change their gateway topology. This situation is akin to the early days of the Internet when forwarding tables were manually configured. A solution is to have the tunnel-complex protocol discover the gateways, negotiate gateway traversal by presenting any credentials needed to satisfy the policies at the gateway, and to configure the tunnels to the gateway so as to form a virtual topology where the traffic flow is governed by the gateway policies. We designate such protocols *discovery protocols*. Cisco's Tunnel Endpoint Discovery (TED) [48] is a discovery protocol that assumes only a single gateway in each organization. The protocol works as follows. Host A sends a packet to Host B. A router acting as a gateway to the domain where host A dwells intercepts the packet and checks to see if there is an existing association with a matching policy. If so, the packet is sent to its destination. Otherwise, a discovery probe is sent to the packet's destination. A gateway protecting Host B intercepts the probe and sends back a reply. The tunnel-establishment protocol IKE is then invoked to set up an IPsec tunnel between the two gateways. More complex discovery protocols are needed in order to traverse nested gateway structures such as the one depicted in Figure 1.5.

1.3 Problem Statements

Tunnel-complex protocols coordinate the creation of a complex of security tunnels in order to satisfy some security policy. This is achieved by invoking tunnel-establishment at different nodes so as to direct the construction of the desired tunnel configuration. Ensuring the functional correctness of these protocols involves verifying that the correct tunnel complex gets constructed. The protocols must also satisfy the traditional security guarantees of integrity, confidentially, and availability. Each of these issues needs to be addressed, yet tunnel-complex protocols have not been subjected to formal analysis and the absence of such foundations will likely encumber attempts to design sophisticated protocols.

Security tunnels are defined by the state of the association and mechanism databases at the tunnel endpoints. The primary functional correctness criteria for tunnel-complex protocols is that they terminate with the association and mechanism databases at selected nodes in a state so as to define the desired tunnel complex. The resulting topology must also be verified as enforcing the intended security policy. If creating the tunnel configuration in Figure 1.7, the protocol must terminate with the association and mechanism databases at the wireless access point, company gateway, and Bob's server each having entries for a pairwise tunnel with Alice. Alice's databases should reflect a nested composition of three tunnels. It would seem that in order to discharge this proof obligation, one need only prove that the tunnel-complex protocol invokes establishment at the proper node with the correct parameters. Yet unexpected interactions between the executing protocols and the state being installed can give rise to problems. In case studies reported in Chapters 2 and 3, we found that these interactions can result in traffic being placed in partially-set-up tunnels creating deadlocks. Such 'friendly fire' incidents where systems are harmed by their own security mechanisms are often appreciated in practical operations. A recent NIST report [114] on Supervisory Control And Data Acquisition (SCADA) security includes a list of documented security incidents (see Section 3.7), which contains as many incidents arising from faults in security protection measures as ones arising from deliberate attacks by adversaries; thereby reinforcing the need to address functional correctness along with traditional security guarantees.

Discovery protocols locate security gateways on the dataflow path and negotiate their traversal. Locating the gateways is a function of the reliability of the network forwarding tables, but gateway traversal is more subtle. A completeness theorem for a discovery protocol would characterize how it delivers the information needed to negotiate gateway traversal and would be a critical correctness criteria. The desired property, and how we formalize it, is an open question, yet one that is central to understanding discovery.

DoS attacks are an attack on the availability of hosts, gateways, and the network itself. An attacker may flood the network with packets in order to consume bandwidth, to entice a node to allocate state in order to consume memory, or to trick a node into performing expensive computations. The TCP SYN attack may have been the first DoS attack generally acknowledged to have affected many Internet users. This attack works as follows, an attacker sends a node a TCP SYN packet with a spoofed return address. The protocol sends an acknowledgement message to the spoofed address, allocates state for the connection, and awaits an acknowledgement, but this will never arrive. Given that it will be a while before connection times out and that the number of connections for a given port are limited, the SYN flooding attack can be used to block a port. Bernstein [10] introduced a countermeasure known as SYN Cookies. The idea is to not allocate state until a round-trip communication is performed with the advertised address. Upon receiving a SYN, a node sends a 'cookie' containing a time-dependent counter, the maximum segment size, and a twenty-four bit value obtained by applying a cryptographic function to the connection four-tuple and time counter. The initiator is expected to send back the cookie and only then is state for the connection allocated. Tunnel-establishment protocols perform costly operations to generate the keys as well as verify signatures and credentials. Such computationally expensive operations can be exploited in a DoS attack similar to the SYN attack. Consequently, cookies are included in many protocols such as IKEv2 to prevent these operations from being performed until after the protocol has verified that the return address is the principal requesting the tunnel. Given that tunnel-establishment protocols form an essential component of tunnel-complex protocols, a DoS attack on the component is an attack on the tunnel-complex protocol itself. Consider the TED protocol discussed above. A distinguished discovery packet arriving at a TED-enabled gateway triggers establishment. Unless sufficient safeguards are in place, a DoS attacker could send the discovery packet followed by the a bogus establishment message in order to entice a gateway to perform costly operations or commit state. Distributed Denial-of-Service (DDoS) attacks often do not depend on address forgery, but instead hijack many nodes, which are in turn used to carry out the attack, making these attacks harder to thwart. One proposed countermeasure is to use client puzzles [72, 6] that force the attacker to solve a small cryptographic puzzle requiring some computational expense. A bot performing many attacks will likely either take too long to solve the puzzle so that the server side times out or the slowness of the system will alert a user that their system has been compromised. Another countermeasure to DDoS attacks uses information theoretic techniques [110] to protect bandwidth so that an increasing amount of work is required to consume a given amount of bandwidth.

Once a tunnel complex has been set up, it should protect the network from unauthorized traffic and therefore protect against DoS attacks. Yet in Chapter 2, we demonstrate how a commonly deployed tunnel complex is subject to a form of DoS attack that we call a *cramming attack*. Here an adversary takes advantage of an 'open ended' tunnel to flood the protected network with packets. Even though the spurious packets are dropped at a tunnel endpoint, unauthorized packets can flood the network, which can have deleterious effects such as undermining the integrity of network accounting systems.

The integrity and security of messages traveling in a tunnel complex is guaranteed by the individual tunnels. These properties must be ensured by the tunnel-establishment protocol that sets up the tunnel. The secrecy and integrity of the shared keys created during tunnel establishment have been the subject of extensive study and we do not address this topic here.

In order to perform a formal analysis addressing the issues raised above, one needs to select a formalism that can express tunnel-complex protocols and reason about properties of interest. Care must be taken in choosing a formalism that can naturally represent essential tunnel structures such as the mechanism and security association databases. Messages traveling from one node to another in a tunnel cannot instantaneously move between nodes as in many formalisms, but must undergo processing as it enters and exits the tunnel. The literature survey given in the next section examines several possible alternatives.

1.4 Related Work

The related work can be broadly classified into the following categories:

- Existing tunnel-complex protocols.
- Reasoning about tunnel-complexes.
- Formal analysis of tunnel-establishment protocols.
- Formal treatments of deadlock.
- Distributed credentials for authorization.
- DoS vulnerabilities of network protocols.
- Alternate formalisms that may be used to analyze tunnel-complex protocols.

In the remainder of this section we shall examine each of these categories and place the related work in relation to the work performed in this document.

1.4.1 Existing Tunnel-Complex Protocols

Tunnel-complex protocols are a relatively recent concept. Although most practical efforts have been dedicated to to the development of centralized management tools such as Solsoft's Policy Manager [112], tunnel-complex protocols are being deployed in products. As we discussed above, Cisco's DMVPN [28] provides the basic functionality of a tunnel-complex protocol that configures a tunnel between two spokes in a hub-and-spoke topology. Discovery protocols have their roots in protocols such as traceroute, which sends out a packet that gets intercepted by nodes on the dataflow path in order to identify the route taken by packets in a network. In the case of discovery protocols, there is a distinguished discovery packet that gets intercepted by security gateways on the dataflow path. Cisco has deployed the aforementioned discovery protocol TED [48]. TED is a great deal simpler than the discovery protocols that we introduce in Chapter 6. Industry seems to avoid complex tunnel topologies due to the difficulty in configuration and both DMVPN and TED are aimed at configuring relatively simple topologies. This conservative approach may, in part, be to due to a recognition that these protocols are poorly understood. We are unaware of any published studies of formal analysis of tunnel-complex protocols.

Tunnel-complex protocols are signaling protocols because they install state at nodes on the dataflow path. Another example of a signaling protocol is the Resource ReSerVation Protocol (RSVP) [20]. This protocol has been subject to several formal treatments. The Specification and Description Language (SDL) was used to formalize a portion of RSVP and to study the interaction been RSVP and routing protocols [49]. The model has two layers, an IP layer modeling routing and message forwarding and a layer modeling the route re-establishment functionality in RSVP. Simulation and state exploration were employed to study how RSVP recovers from link failures. The approach employed in this paper is similar to the one we use in Chapter 2 to model the L3A protocol. On the other hand, the protocols themselves are quite different, which is reflected in the fact that they must model routing, but we elide this detail while modeling the packet header transformations performed by security tunnels. A process algebra is used to model RSVP nodes in [102] as a composition of processes using a number of clever encoding tricks. The primary focus of this work is on modeling arbitrary topologies within a process algebraic framework and proving functional correctness properties. It is interesting to note that the authors speculate that complex signaling protocols could give rise to deadlocks, but do not pursue the issue further.

1.4.2 Reasoning About Tunnel Complexes

Once a tunnel complex has been set up it is necessary to prove that the entire complex satisfies certain security properties. Authentication and confidentiality properties of IPsec tunnels have been formally analyzed in [68]. Confidentiality properties are proved using system invariants and an ‘unwinding set’ is developed to prove authentication properties. Given that this work already addressed this issue, we chose not to address this topic in order to focus our attention on functional and availability properties of tunnel-complex protocols. Yet we believe that the tunnel-calculus could be used as a framework for studying these topics as well.

1.4.3 Formal Analysis of Tunnel Establishment Protocols

Tunnel-establishment protocols are an important component of tunnel-complex protocols. Tunnel-complex protocols invoke tunnel-establishment protocols at different nodes with varying parameters to create the desired tunnel complex. This reflects a natural evolution in computing where a commonly used algorithm or protocol becomes a component in a more complex software system. Tunnel-establishment protocols have been subjected to extensive formal analysis for over a decade. Authentication in key-exchange protocols were rigorously investigated in [38, 8, 7] and [23]. Two establishment protocols intended for use in the IPsec suite of protocols, IKE [76] and Just Fast Keying (JFK) [4], have been formally analyzed [92, 24], and [2]. Recently, Pavlovic and Meadows have analyzed secrecy properties of establishment protocols using a customized logic for deriving cryptographic protocols [100]. All of these papers are primarily focused on verifying the secrecy and integrity properties that must be preserved during key establishment. Recent research [21, 22, 90] has shown that composing security tunnels can result in unexpected interactions that can compromise the cryptographic keys in one or more of the tunnels; and ongoing research is being performed to address this issue. Our approach elides the details of the key exchange in order to focus on functional correctness issues that arise when tunnel-establishment protocols are used as components in tunnel-complex protocols. We are unaware of any other formal analysis focusing on functional properties of establishment protocols.

1.4.4 Distributed Credentials

In order to traverse a security gateway, a principal must be authenticated and authorized to perform the requested action. In the case of discovery protocols, we also require gateways to authenticate themselves. Within our framework, authorization is performed using distributed credentials and assumes the existence of some public key infrastructure. Tunnel-complex protocols are responsible for delivering the credentials that satisfy a particular node's policy. There is a large body of work on distributed credentials that we took inspiration from. Rivest and Lampson proposed a public-key infrastructure called the Simple Distributed Security Infrastructure (SDSI) [106]. SDSI principles are identified with public keys and only things signed by the corresponding private key are recognized. Ellison and Franz, et al. developed the Simple Public Key Infrastructure (SPKI) that provides a mechanism for authorization. SPKI uses authorization certificates to delegate a specific authority from an issuer to a subject. The two efforts have been merged and there is currently a SPKI/SDSI working group in the IETF that has produced several RFCs [44, 46, 45]. The tunnel calculus authorization layer can be viewed as similar to, but simpler, than SPKI/SDSI. There is a substantial body of research dedicated to giving a formal semantics to SPKI/SDSI [1, 61, 62, 85, 29], and [69]. The trust management approach [15] to authorization seeks to create an application independent component to verify if a request is authorized to take some action. Given the local security policy, a set of credentials, and a request, the trust engine returns a decision as to the whether or not the request complies with the policy. PolicyMaker [16, 17] and Keynote [18] are notable examples of this approach. The Query Certificate Manager (QCM) [60, 58, 75, 59] is another system that verifies policies based on certificates. Our model for credentials can be viewed as an abstraction inspired by SPKI and our model of policies are admittedly simple compared to what some others have proposed, but both are sufficient for our purposes. Rather than model the, necessarily complex, machinery used in a specific technology for verifying that a credential set satisfies a given policy, we simply define a relation that acts as a specification that could be satisfied by many of aforementioned proposals.

1.4.5 Formal Treatments of Deadlock

Deadlocks in tunnel-complex protocols are exposed in Chapters 2 and 3 and the tunnel calculus is used to show the absence of a particular deadlock in Chapter 5. The study of deadlocks is almost as old as computer science itself. Early pioneers that introduced concurrent processes into operating systems also introduced machinery such as semaphores and monitors to control access to shared resources [39, 40, 41, 65], but since these mechanisms prevented processes from gaining access to the resource, they gave rise to the possibility that all processes competing for the resource could deadlock. The Dinning Philosophers problem [42] is a classic example of this problem. Concurrency and deadlocks have been studied in the formal methods community for many years. Axiomatic approaches are usually rooted in the Gries-Owiki [57, 70, 71] proof technique. The proof techniques introduced for reasoning about deadlock in the context of concurrent systems can be extended to reason about distributed systems. The texts [5, 109, 27] and [47] contain excellent treatments of reasoning about deadlock in the context of concurrent and distributed programs. Model checking [30, 31] verifies that a transition system is a model for a temporal logic specification. In many cases it is possible to specify the absence of deadlocks as a safety property in temporal logic and use a model checker to find such bugs. Model checking is generally used as a debugging tool where our focus is on proofs of correctness. An alternate approach has its origins in Reynolds' Syntactic Control of Interference [103, 104]. The goal of this program is the design of a powerful Algol-like language in which interference is possible, but syntactically detectable. None of the above approaches have been applied to security tunnels.

1.4.6 DoS

The L3A protocol studied in Chapter 2 creates a tunnel complex to protect the wireless accounting infrastructure from cramming attacks. The basics of accounting and billing for network services is described in [82]. Accounting and billing for Global System for Mobile (GSM) is discussed in [66] and requirements for accounting for Code Division Multiple Access (CDMA) are discussed in [64]. Attacks targeting wireless billing and accounting are a recognized threat. In the so called 'overbilling' attack on the General Packet Radio Service (GPRS) [81, 73], an attacker acting as a mobile station hijacks the IP address of a legitimate mobile station and begins a download from a server on the Internet. The attacker ends the session once the download begins, but since the requested datastream flows to the victim, which gets billed for services that were never requested. This threat arises because the GPRS Transfer Protocol (GTP) provides no security to protect communications between GPRS networks. Several manufactures of firewalls, VPNs, and other equipment advertise that they thwart this attack [74, 101, 67].

Formal methods have been applied to analyze both a protocol's vulnerability to DoS attacks as well as the effectiveness of countermeasures. Abadi, Blanchet, and Fournet perform limited analysis of cookies employed in JFK [2], where they prove that the protocol responder does not commit state until a round-trip has been performed with the initiator. Game theory has also been applied to analyzing DoS countermeasures [87, 9]. Meadow's [89] cost based approach attaches specific values to protocol operations allowing one to analyze the cost of various attacks. This technique has been applied to the Station to Station protocol [89] as well as JFK [111]. Lafrance and Mullins [83] formalize protocols and attackers in a process algebra that incorporates a cost model for cryptographic operations. Assuming that successful DoS attacks are a consequence of a single flaw in a protocol, the authors present a technique for showing whether a single attacker performing only low-cost operations can interfere with the defender's high-cost operations. The problem is formulated in terms of an information flow property. In particular, a theorem that says an attacker's low-cost operations do not interfere with the defender's high-cost operations if the high-cost operations in the system formed from the composition of the defender and attacker simulates those when the defender executes alone. We do not develop a new methodology for studying DoS attacks, but apply Meadow's cost model to analyzing an exhaustive range of attacks. We also state and prove a theorem similar to that in Lafrance and Mullins that says a particular countermeasures are effective against a specific class of attacks.

1.4.7 Alternate Formalisms

Many formalisms have been successfully used to express and reason about security protocols. We now look at a representative sample of these formalisms and discuss why they were are not well suited for reasoning about tunnel-complex protocols. Process algebras such as Communicating Sequential Processes (CSP) [25, 108] and the π-calculus [94, 3, 13] have a long history of being applied to the analysis of protocols and in the last decade they have become a popular vehicle for expressing and analyzing cryptographic protocols. The π-calculus based ProVerif tool [14] provides sophisticated automated assistance to reason about secrecy properties and has been applied to the analysis of the JFK [2] key exchange protocol. Multiset rewriting (MSR) [26, 26] logic is a strongly typed specification language for expressing security protocols and is another possible choice. Processes algebras and and MSR abstract away the details of the communication network and concentrate on the exchange of cryptographic messages. This makes them ideal to analyze if an attacker can gain access to information during an exchange of messages in a cryptographic protocols. Tunnel-complex protocols can be viewed as signaling protocols in that they install state in the SADB and MDB on nodes on the dataflow path. The functional properties that are our focus require us reason about the state of these databases. Consequently, the details that these systems abstract away are those that are needed to express and reason about the class of protocols under study. State machines have long been used to represent protocols. Guttman, Herzog, and Thayer [68] model IPsec tunnels using state machines and formalize authentication and confidentiality properties. On the other had, the databases that are of interest to us are difficult to express in this notation. Logic based approaches have also been applied to protocols. Paulson has applied inductive techniques in HOL to a number of cryptographic protocols [98, 99]. The NRL protocol analyzer (NPA) [91, 92] is another logic-based tool customized for analyzing security protocols. Both the NRL protocol analyzer and the HOL models abstract away the details of communication processing similar to process algebras. A logical approach has also been used to produce detailed and rigorous specification of TCP and UDP [12], but no model of a security tunnel mechanism, such as IPsec, was constructed and the level of detail related to the implementation can be overwhelming if not needed. A custom compositional logic for deriving security protocols and proving secrecy properties has recently been proposed by Mitchell and Pavlovic et al [33, 43]. Given that we do not address those topics, our work would not benefit from the more novel features of this logic. None of the approaches surveyed provides a ready-made framework for expressing and reasoning about tunnel-complex protocols. Leaving us with the option of either customizing an existing formalism or designing a new formalism.

1.5 Contributions

This dissertation contains the first formal treatment of tunnel-complex protocols and discovery protocols in particular. The primary contributions are as follows:

1. The first demonstration of the threat posed by cramming attacks to the wireless accounting infrastructure and a countermeasure in the form of the L3A protocol. (Chapter 2.)

2. A demonstration that formally analyzed tunnel-complex protocols can be implemented on top of the existing Internet security framework and achieve acceptable performance. (Chapter 2.)

3. The introduction of the first formalism designed for expressing tunnel-complex protocols and also designed for reasoning about their functional correctness properties. (Chapters 3 and 4.)

4. The first formal treatment of deadlock issues that arise in tunnel-complex protocols. In particular the following contributions:

   (a) Logical simulations are used to uncover deadlocks in the L3A protocol. (Chapter 2.)

   (b) The tunnel calculus is used to expose a deadlock in a tunnel-establishment protocol. (Chapter 3.)

   (c) Noninterference and progress theorems are formulated and proved showing absence of the aforementioned deadlock for a modified tunnel-establishment protocol. (Chapter 5.)

5. The first demonstration of a completeness property for tunnel-complex protocols. (Chapter 6.)

6. The first study of DoS threats to tunnel-complex protocols and an analysis of countermeasures. (Chapter 7.)

Chapter 2

Layer Three Accounting

The aim of this chapter is to produce a case study of a simple, but realistic, tunnel-complex protocol that went "end-to-end" starting with requirements, then using formal prototypes to debug the design, and finally building an implementation and collecting performance information. Such a controlled study gives us better understanding of the unique demands placed on tunnel-establishment protocols when used as a component of tunnel-complex protocols. Understanding the fundamental difficulties encountered when constructing simple protocols serves to focus further research. The implementation is intended to demonstrate that it is possible to construct tunnel complex protocols using existing computing systems while achieving acceptable performance. The object of this study is a protocol that sets up a collection of tunnels to protect the wireless accounting infrastructure. We begin with an overview of accounting and introduce a form of DoS that can compromise the integrity of the accounting system. We then exhibit a composition of tunnels that we argue thwarts this attack. This is followed by an overview of the Layer 3 Accounting (L3A) protocol that sets up this tunnel complex. The analysis of several iterations of the protocol design is presented. We also present a tear-down protocol. Both L3A and its tear-down protocol were implemented yielding performance that exceeds the standard solution for protecting wireless accounting. The work reported in this chapter appeared in [55] and [54].

2.1 Accounting and Cramming Attacks

All commercial Internet access vendors charge their customers for service. While most conventional wired Internet service providers charge customers a flat monthly charge, the wireless links are sometimes deemed too valuable for such a flat service fee. Vendors typically prefer having the option of charging customers a flat fee or to bill based on the services actually used. Accounting devices are often embedded in the network infrastructure to enable wireless providers to track how much service a user consumes. Emerging protocols for wireless Internet access such as

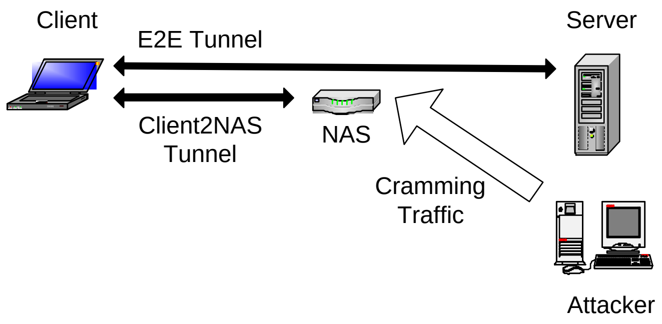

Figure 2.1: Cramming Attack

CDMA2000 [64] have accounting components. RADIUS servers [105] are designed to provide accounting services for protocols such as GPRS. Accounting information is reported to the billing system for computation of the user's charge [82]. It is important that the accounting infrastructure not be compromised otherwise a vendor may not be able to defend itself against a customer challenging his bill. This is a case where profits depend upon having sufficient security in place. The dependence on the integrity of the accounting infrastructure for billing makes it an inviting target for hackers.

A common accounting system is to associate traffic with specific clients on an access network and use a Network Access Server (NAS) to gain access to the network. The NAS is placed at a network bottleneck, such as a wireless access point or router, where it monitors traffic to and from the clients who will be charged for network access. The NAS is typically supported by an authentication and accounting system such as a RADIUS server and collects information about parameters such as throughput of the client, the number of sessions it runs, the duration of its access, or anything else it is able to record. To ensure proper attribution, a tunnel can be placed between the client and the NAS so that each packet from the client is authenticated. Such tunnels are often placed at link layer, but could be placed at virtually any network layer. The client uses its connection through the NAS to visit various sites in the Internet where it finds servers. Clients often secure the link to the server with an additional tunnel, which stretches end-to-end between the client and the server through the NAS. This tunnel provides privacy from, among other things, the NAS itself. As the client makes requests to the server and the server sends its responses through the NAS, the NAS does its accounting. Such a configuration is illustrated in Figure 2.1

The architecture of Figure 2.1 suffers from a gap in its protection of the NAS accounting system. The NAS is able to authenticate all traffic coming from the client and will (typically) drop traffic it receives from any other source on the client-side interface that purports to come from the client but is not authenticated. By contrast, response traffic from the server is unauthenticated by the accounting system. This raises a threat that a node on the Internet could direct false response traffic into the NAS. Since the NAS does not authenticate traffic on the server-side interface, it will typically dispatch this traffic on to the client. The client will probably discard the traffic since it will not match its tunnel to the server, but by the time this traffic reaches the client, it has been attributed to it by the NAS thus compromising the integrity of the accounting database. This constitutes a form of DoS, but instead of being denied service, clients are billed for service that was never used. We refer to this form of attack as a *cramming attack*.

The actual details of the cramming attack depend on the network architecture and details of the accounting mechanism. The seriousness of the threat depends on how response traffic is forwarded to the client by the NAS. For networks that use globally routable network IP addresses and allow arbitrary services to be run on the clients (*i.e.* outside hosts can initiate connections to these services), the cramming attack is easy to perform in the absence of additional firewalling mechanism at the NAS. Firewalling on the client side may still be in use, but as the NAS will not be aware of it, it will forward any packet (and account for it) on to the client host.

If Network Address Translation (NAT) is being used, a cramming attack is more complicated. For purposes of this discussion we only consider the TCP protocol: details for UDP are similar. As NAT is used to share one globally routable address with hosts having private addresses, an incoming packet will only be forwarded if the NAT router determines it to be a part of an existing connection initiated earlier by a client. Connections are identified by a 4-tuple which response packets must use. As the destination IP in the response packets will be that of the NAS address, three remaining values need to be determined by the attacker: the IP address of the server, the server port number, and the client port number. Guessing these values for a particular client's connection is challenging for an off-path attacker (that is, an attacker that is not on the routing path between the NAS and the server).

However, some tricks can be used to make the network vulnerable to cramming attacks against random clients. For networks that support a large number of clients, many users are likely to be connected to relatively popular services on the web such as popular search engines and portals (*e.g.* `google.com`), instant messaging services (*e.g.* AIM), IMAP and POP mail access (*e.g.* `gmail.com`, `yahoo.com`), and so on. Thus, the attacker has a large number of fixed server IP/port pairs to choose from as possible endpoints for different connections a NAT NAS might be tracking. Only the client port information needs to be guessed. Client ports are often chosen from a fixed set of ranges (ephemeral port ranges) whose exact values are dependent on the particular OS and configuration. By using different client port ranges and sending out packets with different client port values picked from probable ranges, it is possible to get response packets past the NAS and hence successfully perform the cramming attack. There are some NAT implementations that make this a very effective approach. For instance, if port numbers are allocated sequentially and there is an insider behind the NAS, then an active port number can be guessed easily. Even if there is no such edge, a brute force attack can achieve some success. On a Pentium 4 running Linux 2.6.10 at 2.4 GHz with 1GB of RAM we were able to send packets with a 1.4Kb payload and varying port values at a rate of around 10,000 packets/second with code that had no optimizations and no changes to the drivers or the kernel. Thus, a brute force attack on the client port numbers can be performed in a small amount of time. Any technique for telling if a cramming packet 'hit' a client would make such an attack quite effective. For instance, in monitoring link-layer communication one might be able to detect if a cramming packet was consumed by a device.

The time window in which the attack is successful depends on the length of the time period for which the NAT router maintains state information for each connection. As this state information is the one that is used to ascertain whether to forward a certain response packet to a client, it is maintained for at least the period of the connection. For connections that only last momentarily (*e.g.* HTTP), it is important that the attack take place in the period when the NAT router still has this state information stored. The NAT standard [113] recommends that the NAT router maintain state for at least another 4 minutes (2 * Maximum Segment Lifetime) after it thinks that the connection has terminated. As a NAT router can never be sure whether the connection tear-down packets it saw on the wire actually reached the destination host, it continues to forward packets for that connection for a little while after the observed tear down (to enable retransmissions). Our experiments with a Linksys wireless router doing NAT showed that connection state was maintained for 7 more minutes after the actual connection was terminated. Thus it was possible to cram packets into this connection after its actual termination for another 7 minutes. These factors contribute to increasing the length of the vulnerability time window and hence give plenty of time to the attacker for a brute force attack.

2.2 Requirements

Having shown that the tunnel configuration most commonly used to protect the wireless accounting infrastructure is vulnerable to cramming attacks, we embark on the design of a tunnel-complex protocol that sets up a tunnel configuration to protect the NAS from cramming attacks. Given that the protection stems from design decisions made about the tunnel composition, we would like these decisions to be guided by some design principles. We now introduce several properties that fulfill this role.

If all egress traffic flowing from inside to outside a node's administrative domain must be be authenticated and authorized, then we say that the node enforces the *egress authenticated traversal* property. If all traffic flowing from outside to inside a node's administrative domain must be authenticated and authorized, then we say that the node enforces the *ingress authenticated traversal* property. If both properties

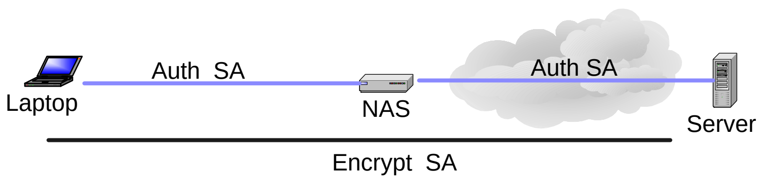

Figure 2.2: Base SA configuration

are satisfied, then we say that the nodes satisfy the *authenticated traversal* property. In the tunnel configuration displayed in Figure 2.1, the NAS satisfies the egress authenticated traversal property, but fails to satisfy ingress authenticated traversal and this is the source of the cramming attack.

In order to satisfy the authenticated traversal property the NAS should require all egress traffic to be authenticated via an association flowing from the client to the NAS and all ingress traffic to be authenticated via an association flowing from the server to the NAS. To preserve end-to-end security an association should encrypt traffic flowing between the client and the NAS. The resulting situation is one where the end-to-end association must tunnel through the two authentication tunnels. This configuration of associations can be seen in Figure 2.2.

2.3 Protocol Overview

Recall the following facts from Chapter 1. Security associations define a collection of cryptographic transforms that are applied to each packet traveling in that association and create a virtual secure channel. Security associations are kept in the SADB at each node. Security mechanisms direct traffic into security associations based on selection criteria such as source and destination address. These filters are maintained in each node's MDB. A tunnel is set up using a tunnel-establishment protocol that establishes symmetric keys used to encrypt and authenticate traffic. In this chapter we develop a tunnel-establishment protocol called Estab that is similar to IKEv2 [76], but with several significant differences. Estab is designed with the intention of being used as a component in a tunnel-complex protocol. This means that it installs entries in the SADB as well as entries in the MDB as directed by the tunnel-complex protocol. In contrast, IKE is typically invoked if a packet matches matches a filter in the MDB and there is no existing association. Another difference is that the Estab protocol accommodates nested tunnels in the following way. If there is an existing entry in the mechanism database for traffic flowing from a to b and the Estab is directed to create a new association with the same MDB entries, then the new association is assumed to be nested inside of the existing association.

Associations are unidirectional so we have to ask if a single execution of our key exchange should establish a single unidirectional association that only protects the flow of information in one direction or should it establish two associations that protect the flow of information in both directions. We considered both possibilities in the course of our research. The first model of Estab that we shall present will set up unidirectional associations while the second model of Estab will set up bidirectional associations.

We must also decide when the SADB and MDB are updated. There are three options that we consider. The first option is to return information exchanged in the protocol and let the databases be modified at a higher level. This breaks modularity of the system and reduces the information that can be known by each side upon termination. Suppose both nodes participating in the key exchange simply updated their databases after they had completed sending messages, then upon termination of the protocol neither party would be sure that an association has been established. The second option is illustrated in Figure 2.3 where the responder writes the information for the $\mathsf{I} \rightarrow \mathsf{R}$ association to the databases before sending the final message. As a consequence, the initiator can be assured that the $\mathsf{I} \rightarrow \mathsf{R}$ association is established when the protocol at the initiator terminates. The third option adds an additional acknowledgment from the initiator to the responder. This message is sent after the initiator has completed all of its database updates. Upon termination of the protocol, the initiator knows that the $\mathsf{I} \rightarrow \mathsf{R}$ association is established and the responder knows that both associations are established. (The initiator does not know whether the responder has completed updating its databases for the $\mathsf{R} \rightarrow \mathsf{I}$ association.) Adding a second acknowledgment from the responder to the initiator results in both parties knowing all security associations are established upon termination. We selected the *second* option, because it provides some knowledge about the state of the databases, and at the same time provides the weakest guarantee of the three acceptable choices with a minimum number of messages. A protocol specification that was correct using this choice would remain correct if the weaker key exchange protocol were replaced by one offering stronger guarantees.

The tunnel configuration shown in Figure 2.1 can be set up by having the client initiate establishment with the NAS and then initiate establishment with the server. Setting up the tunnel configuration displayed in Figure 2.2 is not as easy because the client cannot directly set up the tunnel between the NAS and the server. A solution is to design a tunnel-complex protocol that sets up the desired tunnel complex.

It is assumed that the client knows the identity/address of both the NAS and the server. In Chapter 6 we present protocols that discover the nodes with which they set up tunnels and see that this introduces additional complexities.

Note that the client has a relationship with both the server and the NAS and must authenticate itself with each of these nodes during different stages of the protocol. We have already established that there will be an authentication association between the client and the NAS, an authentication association between the NAS and the server, and an association providing both authentication and encryption between the client and the server. It is clear that the client will authenticate itself to the NAS during the key exchange. Similarly, the client will authenticate to the server as part of a key

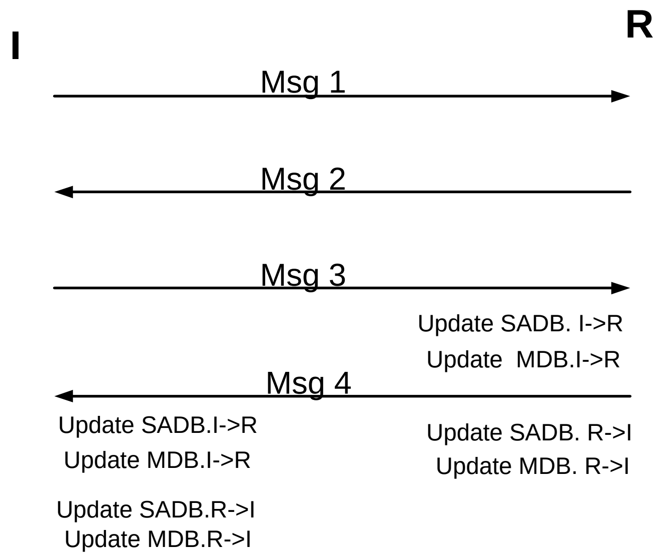

Figure 2.3: Write after message three.

exchange. It is less clear how the NAS will authenticate itself to the server, since it needs to communicate to the server that it is establishing an association on behalf of the client. We resolve this problem by having the client send the NAS a credential that the NAS will present to the server on behalf of the client. If the server verifies that the credential is from a valid user, then it allows the association between the NAS and server to be established.

We can now give an outline of the L3A protocol.

Skeleton of L3A Protocol

Client initiates protocol The client identifies the server that it desires to communicate with and the NAS that will deliver access to the Internet. The client then invokes Estab to establish an association between the client and the NAS. The client must pass to the NAS the credential that the NAS will present to the server as well as the address of the server.

Establish NAS-server SA Upon notification that the client-NAS Estab exchange is complete, the NAS gets the address of the server and the credential. The NAS then invokes Estab to establish an association between the NAS and the server. The NAS presents the credential to the server. If the credential is not valid, then the protocol is terminated.

Establish server-client SA Upon notification that the NAS-server Estab exchange is complete, the server invokes a key exchange with the client.

2.4 Architecture of the Formal Model

Prototypes of the L3A and Estab protocols were constructed early in the design phase. The prototypes were constructed in Maude [32, 34]. Maude supports the specification of complete designs in a modular way. In modeling such a large system, it is important to keep in mind to only model those aspects of the system that are of interest. For example, we only model message forwarding and message delivery in IP. Details such as fragmentation are ignored. Modeling tunnels was necessary since the L3A and Estab protocols can only be judged correct if they correctly set up the desired security associations and mechanisms. The Estab and L3A protocols are built on top of this framework. Being able to construct such complete models is particularly useful when modeling protocols that rely on lower layer protocols since the interaction between the layers may be a source of errors. We now turn our attention towards the architecture of the formal model.

In keeping with good software engineering practices, the design of our formal model is modular and constructed as a hierarchy of abstractions reflecting the structure of the system being modeled. Figure 2.4 shows the components of our model

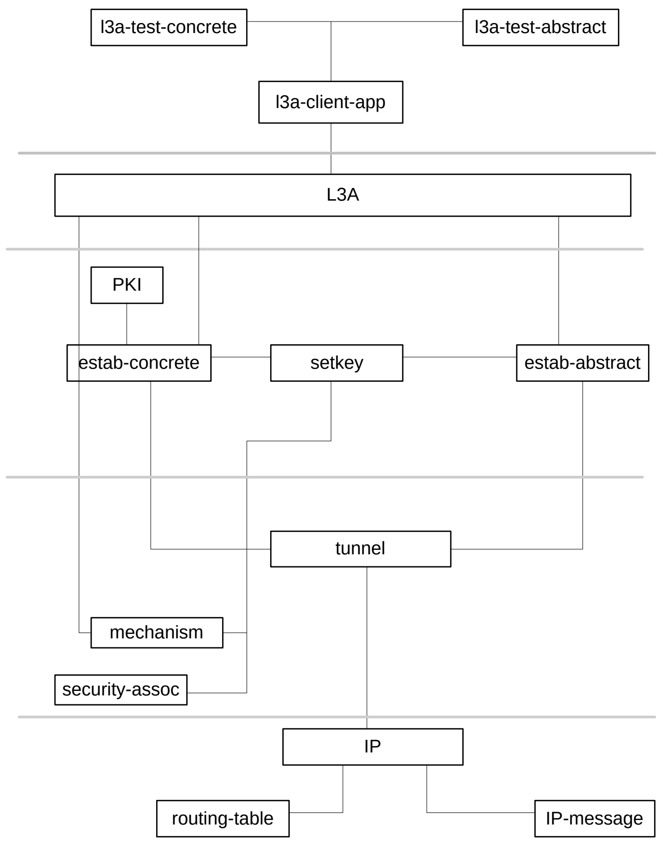

Figure 2.4: Architecture of the model

and how they are related. Note that the lightly shaded horizontal lines separate the different layers of the model hierarchy. (In the discussion that follows, the names of software modules are given in typewriter font.) The lowest layer of our system, `IP`, models the sending and receiving of IP messages. Routing is modeled at this layer. At the next layer the `tunnels` module models network layer security tunnels that behave similar to IPsec. We do not attempt to provide a concrete model of encryption since we are only concerned about ensuring that the proper headers are applied and not with the concrete cryptographic transformations that get applied to the packets once they arrive at their destination. On the other hand, we must model the databases that maintain state of the security associations and mechanisms, because their state defines the security associations and mechanisms. At the level of the key exchange we include a module `setkey` that provides an interface to the databases. This module encapsulates the updating of both the SADB and MDB databases. The `PKI` module provides a limited public key infrastructure. The stub key exchange module `estab-abstract` allows L3A development to be performed in parallel with the development of Estab. The full Estab protocol is modeled by the module `estab-concrete`, which is a refinement of `estab-abstract`. The L3A protocol is formally modeled by the module `L3A`. Observing the diagram we see lines from both `L3A` and `setkey` to the `mechanism` module, which defines the MDB, but only `setkey` has a connection to the `security-assoc` module, which defines the SADB. This is because mechanisms are created and modified by both `estab-concrete` and `L3A`, while the security associations are only modified by `estab-concrete` via the `setkey` module. Constructing different test cases is facilitated by the module `l3a-client-app`. At the top level the `l3a-test-concrete` and `l3a-test-abstract` modules import the other modules and define the system configuration. The abstract and concrete versions differ in that the abstract version imports `estab-abstract`, while the concrete version imports `estab-concrete`. A bonus of the modular nature of our design is that we can reuse portions of the model when designing other protocols.

The module `estab-concrete` is composed of an initiator processes and a responder daemon. Each node in the system must start its responder daemon before the protocol may be executed. The `L3A` module defines three processes—one process to be run at each of the nodes (client, server, NAS). The `L3A` module is invoked by the `l3a-client-app` module at the client, but the processes at the NAS and the server are run as daemons. These daemons wait for notification from the establishment responder daemon. After receiving such a notification, the process executes its portion of the L3A protocol.

To analyze the specification using Maude we need to specify a concrete initial state, which contains information about the nodes (which can act as hosts or security gateways) and an enumeration of all subnets representing the network topology. We use the network topology given in Figure 2.5. Furthermore, the initial state contains, for each node: the network interfaces, the routing table, the trusted certificate authorities, the initial SADB state, and the initial MDB state. All this constitutes

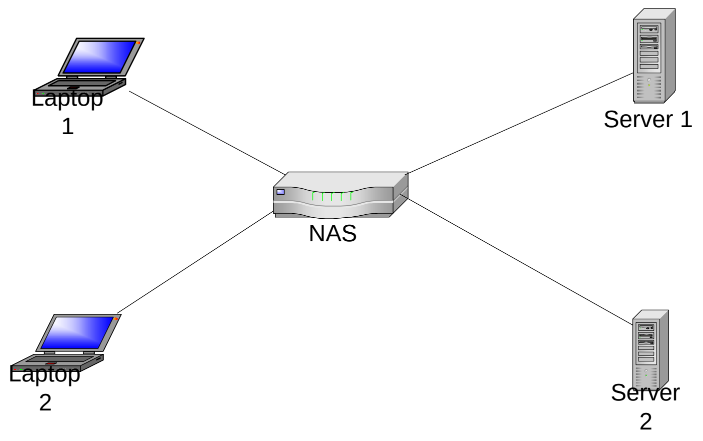

Figure 2.5: Model network topology.

a multiset.

2.5 Formal Simulation

This section records the evolution of the L3A tunnel-complex protocol. Each design presented in this section was implemented in the Maude framework described above. At each stage in the evolution of the protocol's design we describe both the design and the results of running logical simulation.

2.5.1 First Prototype

The L3A tunnel-complex protocol is dependent on the Estab protocol to set up tunnels. Therefore, our first task is to design the Estab protocol based on the design decisions made above. The Estab protocol design is driven by the demands of the higher-level tunnel-complex protocol. Does the L3A protocol demand that its establishment protocol set up a bidirectional pair of security associations or will a single unidirectional association suffice? Recall that the requirements demand a tunnel complex that provide the following protections. To ensure privacy between the client and the server we must have associations performing encryption and authentication going in both directions. To satisfy the requirements there must be a security association flowing from the client to the NAS and a security association flowing from the server to the NAS. The traffic is authenticated by the end-to-end association so there does not appear to be a need to have associations going the opposite direction. Consequently, it seems that it would suffice to have the establishment protocol set up a single association.

The client must pass a credential as well as the address of the server to the NAS and the NAS must then pass this credential to the server. To accommodate this

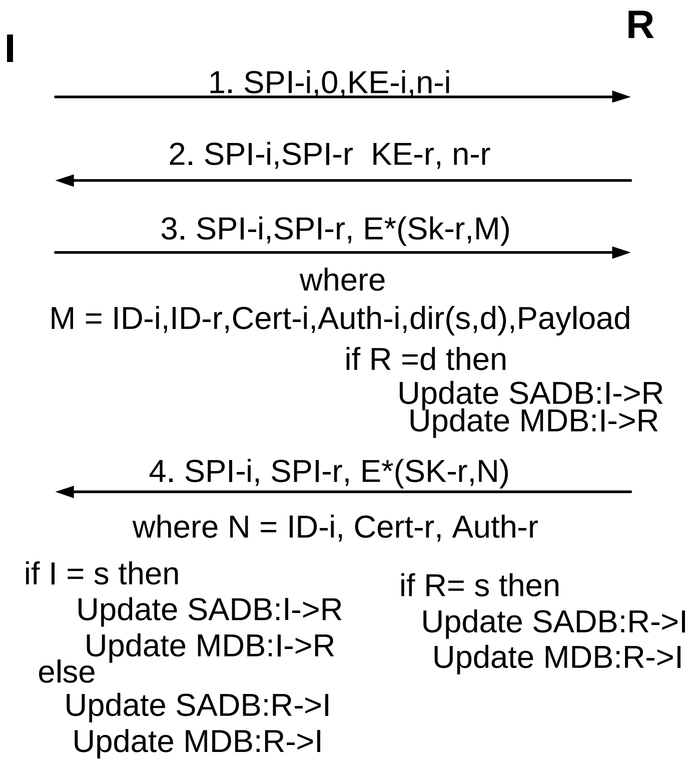

Figure 2.6: Estab V 1.0

data we add a payload field to the third message of the Estab protocol. Since the Estab exchange will create a single association and this association need not flow from initiator to responder, the Estab exchange needs to include a field indicting whether the security association flows from the initiator to responder or visa-versa. Another consequence of establishing only a single association is that each node only needs to update its SADB and MDB once.

The first version of the Estab protocol is given as follows: Let K be a symmetric key. We write $S(K, M)$ for a signature function (such as HMAC) and $E(K, M)$ for an encryption function (such as triple DES). Assuming K_a and K_e are keys for authentication and encryption respectively, we abbreviate:

$$S^*(K_a, M) \overset{\text{def}}{=} M, S(K_a, M)$$
$$E^*((K_a, K_e), M) \overset{\text{def}}{=} S^*(K_a, E(K_e, M)).$$

The Estab protocol is illustrated in Figure 2.6. Details of the protocol description are as follows.

Estab Protocol V 1.0

Initiation The protocol has two principals: an initiator I and a responder R. Principal I generates a nonce n_{I}, a SPI value $\mathrm{SPI}_{\mathsf{I}}$ for the $\mathsf{R} \rightarrow \mathsf{I}$ association, and the Diffie-Hellman value KE_{I}. The initiator then sends the message

Msg 1 $\mathsf{I} \rightarrow \mathsf{R} : \mathrm{SPI}_{\mathsf{I}}, 0, \mathrm{KE}_{\mathsf{I}}, n_{\mathsf{I}}$

If R gets a message of this form, it generates a SPI value $\mathrm{SPI}_{\mathsf{R}}$ for the $\mathsf{I} \rightarrow \mathsf{R}$ association, a nonce n_{R}, and a Diffie-Hellman value KE_{R}. The responder then sends the message

Msg 2 $\mathsf{R} \rightarrow \mathsf{I} : \mathrm{SPI}_{\mathsf{I}}, \mathrm{SPI}_{\mathsf{R}}, \mathrm{KE}_{\mathsf{R}}, n_{\mathsf{R}}$

Generate Keys Both sides can now generate SKEYSEED from which all the cryptographic keys for the resulting SAs are derived. Separate keys for authentication and encryption are computed for both directions. These keys are known as SK_e^{I}, SK_a^{I}, SK_e^{R}, and SK_a^{R} for the encryption and authentication of the initiator and responder tunnels. When missing the subscript, SK^{I} and SK^{R} denote a pair of authentication and encryption keys.

Notice that we generate two key pairs, one for each direction, even though the protocol only sets up a unidirectional tunnel. Given that the direction of the association is not known at this point of the protocol execution, we generate a pair for each possible direction and discard the unused pair once the direction is known. This design decision was, in part, to follow standard protocols such as IKE and, in part, to accommodate future modifications to the protocol.

The initiator now fetches its certificate Γ_{I}. It also generates

$$\mathrm{Auth}_{\mathsf{I}} = S(SK_a^{\mathsf{I}}, (\mathrm{Msg1}, n_{\mathsf{R}}, S(SK_a^{\mathsf{I}}, \mathrm{ID}_{\mathsf{I}}))),$$

where ID_{I} is the identity of the initiator. This proves the initiator's knowledge of the secret corresponding to ID_{I} and integrity protects the contents of the first message. The mechanism selectors for the resulting SAs are also generated. The initiator then forms a pair (s,d) indicating the source s and destination d of the SA and sends the message

Msg 3 $\mathsf{I} \rightarrow \mathsf{R} : \mathrm{SPI}_{\mathsf{I}}, \mathrm{SPI}_{\mathsf{R}}, E^*(SK^{\mathsf{R}}, M)$

where $M = (\mathrm{ID}_{\mathsf{I}}, \mathrm{ID}_{\mathsf{R}}, \Gamma_{\mathsf{I}}, \mathrm{Auth}_{\mathsf{I}}, \mathrm{dir}(s,d), \mathit{payload}))$. Upon receiving a message of this form, the responder checks the signature of the message, decrypts the message, and verifies $\mathrm{Auth}_{\mathsf{I}}$. If R is the destination of the association, then the MDB and SADB are updated.

The responder then fetches its certificate Γ_{R}. It also generates

$$\mathrm{Auth}_{\mathsf{R}} = S(SK_a^{\mathsf{R}}, (\mathrm{Msg2}, n_{\mathsf{I}}, S(SK_a^{\mathsf{R}}, \mathrm{ID}_{\mathsf{R}}))),$$

where ID_{R} is the identity of the responder. The responder then sends the message

Msg 4 $\mathsf{R} \rightarrow \mathsf{I} : \mathrm{SPI}_{\mathsf{I}}, \mathrm{SPI}_{\mathsf{R}}, E^*(SK_{\mathsf{I}}, (\mathrm{ID}_{\mathsf{I}}, \Gamma_{\mathsf{R}}, \mathrm{Auth}_{\mathsf{R}}))$

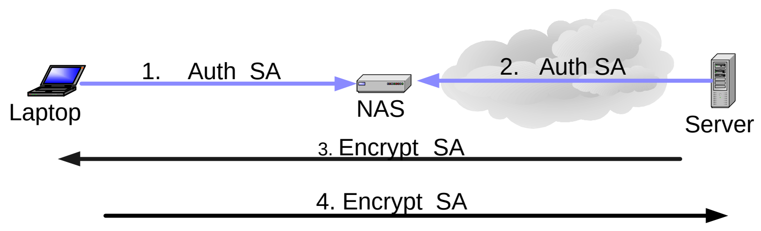

Figure 2.7: L3A V 1.0

If R is the source of the association, then the SADB and MDB databases are updated for the $\mathsf{R} \to \mathsf{I}$ association.

If I receives a message of this form, it checks the signature, decrypts the message, and verifies $\mathrm{Auth}_{\mathsf{R}}$. The MDB and SADB are updated according to the direction of the SA.

Having designed the establishment component, we now refine the basic skeleton of the L3A tunnel-complex protocol given above into the first version of the protocol. The L3A tunnel-complex protocol is invoked at a client. The L3A protocol invokes Estab at the client to set up the $\mathsf{C} \to \mathsf{NAS}$ tunnel. The L3A protocol invokes Estab at the NAS set up the $\mathsf{S} \to \mathsf{NAS}$ tunnel. The protocol invokes Estab at the server and client to create tunnels $\mathsf{S} \to \mathsf{C}$ and $\mathsf{C} \to \mathsf{S}$ respectively. The direction of the association being set up as well as the payload are passed as parameters from the tunnel-complex protocol to the establishment protocol.

Version one of the L3A protocol follows and is illustrated by Figure 2.7.

L3A Protocol V 1.0

Client initiates protocol The client C identifies the server S that it desires to communicate with and the NAS that will provide access to the Internet. The client then invokes Estab to establish an association from the client to the NAS with the parameters: $\mathit{payload} = \mathit{cred}, \mathsf{S}$ and $\mathit{dir} = \mathrm{dir}(\mathsf{C}, \mathsf{NAS})$. When the key exchange at the client has terminated it updates its MDB with a filter saying that all traffic from the client to the server should flow through the $\mathsf{C} \to \mathsf{NAS}$ association.

Establish Server-to-NAS SA Upon notification that a key exchange with the client is complete, the NAS extracts the address of the server and the credential. It then updates its MDB with an entry saying that all traffic from the client to the server should flow through the $\mathsf{C} \to \mathsf{NAS}$ SA. The NAS then invokes Estab to establish an SA from the NAS to the server. The two parameters to Estab are: $\mathit{payload} = \mathit{cred}$ and $\mathit{dir} = \mathrm{dir}(\mathsf{S}, \mathsf{NAS})$. Upon termination of the key exchange, the NAS updates its mechanism database to reflect the fact that all traffic flowing from the server to the client should travel in the $\mathsf{S} \to \mathsf{NAS}$ association.

Establish Server-to-Client SA Upon notification that a key exchange with the NAS has occurred, the server extracts the credential that had been passed by the client. If the credential is valid, the server updates its MDB with an entry saying that all traffic flowing from the server to the client should travel in the $\mathsf{S} \rightarrow \mathsf{NAS}$ association. The server then invokes Estab to set up an association from the server to the client. The two parameters to Estab are: $payload = \mathsf{empty}$ and $dir = \mathrm{dir}(\mathsf{S}, \mathsf{C})$.

Establish Client-to-Server SA Upon notification that the Estab exchange with the server has occurred, the client invokes Estab to create an association from the client to the server. The parameters to Estab are: $payload = \mathsf{empty}$ and $dir = \mathrm{dir}(\mathsf{C}, \mathsf{S})$.

We now look at each of the solutions produced by logical simulation of the protocol in some detail.

- Solution 1.

  The NAS SADB has no entry for the association $\mathsf{S} \rightarrow \mathsf{NAS}$ while the server SADB does have an association $\mathsf{S} \rightarrow \mathsf{NAS}$ entry. This means that a message sent from the server to the NAS would have a header applied at the server, but there is there no corresponding entry in the NAS SADB; so the message will be dropped. We refer to the phenomena where packets get dropped because the state for a tunnel is only installed at one of the two nodes of the pair-wise tunnel as a 'partially set up tunnel'. In this case, the fourth message of the $\mathsf{S} \rightarrow \mathsf{NAS}$ Estab exchange gets caught in the partially set up tunnel $\mathsf{S} \rightarrow \mathsf{NAS}$. The first message of the $\mathsf{S} \rightarrow \mathsf{C}$ Estab exchange also gets caught in the partially set up $\mathsf{S} \rightarrow \mathsf{NAS}$ association.

- Solution 2.

  The first message of the $\mathsf{S} \rightarrow \mathsf{C}$ Estab exchange gets caught in a partially set up $\mathsf{S} \rightarrow \mathsf{NAS}$ association. Both the mechanism databases and the association databases look correct, but further analysis shows that this was misleading. There is a concurrency problem. The SADB on the NAS was not updated until after the packet arrived. Consequently, that packet arrived in a partially set up tunnel as was dropped.

- Solution 3. At each node the entries in the SADB and MDB correspond to the associations and mechanisms given in the requirements. For example, at the client there are entries in the SADB for the $\mathsf{C} \rightarrow \mathsf{NAS}$, $\mathsf{C} \rightarrow \mathsf{S}$, and $\mathsf{S} \rightarrow \mathsf{C}$ associations. There are filters that say that all traffic flowing from the client to the NAS travels in the $\mathsf{C} \rightarrow \mathsf{NAS}$ association; all traffic flowing from the client to the server travels in the $\mathsf{C} \rightarrow \mathsf{S}$ association, which is tunneled through the $\mathsf{C} \rightarrow \mathsf{NAS}$ association; all traffic flowing from the server to the client travels in the $\mathsf{S} \rightarrow \mathsf{C}$ association. The server and the NAS have similar SADB and MDB entries. Since the SADB and MDB at each node have the correct entries as dictated by the requirements, we consider the solution to be correct.

In the next section we shall revise the protocol in an attempt to correct the problems uncovered by the exhaustive search.

2.5.2 Revising the Prototype

The first version of our protocol exhibited several errors. The errors uncovered in formal analysis seemed to (at least partially) stem from the fact that the server has updated its SADB/MDB before the NAS. The first solution produced by exhaustive search seemed to us rather vexing. The fourth Estab message gets caught in a partially set up tunnel.

Analysis revealed that the problem was not caused by the protocol itself, but by the way we modeled communication. It turned out that our model of IP semantics was too weak. The actual semantics for sending a message is that the send function does not return until the message is completely processed by IP. We checked the code! Our model allowed the sender of an IP message to continue processing as soon as the send call was made while the message was processed concurrently by IP. Consequently, the additional concurrency in our model resulted in cases where the association is set up on the responder side before message 4 is actually out of the door. As a result of this behavior, the association is applied to the outgoing message, but there is no entry in the association initiator's SADB. We corrected our models of IP to match the actual semantics. Given that the previous version of the protocol had been tested with an unrealistic model for IP, it seems reasonable to limit our corrections to this problem to see if fixing the model solved all of our problems.

Performing an exhaustive search of the state space yields four solutions. It may seem like things are now worse since there are more solutions than the first version of the protocol. The first three solutions are all incorrect because the first message of the $\mathsf{S} \rightarrow \mathsf{C}$ Estab exchange gets caught in a partially set up $\mathsf{S} \rightarrow \mathsf{NAS}$ association. The problem is due to the fact that the $\mathsf{S} \rightarrow \mathsf{NAS}$ is set up on the server side and the server goes ahead with the $\mathsf{S} \rightarrow \mathsf{C}$ Estab exchange before the NAS has set up the association on its side. The fourth solution is correct.

2.5.3 Second Revision

The problem with the previous revision to the protocol seems due to the fact that the server can finish processing setting up the $\mathsf{S} \rightarrow \mathsf{NAS}$ and begin the $\mathsf{S} \rightarrow \mathsf{C}$ Estab exchange before the NAS has set up the $\mathsf{S} \rightarrow \mathsf{NAS}$ association on its side. As a consequence, the first message of the $\mathsf{S} \rightarrow \mathsf{C}$ key exchange gets caught in a partially set up tunnel. The obvious solution to this problem is to force the server to wait until the NAS has completed its processing. We add a single message that the NAS sends to the server when it has completed its half of the Estab protocol. The server

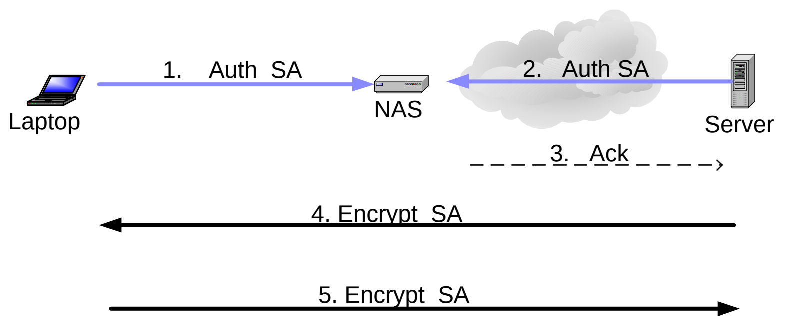

Figure 2.8: L3A V 2.0

waits to receive this message before it continues with any processing. The revised protocol is illustrated in Figure 2.8. Performing a search of the state space now yields only the correct solution.

2.5.4 Further Evolution

Further refinements of the protocol can be made. The acknowledgment sent from the NAS to the server eliminated undesirable concurrent interleavings of the system, but the message does not travel in an authenticated DoS resistant security association. This violates one of the primary requirements of the protocol. Furthermore, it was also pointed out to us that the NAS would need to send maintenance traffic to the client and server. This traffic would need to travel in authenticated security associations as well. As a result of these requirements we revise our design. It now seems obvious that the key exchange should establish a pair of security associations with one going in each direction. This would bring our protocol in line with protocols such as IKE.

As we modify Estab and L3A we continue running logical simulations on a variety of scenarios. We find that the client could finish setting up the $\mathsf{C} \leftrightarrow \mathsf{S}$ association before the server. One possibility is to ignore this fact and assume that an L3A user would just back off and retry later if they discovered that the protocol was not yet set up. Given our previous problems with interleaving concurrency, we choose to add a Fin message that would let the client know when the protocol had terminated. For consistency and uniformity we add an acknowledgment from the client to the NAS after the client had finished the $\mathsf{C} \leftrightarrow \mathsf{NAS}$ key exchange. This allows us to remove the payload field from Estab and include that information as part of the acknowledgments. The direction field is no longer needed since each Estab invocation establishes a pair of associations. The new version of Estab is illustrated in Figure 2.9.

In the revised L3A design, the client begins by performing a key exchange with the NAS. After the Estab exchange between the client and the NAS has terminated, the

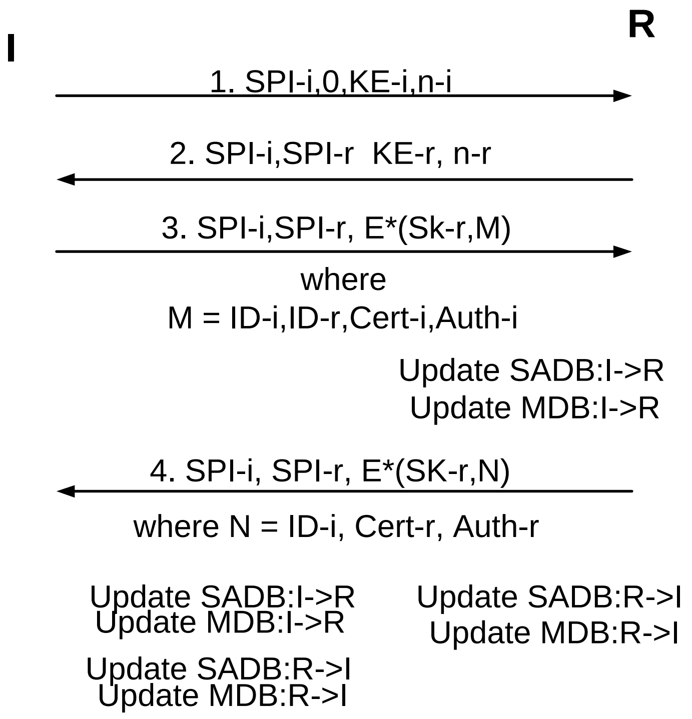

Figure 2.9: Estab V 2.0

client sends a Req message to the NAS containing the credential and server address. The NAS does not start the key exchange with the server until after receiving this message. Once the key exchange between the NAS and the server is complete, the NAS sends the server an Ack message with the credential. If the credential is valid, the server initiates a key exchange with the client to establish the two associations ($\mathsf{S} \rightarrow \mathsf{C}$ and $\mathsf{C} \rightarrow \mathsf{S}$). Upon termination of this key exchange, the server sends a Fin message to the client indicating that the protocol has terminated. The modified protocol is illustrated in Figure 2.10.

The L3A protocol design can be modified to accommodate reuse of security associations. For example, suppose several clients gain access to the network via the same NAS and all go to the same server. The $\mathsf{NAS} \leftrightarrow \mathsf{S}$ associations can be shared among the clients. Similarly, the $\mathsf{C} \leftrightarrow \mathsf{NAS}$ can be reused when a client connects to multiple servers. At each node, processing for establishing security associations is now structured as a set of conditional blocks with each block handling a different case. For example, the NAS has to consider cases where its first action of the protocol is processing a Req message, because the client is reusing an association. Alternatively, the NAS first action could be the notification that a key exchange with the client has occurred.

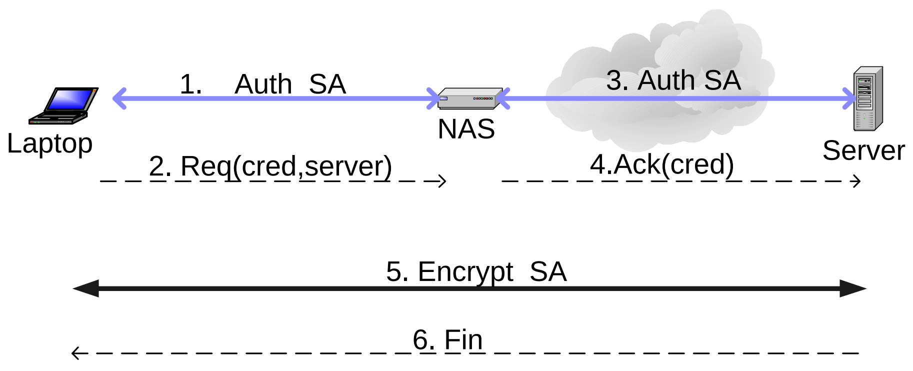

Figure 2.10: L3A V 3.0

2.6 L3A Tear Down

The L3A protocol installs state at the client, NAS, and server as it sets up the tunnel complex. When communication between the client and the server ends, the system should tear down the tunnels and clean up any state from the unused connection to prevent resources from being exhausted. Given that tunnels may be shared among different connections, care must be taken to ensure that tunnels that are used by other connections are not also torn down. In this section, we develop a *tear-down protocol* that tears down a tunnel complex that had been set up by L3A. An alternative solution, would have been to have each node autonomously tear down used tunnels based on timeouts. The design decision to engineer a protocol to perform this task was primarily driven by a desire to better understand tunnel-complex protocols and the realization that we needed such a protocol to perform the latency experiments reported in the next section. We do not consider tear-downs in the remaining of dissertation.

There are two primary requirements that the protocol must satisfy. The first requirement is that the tear-down protocol preserves tunnels shared with other connections. The second requirement is that all commands to delete a tunnel arrive in the tunnel being deleted. As with L3A, the design of the tear-down protocol was prototyped and debugged in Maude.

The steps of the protocol are illustrated in Figure 2.11. Note that each association has a label a - f. In step 1 the client initiates the protocol sending a message to the server to delete the association flowing from the client to the server (labeled e). The server removes this and sends a message to the client to remove both associations flowing between the client and server (labeled e and f). When this message has been sent the server removes association f. The client then sends the NAS a request to tear-down the associations between the NAS and the server (step 3). The NAS then initiates the tear down of associations e and f (steps 4 and 5). When these associations have been removed the NAS sends an acknowledgement to the client (step 6) and the client initiates a tear down of the associations flowing between the

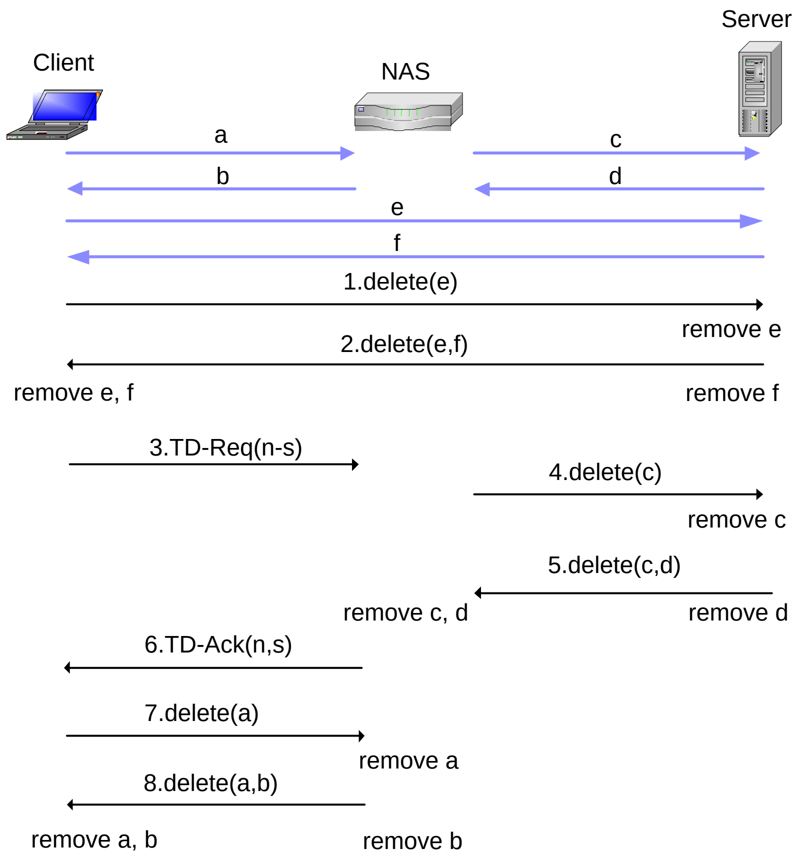

Figure 2.11: L3A Tear Down

client and the NAS (steps 7 and 8).

A sketch of the protocol follows. We label the associations a-f as in the figure. In practice, these identifiers would be the SPIs for each association. The protocol works as follows.

Tear Down Protocol V 1.0

Client initiates protocol The client initiates the protocol by sending to the server

Msg1 $\mathsf{C} \rightarrow \mathsf{S} : \mathsf{delete}(e)$ Upon receiving a message of this form, the server deletes the association and mechanism for e and sends the message

Msg2 $\mathsf{S} \rightarrow \mathsf{C} : \mathsf{delete}(e, f)$ and removes the association and mechanism for f. Upon receiving a message of this form, the client deletes the e and f associations as well as their mechanisms. The client then sends the message

Msg3 $\mathsf{C} \rightarrow \mathsf{NAS} : \mathrm{TDReq}(\mathsf{NAS}, \mathsf{S})$ Upon receiving a message of this form, the NAS sends a message

Msg4 $\mathsf{NAS} \rightarrow \mathsf{S} : \mathsf{delete}(\mathsf{C}, c)$ Upon receiving a message of this form, the server removes the mechanism for all traffic flowing from $\mathsf{S} \rightarrow \mathsf{C}$. If there are no MDB entries for other clients, the association is deleted. The server sends the message

Msg5 $\mathsf{S} \rightarrow \mathsf{NAS} : \mathsf{delete}(\mathsf{C}, \mathsf{S}, c, d)$ If the c association was removed, the d association is now removed. Upon receiving this message the NAS removes the mechanism for $\mathsf{S} \rightarrow \mathsf{C}$ and removes c and d if the tunnel is not in use by another client. The NAS then forms the following message

Msg6 $\mathsf{NAS} \rightarrow \mathsf{C} : \mathrm{TDAck}(n, s)$ Upon receiving a message of this form, the client checks if there are sessions with other servers and if not, then the client forms

Msg7 $\mathsf{C} \rightarrow \mathsf{NAS} : \mathsf{delete}(a)$ Upon receiving a message of this form, the NAS deletes the association and mechanism for e and sends the message

Msg8 $\mathsf{NAS} \rightarrow \mathsf{C} : \mathsf{delete}(a, b)$ and removes the association and mechanism for b. Upon receiving a message of this form, the client removes the a and b associations as well as their MDB entries.

The command tear-down request might seem like an obvious vulnerability. The first line of defense, is that all tear-down commands must arrive in a tunnel and are thus authorized and authenticated. It remains to demonstrate that a principal acting as the endpoint of a tunnel cannot tear down a tunnel connecting two other principals. Suppose Alice has set up a tunnel complex to communicate with Bob and Ted is connected to Carol through the same NAS, and Ted launches a DoS attack against the Alice-Bob connection by sending a command to the NAS to tear-down the connection between the NAS and Alice. If Ted attempts to spoof Alice's address the authentication tunnel will reject the message at the NAS. Otherwise, the message arrives with a header saying it is from Ted and the NAS sends Bob a message to remove the mechanism for the Ted to Bob connection, yet since there is no such mechanism entry the command is ignored. If there were such a connection, then the NAS-Bob association would not be removed since Alice still has a connection to that node.

2.7 Implementation

In order to demonstrate the practicality of tunnel complex protocols, an implementation of L3A and Estab was built. The implementation was written in C and structured as a collection of processes running on the client, NAS, and server. The cryptographic operations of L3A are performed using the standard OpenSSL library. Our implementation uses IPsec tunnels that are set up with our Estab protocol. Updates to the kernel's SADB made by L3A are communicated through the PF_KEY Key Management API, as described in RFC 2367 [88]. Interestingly, this RFC does not completely describe the manipulation of the MDB, so a trial and error approach was required in order to interface with L3A correctly.

We implemented the three principals of L3A (client, NAS, server) on three different machines, each running FreeBSD 4.8 and connected with a megabit/second network link. In our testbed, the client and server both are Micron 600MHz Pentium IIIs with 128MB of memory, and the NAS is a Dell 1.3 GHz Pentium IV with 256MB of memory.

The experiments run on our testbed were designed to give performance results of the L3A protocol that can be compared to the performance of other solutions to the accounting problem. The first set of tests measures the raw throughput of client-server communication in 4 cases. The first case, Base, is a baseline, with no IPsec at all, in order to quantify the maximum possible throughput of the connection. The second, End-to-end, utilizes IPsec end-to-end encryption and authentication, without accounting guarantees. The third case, labeled as Typical, is the configuration of tunnels seen in most current accounting systems. The client maintains an encrypted tunnel with both the NAS and the server. The final case, L3A, uses the tunnels set up by L3A. This entails three tunnels rather than the two used in the Typical case, but encryption is performed only end-to-end. The second set of tests measures the latency of the tunnel set up. Three cases are considered in these tests. In the simplest case, End-to-end, just the client-server tunnel is set up. The next case, L3A w/ Reuse, reflects the common case, where a client-NAS tunnel already exists, and the remaining NAS-server and client-server tunnels are created and torn down by L3A, leaving the client-NAS tunnel for another L3A session. The final case, L3A w/o Reuse, describes the latency of L3A when the client-NAS tunnel is not reused, and all three tunnels are created and torn down by L3A.

The results of the throughput tests appear in Figure 2.12. The Base case where

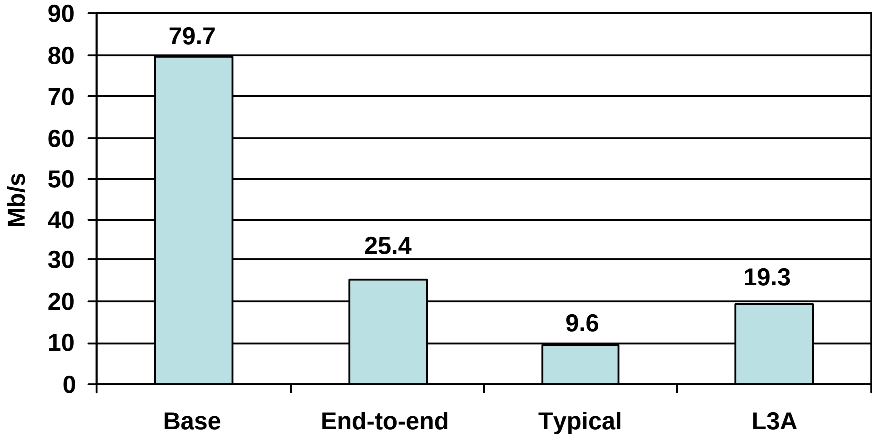

Figure 2.12: Throughputs

data is sent in the clear has the best performance as expected. In the End-to-end case, encryption has a significant impact on throughput, reducing it to barely a third of the unencrypted rate. This is particularly pronounced in the third case, Typical, where double encryption degrades performance to about a third of the End-to-end case because of the double encryption burden it places on the client. In the case of L3A only the end-to-end tunnel performs encryption and the other two perform authentication. The graph reflects the fact that the cost of an authenticated tunnels is much less than that of a tunnel that performs encryption. Consequently, the throughput performance of L3A is 101% better than that of the Typical configuration and only 32% lower than the case with no accounting. We did not measure the case where there are large numbers of clients, but in our tests with one client, the NAS was only lightly loaded.

The results from the latency tests appear in Figure 2.13. For each scenario, the data reflects the time taken to both set up and tear down all appropriate tunnels. The latency cost of establishing tunnels for accounting is 142% greater than that of end-to-end protection alone, but in the most common case, when there is already a tunnel between the client and NAS, it will be only 48% longer.

Our experiments show that L3A accounting costs about 160ms for both set up and tear down. This is about 2.4 times more than the same operations for an IPsec tunnel alone. However, tunnel reuse in L3A reduces this to a factor of 1.5 in the common case where the client-to-NAS tunnel already exists. On the other hand, L3A improves bulk traffic performance by 100% over a naive (but typical) approach to accounting where accounting uses an encrypted tunnel to the NAS. From these results we conclude that tunnel complex protocols can be efficiently implemented using the existing IP security infrastructure.

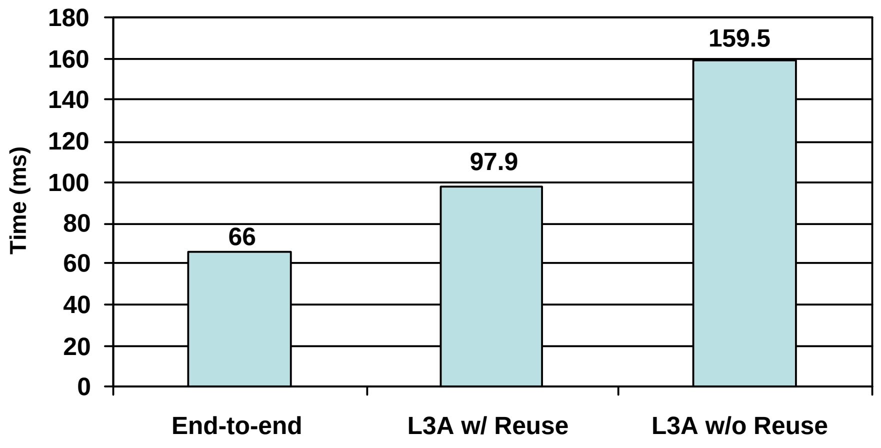

Figure 2.13: Latencies

2.8 Conclusion

In this chapter we introduced a form of DoS attack against the wireless accounting infrastructure that we call cramming attacks and argue that it is possible for an attacker to effectively succeed using such an attack. We presented the design and implementation of the L3A protocol that sets up a tunnel complex that thwarts cramming attacks as well as a corresponding tear-down protocol. During the course of this investigation we found that nontrivial functional correctness issues were the predominant concern. In particular, deadlocks that arose from unexpected interactions between the protocol execution and the state being installed at the nodes. Although logical simulations were quite helpful in debugging the protocols, it would also be useful to formulate and prove various functional correctness properties as well as reason about availability, yet we found existing formalisms lacking when it came to reasoning about the interaction of state at a node and an executing protocol. This observation lead to the research that is presented in the next several chapters.

Chapter 3

Modeling Tunnels

This chapter introduces an abstract model of tunnels. The proposed model includes the fundamental structures that define the state of a tunnel as well as the header processing performed by tunnels. On the other hand, the details of cryptographic operations performed by the tunnels are elided in order to focus on functional correctness and DoS properties. We sketch the design of a tunnel-establishment protocol intended for use as a component in tunnel-complex protocols operating in an environment where authenticated traversal is enforced. We demonstrate the utility of our approach to formalizing tunnels by exhibiting a deadlock that can arise when the establishment protocol is used in a peer-to-peer fashion. After considering several possible techniques for avoiding the deadlock, we introduce the notion of a 'session' distinguished by a unique identifier and explain how our model of tunnels can be modified to accommodate this concept and argue that establishment deadlock is avoided in the resulting system. The discussion is kept rather informal throughout this chapter in order to keep the focus on key concepts; a formal calculus incorporating these ideas is presented in the next chapter.

3.1 Abstract Foundations

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.The behavior of the constructor and destructor pair constitute the essence of a security association. 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 a tunnel-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. Although this level of abstraction is sufficient to model the basic concept of a tunnel, a somewhat more concrete refinement is needed for our purposes.

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 payload. 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. A security association is defined to be a constructor and destructor pair. Each association has a security parameter index ι 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 defines the associations active at a node and is modeled as a set of constructors and destructors that get applied at that 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.

An association flowing from node a to node b with identifier ι is characterized by a pair composed of a constructor $\mathsf{Out}(b, \iota)$ residing in the association database at node a and a destructor $\mathsf{In}(a, \iota)$ residing in the association database at b. To convey the behavior of these operators we shall informally describe the operation of constructors and destructors as follows. The constructor $\mathsf{Out}(b, \iota)$ applied to the packet $\mathsf{P}(c, d, y) @ a$ will create a secure packet as given in the equation

$$\mathsf{Out}(b, \iota)(\mathsf{P}(c, d, y)) \quad = \quad \mathsf{P}(a, b, \mathsf{S}(\iota, \mathsf{P}(c, d, y))),$$

where all free variables are assumed to be universally quantified. The destructor $\mathsf{In}(a, \iota)$ applied to a packet created by the application of the constructor $\mathsf{Out}(b, \iota)$ will remove the secure header yielding the original packet. This means that the destructor can be viewed as being the inverse of the constructor so

$$\mathsf{In}(a, \iota)(\mathsf{Out}(b, \iota)(p)) = p,$$

where all free variables are assumed to be universally quantified.

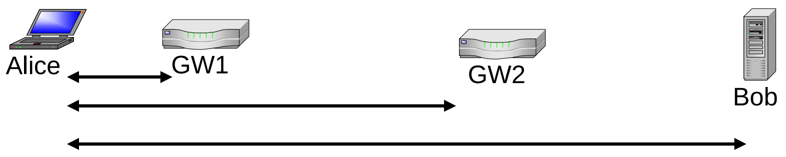

Figure 3.1: Nested Tunnels 1

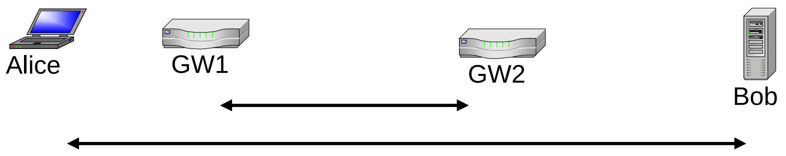

Figure 3.2: Nested Tunnels 2

Tunnels may be nested inside of one another as illustrated in Figures 3.1 and 3.2. A packet at Alice traveling to Bob in the tunnel configuration depicted in Figure 3.1 will first have the constructor for the Alice-Bob association applied. The resulting packet will then have the constructor for the Alice-GW2 association applied, and finally, the constructor for the Alice-GW1 constructor is applied. The entry in the mechanism database at Alice would be

$$A \longrightarrow B : [\mathsf{Out}(B, \iota_1), \mathsf{Out}(GW2, \iota_2), \mathsf{Out}(GW1, \iota_3)],$$

where ι_1, ι_2, and ι_3 are valid SPI values for the associations in question. Applying this bundle to the packet $\mathsf{P}(A, B, y)$ yields

$$\mathsf{P}(A, GW1, \mathsf{S}(\iota_3, \mathsf{P}(A, GW2, \mathsf{S}(\iota_2, \mathsf{P}(A, B, \mathsf{S}(\iota_1, \mathsf{P}(A, B, y))))))).$$

Applying the destructor at $GW1$ strips off the outer header yielding

$$\mathsf{P}(A, GW2, \mathsf{S}(\iota_2, \mathsf{P}(A, B, \mathsf{S}(\iota_1, \mathsf{P}(A, B, y))))),$$

and applying the destructor at GW2 yields

$$\mathsf{P}(A, B, \mathsf{S}(\iota_1, \mathsf{P}(A, B, y))),$$

and applying the destructor at Bob yields the original packet $\mathsf{P}(A, B, y)$. A packet traveling in the tunnel configuration depicted in Figure 3.2 will first have a constructor for the Alice-Bob association applied at Alice. When the packet arrives at GW1 a constructor is applied for the GW1-GW2 association. So the Alice-Bob tunnel is nested inside of the GW1-GW2 tunnel. At GW2 a destructor is applied yielding

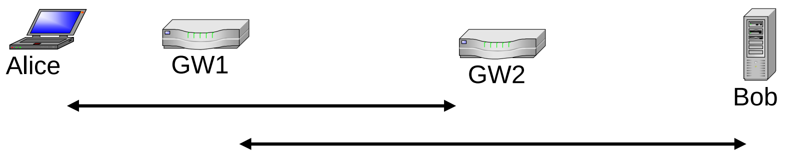

Figure 3.3: Nested Tunnels 3

a packet traveling only in the Alice-Bob association. Finally, a destructor applied at Bob for the inner tunnel yields the original packet. Figure 3.3 illustrates an ill-formed tunnel since neither tunnel completely encompasses the other. To see why this is undesirable notice that a packet traveling in this tunnel complex would enter the Alice-GW2 tunnel and then at GW1 would enter the GW1-Bob tunnel so the packet arrives at GW2 and is forwarded to Bob where the header for the GW2-Bob tunnel is removed, but the resulting packet is destined for GW2 and the packet is dropped because the destination is not Bob.

3.2 Establishment for Discovery

Tunnel establishment has the following components: the authorization and authentication of the tunnel at both nodes, the creation of the association, the updating of the association and mechanism databases, and the establishment of shared cryptographic keys by way of a key exchange protocol. The focus of our model is on the first three components, and given that our formalism 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 a constructor X.

Our establishment protocol is assumed to be triggered by a tunnel-complex protocol. In particular, a discovery protocol that will discover gateways on the dataflow path and set up tunnels to negotiate their traversal. We have seen in Chapter 2 that enforcing the ingress authenticated traversal property can protect networks from the unexpected consequences of allowing unauthenticated/unauthorized traffic to traverse a gateway and enter a seemingly protected network. This seems to comply with the general notion that network administrators should control what traffic they allow on their networks. Enforcing the egress authenticated traversal property helps prevent extrusion attacks. Hence our design should be compatible with the authenticated traversal property. Since the protocol is intended to be used as a discovery protocol component, we require that the establishment protocol deliver credentials needed to negotiate the traversal of the gateway. In addition, the newly discovered gateway is not trusted, so discovery protocols should deliver credentials that prove that a newly discovered gateway belongs to a trusted administrative entity.

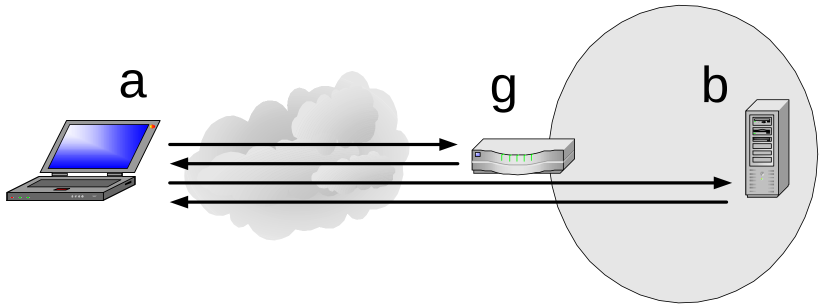

Figure 3.4: Road Warrior

Let us illustrate the basic ideas with the road warrior scenario shown in Figure 3.4, 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 Θ enforced by g. So Alice must present a credential Ξ to the gateway in order to demonstrate that she satisfies the policy. The gateway must also present its credentials Ξ' 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 SADB and MDB at both a and g. Although we have elided the cryptographic operations performed by the tunnels, it is necessary to include digital signatures in the establishment messages for when we analyze DoS threats in Chapter 7. 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 Ξ, and the filter selectors a and b. This message is formally expressed as a term $\mathsf{P}(a, g, \mathsf{X}(\mathsf{Req}(a, b, \iota_a, \Xi, \mathit{Sig})))$, where Sig is a digital signature.

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

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 $\Sigma \cup \mathsf{In}(a, \iota_g)$. A packet filter is added to the responder's inbound mechanism database Π^i indicating 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 Ξ'. This message is formally expressed as a term $\mathsf{P}(g, a, \mathsf{X}(\mathsf{Rep}(a, b, \iota_a, \iota_g, \Xi', \mathit{Sig}')))$, where Sig' is a digital signature.

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

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

Rep Received: Upon receiving the reply message, the initiator verifies the signature and calls upon an oracle to verify that Ξ' satisfies its policy Φ 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 decapsulated packet $\mathsf{P}(a, b, y)$ is sent on towards the server.

In the above model, the responder will verify the signature on any request message it receives and the initiator will verify the signature on any reply message purported to come from the responder. The verification of digital signatures and credentials are relatively costly operations that can be exploited by an adversary executing a denial-of-service attack. For instance, an attacker could simply send many bogus response messages to the responder processes, forcing the node to perform costly operations denying resources to legitimate users. This issue will be addressed in some depth in Chapter 7.

3.3 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.

	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 3.5: Deadlock Scenario

Table 3.5 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 and signatures in the messages are not included 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 the deadlock does not reoccur. The overhead and complexity of this solution might be acceptable if the problem is 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 [77] 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 *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 v into association ι flowing from a to b is written as $s \longrightarrow d : v : [\mathsf{Out}(b, \iota)]$. 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 $\mathsf{P}(a, b, \mathsf{S}(v, \iota, \mathsf{P}(s, d, y)))$, where the secure header identifies both the session v 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 v and Bob initiates the establishment protocol for session u. The first message sent by Alice is represented by the term $\mathsf{P}(a, b, \mathsf{X}(\mathrm{Req}(a, b, v, \iota_a)))$ and includes the session identifier. The filter installed at node b during session v would have the form $a \longrightarrow b : v : [\mathsf{In}(a, \iota_b)]$. When the establishment reply message $\mathsf{P}(a, b, \mathsf{X}(\mathrm{Rep}(b, a, u, \iota'_b, \iota'_a)))$ for session u arrives at node b, the packet will not match the filter installed in session v 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. On the other hand, associations may be shared across sessions to improve efficiency. The SPI is bound to the association not the session. In which case, packets belonging to different sessions traveling in a single association will have the same SPI but different session identifiers.

An observant reader may notice that our solution allows a pair of tunnels (each belonging to a different session) to be set up between a and b. In practice, this would not be a problem as traffic could safely travel in either, where the session identifier would be used to select the tunnel in which traffic would travel. On the other hand, from a general policy standpoint it might be deemed undesirable to maintain two associations flowing in each direction. One could detect if such a pair of tunnels was set up and change one of the mechanism entries at each node so that both sessions use the same association and tear down the unneeded association.

While the are many similarities between the IPsec architecture and the model for security tunnels presented in this chapter, there are also several differences. In the IPsec architecture, IKE is triggered when an outbound packet matches a filter entry in the mechanism database (IPsec policy database) and the corresponding associations (bundle entries) have yet to be set up. The expectation is that mechanism filter entries are set up by an administrator ahead of time, although, they may be altered during IKE execution. Our system presupposes the existence of tunnel-complex protocols that invoke tunnel-establishment protocols to set up pairwise tunnels and it is the establishment protocol that installs mechanism database entries. In IPsec, IKE traffic is exempted from the mechanism (IPsec policy) filters in order to prevent an infinite loop of calls to IKE. In order to accommodate the enforcement of authenticated traversal, our model does not exempt establishment traffic from the filter mechanisms. IPsec seems moving away from directly supporting nested tunnels and IKE does not support the construction of nested tunnels. Yet many useful tunnel complexes are formed from such a composition of tunnels. Therefore, we not only incorporate support for nested tunnels, but, as presented in the next chapter, our tunnel-establishment protocol also supports nested tunnel creation. By eliding the details of the cryptographic transformations performed by the tunnels, we need not perform a key exchange as part of establishment. As a result, our tunnel-establishment protocol has only two messages rather than the four seen in IKE. We had included these messages in the L3A establishment protocol, but seeing that key secrecy and integrity is not the focus of our study, the added messages were viewed as extraneous and were best abstracted away. Our decision not to produce an exact model of IPsec was due to the fact that we felt a more significant contribution would result if we focus on tunnel-complex protocols and were not encumbered by the constraints imposed by an existing standard; so we constructed a model for security tunnels that supports tunnel-complex protocols, which could inform the evolution of future protocols.

3.4 Conclusion

This chapter presented an overview of a scheme for modeling security tunnels that abstracts away cryptographic details, but models the databases (SADB and MDB) that define the state of the tunnels. A tunnel establishment protocol is illustrated that acts at the same level of abstraction. We demonstrate that the model can aid in uncovering errors that arise from interactions between tunnel-complex protocols and the state they install at nodes. The presentation in this chapter was kept at a rather high-level avoiding details of packet processing and authorization. In the next chapter, our model for tunnels becomes the foundation of a formalism called the tunnel calculus. In Chapter 5, the tunnel calculus is used prove that our modified establishment protocol does indeed avoid deadlock. The work reported in this chapter appeared in [53].

Chapter 4

Tunnel Calculus

In this chapter, we build upon the model for tunnels given above to construct a formalism called the *tunnel calculus.* The tunnel calculus is a domain specific formalism for expressing tunnel-complex protocols, reasoning about their functional correctness, and analyzing their vulnerability to DoS attacks. The semantic underpinnings of the tunnel calculus are similar to the Chemical Abstract Machine [11]. The core of the tunnel calculus is composed of the following four layers: the lowest layer models packet forwarding, the next layer models the state of tunnels at a node as well as the packet header processing performed by security tunnels, the next layer specifies how authorization is performed using distributed credentials, the top layer models tunnel-establishment. A precise definition of each of these layers is given in this chapter. The semantics of the tunnel calculus gives rise to a trace theory used for formal reasoning. The tunnel-calculus presented in this chapter is the foundation upon which the rest of this dissertation is built.

This chapter is organized into four sections: the first section defines the structure of and the grammar of the tunnel calculus, the second section presents the rewrite rules and semantic functions that define the operational semantics of the core layers of the tunnel calculus, the third section we defines a trace theory induced by the operational semantics, and the fourth section presents a collection of propositions.

4.1 Grammar and Structure

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 semantic functions 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 [11]. The tunnel calculus is obtained by instantiating the tuple with specific types, elements, terms, equations, and rules.

The types of the calculus are given in Table 4.1. The basic types include the ad-

Node	$a \in \mathit{Node}$	Domains	$d \in \mathit{Domain}$
Message	$m \in \mathit{Msg}$	Session	$u \in \mathit{Session}\ +\ -\infty$
Forwarding Table	$f \in \mathit{Addr} \rightsquigarrow \mathit{Addr}$	Booleans	$B \in \mathit{Boolean}$
Pub/Priv Key	$K, K^{-1} \in \mathit{Key}$	Request Ident	$k \in \mathit{Identifier}$
Signature	$g \in \mathit{Sig}$	Security Parameter Index	$\iota \in \mathit{SPI}$

Table 4.1: Tunnel Calculus Types

dresses of nodes in the network a, generic messages m, SPI values ι, and Booleans B. Although we generally elide cryptography in this dissertation, we do include digital signatures on establishment messages for the purposes of analyzing DoS threats in Chapter 7 and hence we include a type digital signatures g. Public key cryptography is used both to identify principals and to generate and verify signatures so types are included for public keys K and private keys K^{-1}. The type *Domain* represents an address range d. The forwarding table f is a partial function ($\rightsquigarrow$) from the destination address to the address of the next hop. A variable denoting the session identifier u may be either of type *Session* or have the value $-\infty$ indicating that no session identifier has been assigned to that variable.

The syntactic class of *elements* is specified in Table 4.2. The elements are the basic structures used to model packets, messages, associations, and mechanisms. Establishment messages contain addresses, SPI values, session identifiers, a set of credentials, and a digital signature to ensure that the message is authenticated. Discovery messages contain an address and a session identifier. The discovery and establishment messages must undergo special processing and are distinguished by the X and C constructors. A packet $\mathsf{P}(a, a, y)$ is a triple composed of source address, destination address, and payload. The payload y can be a message m, a secure message $\mathsf{S}(u, \iota, p)$, or a distinguished message χ. An association constructor has the form $\mathsf{Out}(a, \iota)$ and an association destructor has the form $\mathsf{In}(a, \iota)$. 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. A mechanism is a triple consisting of a packet filter ψ, a session identifier u, and a bundle of associations. A bundle is a list of inbound β^i or outbound β^o associations. An inbound mechanism database Π^i is a list of inbound security mechanisms π^i and an outbound security mechanism database is a list of outbound security mechanisms π^o. Together, Σ, Π^i, and Π^o

Est Req/Rep	κ	::=	$Req(a, a, u, \iota, \Xi, g) \mid Rep(a, a, u, \iota, \iota, \Xi, g)$
Disc	δ	::=	$\mathrm{Disc}(a, u)$
Distinguish	χ	::=	$\mathsf{C}(\delta) \mid \mathsf{X}(\kappa)$
Sec	s	::=	$\mathsf{S}(u, \iota, p)$
Payload	y	::=	$m \mid s \mid \chi$
Packet	p	::=	$\mathsf{P}(a, a, y)$
Assoc In	σ^i	::=	$\mathsf{In}(a, \iota)$
Assoc Out	σ^o	::=	$\mathsf{Out}(a, \iota)$
Assoc DB	Σ	::=	$\mathsf{Assoc}(\{\sigma^{\mathsf{i}}, \ldots, \sigma^{\mathsf{i}}\} \cup \{\sigma^{\mathsf{o}}_{\mathsf{1}}, \ldots, \sigma^{\mathsf{o}}_{\mathsf{m}}\})$
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	::=	$\omega \rightarrow \omega$
Mech In	π^i	::=	$\mathsf{Mech}(\psi : u : \beta^i)$
Mech Out	π^o	::=	$\mathsf{Mech}(\psi : u : \beta^o)$
Mech DB In	Π^i	::=	$\mathsf{MechIn}[\pi^i_1, \ldots, \pi^i_n]$
Mech DB Out	Π^o	::=	$\mathsf{MechOut}[\pi^o_1, \ldots, \pi^o_m]$
Key List	$\overline{K}$	::=	$K, K, \ldots, K$
Policy Selector	η	::=	$\omega \leftrightarrow \omega$
GW Policy	θ	::=	$\mathsf{Pol}\langle *, \eta\rangle \mid \mathsf{Pol}\langle \overline{K}, \eta\rangle$
Policies	Θ	::=	$\mathsf{Pols}[\theta, \ldots, \theta]$
Disc Pol	ϕ	::=	$\mathsf{Disc}\langle K \mid \overline{K}\rangle$
Disc Pols	Φ^u	::=	$\mathsf{Discs}\{\phi_1, \ldots, \phi_n\}^u$
Credential	ξ	::=	$\mathsf{Cred}\langle K, K\rangle$
Credentials	Ξ	::=	$\mathsf{Creds}\{\xi, \ldots, \xi\}^a \mid \mathsf{Creds}\{\xi, \ldots, \xi\}^u$
Resumption	z	::=	$\beta \mid a \mid p \mid \iota \mid k$
Resumption Term	Z	::=	$\langle\rangle \mid \langle z, \ldots, z\rangle$

Table 4.2: Tunnel Calculus Elements

define the state of tunnels at a node. Gateway policies θ have the form of a list of public keys and a selector. Each node maintains a list of polices in the structure Θ. Discovery policies ϕ take the form of a pair formed from a public key and a list of keys. Each session maintains a set of discovery policies Φ. Credentials ξ formed form a pair of two public keys. Both nodes and sessions maintain credential sets that are represented as Ξ^a and Ξ^u. Although credentials and discovery polices would likely contain digital signatures, this is elided for the sake of conciseness and the fact that doing so does not alter our treatment of the topic. Resumption terms hold 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 4.3. Packets p, resumption

Term	t	$::=$	$p \mid Z \mid \Sigma \mid \Pi^i \mid \Pi^o \mid \Theta \mid \Phi \mid \Xi \mid$
To/Ack/From Fwd			$\downarrow_{\mathrm{ip}(k)} p \mid \upharpoonright_{\mathrm{ip}(k)} \mid \upuparrows_{\mathrm{ip}} p \mid$
To/Ack/From Sec			$\downarrow_{\mathrm{sec}(u,k)} p \mid \upharpoonright_{\mathrm{sec}(k)} \mid \upuparrows_{\mathrm{sec}(u)} p \mid$
To/Ack Establish			$\downarrow_{\mathrm{est}(u,k)} \mathsf{E}(a,a,a) \mid \upharpoonright_{\mathrm{est}(k)} \mid$
To/Ack Est Resp			$\downarrow_{\mathrm{eresp}(u,k)} \mid \upharpoonright_{\mathrm{eresp}(k)} \mathsf{R}(a)$
To/Ack Authorization			$\downarrow_{\mathrm{auth}(u,k)} \mathsf{Ai}(a,a,a,a,\Theta,\Xi) \mid$
			$\upharpoonright_{\mathrm{auth}(k)} \mathsf{GWPol}(u,B) \mid$
			$\downarrow_{\mathrm{auth}(u,k)} \mathsf{Ar}(a,a,a,a,\phi,\Xi) \mid$
			$\upharpoonright_{\mathrm{auth}(k)} \mathsf{DisPol}(u,B) \mid$
To/Ack Discovery			$\downarrow_{\mathrm{dis}(u,k)} \mathsf{D}(a,a) \mid \upharpoonright_{\mathrm{dis}(k)}$
Node Term	nt	$::=$	$t @ a$

Table 4.3: Tunnel Calculus Terms

terms Z, the association database Σ, the mechanism databases Π^i and Π^o, discovery policies Φ, gateway policies Θ, and credential sets Ξ 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 $\upuparrows_{\mathrm{ip}}$. The $\upharpoonright_{\mathrm{ip}(k)}$ term acknowledges that the forwarding layer has completed processing.

Node terms have a grammar $nt ::= t @ a$, 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. Every node in the network must contain node terms specifying the forwarding table, association database, and mechanism databases. Protocols modeled as rewrite rules update this state as they execute.

A rewrite rule has the form

$$\begin{aligned}
&t_1 \mathbin{@} a_1, \ldots, t_n \mathbin{@} a_n \longrightarrow t'_1 \mathbin{@} a'_1, \ldots t'_m \mathbin{@} a'_m \quad \text{if } E \\
&\text{if } E' \text{ then id } = z \text{ else } (z \mid \text{ is new}) \\
&\text{where } t'' = h \\
&\text{new } u \mid \iota \mid k
\end{aligned}$$

where id is and identifies, E and E' are Boolean expressions, and h is a semantic function. If the rule has the optional if E, then the rule only fires if E evaluates to true. If the rule has an optional

$$\text{if } E' \text{ then id} = z_1 \text{ else } (z_2 \mid \text{ is new}),$$

then the value of identifier id in the rule will take on the value of element z_1 if the predicate E' is true and if E' is false, the identifier id takes on the value given by element z_2 or generates a new value. If the rule contains the optional

$$\text{where } t'' = h,$$

then unification is performed matching the term t'' against the value returned by the semantic function h. Suppose the rule contained

$$\text{where } Y(var_1, var_2) = h.$$

If h did not return a term that could unify against $Y(var_1, var_2)$, then the rule would not be executed. If h returns $Y(c, d)$, then var_1 and var_2 will be assigned the values c and d respectively. The *new* operator generates a unique value, which is formally stated as follows: if $x = \text{new } u$, then then x is not a subterm of any term previously written to the multiset and

$$x = \text{new } u \textsf{ and } y = \text{new } v \implies x \neq y.$$

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

Recall that rules in our system have the form

$$t_1 \mathbin{@} a_1, \ldots, t_n \mathbin{@} a_n \longrightarrow t'_1 \mathbin{@} a'_1, \ldots, t'_m \mathbin{@} a'_m.$$

To prevent the rules from being too cumbersome to read, the following notational conventions are employed. When all the node terms in a rule are located at the same node a we drop the $@\,a$ on each node term and write the rule as

$$\vdash_a t_1, \ldots, t_n \longrightarrow t'_1, \ldots, t'_m,$$

where the node's address is a subscript to the turnstile. If terms are used in a rule, but not consumed, then then they are written to the left of the turnstile

$$t_1 \,@\, a_1, \ldots, t_i \,@\, a_i \vdash t_{i+1} \,@\, a_{i+1}, \ldots, t_n \,@\, a_n \longrightarrow t_1' \,@\, a_1' \ldots, t_m' \,@\, a_m'.$$

Many rules combine these two shorthand notations and have the form

$$t_1, \ldots, t_i \vdash_a t_{i+1}, \ldots, t_n \longrightarrow t_1' \ldots, t_m'.$$

terms	$\bar{t}$	::=	$t_1, \ldots, t_n$
node terms	$\overline{nt}$	::=	$nt_1, \ldots, nt_m$
basic rule	br	::=	$[\overline{nt}] \vdash \overline{nt} \longrightarrow \overline{nt}$
			$\mid [\bar{t}] \vdash_a \bar{t} \longrightarrow \bar{t}$
rewrite rule	rr	::=	br [if E]
			[if E' then id $= z$ else $(z \mid$ is new$)$]
			[where $t = h$]
			[new $(u \mid \iota \mid k)$]

Table 4.4: Tunnel Calculus Rewrite Rules

A formal grammar for the rewrite rules is given in table 4.4.

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 rewriting logic 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 holds state. Consider the example

$$\begin{array}{ll} \mathbf{1)} & t_1 \,@\, a \longrightarrow t_2 \,@\, a, \langle x_1, x_2 \rangle \,@\, a. \\ \mathbf{2)} & \langle x_1, x_2 \rangle \,@\, a, t_3 \,@\, a \longrightarrow t_4 \,@\, a, \langle x_3 \rangle \,@\, a. \\ \mathbf{3)} & \langle x_3 \rangle \,@\, a, t_5 \,@\, a \longrightarrow t_6 \,@\, a. \end{array}$$

The term t_3 contains elements x_1 and x_2 and the term t_5 contains the element x_3. The first rule writes a resumption term $\langle x_1, x_2 \rangle$. The second rule will not execute until such a term appears in the multiset. The fact that t_3 contains the elements x_1 and x_2 means that the rule consumes two node terms both containing x_1 and x_2. So the resumption term produced in **1**) is consumed in **2**) and the order of execution of the two rules is determined. The values of x_1 and x_2 should be chosen so that only the desired instance of t_3 may possess them. Session identifiers and acknowledgment identifiers are good choices in that they are unique to a particular run of the protocol. Given these constraints, we can deduce that the execution of the second rule has been ordered to come after the execution of the first. Similarly, the execution third rule is delayed until the resumption term $\langle x_3 \rangle$ appears in the multiset along with a term t_5 containing element x_3.

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 in layers that form a set of primitives that are used to express a tunnel-complex protocol. Each layer of the calculus is intended to abstractly model some layer or module in the network stack. The lowest layer is the *forwarding layer* (ip) modeling the movement of packets performed by IP. The *secure layer* (sec) models the tunnel processing described in Chapter 3. The *authorization layer* (auth) verifies that a set of credentials satisfy a given policy. The *establishment layer* (est, eresp) is an abstraction of establishment protocols such as IKE and establishes a bidirectional tunnel between two nodes and updates the association and mechanism databases accordingly. The forwarding, secure processing, and establishment layers are structured as having an initiator and a responder process. The forwarding and secure layer responder processes run as a daemon. The authorization layer behaves like a function and does not have a responder. These layers form a foundation upon which tunnel-complex protocols are built.

The first rule of an initiator (and the establishment responder) always has the form of a rewrite rule with $\downarrow_I$ on the left of the arrow, where $I = \{\text{ip, sec, auth, eresp, est}\}$. 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 to acknowledge termination. Responder processes await the reception of a message from the initiator before performing any action. If a responder is running as a daemon, information is passed to a higher level by writing an $\Uparrow_I$ term. Otherwise, information is passed in an $\uparrow_I$ 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. This prevents confusion that may result from many $\uparrow_I$ terms being in the multiset. In the forwarding layer, these terms have the form $\downarrow_{\text{ip}(k)}$ and $\uparrow_{\text{ip}(k)}$. The $\downarrow$ terms of the remaining layers are also annotated with the session identifier. To see how the layers interact consider what happens when a packet $p\,@\,a$ is sent to node b via the secure processing layer. The secure layer applies the appropriate constructors to the packet and sends it to the forwarding layer; the forwarding layer forwards it to the next node where it gets 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:

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

and at node b they will add the terms:

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

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 [55].

The following two assumptions hold throughout the remainder of this dissertation. The first assumption is that all messages sent at layers higher than the secure processing layer are sent via the secure processing layer. The second assumption is that session identifiers are always introduced using the *new* operator, ensuring their uniqueness.

4.2 Core Layers of the Tunnel Calculus

In this section we give a precise definition of each of the four core layers of the tunnel calculus. In the case of the authorization layer, we give a relation defining what it means a credential set to satisfy a policy. For the other three layers, a collection of rewrite rules are defined, which define the operational semantics of the layer.

4.2.1 Forwarding Layer

The forwarding layer models the movement of packets based on a forwarding table and serves as an abstraction of the IP layer. We do not attempt to model packet fragmentation or routing. The semantics of this layer is concisely expressed using two rules.

Rule F.1.1 $\quad \mathsf{F}(f) \,@\, a \ \vdash\ \downarrow_{\mathrm{ip}(k)} \mathsf{P}(b,c,y) \,@\, a \longrightarrow \mathsf{P}(b,c,y) \,@\, f(c),\ \uparrow_{\mathrm{ip}(k)}\ @\, a.$

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. The forwarding layer responder daemon is specified by the rule.

Rule F.2.1 $\quad \vdash_a \ p \longrightarrow \Uparrow_{\mathrm{ip}}\, p$

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

4.2.2 Secure Processing Layer

The secure processing layer performs the packet processing associated with tunnels. When the secure processing layer is invoked ($\downarrow_{\mathrm{sec}(u,k)}\ p$) to send a packet to its destination, the secure processing layer initiator performs the following actions: it first consults the outbound mechanism database Π^o to obtain a bundle of associations, it then applies the constructors in the bundle to the packet, and, finally, it dispatches the resulting packet to the forwarding layer to move it to the other end of the tunnel. The initiator must await the forwarding layer acknowledgment before writing the acknowledgment that it has completed processing. The secure layer responder daemon is more complex. A packet arriving at a node may be a distinguished message such as a discovery or establishment packet, a packet legitimately arriving in the clear, or a packet traveling in one or more security associations. If the packet is arriving in a security association, then 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. This processing models the application of destructors. If the decapsulated packet is a distinguished packet, then 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 a packet arrives at a gateway in a tunnel and the gateway is not the final destination, then a $\downarrow_{\mathrm{sec}(u,k)}\ p$ term is written in order to send the packet towards its destination.

The following two rewrite rules express the semantics of the secure processing layer initiator.

Rule S.1.1 $\quad \Pi^o \ \vdash_e \ \downarrow_{\mathrm{sec}(u,k)} \mathsf{P}(b,c,y) \longrightarrow$
$\downarrow_{\mathrm{ip}(k')}\ \mathrm{Nest}(\mathrm{BndlSel}(b,c,u,\Pi^o), e, u, \mathsf{P}(b,c,y)), \langle k, k', u\rangle$
$\text{new}\ \ k'$.

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) that apply to the packet. The semantic function Nest applies the appropriate constructors and encapsulates the packet in the proper headers creating a packet p'. The term $\downarrow_{\mathrm{ip}(k')}\ p'$ is written to the multiset indicating that the packet is ready for dispatch by the forwarding layer with the acknowledgment identifier k' generated by the *new* operator. A resumption term is written to the multiset containing the session identifier u and the two acknowledgment identifiers k and k'.

Rule S.1.2 $\quad \vdash_e \ \langle k, k', u\rangle,\ \uparrow_{\mathrm{ip}(k')} \longrightarrow \uparrow_{\mathrm{sec}(k)}$

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; however, it is not an acknowledgment of delivery.

The secure layer inbound processing rules call Strip to process any messages arriving in a tunnel and the rule that gets executed depends on the shape of the returned value. If a packet is a control packet or an establishment packet, then it gets passed up for processing. If the packet is traveling in an invalid security association, there is no matching pattern and the packet is effectively dropped. A valid packet destined for this node is passed up for processing. A valid packet destined for a different node is sent on toward its destination.

The following six rules define the processing performed by the secure layer responder.

Rule S.2.1

$$\begin{array}{lll} \Sigma, \Pi^i & \vdash_e & \Uparrow_{\text{ip}} p \longrightarrow \\ & & \Uparrow_{\text{sec}(u)} \mathsf{P}(b, c, \mathsf{X}(\kappa)) \\ & & \text{where } \mathsf{Exchange}(\mathsf{P}(b, c, \mathsf{X}(\kappa)), \beta) = \text{Strip}(\Sigma, e, -\infty, p, \mathsf{Bndl}[]) \\ & & \text{if } \mathsf{Mech}(b \to c : u : \beta) \in \Pi^i \text{ or} \\ & & \quad (\nexists \beta' \neq [].\mathsf{Mech}(b \to c : u : \beta') \in \Pi^i \text{ and } \beta = []). \end{array}$$

If the decapsulated packet is an establishment message and there is a matching entry in the mechanism database, which indicates it arrived in a valid tunnel, or there is no matching entry and an empty bundle value has been returned by Strip, which indicates that the packet arrived in the clear, then pass the exchange packet up for further processing.

Rule S.2.2

$$\begin{array}{lll} \Sigma, \Pi^i & \vdash_e & \Uparrow_{\text{ip}} p \longrightarrow \\ & & \Uparrow_{\text{sec}(u)} \mathsf{P}(b, c, \mathsf{X}(\kappa)) \\ & & \text{where } \mathsf{ConMsg}(\mathsf{P}(b, c, \mathsf{X}(\kappa)), \beta) = \text{Strip}(\Sigma, e, -\infty, p, \mathsf{Bndl}[]) \\ & & \text{if } \mathsf{Mech}(b \to c : u : \beta) \in \Pi^i \text{ or} \\ & & \quad (\nexists \beta' \neq [].\mathsf{Mech}(b \to c : u : \beta') \in \Pi^i \text{ and } \beta = []). \end{array}$$

The processing for control packets is the same as for establishment packets.

Rule S.2.3

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

Decapsulates a packet that is neither a control packet nor an establishment packet.

Rule S.2.4

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

If the packet was traveling in a valid association and it is destined for this node, then pass the packet up for further processing.

Rule S.2.5

$$\begin{array}{lll} \Pi^i & \vdash_e & \langle \mathsf{P}(b,c,y),\beta,u\rangle \longrightarrow \downarrow_{\mathrm{sec}(u,k_1)} \mathsf{P}(b,c,y),\ \langle u,k_1\rangle \\ & & \text{if } e \neq c \text{ and } \mathsf{Mech}(b \to c : u : \beta) \in \Pi^i \\ & & \text{new } \ k_1. \end{array}$$

If the packet was traveling in a valid association and it is not destined for this node, then invoke the secure layer to send the packet towards its destination.

Rule S.2.6

$$\vdash_e \ \langle u,k_1\rangle,\ \uparrow_{\mathrm{sec}(k_1)} \ \to \cdot$$

Upon receiving the acknowledgment that the message has been sent, the protocol terminates.

Semantic Functions

The secure processing layer makes use of several auxiliary semantic functions. The secure layer initiator rules are made concise by the using the functions BndlSel and Nest to wrap a packet in the proper header. The secure layer responder uses the function Strip to apply destructors to traffic arriving in an association. The precise definitions are given below.

When processing an outbound packet, the function BndlSel consults the outbound mechanism database Π^o to determine what, if any, bundle to apply to the packet.

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

$$\begin{array}{l} \mathrm{BndlSel}(b,c,u,\Pi^o) = \beta \\ \qquad \text{if } \mathsf{Mech}(b \to c : u : \beta) \in \Pi^o \\ \mathrm{BndlSel}(b,c,u,\Pi^o) = \mathsf{Bndl}[\,] \ \text{ otherwise} \end{array}$$

This function takes as parameters the packet's source and destination addresses, the session identifier, and the outbound mechanism database. It fetches from the mechanism database the bundle from the mechanism entry having a matching packet filter and session identifier. If there are no matching entries, the message is sent in the clear.

When a bundle is applied to an outbound packet, a new packet is created, which is composed of nested packets. For instance, if the bundle

$$\mathsf{Bndl}[\mathsf{Out}(c_2, \iota_2), \mathsf{Out}(c_1, \iota_1)]$$

is applied to the packet $p = \mathsf{P}(b, c, y)$ at node a in session u, then the following packet is generated:

$$\mathsf{P}(a, c_1, \mathsf{S}(u, \iota_1, \mathsf{P}(a, c_2, \mathsf{S}(u, \iota_2, \mathsf{P}(b, c, y))))).$$

This action is performed by the Nest function.

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

$$\begin{gathered}\text{Nest}(\mathsf{Bndl}[], e, u, p) = p \\ \text{Nest}((\mathsf{Out}(d, \iota) :: \beta), e, u, p) = \text{Nest}(\beta, e, u, \mathsf{P}(e, d, \mathsf{S}(u, \iota, p)))\end{gathered}$$

The parameters are the bundle β that is to be applied to the packet, the address e of the current node, the session identifier u, and the packet p to which the bundle is applied. The function applies the bundle $\mathsf{Out}(d, \iota) :: \beta$ to the packet p in a recursive manner by taking the head of the bundle list $\mathsf{Out}(d, \iota)$, and creating a new packet with source address e, destination address d, and a secure message with session u and SPI ι. The function is then called again with the newly created packet as a parameter. If the bundle is empty, the function just returns the packet.

The secure layer responder uses a semantic function Strip to remove and verify secure packet headers. When Strip is initially called the session number is unknown and the parameter is assumed to be set to $-\infty$. Control and exchange packets contain the session number even if not traveling in a tunnel. The function strip Strip returns $-\infty$ if called with a packet that is not a control packet, an exchange packet, or a packet traveling in a tunnel.

$$\begin{aligned}&\text{Strip} : \mathit{Associations} \times \mathit{Addr} \times \mathit{Packet} \times \mathit{Bundle} \rightsquigarrow \\ &\mathit{Packet} \times \mathit{Sessions} \times \mathit{Bundle} \; + \; \mathsf{Exchange}(\mathit{Packet} \times \mathit{Bundle}) \\ &\qquad\qquad + \; \mathsf{ConMsg}(\mathit{Packet} \times \mathit{Bundle})\end{aligned}$$

$$\begin{array}{rcl}\text{Strip}(\Sigma, e, u', \mathsf{P}(b, c, \mathsf{C}(u, m)), \beta) & = & \mathsf{ConMsg}(\mathsf{P}(b, c, \mathsf{C}(u, m)), \beta). \\ \text{Strip}(\Sigma, e, u', \mathsf{P}(b, c, \mathsf{X}(u, m)), \beta) & = & \mathsf{Exchange}(\mathsf{P}(b, c, \mathsf{X}(u, m)), \beta). \\ \text{Strip}(\Sigma, e, u', \mathsf{P}(b, c, \mathsf{S}(u, \iota, y)), \beta) & = & \text{Strip}(\Sigma, e, y, u, \mathsf{In}(b, \iota) :: \beta) \\ & & \text{if } e = c \ \mathsf{and} \ \mathsf{In}(b, \iota) \in \Sigma. \\ \text{Strip}(\Sigma, e, u', \mathsf{P}(b, c, \mathsf{S}(u, \iota, y)), \beta) & & \text{is undefined} \\ & & \text{if } e = c \ \mathsf{and} \ \mathsf{In}(b, \iota) \notin \Sigma. \\ \text{Strip}(\Sigma, e, u', \mathsf{P}(b, c, y), \beta) & = & (\mathsf{P}(b, c, y), u', \beta) \\ & & \text{otherwise.}\end{array}$$

The first equation says that if the packet is a control packet, then return

$$\mathsf{ConMsg}(p, \beta).$$

The second equation returns $\mathsf{Exchange}(p, \beta)$ if the message is an exchange packet. The third equation removes the outer secure header and checks to see if the destination given in the outer header is the same as e (the node doing the processing). If so and the association is valid (a member of Σ), then recursively call Strip with the association of the header that was just removed added to the bundle. The fourth equation covers the case where the header has an invalid association and the packet is just dropped. The fifth equation simply returns the packet because either all the secure headers have been removed or the outer header indicates a different destination address.

The tunnel calculus is designed to accommodate nested tunnels. In the context of tunnel-complex protocols, there must be a design decision made as to when one tunnel is nested inside another. The tunnel calculus implements the assumption that if session u creates a tunnel, but there already exists a entry in the mechanism database for that selector, then the newly created tunnel is nested inside of the existing tunnel. This assumption is built into the $\otimes$ operator that defines how a mechanism entry gets added to a mechanism database. Consider the situation where the mechanism $\mathsf{Mech}(a \to b : u : \mathsf{Bndl}[\mathsf{Out}(b, \iota_b)]$ is added to the outgoing mechanism database Π^o. If there is no entry in Π^o with selector $a \to b$ and session identifier u, then simply add the entry to the database. This is reflected in the first equation below. If there is an entry already in Π^o with selector $a \to b$ and session identifier u, then we do not want to add a new entry. Instead, the association $\mathsf{Out}(b, i_b)$ is added to the bundle of the existing entry to create nested tunnels. Note that this action is not performed if the association is already in the bundle.

$$\begin{array}{rcl}
\mathsf{Mech}(\psi : u : \mathsf{Bndl}[\sigma]) \otimes \Pi & = & \mathsf{Mech}(\psi : u : \mathsf{Bndl}[\sigma]) :: \Pi \\
\text{if } \mathsf{not} \ (\exists \pi = \mathsf{Mech}(\psi' : u' : \beta') \in \Pi. & & (\psi = \psi' \ \mathsf{and} \ u = u')) \\
\mathsf{Mech}(\psi : u : \mathsf{Bndl}[\sigma]) \otimes \Pi & = & \mathsf{Mech}(\psi : u : (\sigma :: \beta')) :: (\Pi - \pi) \\
\text{if } \exists \pi = \mathsf{Mech}(\psi' : u : \beta') \in \Pi. & & (\psi = \psi' \ \mathsf{and} \ u = u' \ \mathsf{and} \ \sigma \notin \beta')
\end{array}$$

4.2.3 Authorization Layer

Security gateways are presumed to enforce policies governing the flow of ingress traffic into its administrative domain and egress traffic exiting its administrative domain. The authenticated traversal property states that all ingress and egress traffic should be authenticated and authorized as satisfying the policy at a gateway. Gateway policies precisely define the principals that are allowed to undertake specific traffic flows.

Consider a discovery protocol that has just discovered a gateway on the dataflow path. The gateway protects its administrative domain against unauthorized egress or ingress traversal; so the discovery protocol must present credentials that satisfy the gateway's policies in order to traverse this gateway. But should the protocol trust this unknown gateway. Assume that the gateway is malicious and that it has inserted itself as a man-in-the middle, if no end-to-end encryption is employed in the final tunnel configuration, then the attacker can listen in on subsequent communication. If end-to-end encryption is used, then the attacker can still monitor the traffic flow, which is undesirable in and of itself. To prevent this from occurring we have introduced the notion of discovery policy. A discovery policy is simply the list of administrative entities with which a protocol session is willing to communicate. The host initiating discovery is assumed to initialize the session discovery policy as part of normal protocol execution and the protocol can add to the list of acceptable administrative entities as the protocol executes.

Distributed credentials are used to satisfy both discovery polices and gateway policies. Tunnel-complex protocols are responsible for delivering the proper credentials to a node. When a tunnel-complex protocol begins execution it only has available to it the credentials available at the node that initiated the protocol. Additional credentials may be obtained as new gateways on the dataflow path are discovered. In this sense, tunnel-complex protocols can be viewed as credential delivery mechanisms.

In this section, we give a precise definition of gateway policies, discovery polices, and credentials as well as what it means for a credential set to satisfy a policy. Collectively these definitions determine the tunnel calculus *authorization layer*. We assume a public key infrastructure, but elide the details. The resulting calculus is similar to SPKI/SDSI [46] in content and objective, but is simplified and specialized to this application to make its description self-contained.

Gateway Policies

A gateway policy $\theta = \mathsf{Pol}\langle K_1, \ldots, K_n : \eta\rangle$ specifies a list of principals $K_1, \ldots, K_n$ that is authorized by a gateway to communicate between the addresses given in the selector η. To illustrate the purpose of a gateway policy, consider the case where a tunnel-complex protocol has been employed to enable communication between s and d. Assume a gateway on the dataflow path between s and d has the policy $\mathsf{Pol}\langle K_x, s \leftrightarrow d\rangle$. If the protocol delivers a credential for K_x to the gateway, then a tunnel is set up allowing traffic flowing between s and d to tunnel through the gateway. Policies can restrict communication to be bidirectional $a \leftrightarrow b$ or unidirectional $a \mapsto b$. For instance, a policy at a gateway may say that Alice represented as principal K_A is allowed to communicate between the address a_1 and an address a_2. This is formalized as $\mathsf{Pol}\langle K_A : a_1 \leftrightarrow a_2\rangle$. A more liberal policy may state that any principal can communicate between these same address ranges, and would be written as $\mathsf{Pol}\langle * : a_1 \leftrightarrow a_2\rangle$. Each gateway maintains a list of policies $\Theta = \mathsf{Pols}[\theta_1, \ldots, \theta_n]$ that it enforces. For simplicity, we assume that the policies at each gateway have disjoint traffic selectors. Hence, there is at most one policy at a gateway for a given traffic flow. We write $\Theta_\eta \mathbin{@} a$ to denote the policy at node a that matches the selector η.

We could have employed a naming system similar to the one in SPKI/SDSI to avoid explicit mention of address ranges by binding them to domain names. This would allow us to write policies such as $\mathsf{Pol}\langle K_A : K_{a_1} \leftrightarrow K_{a_2}\rangle$. However, it is essential for policies to be expressed in terms of address ranges ultimately since a packet encrypted from end-to-end shows little else on which policy could be based.

Discovery Policies

Gateway discovery poses the quandary of whether a newly-discovered gateway should be trusted. To counter the threat of rouge gateways, discovery protocols verify that a newly discovered gateway is trusted to continue executing the protocol by checking a list of administrative entities trusted by the protocol. These entities are defined by the discovery policy maintained at each host. A discovery policy ϕ says that principal K is willing to communicate with principals $K_1, \ldots, K_n$. This is formalized as $\mathsf{Disc}\langle K \mid K_1, \ldots, K_n\rangle$.. The structure Φ^u contains the set of discovery policies for session u. The discovery policy at node a is often denoted ϕ^a. It is left to the discovery protocol designer to define who gets to contribute to a session's discovery policy. For instance, in a very restrictive design, each gateway on the path must belong to one of the administrative entities listed in the protocol initiator's discovery policy. A more liberal design would allow gateways on the path to contribute to the session's discovery policy. In this case, a newly discovered gateway would be acceptable if it were listed in the discovery policy of the initiating host or in the discovery policy of any previously discovered gateways.

The set of discovery policies Φ^u for session u is typically initialized to the set of policies at the initiating node when the protocol begins executing. Depending on the protocol, the set may grow as newly discovered nodes are approved.

A fuller authorization system may require a more sophisticated structure for discovery policies than that provided in the current system, which may be inadequate for cases where the trusted gateways would be inferred from a delegation chain, general attributes, or a similar technique, but we choose to avoid that complexity here.

Credentials

Credentials specify a relationship between two principals or delegate authority from one principal to another. For instance, Alice may have a credential that says she belongs to Acme Inc. A credential $\xi = \mathsf{Cred}\langle K_S, K_I\rangle$ defines a relation such as: K_I 'delegates' to K_S, or K_S 'is a member of' K_I, or K_S 'speaks for' K_I. Principal K_I

is called the *issuer* and K_S the *subject*. Given a credential that says K_S speaks for K_I, we often write $K_S \Rightarrow K_I$. The credential set for node a is denoted Ξ^a. The credential set presented to gateways during an execution of discovery protocol session u is denoted Ξ^u.

Given a credential set defining $K_n \Rightarrow K_{n-1}$, and $K_{n-1} \Rightarrow K_{n-2}$, and $K_{n-2} \Rightarrow K_{n-3}$, and $K_{n-3} \Rightarrow K_{n-4}$, and $\ldots$, $K_3 \Rightarrow K_2 \Rightarrow K_1$, and $K_3 \Rightarrow K_m$, and $K_2 \Rightarrow K_{m'}$, one can from a tree

$$\begin{tikzcd}
 & & & & & K_m & & \\
\Xi \vdash K_n \arrow[r, Rightarrow] & K_{n-1} \arrow[r, Rightarrow] & K_{n-2} \arrow[r, Rightarrow] & \cdots \arrow[r, Rightarrow] & K_3 \arrow[ur, Rightarrow] \arrow[rr, Rightarrow] & & K_2 \arrow[r, Rightarrow] \arrow[dr, Rightarrow] & K_1. \\
 & & & & & & & K_{m'}
\end{tikzcd}$$

The tree formed from the credential set Ξ is denoted $\mathcal{T}(\Xi)$. Given $\mathcal{T}(\Xi)$, a *chain* is defined as a path from the root of the tree (term closest to the $\vdash$) to a leaf. Whenever a tree if formed we assume that a signature on the credentials are checked to verify the integrity of the credentials, but this has been elided in the current treatment.

Satisfaction

Having defined gateway policies, discovery policies, and credentials, it is now possible to define what it means for a given credential set to satisfy a given gateway or discovery policy. The authorization layer is treated as a function call by the establishment layer. Instead of giving a set of rewrite rules for this layer, a satisfaction relation is given specifying what it means for a credential set to satisfy a given policy. The case of discovery policies and gateway polices are considered separately.

To satisfy a gateway policy one must verify that a given credential set contains a key listed in the policy. For instance, if the policy at a gateway G_3 says K_M is allowed to traverse the gateway and the protocol initiated by Alice delivers the credential $K_A \Rightarrow K_M$ saying she belongs to principal M, then the policy is satisfied, but to ensure integrity we require that there be a delegation chain from G_3 to A such as

$$K_{G_3} \Rightarrow K_{G_2} \Rightarrow K_{G_1} \Rightarrow K_A \Rightarrow K_M.$$

It remains to formalize this concept. Given the policy Θ_η for the traffic flow η and a credential set Ξ^u, the satisfaction relation

$$\Xi^u \models_{K_a} \Theta_\eta$$

is defined to be true if there exists a chain in $\mathcal{T}(\Xi^u)$ rooted at K_a that contains one of the keys in Θ_η. The relation is defined to be false otherwise.

The interface for the authorization layer for gateway policies is defined as

$$\downarrow_{\mathrm{auth}(u,k)} \mathsf{Ai}(a, b, s, d, \Theta, \Xi^u),$$

where a is the address of the establishment initiator, b is the address of the establishment responder, and s and d are the address used to select the policy that applies to this traffic flow. The authorization layer returns $\uparrow_{\text{auth}(k)} \mathsf{GWPol}(u, \text{true})$ if

$$\Xi^u \models_{K_a} \Theta_{s \leftrightarrow d}$$

and $\uparrow_{\text{auth}(k)} \mathsf{GWPol}(u, \text{false})$ otherwise.

In order for node a to prove that it satisfies the discovery policy Φ^u at node b, node a must send its credential set Ξ^a to b. These credentials define the administrative domain to which the node belongs. For instance, a may belong to the accounting department (K_N) of Coyote corporation (K_C) and so the credential set can form the chain $K_a \Rightarrow K_N \Rightarrow K_C$. If the discovery policy contains K_C, then it is satisfied by the given credential chain. Satisfaction for discovery policies is formalized as follows. Given a set of discovery policies

$$\Phi^u = \mathsf{Discs}\{\mathsf{Disc}\langle K_I \mid K_1, \ldots, K_n\rangle, \ldots, \mathsf{Disc}\langle K_G \mid K'_1, \ldots, K'_m\rangle\}$$

and a set of credentials Ξ^a from node a, the satisfaction

$$\Xi^a \models_{K_a} \Phi^u$$

is defined to be true if $\mathcal{T}(\Xi^a)$ is a tree rooted at K_a that contains a chain that contains one of the keys listed in the discovery policies in Φ^u. The relation is defined to be false otherwise.

The interface for the authorization layer for discovery policies is given as

$$\downarrow_{\text{auth}(u,k)} \mathsf{Ar}(a, b, s, d, \Phi^u, \Xi^a),$$

where a is the address of the establishment initiator, b is the address of the establishment responder, and s is the originating source and d is the destination addresses of the protocol. The authorization layer returns $\uparrow_{\text{auth}(k)} \mathsf{DisPol}(u, \text{true})$ if

$$\Xi^a \models_{K_b} \Phi^u$$

and $\uparrow_{\text{auth}(k)} \mathsf{DisPol}(u, \text{false})$ otherwise.

4.2.4 Establishment Layer

Tunnel establishment is the process of setting up a bidirectional tunnel 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 for the associations by way of a key exchange protocol [93, 19]. The focus in this dissertation is on the first two components, therefore, we elide 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. The protocol is only successful if both participants can present credentials that satisfy the policy at the other.

This protocol has been designed with the intention that it will be used as a component in discovery protocols. The discovery protocol is assumed to invoke the establishment responder when it sends out a distinguished discovery packet and the gateway that intercepts the packets invokes the establishment initiator process.

The establishment layer messages are digitally signed to ensure their integrity. Although it has been our preference throughout this dissertation to elide cryptographic operations whenever practical, in this case we need to make them explicit to facilitate the analysis of DoS threats performed in Chapter 7. The function $\mathrm{Sign}(K_a^{-1})$ produces a signature of the message sent in the given rule. The function $\mathrm{CheckSig}(K_b, g)$ is defined by the equation

$$\begin{cases} \mathrm{CheckSig}(K_a, Sign(K_a^{-1}, p)) & = \quad \mathrm{true} \\ \mathrm{Otherwise} & \quad\;\; \mathrm{false.} \end{cases}$$

Assume that node a is the establishment initiator, node b is the establishment responder, the session identifier is u, and that the protocol will install packet filters s and d. The tunnel establishment protocol works as follows. The establishment initiator a generates a SPI ι_a to identify the association flowing from the responder to the initiator and forms an establishment request message $\mathsf{Req}(s, d, u, \iota_a, \Xi^a, g)$, where Ξ^a is the initiator's credential set and g is a signature generated using the private key of a. Upon receiving a message of this form, the responder b calls the authorization layer to verify that Ξ^a satisfies the discovery policy Φ^u. If so, the responder generates a SPI ι_b to identify the association flowing from a to b and makes the appropriate entries in the association and inbound mechanism database for this association. The responder then forms an establishment response message $\mathsf{Rep}(s, d, u, \iota_a, \iota_b, \Xi^u, g')$, where Ξ^u is the credential set for protocol session u and g' is a signature generated using the private key of b. Once the establishment response message has been sent, the responder adds entries in the association and outbound mechanism database for the association flowing from the responder to the initiator. Upon receiving the establishment response message, the initiator a calls the authorization layer to verify that Ξ^u satisfies its policy $\Theta_{s\leftrightarrow d}$. If so, entries for the two associations are added to the association and mechanism databases.

The rules for the establishment initiator process are given as follows.

Rule E.1.1

$$\begin{array}{lll}\Xi^a & \vdash_a & \downarrow_{\mathrm{est}(u,k_1)} \mathsf{E}(b,s,d) \longrightarrow \\ & & \downarrow_{\mathrm{sec}(u,k_2)} \mathsf{P}(a,b,\mathsf{X}(\mathsf{Req}(s,d,u,\iota_a,\Xi^a,g))), \\ & & \langle u,a,b,s,d,k_1,k_2,\iota_a\rangle \\ & & \text{if } \exists \mathsf{In}(b,\iota_x) \in \Sigma \text{ then } \iota_a = \iota_x \text{ else } \iota_a \text{ is new} \\ & & \text{new } \ k_2 \\ & & \text{where } g = \mathrm{Sign}(K_a^{-1}).\end{array}$$

The initiator a invokes the establishment layer by writing a $\downarrow_{\mathrm{est}(u,k)} \mathsf{E}(b,s,d)$ term, where b is the responder and s and d are the packet filters to be installed in the mechanism database. 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 a signed establishment request message to node b. The semantic function sign produces a signature of the message being sent using the private key of a.

Rule E.1.2

$$\begin{array}{lll}\Theta & \vdash_a & \langle u,a,b,s,d,k_1,k_2,\iota_a\rangle, \\ & & \uparrow_{\mathrm{sec}(k_2)}, \Uparrow_{\mathrm{sec}(u)} \mathsf{P}(b,a,\mathsf{X}(\mathsf{Rep}(s,d,u,\iota_a,\iota_b,\Xi^u,g'))) \longrightarrow \\ & & \downarrow_{\mathrm{auth}(u,k_3)} \mathsf{Ai}(a,b,s,d,\Theta,\Xi^u), \\ & & \langle u,a,b,s,d,k_1,k_3,\iota_a,\iota_b\rangle \\ & & \text{new } \ k_3 \\ & & \text{if } \ \mathrm{CheckSig}(K_b,g').\end{array}$$

Upon receiving the establishment response message, the initiator a verifies the signature and invokes the authorization layer to verify that the credential Ξ^u satisfies the gateway policy $\Theta_{a\leftrightarrow b}$. We define $\mathrm{CheckSig}(K,\mathrm{Sign}(K^{-1}))$ as true.

Rule E.1.3

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

If the authorization layer returns true, then update the association and mechanism databases for both associations and write the establishment acknowledgment term.

The establishment responder rules are given below are in some sense the mirror image of the initiator rules.

Rule E.2.1

$$\begin{array}{rl}
\Phi^u \vdash_b & \downarrow_{\mathrm{eresp}(u,k_1)} , \\
& \Uparrow_{\mathrm{sec}(u)} \mathsf{P}(a, b, \mathsf{X}(\mathsf{Req}(s, d, u, \iota_a, \Xi^a, g)) \longrightarrow \\
& \downarrow_{\mathrm{auth}(u,k_2)} \mathsf{Ar}(a, b, s, b, \Phi^u, \Xi^a), \\
& \langle u, a, b, s, d, \iota_a, k_1, k_2 \rangle \\
& \text{new } \; k_2 \\
& \text{if CheckSig}(K_a^{-1}, g).
\end{array}$$

Upon the arrival of an establishment request message, the signature is verified and the authorization layer is invoked to verify that the initiator's credential Ξ^a satisfies the discovery policy Φ^u.

Rule E.2.2

$$\begin{array}{rl}
\Xi^b,\; \Xi^u \vdash_b & \langle u, a, b, s, d, \iota_a, k_1, k_2 \rangle, \\
& \uparrow_{\mathrm{auth}(k_2)} \mathsf{DisPol}(u, \mathrm{true}),\; \Sigma,\; \Pi^i \;\longrightarrow \\
& \Sigma \cup \mathsf{In}(a, \iota_b), \\
& \mathsf{Mech}(d \to s : u : \mathsf{Bndl}[\mathsf{In}(a, \iota_b)]) \otimes \Pi^i, \\
& \downarrow_{\mathrm{sec}(u,k_3)} \mathsf{P}(b, a, \\
& \quad \mathsf{X}(\mathsf{Rep}(s, d, u, \iota_a, \iota_s, \Xi^u \cup \Xi^b \cup \{K_a \Rightarrow K_b\}, g'))), \\
& \langle u, a, b, s, d, \iota_a, \iota_b, k_1, k_3 \rangle \\
& \text{new } \; k_3 \\
& \text{if } \exists \mathsf{In}(a, \iota_x) \in \Sigma \text{ then } \iota_b = i_x \text{ else } \iota_b \text{ is new} \\
& \text{where } g = \text{Sign}(K_b^{-1}).
\end{array}$$

Rule E.2.2 only executes if the authorization layer verifies that the discovery policy is satisfied. If there is an existing association flowing from the initiator to the responder, then it gets reused. Otherwise, a new association is generated. 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

$$\begin{array}{rl}
\vdash_b & \langle u, a, b, s, d, \iota_a, \iota_b, k_1, k_3 \rangle, \Sigma, \Pi^o, \uparrow_{\mathrm{sec}(k_3)} \;\longrightarrow \\
& \uparrow_{\mathrm{eresp}(k_1)} \mathsf{R}(a),\; \Sigma \cup \{\mathsf{Out}(a, \iota_a)\}, \\
& \mathsf{Mech}(s \to d : u : \mathsf{Bndl}[\mathsf{Out}(a, \iota_a)]) \otimes \Pi^o.
\end{array}$$

Upon acknowledgment that the reply has been sent, entries are made in the association and mechanism databases for the association flowing from b to a.

4.3 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' is a multiset of terms in M matching L and R' is a multiset matching the pattern R. 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** $\mathbf{X}(u)(L', R')(a)$ or

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

When the context is clear we drop the redux, contractum, and node. If all the rules belong to the same session, then we drop the session identifier as well and just write $M \xrightarrow{\mathbf{X}} M$. The sequential application of **Rules** $\mathbf{X}_1(u_1), \ldots, \mathbf{X}_n(u_n)$ to the multiset M_1 is written as

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

The sequence of multisets $M_1, M_2, \ldots, M_{n+1}$ is called a *trace* of the 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.

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 $\mathsf{F}(f)$).

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.

Suppose $T = M_1, \ldots, M_n$, we say $M_i \in T$ if M_i is identical to M_i in T and we say that for a term t, $t \in T$ if there exists a multiset $M_i \in T$ such that $t \in M_i$. We denote the set of elements appearing in a multiset M as $\mathfrak{L}(M)$.

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 a protocol. Yet the protocol is executing on a distributed network of nodes. A trace of the establishment protocol must record that the establishment initiator has sent the request message before it records that the message has been received at the establishment responder and it must record that the reply has been sent by the establishment responder and received by the establishment initiator before the establishment initiator writes state. This is due to the causal ordering induced by the messages [84]. 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 [35] 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 [95]. 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. This is formally stated as

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

where $F(f), \Sigma, \Pi^i$, and Π^o represent all possible terms of that form. The infinite multiset of all the state elements is defined as

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

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 $\mathbf{X} \parallel \mathbf{Y}$.

4.4 Putative Properties

In this section, we formulate and prove a collection of propositions that express properties of the tunnel calculus that we would expect to be true. Throughout this section, we assume that none of the nodes act maliciously, but obey the rules of the protocol.

The first property asserts the uniqueness of acknowledgment identifiers and follows from inspecting the rules of the tunnel calculus and observing that acknowledgment operators are always generated by the *new* operator.

Proposition 4.1 (Uniqueness of Identifiers) *Consider a trace $T = M_1, M_2, \ldots, M_n$. Let t be a term that begins with $\downarrow_{I(u,k)}$ or $\downarrow_{\mathrm{ip}(k)}$, where $I = \{\mathrm{sec}, \mathrm{auth}, \mathrm{eresp}, \mathrm{est}\}$. If $t \notin M_{i-1}$ and $t \in M_i$ and t' has the form $\downarrow_{\mathrm{ip}(k')}$ or $\downarrow_{I(u,k')}$, where $t' \in M_j (j > i)$ and $t' \notin M_{j-1}$, then $k' \neq k$.* □

In the tunnel calculus, all messages contain session identifiers. This restriction could have been weakened to accommodate traffic existing outside of our convention that all traffic travels in a tunnel or is part of a protocol setting up a tunnel, but at a cost in complexity that we felt obscured the focus of the work.

Proposition 4.2 (Messages Contain Session Identifiers.) *Consider a trace $T = M_1, \ldots, M_n$ that records the execution of any of our tunnel calculus rules. If packet $p \in T$ and $p \notin \mathfrak{L}(M_1)$, then there exists a session identifier u such that $u \in p$. That is, u appears in packet p.*

Proof: From inspection of the rules we can see that all messages sent in an association contain a session identifier in the header. Establishment packets do not travel in an association, but both the establishment request and establishment response messages contain the session identifier. Discovery packets also contain the session identifier. Since these are the only packets allowed in our system, the theorem holds. □

The following property 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. Note that this is not the case for rewrite systems in general, but the tunnel calculus was designed with such analysis in mind.

Lemma 4.3 *If $M \longrightarrow M'$, then there exists only one rule* ***X*** *whose application to M ($M \xrightarrow{\boldsymbol{X}} M'$) could have produced M'.*

Proof: Recall that $M' = M - L \cup R$ where L and R are the redux and contractum of the rule in question. Knowing M and M' allows us to discern both the redux and contractum. All that remains to show is that the redux and contractum pair can only match one possible rule.

In many cases, the left-hand side of a rule has a pattern that distinguishes it from all the other rules. In such cases, one can deduce the rule that was applied in $M \longrightarrow M'$ by observing the term(s) that were removed from M in the application of the rule.

Rules F.1.1, **S.1.1**, and **E.1.1** all have the form of a left-hand side with a single $\downarrow$-term of the form $\downarrow_{\mathrm{ip}(k)}$, $\downarrow_{\mathrm{sec}(u,k)}$, or $\downarrow_{\mathrm{est}(u,k)}$. No other rules have these terms on the left-hand side. If $M \longrightarrow M'$ removes one of these terms, then we know the rule that was applied respectively.

Rule F.2.1 is the only rule with a packet on the left-hand side. If $M \longrightarrow M'$ removes a packet from M, then we know that it was an application of **Rule F.2.1**.

Rule E.2.1 is the only rule with a $\downarrow_{\mathrm{eresp}(u,k)}$ term and a $\Uparrow_{\mathrm{sec}(u)}$ term on the left hand side. If $M \longrightarrow M'$ removes two terms of this form, then it is an application of **Rule E.2.1**.

The left-hand side of **Rule E.1.2** is a resumption term of type

$$\mathit{Session} \times \mathit{Addr} \times \mathit{Addr} \times \mathit{Addr} \times \mathit{Addr} \times \mathit{Identifier} \times \mathit{Identifier} \times \mathit{SPI},$$

a $\uparrow_{\mathrm{sec}(k)}$ term, and a $\Uparrow_{\mathrm{sec}(u)}$ term. Although **Rule E.2.2** also has an eight-tuple on the left-hand side, this tuple has a different type. The other terms on the left-hand of **Rule E.2.2** differ from those in **Rule E.1.2**. No other rule has a pattern on the left-hand side that is similar to that of **Rule E.1.2**. Consequently, if $M \longrightarrow M'$ removes three terms of the form in **Rule E.1.2** from M, then we can conclude that it is an application of that rule.

The reasoning for **Rules E.1.3, E.2.2, S.1.2, S.2.6** and **E.2.3** is similar to that for the previous rule.

The four rules that compose the secure layer responder are more complex than the others we have considered. **Rules S.2.1**, **S.2.2**, and **S.2.3** each have the term $\Uparrow_{\text{ip}} p$ on the left of the arrow. Although each rule also contains terms on the left of the turnstile, these terms are not actually removed from the multiset and thus cannot be of help in identifying the rule that was applied. Conditionals and pattern matching determine the rule that gets executed. If the only term removed from the multiset in $M \longrightarrow M'$ is of the form $\Uparrow_{\text{ip}} p$, we can determine the rule that was applied by the shape of the packet p.

- If p is an establishment packet, then it was an application of **Rule S.2.1**.
- If p is a control packet, then it was an application of **Rule S.2.2**.
- Otherwise, it was an application of **Rule S.2.3**.

Rules S.2.4 and **S.2.5** both have a left-hand side consisting of a single resumption term composed of a packet, a bundle, and a session identifier. So it is not possible to determine the rule that was invoked by solely observing the term that was removed from the multiset. Since the right-hand side of the two rules are sufficiently different, one can deduce the rule that was applied, but observing both the terms removed and added by $M \longrightarrow M'$. If the resumption term is removed from the multiset and a $\Uparrow_{\text{sec}(u)}$ term is added the multiset, then $M \longrightarrow M'$ is an application of **Rule S.2.4**. If the resumption term is removed from the multiset and a $\downarrow_{\text{sec}(u,k)}$ term is added to the multiset, then $M \longrightarrow M'$ is an application of **Rule 2.5**. $\Box$

Given a trace $T = M_1, M_2, \ldots, M_n$, in which term t appears. Define $T \dagger (t)$ to be the index of the multiset in which the term t first appears. We assume that $T \dagger (t)$ asserts the existence of $t \in T$. Define the preorder $t_1 \prec_T t_2$ on terms as follows:

$$t_1 \prec_T t_2 \overset{def}{=} t_1, t_2 \notin M_1 \text{ and } T \dagger (t_1) < T \dagger (t_2) \text{ and } t_1 \notin M_j,$$

where $j = T \dagger (t_2)$. This says that the first occurrence of term t_1 in a trace occurs before the first occurrence of term t_2 and that t_1 was consumed before t_2 was produced. We drop the subscript and write $t_1 \prec t_2$ instead of $t_1 \prec_T t_2$ when the context is clear.

The following is a property of the forwarding layer and says that if a forwarding layer packet is received at a node, then it was sent from some node and that the $\downarrow_{\text{ip}(k)}$ term is consumed before the corresponding acknowledgment and receive terms appear in the trace.

Proposition 4.4 *Consider a trace $T = M_1, \ldots, M_n$, if $T \dagger (\Uparrow_{\mathrm{ip}} p \,@\, b) \;=\; j+1$, then the following holds:*

$$\begin{aligned} \downarrow_{\mathrm{ip}(k)} p \,@\, a \;&\prec_T\; p \,@\, b && (4.1)\\ p \,@\, b \;&\prec_T\; \Uparrow_{\mathrm{ip}} p \,@\, b && (4.2)\\ \downarrow_{\mathrm{ip}(k)} p \,@\, a \;&\prec_T\; \uparrow_{\mathrm{ip}(k)} \;@\, a, && (4.3) \end{aligned}$$

where $\downarrow_{\mathrm{ip}(k)} p \,@\, a,\;\; p \,@\, b,\; \Uparrow_{\mathrm{ip}} p \,@\, b,\; \downarrow_{\mathrm{ip}(k)} p \,@\, a \notin \mathfrak{L}(M_1)$.

Proof: It follows from the assumption and inspection of the rules that $M_j \longrightarrow M_{j+1}$ must be an application of **Rule F.2.1** ($M_j \xrightarrow{\textbf{F.2.1}} M_{j+1}$), which consumes a $p \,@\, b \in M_j$ and rewrites a $\Uparrow_{\mathrm{ip}} p \,@\, b$ term in M_{j+1}; hence, the ordering expressed in (4.2) holds. It also follows from the tunnel calculus rules that an application of **Rule F.1.1** must have been occurred to produce the $p \,@\, b$ term and that this must have occurred in a prefix of $M_1, \ldots, M_j$, say at $M_i \xrightarrow{F.1.1} M_{i+1}$ $(i < j)$, where a term of the form $\downarrow_{\mathrm{ip}(k)} p \,@\, a$ was removed from the multiset and a term of the form $p \,@\, b$ was written to the multiset along with a $\uparrow_{\mathrm{ip}(k)} \;@\, a$ term. Hence, we can conclude that the orderings (4.1) and (4.3) are true. □

We now examine two properties of the secure processing layer. The first property says that the system is constructed so that if a destructor was applied to a packet, then the corresponding constructor must have been applied.

Proposition 4.5 *Consider a trace $T = M_1, \ldots, M_n$, where M_1 is a virgin network. If the trace records the application of a destructor $\mathsf{In}(a, \iota)$ to packet p' at node b via the application of a rule $M_i \xrightarrow{X_i} M_{i+1}$, where X_i is **Rule S.2.1**, **S.2.2**, or **S.2.3**, then the trace must have recorded $M_j \xrightarrow{S.1.1} M_{j+1}$ that applies the corresponding $\mathsf{Out}(b, \iota)$ constructor, where $j < i$.*

Proof: Rules S.2.1, S.2.2, and **S.2.3** all call the strip function. The third equation of the strip function must have been applied because this is the only rule that actually applies a destructor. This equation only executes if the packet has a secure header with SPI value ι and the matching destructor is in the mechanism database. Consequently, the matching constructor must have been applied, but inspecting the rules of the tunnel calculus we see that only can only be done by the application of the bundle and nest functions called in **Rule S.1.1**. □

The second secure layer property says that if a secure message is received at a node, then it was sent from some node at some point in the past and that the $\downarrow_{\mathrm{sec}(u,k)}$ term is consumed before the corresponding acknowledgment and receive terms appear in the trace.

Proposition 4.6 *Consider a trace* $T = M_1, \ldots, M_n$, *if* $T \dagger (\Uparrow_{\mathrm{sec}(u)} p \mathbin{@} b) = j+1$, *then the following holds:*

$$\downarrow_{\mathrm{sec}(u,k)} p \mathbin{@} a \prec_T \Uparrow_{\mathrm{sec}(u)} p \mathbin{@} b \tag{4.4}$$

$$\downarrow_{\mathrm{sec}(u,k)} p \mathbin{@} a \prec_T \uparrow_{\mathrm{sec}(k)} \mathbin{@} a, \tag{4.5}$$

where $\downarrow_{\mathrm{sec}(u,k)} p \mathbin{@} a$, $\Uparrow_{\mathrm{sec}(u)} p \mathbin{@} b$, $\uparrow_{\mathrm{sec}(k)} \mathbin{@} a \notin \mathfrak{L}(M_1)$.

Proof: It follows from inspection of the rules that $M_j \longrightarrow M_{j+1}$ must be an application of **Rules S.2.1, S.2.2,** or **S.2.4**. **Rules S.2.1** and **S.2.2** directly consume a $\Uparrow_{\mathrm{ip}} p'$ term and produce the $\Uparrow_{\mathrm{sec}(u)} p \mathbin{@} b$ term, where p is the packet returned by calling strip on p'. If $M_j \longrightarrow M_{j+1}$ is an application of **Rule S.2.4**, then it must have consumed a resumption term produced by some previous application of **Rule S.2.3**, which consumed a $\Uparrow_{\mathrm{ip}} p'$, where p is the packet returned by calling strip on p'. So in each of these three cases $\Uparrow_{\mathrm{ip}} p' \prec_T \Uparrow_{\mathrm{sec}(u)} p$. From proposition 4.4, we can conclude that there had to have been a $\downarrow_{\mathrm{ip}(k')} p' \mathbin{@} a$ term written to the multiset such that $\downarrow_{\mathrm{ip}(k')} p' \mathbin{@} a \prec_T \Uparrow_{\mathrm{ip}} p' \mathbin{@} b$. Given that we have assumed that all messages are sent via the secure processing layer, we can conclude that the trace must record an instance **Rule S.1.1** producing the aforementioned $\downarrow_{\mathrm{ip}(k')} p' \mathbin{@} a$ term and consuming a $\downarrow_{\mathrm{sec}(u,k)} p \mathbin{@} a$ term so

$$\downarrow_{\mathrm{sec}(u,k)} p \mathbin{@} a \prec_T \downarrow_{\mathrm{ip}(k')} p' \mathbin{@} a.$$

Hence

$$\downarrow_{\mathrm{sec}(u,k)} p \mathbin{@} a \prec_T \downarrow_{\mathrm{ip}(k')} p' \mathbin{@} a \prec_T \Uparrow_{\mathrm{ip}} p' \mathbin{@} b \prec_T \Uparrow_{\mathrm{sec}(u)} p \mathbin{@} b,$$

and the ordering (4.4) follows from transitivity.

Given that **Rule S.1.1** will have removed the $\downarrow_{\mathrm{sec}(u,k)} p \mathbin{@} a$ term from the multiset when the packet is sent down to the forwarding layer and the $\uparrow_{\mathrm{sec}(k)} \mathbin{@} a$ term is not produced in **Rule S.1.2** until after the forwarding layer has sent the packet. Whence the term $\uparrow_{\mathrm{sec}(k)} \mathbin{@} a$ is not produced until after the $\downarrow_{\mathrm{sec}(u,k)} p \mathbin{@} a$ is consumed; and we can conclude that the ordering (4.5) h. □

The following is a property of the establishment layer and says that if establishment request is received at a node, then it must have been sent by some node; and that if the establishment reply is received at a node, then an establishment reply was sent from a node that previously received an establishment request.

Proposition 4.7 *Suppose* $T = M_1, \ldots, M_n$ *records the execution of the establishment protocol with node* a *as initiator and node* b *as the responder executing in session* u, *where* M_1 *is a virginal network and* $M_1 \overset{E.2.1(u)(b)}{\longrightarrow} M_2$ *and* $M_2 \overset{E.1.1(u)(a)}{\longrightarrow} M_3$. *If*

$$T \dagger (\Uparrow_{\mathrm{sec}(u)} \mathsf{P}(a, b, \mathsf{X}(\mathsf{Req}(s, d, u, \iota_a, \Xi^a, g))) \mathbin{@} b) = i+1,$$

then the following holds:

$$\begin{array}{rl} & \downarrow_{\mathrm{sec}(u,k)} \mathsf{P}(a, b, \mathsf{X}(\mathsf{Req}(s, d, u, \iota_a, \Xi^a, g))) \,@\, a \\ \prec_T & \Uparrow_{\mathrm{sec}(u)} \mathsf{P}(a, b, \mathsf{X}(\mathsf{Req}(s, d, u, \iota_a, \Xi^a, g))) \,@\, b. \end{array} \tag{4.6}$$

If

$$T \dagger (\Uparrow_{\mathrm{sec}(u)} \mathsf{P}(b, a, \mathsf{X}(\mathsf{Rep}(s, d, u, \iota_a, \iota_b, \Xi^u, g'))) \,@\, a) = j + 1,$$

then the following holds:

$$\begin{array}{rll} & \Uparrow_{\mathrm{sec}(u)} \mathsf{P}(a, b, \mathsf{X}(\mathsf{Req}(s, d, u, \iota_a, \Xi^a, g))) \,@\, b & \\ \prec_T & \downarrow_{\mathrm{sec}(u,k')} \mathsf{P}(b, a, \mathsf{X}(\mathsf{Rep}(s, d, u, \iota_a, \iota_b, \Xi^u, g'))) \,@\, b & (4.7) \\ \prec_T & \Uparrow_{\mathrm{sec}(u)} \mathsf{P}(b, a, \mathsf{X}(\mathsf{Rep}(s, d, u, \iota_a, \iota_b, \Xi^u, g'))) \,@\, a. & (4.8) \end{array}$$

where

$$\left.\begin{array}{l} \downarrow_{\mathrm{sec}(u,k)} \mathsf{P}(a, b, \mathsf{X}(\mathsf{Req}(s, d, u, \iota_a, \Xi^a, g))) \,@\, a \\ \Uparrow_{\mathrm{sec}(u)} \mathsf{P}(a, b, \mathsf{X}(\mathsf{Req}(s, d, u, \iota_a, \Xi^a, g))) \,@\, b \\ \downarrow_{\mathrm{sec}(u,k')} \mathsf{P}(b, a, \mathsf{X}(\mathsf{Rep}(s, d, u, \iota_a, \iota_b, \Xi^u, g'))) \,@\, b \\ \Uparrow_{\mathrm{sec}(u)} \mathsf{P}(b, a, \mathsf{X}(\mathsf{Rep}(s, d, u, \iota_a, \iota_b, \Xi^u, g'))) \,@\, a \end{array}\right\} \notin \mathfrak{L}(M_1).$$

Proof: It follows from inspection of the rules that $M_i \longrightarrow M_{i+1}$ and $M_j \longrightarrow M_{j+1}$ must have been applications of **Rules E.2.2** and **E.1.2** respectively. The ordering (4.6) follows directly from proposition 4.6. It follows from inspection of the rules that **Rule E.2.1** must be executed before **Rule E.2.2** is executed. That is the establishment request must have been received before the establishment reply is sent so the ordering (4.7) holds. The ordering (4.8) follows from proposition 4.6. □

4.5 Conclusion

In this chapter, we have presented a formal definition of the tunnel calculus, which provides an operational semantics for a protocol stack including the processing associated with security tunnels. Among the layers of the tunnel calculus are an authorization layer, providing a SPKI like calculus for expressing polices, credentials, and satisfaction, and an establishment layer that models the up a pair-wise tunnel. We also introduced a trace theory induced by the tunnel-calculus semantics. In addition to the descriptive part of the chapter, we prove a number of basic propositions about the system. With the exception of the authorization layer and several of the putative properties, the work reported in this chapter appeared in [53].

Although we asserted in Chapter 3 that session identifiers prevent establishment deadlock, there was no mathematical proof that this was indeed true. This is carried out in the next chapter using the machinery of the tunnel calculus developed here. Subsequent chapters apply the tunnel calculus for reasoning about the functional correctness properties of discovery protocols and reasoning about their vulnerability to denial of service.

Chapter 5

Noninterference and Progress

The tunnel calculus presented in the previous chapter incorporates the concept of session identifier that we introduced in Chapter 3 in order to avoid establishment deadlock. In this chapter, we prove that the deadlock in question is indeed avoided. The chapter is organized as follows. We first prove that the application of two rules in distinct sessions are independent in the sense that we defined in the previous chapter. Next, we formulate and prove the session matching property that says that the packet filters in the MDB only match packets in the specified session. This is followed by a simulation lemma, which enables us to prove the equivalence of traces. We then introduce noninterference and progress theorems that together say that the execution of an establishment protocol in one session does not interfere with the sending and receiving of messages in another establishment session.

5.1 Independence Between Sessions

In this section we demonstrate a lemma that says that the redux and contractum of the application of rules executing in different sessions are disjoint. A consequence of this lemma is the independence between sessions theorem that says that two operations executing in different sessions are independent.

Let **Rule** $\mathbf{X}(u)$ and **Rule** $\mathbf{Y}(v)$ denote any two rules in the tunnel calculus executing in sessions u and v respectively.

Lemma 5.1 *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*

$$(L_1 - \mathcal{H}(a)) \cap (L_2 - \mathcal{H}(b)) = \emptyset \quad \textit{and} \tag{5.1}$$
$$(L_1 - \mathcal{H}(a)) \cap (R_2 - \mathcal{H}(b)) = \emptyset. \tag{5.2}$$

That is, neither the execution of **Rule** $\mathbf{X}(u)$ nor **Rule** $\mathbf{Y}(v)$ will consume terms that would otherwise have been consumed by the execution of the other unless those terms represent shared state. The proof that follows is composed of two components that correspond to the case where $a \neq b$ and the case where $a = b$. In the first case, only the rules of the forwarding layer need be considered. In the second case, each of the rules must be considered.

Proof: Assume that $a \neq b$. If neither $\mathbf{X}(u)(a)$ nor $\mathbf{Y}(v)(b)$ is an instance of **Rule F.1.1**, then the node terms in their reduxes and contractums are located at different nodes and are hence disjoint, and (5.1) and (5.2) hold. Consequently, we need only be concerned about the case where at least one of $M_i \longrightarrow M_{i+1}$ or $M_j \longrightarrow M_{j+1}$ is an application of **Rule F.1.1**. Suppose $M_j \overset{\mathbf{Y}(v)(b)}{\longrightarrow} M_{j+1}$ is an application of **Rule F.1.1**. If $\mathbf{X}(u)(a)$ is not a rule in the forwarding layer, then the redux and contractum of $\mathbf{X}(u)(a)$ is disjoint from the redux and contractum of $\mathbf{Y}(v)(b)$ and (5.1) and (5.2) hold. If $\mathbf{X}(u)(a)$ is also an instance of **Rule F.1.1**, then their redux are disjoint since $a \neq b$, and thus (5.1) holds. The right-hand side of **Rule F.1.1** is just a packet and has a pattern that is distinctly different from the from the left-hand side of the same rule so (5.2) holds. Suppose $\mathbf{X}(u)(a)$ is an instance of **Rule F.2.1**. If the application of $\mathbf{Y}(\textbf{Rule F.1.1}(v)(b))$ moves the packet to a node other than a, then (5.1) and (5.2) hold. Otherwise, it is possible that an application of $\mathbf{X}(\textbf{Rule F.2.1}(u)(a))$ consumes the packet produced by an application of $\mathbf{Y}(\textbf{Rule F.1.1}(v)(b))$. From proposition 4.2 the packet that gets consumed must contain the session identifier v. Since the session identifier in R_2 is v, the term must belong to session v, but this contradicts the assumption that $\mathbf{X}$ $(\textbf{Rule F.2.1}(u)(a))$ is executing in session u. Hence the lemma holds if $a \neq b$.

Assume $a = b$. The proof proceeds by case analysis on the rules of the tunnel calculus. Let

$$\begin{aligned} L_1' &= L_1 - \mathcal{H}(a) \\ R_1' &= R_1 - \mathcal{H}(a) \\ L_2' &= L_2 - \mathcal{H}(a) \\ R_2' &= R_2 - \mathcal{H}(a). \end{aligned}$$

Rather than inspect each rule we analyze the structure of L_1' and demonstrate that for all L_2' that may be consumed by $M_j \longrightarrow M_{j+1}$ in v, $L_1' \cap L_2' = \emptyset$. It is also shown that $L_1' \cap R_2' = \emptyset$, where R_2' is any contractum produced by the application of rule $\mathbf{Y}$ in session v.

Consider the case where L_1' is of the following form:

$\downarrow_{\mathrm{ip}(k)}$ **(Rule F.1.1).** If $M_j \overset{\mathbf{Y}(v)}{\longrightarrow} M_{j+1}$ is not an application of **Rule F.1.1**, then $L_1' \cap L_2' = \emptyset$ since no other rule has a $\downarrow_{\mathrm{ip}(k)}$ term on the left-hand side. We must show that terms that would otherwise have been consumed by $M_j \overset{\textbf{F.1.1}(v)}{\longrightarrow} M_{j+1}$ are not consumed by $M_i \overset{\textbf{F.1.1}(u)}{\longrightarrow} M_{i+1}$ and vice versa. Now $M_i \overset{\textbf{F.1.1}(u)}{\longrightarrow} M_{i+1}$ can only consume terms containing session identifier u. If $M_j \xrightarrow{\mathbf{Y}(v)} M_{j+1}$ is an instance of **Rule F.1.1**, then the two have a left-hand side of the same form. From proposition 4.2, the packets in the respective $\downarrow_{ip}\, p$ terms contain their respective session identifiers. If $M_i \longrightarrow M_{i+1}$ were to consume a $\downarrow_{ip(k)}$ term containing session identifier v, then the term would belong to session v and $M_i \xrightarrow{\mathbf{F.1.1}(u)} M_{i+1}$ would be executing in session v, contradicting the assumption that it is executing in session u. Likewise, $M_j \xrightarrow{\mathbf{F.1.1}(v)} M_{j+1}$ will not consume a $\downarrow_{ip(k)}$ term for session u. So $L_1' \cap L_2' = \emptyset$ and (5.1) holds.

To show that (5.2) holds we only need consider the case where $M_j \xrightarrow{\mathbf{Y}(v)} M_{j+1}$ is an instance of **Rule S.1.1**. This rule produces a $\downarrow_{\mathrm{ip}(k)}\, p$ term that may be consumed by $M_i \longrightarrow M_{i+1}$. But given that the packet must contain the session identifier, if $M_i \xrightarrow{\mathbf{F.1.1}(u)} M_{i+1}$ consumes a $\downarrow_{\mathrm{ip}(k)}\, p$ term from session v, then $M_i \xrightarrow{\mathbf{F.1.1}(u)} M_{i+1}$ would be executing in session v, contradicting the fact that $M_i \xrightarrow{\mathbf{X}(u)} M_{i+1}$ is in session u. Whence $L_1' \cap R_2' = \emptyset$ and (5.2) holds.

$\downarrow_{I(u,k)}$**, where** $I = \{\mathrm{sec}, \mathrm{est}\}$ **(Rules S.1.1 and E.1.1).** In order to show $L_1' \cap L_2' = \emptyset$, we need only consider the case where $M_i \xrightarrow{\mathbf{X}(u)} M_{i+1}$ and $M_j \xrightarrow{\mathbf{Y}(v)} M_{j+1}$ both consume terms of this form. If $M_i \longrightarrow M_{i+1}$ were to consume a $\downarrow_{I(v,k)}$ term containing session identifier v, then the term would belong to session v and $M_i \xrightarrow{\mathbf{X}(u)} M_{i+1}$ would be executing in session v contradicting the assumption that it is executing in session u. Likewise, $M_j \xrightarrow{\mathbf{Y}(v)} M_{j+1}$ will not consume a $\downarrow_{I(u,k)}$ term. So $L_1' \cap L_2' = \emptyset$ and (5.1) holds. Session v cannot produce a term with session identifier u due to the uniqueness of session identifiers and because such a term would belong to session u if it were produced. So session v cannot have produced a term $\downarrow_{I(u,k)}$ containing session identifier u and consumed by $M_i \longrightarrow M_{i+1}$ in u. It follows that $L_1' \cap R_2' = \emptyset$ and (5.2) holds.

$p\,@\,a$ **(Rule F.2.1).** The reasoning for this case is similar to the previous case.

$\downarrow_{\mathrm{eresp}(u,k)}\,, \Uparrow_{\mathrm{sec}(u)}\, p$ **(Rule E.2.1).** The reasoning for this case is similar to the previous cases.

$\langle \mathsf{P}(b,c,y), \beta, u\rangle$ **(Rule S.2.4, S.2.5).** Terms of this form are produced by the semantic functions, not by any rewrite rule, but assume that such a term was produced in v. If it had been consumed in L_1', then that rule would have executed in session v since that session identifier is part of the term. This contradicts the assumption that $M_i \longrightarrow M_{i+1}$ executes in session u. Hence, $L_1' \cap R_2' = \emptyset$ and (5.2) holds.

Suppose both $M_i \xrightarrow{\mathbf{X}(u)} M_{i+1}$ and $M_j \xrightarrow{\mathbf{Y}(v)} M_{j+1}$ are both an instance of one of the rules under consideration, then it is possible that an application of one could consume the term intended for the other, but the term contains the session identifier and then $M_i \xrightarrow{\mathbf{X}(u)} M_{i+1}$ would be in session u or $M_j \xrightarrow{\mathbf{Y}(v)} M_{j+1}$ would be in session u contradicting our assumption that $M_i \longrightarrow M_{i+1}$ is executing in session u and $M_j \longrightarrow M_{j+1}$ is executing in session v. Hence $L'_1 \cap L'_2 = \emptyset$ and (5.1) holds.

$\Uparrow_{\mathrm{ip}} p$.**(S.2.1, S.2.2, S.2.3).** Although the forwarding layer has no concept of session, each of the messages sent in a session does contain a session identifier as shown in proposition 4.2. We have assumed $M_i \longrightarrow M_{i+1}$ is in session u and $M_j \longrightarrow M_{j+1}$ is in v. Suppose $M_i \longrightarrow M_{i+1}$ consumes a $\Uparrow_{\mathrm{ip}} p$ term belonging to session v, then $M_i \longrightarrow M_{i+1}$ must be in session v, but this contradicts our assumption that it was in session u. Similarly $M_j \longrightarrow M_{j+1}$ cannot consume a term belonging to u. Hence $L_1 \cap L_2 = \emptyset$ and (5.1) holds. **Rule F.2.1** is the only rule that produces a $\Uparrow_{\mathrm{ip}} p$. From proposition 4.2 we know that the term produced by an application of the rule contains the session identifier v. So if $M_i \xrightarrow{\mathbf{X}(u)} M_{i+1}$ consumes the $\Uparrow_{\mathrm{ip}} p$ term, then $M_i \longrightarrow M_{i+1}$ is executing in session v rather than in session u, which contradicts our assumption. Hence $L_1 \cap R_2 = \emptyset$ and (5.2) holds.

$\langle \ldots \rangle, l_2, \ldots, l_n$ **(Rules S.1.2, S.2.6, E.1.2, E.1.3, E.2.2, E.2.3.)** Among the elements in the resumption term are the session identifier and one or more acknowledgment identifiers. The l_i terms on the left side of the rule have the form $\uparrow_{I(k)}, \Xi, \Theta, \text{or} \Uparrow_{\mathrm{sec}(u)} p$, where $I = \{\mathrm{ip}, \mathrm{sec}, \mathrm{auth}, \mathrm{eresp}, \mathrm{est}\}$. We must show $l_i \cap L'_2 = \emptyset$ and $l_i \cap R'_2 = \emptyset$.

In the following, we only examine the case where $l_i = \uparrow_{I(k)}$. Inspecting the rules we see that $\uparrow_{I(k)}$ terms only appear in conjunction with resumption terms that contain both the session identifier and the acknowledgment identifier k. For a match to occur the value of session identifier k in the $\uparrow_{I(k)}$ term must match the value in the resumption term. It follows from the Uniqueness of Acknowledgment Identifiers 4.1 that sessions u and v cannot generate the same acknowledgement identifier value and hence no term produced in session v will contain an acknowledgment identifier matching k. So $l_i \cap R'_2 = \emptyset$ and (5.2) holds. From the Uniqueness of Acknowledgment Identifiers 4.1 we can conclude that the acknowledgment identifiers appearing in the resumption terms in the redux of $M_j \xrightarrow{\mathbf{Y}(v)} M_{j+1}$ must differ from those that appear in the resumption term in $M_i \xrightarrow{\mathbf{X}(u)} M_{i+1}$. Consequently, $\uparrow_{I(k)}$ terms are only consumed by rules executing in the session that generated the acknowledgment identifier. Hence $l_i \cap L'_2 = \emptyset$ and (5.1) holds.

The lemma follows from the above case analysis. □

The following is implied by Lemma 5.1, the definition of $\prec$ and the definition of independence.

Corollary 5.2 (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 $\boldsymbol{X}(u)(a) \parallel \boldsymbol{Y}(v)(b)$.* $\square$

To understand these two results, consider the application of rules $\mathbf{X}_1(u)$, $\mathbf{X}_2(u)$, and $\mathbf{X}_3(u)$ in session u and $\mathbf{Y}_1(v)$, $\mathbf{Y}_2(v)$, and $\mathbf{Y}_3(v)$ in session v. Suppose the rules have the following dependencies:

$$\mathbf{X}_1(u) \prec \mathbf{X}_2(u) \prec \mathbf{X}_3(u)$$
$$\mathbf{Y}_1(v) \prec \mathbf{Y}_2(v) \prec \mathbf{Y}_3(v).$$

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 $\mathbf{X}_2, \mathbf{X}_1, \mathbf{Y}_3, \mathbf{X}_3, \mathbf{Y}_2, \mathbf{Y}_1$.

5.2 Session Matching Property

Recall that the defect exposed in Section 3.3 arose because packets from one session match the packet filters installed during establishment for another session. The modifications to the session layer incorporated into the tunnel calculus purportedly prevented this from occurring. The session matching property demonstrates that this is indeed so.

Lemma 5.3 (Session Matching Property) *Let $T = M_1, \ldots, M_n$ be a trace and assume session u is active in T. Suppose $M_i \longrightarrow M_{i+1}$ is an application of **Rule S.1.1**(u), where the outbound message being processed is the term*

$$\downarrow_{\mathrm{sec}(u,k)} \mathsf{P}(b, c, y) \,@\, a.$$

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

Suppose $M_i \longrightarrow M_{i+1}$ is an application of one of the following:

- ***Rule S.2.1**(u), where the inbound message is the term $\mathsf{P}(b, c, \mathsf{X}(u, m))$*
- ***Rule S.2.2**(u), where the inbound message is the term $\mathsf{P}(b, c, \mathsf{C}(u, m))$*
- ***Rule S.2.4**(u), where the inbound message is the term $\mathsf{P}(b, c, \mathsf{S}(u, \iota, p))$*
- ***Rule S.2.5**(u), where the inbound message is the term $\mathsf{P}(b, c, \mathsf{S}(u, \iota, p))$.*

If the rule executes due to a matching entry in the inbound mechanism database Π^i*, then the matching database entry must have the form*

$$\mathsf{Mech}(\psi : u : \beta^i),$$

where $b \longrightarrow c \in \psi$*.*

Proof: Consider the case of inbound and outbound traffic separately.

Outbound traffic. Suppose the term $\downarrow_{\sec(u,k)} \mathsf{P}(b, c, y)$ is in the multiset at M_i and consumed by an application of the specified rule at $M_i \longrightarrow M_{i+1}$, where message y is a control packet, an exchange, or a regular data packet. **Rule S.1.1** uses the semantic function BndlSel to produce the bundle that is applied to the outgoing message. The function BndlSel will return the bundle β of an entry $\mathsf{Mech}(b \to c : u : \beta) \in \Pi^o$. If there is no entry with a matching session number, then BndlSel returns $\mathsf{Bndl}[]$, but since there was no match the property does not apply.

Inbound traffic. Suppose the term $\Uparrow_{\text{ip}} p @ e$ is in the multiset M_i and consumed by one of the specified rules at $M_i \longrightarrow M_{i+1}$. The semantic function Strip removes the headers with destination field e. It also checks that these are valid associations. The value returned by strip will determine what rule executes.

Suppose strip returns $\mathsf{Exchange}(\mathsf{P}(b, c, \mathsf{X}(u, m)), \beta)$. **Rule S.2.1** applies, but it will only execute if either there is an entry in Π^i with source-destination field $b \longrightarrow c$ and session u or the value β returned was $[]$ and there is no matching entry in Π^i. The first disjunct ensures that a packet arriving in a tunnel has a matching filter entry in Π^i and that the source, destination, and session identifiers match. Thus there will be no match if the packet and mechanism entry do not have a matching session identifier. The second disjunct ensures that an establishment packet arriving at a node and not traveling in a tunnel gets processed anyway, as expected, but it is not processed as belonging to a different session; so the property still holds.

Suppose strip return $\mathsf{ConMsg}(\mathsf{P}(b, c, \mathsf{C}(u, m)), \beta)$, then **Rule S.2.2** applies. This case is similar to that for exchange packets.

Suppose that strip returns $(\mathsf{P}(b, c, y), u, \beta)$, then either rule **S.2.4** or **S.2.5** will execute. In either case, the rule only executes if there is an entry $\mathsf{Mech}(b \to c : u : \beta) \in \Pi^i$.

The property follows from the cases shown above. □

This result tells us that the tunnel calculus was designed so that a message in one session will not be processed by a different session's MDB entry.

5.3 Trace Equivalence

We define what it means for two terms and two multisets to be equivalent. Since acknowledgment and SPI values are generated via an application of the *new* operator, these values will differ in every trace. Instead, the observer must verify that the two traces record the same messages sent and delivered for u up to α-equivalence of SPI and acknowledgment identifier values. 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 in t such that $t\rho \equiv t'$. Given a multiset M, $M\rho = \{\!|t\rho | t \in M|\!\}$ and $M_1 \sim M_2$ if and only if there exists a substitution ρ such that $M_1\rho = M_2$.

Let $\overline{M}$ denote a sequence of multisets $\overline{M} = M_1, \ldots, M_n$. Let intersection distribute over a sequence of multisets so that $\overline{M} \cap \mathcal{V} = M_1 \cap \mathcal{V}, \ldots, M_n \cap \mathcal{V}$ for some multiset of terms $\mathcal{V}$.

Noninterference among sessions is defined in terms of 'observable' activity. In this case, the observables are the secure messages sent and received during a session. Define $\mathcal{Q}(u)$ to be the infinite set 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. The values of a, b, y, and k can take any legal value.

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

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

and

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

where $M_1 \sim M_1'$. Suppose the traces $T = M_1, M_2, M_3$ and $T' = M_1', M_2', M_3'$ both record the same v messages modulo the aforementioned α-equivalence. If a term $t \in \mathcal{Q}(v)$ was produced by $M_2 \xrightarrow{\mathbf{Y}(v)} M_3$ and a corresponding term t' was produced by $M_1' \xrightarrow{\mathbf{Y}(v)} M_2'$, where $t' \sim t$, then $T \cap \mathcal{Q}(v) = \{\!|t|\!\}$ and $T' \cap \mathcal{Q}(v) = \{\!|t'|\!\}, \{\!|t'|\!\}$ because the t' term must still be in the multiset M_3' since the application of a rule in session v will not remove a u term. Although this satisfies our notion of the two traces containing the same messages in $\mathcal{Q}(v)$, $\{\!|t|\!\} \not\sim \{\!|t'|\!\}, \{\!|t'|\!\}$. The introduction of the filter 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. For example, $(\!|\emptyset, \{\!|1, 2, 2|\!\}, \{\!|1, 2, 2|\!\}, \{\!|3, 4|\!\}|\!) = \{\!|1, 2, 2|\!\}, \{\!|3, 4|\!\}$. So two traces are said to be v-observationally equivalent if $(\!|T \cap \mathcal{Q}(v)|\!) \sim (\!|T' \cap \mathcal{Q}(v)|\!)$.

5.4 Simulation

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

Lemma 5.4 (Simulation Lemma) *If $M_1 \xrightarrow{\boldsymbol{X}} M_2$ and $M_1 \sim M_1'$, then there exists M_2' such that $M_2 \sim M_2'$ and $M_1' \xrightarrow{\boldsymbol{X}} M_2'$.*

Proof: Apply **Rule X**$(u)(a)$ to M_1' yielding M_2'. Recall that the variables on the right-hand side of a rule must either appear in the left-hand side of a rule or the values must be generated using a *new* command. Given that $M_1 \sim M_1'$ and the fact that SPIs and acknowledgment identifiers are generated by *new*, we can conclude that M_2' only differs from M_2 in the SPI and acknowledgment identifier. Consequently, $M_2 \sim M_2'$. $\square$

5.5 Observational Commutativity

We have established that the operations that occur during two distinct sessions are independent. This means that if a session u runs concurrently with another session v, then there may be many possible legal traces reflecting different interleavings of the execution of two traces. If the session v does not interfere with session u, then from the point of view of an observer, session u should send and receive the same messages regardless of the interleaving of the activity in session u. Formulating this in terms of traces, we say that traces representing different interleavings of the two sessions record the same messages sent and received in session u.

We start by considering the case in which the trace records the execution of only two rules, where each rule is executing in a distinct session. This is generalized to a case where we only consider the first two operations in a longer trace. Finally, the general observational commutativity theorem is presented.

Lemma 5.5 *Suppose u and v are distinct session identifiers and*

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

and $M_1 \sim M_1'$. Then there exist M_2', M_3' such that

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

and

$$(\!|(M_1, M_2, M_3) \cap \mathcal{Q}(v)|\!) \sim (\!|(M_1', M_2', M_3') \cap \mathcal{Q}(v)|\!).$$

Proof: Let $T = M_1, M_2, M_3$ and $T' = M_1', M_2', M_3'$ where $M_1 \sim M_1'$.

Assume that terms representing shared state do not appear in **Rule X**(u) and **Rule Y**(v). It follows from Lemma 5.1 that the application of **Rule X**(u) will not consume terms in the redux of an application of **Rule Y**(v) and vice versa. Nor is any term in the contractum of an application of **Rule X**(u) in the redux of an application of **Rule Y**(v). It follows from Independence Between Sessions 5.2 that $\mathbf{X}(u) \parallel \mathbf{Y}(v)$.

Recall that the rules of the tunnel calculus have the structure where all variables on the right-hand side of a rule appear on the left-hand side or have their values generated by the *new* operator. Given that $M_1 \sim M_1'$, we can conclude that $\mathcal{Q}(v)$ terms in T and T' only differ by SPI and acknowledgement identifier values.

We now consider the situation where shared state is involved. First consider the case where sessions u and v interact by way of the association database. Suppose in trace T, $\mathbf{X}(u)$ writes state to the association database and $\mathbf{Y}(v)$ is an instance of **Rule E.1.1** or **Rule E.2.2** that reuses the SPI written by $\mathbf{X}(u)$. When the order of execution is swapped in T', rule $\mathbf{Y}$ will not reuse the SPI. So in one case, the SADB Σ has the single entry, say, $\mathsf{In}(a, \iota)$, and in the other case, Σ would have entries $\mathsf{In}(a, \iota)$ and $\mathsf{In}(a, \iota')$, but since these elements only differ by SPI value and the SADB is represented as a set, we can conclude that

$$\{\mathsf{In}(a, \iota)\} \sim \{\mathsf{In}(a, \iota), \mathsf{In}(a, \iota')\}.$$

The reasoning for other cases where shared state involves the SADB is similar. Consequently, $\mathcal{Q}(v)$ terms in T and T' will only differ by SPI and acknowledgement identifiers.

We now examine the case where the two sessions may interact by sharing the MDB. These databases only get updated via an application of **Rule E.1.4, E.2.3,** or **E.2.4**. Suppose **Rule X** or **Rule Y** is an application of one of these rules. It follows from reasoning similar to that above, if $\mathcal{Q}(v)$ terms appear in both T and T', then they only differ by SPI and acknowledgement identifiers. It remains to show that swapping the order of execution of $\mathbf{X}(u)$ and $\mathbf{Y}(v)$ will not interfere with the sending and receiving of messages in v. Assume that term $t' \in \mathcal{Q}(v)$ appears in $T' = M_1', M_2', M_3'$, but no corresponding term t ($t \sim t'$) appears in $T = M_1, M_2, M_3$. This means that a mechanism entry for session u has prevented a message from being sent or received in session v, but the Session Matching Property 5.3 prevents such 'interference'. The case where a term $t \in \mathcal{Q}(v)$ appears in T, but a corresponding term does not appear in T' is similarly averted.

Having established that the two trace sequences M_1, M_2, M_3 and M_1', M_2', M_3' contain $\mathcal{Q}(v)$ terms differing only in SPI and acknowledgment identifiers we can apply the following

$$(\!| M_1, M_2, M_3 \cap \mathcal{Q}(v) |\!) = M_1 \cap \mathcal{Q}(v), M_3 \cap \mathcal{Q}(v)$$

and

$$(\!| M_1', M_2', M_3' \cap \mathcal{Q}(v) |\!) = M_1' \cap \mathcal{Q}(v), M_2' \cap \mathcal{Q}(v).$$

to conclude

$$(\!| M_1, M_2, M_3 \cap \mathcal{Q}(v) |\!) \sim (\!| M_1', M_2', M_3' \cap \mathcal{Q}(v) |\!).$$

$\square$

The following *absorption property* is useful. Its proof is a straightforward application of the definition.

$$\begin{array}{rl} & (\!| M_1, \ldots, (\!| M_i, \ldots, M_j |\!), \ldots, M_n |\!) \\ = & (\!| M_1, \ldots, M_i, \ldots, M_j, \ldots, M_n |\!). \square \end{array}$$

The next result generalizes the previous lemma to a longer trace.

Lemma 5.6 *Let $T = M_1, \ldots, M_n$ be a trace*

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

where $u_1 \neq u_2$. Then there is a trace $T' = M'_1, \ldots, M'_n$, where $M_1 \sim M'_1$, that has the form

$$M'_1 \xrightarrow{\boldsymbol{X}_2(u_2)} M'_2 \xrightarrow{\boldsymbol{X}_1(u_1)} M'_3 \xrightarrow{\boldsymbol{X}_3(u_3)} \cdots \xrightarrow{\boldsymbol{X}_{n-1}(u_{n-1})} M'_n,$$

where $(\!|T \cap \mathcal{Q}(u_2)|\!) \sim (\!|T' \cap \mathcal{Q}(u_2)|\!)$.

Proof:

$$\begin{array}{ll}
 & (\!|T \cap \mathcal{Q}(u)|\!) \\
 & \text{Expanding definitions and filter absorption} \\
= & (\!|(\!|M_1, M_2, M_3 \cap \mathcal{Q}(u_2)|\!), M_4, \ldots, M_n \cap \mathcal{Q}(u_2)|\!) \\
 & \text{Swap } \mathbf{X}_1(u_1) \text{ and } \mathbf{X}_2(u_2) \\
 & \quad \text{then apply Lemma 5.5 and the Simulation Lemma 5.4 as needed.} \\
\sim & (\!|(\!|M'_1, M'_2, M'_3 \cap \mathcal{Q}(u_2)|\!), M'_4, \ldots, M'_n \cap \mathcal{Q}(u_2)|\!) \\
 & \text{Definitions and filter absorption} \\
= & (\!|T' \cap \mathcal{Q}(u_2)|\!).
\end{array}$$

□

We now present the Observational Commutativity Theorem that generalizes the previous result.

Theorem 5.7 (Observational Commutativity) *Let $T = M_1, \ldots, M_n$ be a trace*

$$\begin{array}{l} M_1 \xrightarrow{\boldsymbol{X}_1(u_1)} M_2 \xrightarrow{\boldsymbol{X}_2(u_2)} M_3 \xrightarrow{\boldsymbol{X}_3(u_3)} \cdots \xrightarrow{\boldsymbol{X}_{i-1}(u_{i-1})} M_i \xrightarrow{\boldsymbol{X}_i(u_i)} \\ M_{i+1} \xrightarrow{\boldsymbol{X}_{i+1}(u_{i+1})} \cdots \xrightarrow{\boldsymbol{X}_{n-1}(u_{n-1})} M_n, \end{array}$$

where $u_i \neq u_1, \ldots, u_{i-1}$. Then there is a trace $T' = M'_1, \ldots, M'_n$, where $M_1 \sim M'_1$ and

$$\begin{array}{l} M'_1 \xrightarrow{\boldsymbol{X}_i(u_i)} M'_2 \xrightarrow{\boldsymbol{X}_1(u_1)} M'_3 \xrightarrow{\boldsymbol{X}_2(u_2)} \cdots \xrightarrow{\boldsymbol{X}_{i-2}(u_{i-2})} M'_i \xrightarrow{\boldsymbol{X}_{i-1}(u_{i-1})} \\ M'_{i+1} \xrightarrow{\boldsymbol{X}_{i+1}(u_{i+1})} \cdots \xrightarrow{\boldsymbol{X}_{n-1}(u_{n-1})} M'_n, \end{array}$$

where $(\!|T \cap \mathcal{Q}(u_i)|\!) \sim (\!|T' \cap \mathcal{Q}(u_i)|\!)$.

Proof: By induction on i.

Base case $i = 2$ is covered by Lemma 5.6.

Assume that the for $k-1$ show it holds for k. Let

$$\begin{aligned} T &= M_1 \xrightarrow{\mathbf{X}_1(u_1)} M_2 \xrightarrow{\mathbf{X}_2(u_2)} M_3 \xrightarrow{\mathbf{X}_3(u_3)} \cdots \\ & \xrightarrow{\mathbf{X}_{k-1}(u_{k-1})} M_k \xrightarrow{\mathbf{X}_k(u_k)} M_{k+1} \xrightarrow{\mathbf{X}_{k+1}(u_{k+1})} \cdots \xrightarrow{\mathbf{X}_{n-1}(u_{n-1})} M_n. \end{aligned}$$

Applying the induction hypothesis we can transform the T into

$$\begin{aligned} T'' &= M_1'' \xrightarrow{\mathbf{X}_1(u_1)} M_2'' \xrightarrow{\mathbf{X}_k(u_k)} M_3'' \xrightarrow{\mathbf{X}_2(u_2)} \cdots \\ & \xrightarrow{\mathbf{X}_{k-2}(u_{k-2})} M_k'' \xrightarrow{\mathbf{X}_{k-1}(u_{k-1})} M_{k+1}'' \xrightarrow{\mathbf{X}_{k+1}(u_{k-1})} \cdots \xrightarrow{\mathbf{X}_{n-1}(u_{n-1})} M_n'', \end{aligned}$$

where $M_1 \sim M_1''$ and $M_2 \sim M_2''$. The trace T' is produced by an application of Lemma 5.6 and the result follows. □

5.6 Independence Within a Session

We have seen that within a session there are rules that may execute at different nodes that are independent. For instance, the application of **Rules E.1.3** and **E.2.3**. The following result says that this does not affect the semantics of the trace of the establishment protocol.

Lemma 5.8 (Independence within a Session) *Suppose the trace $T = M_1, \ldots, M_n$ records the execution of a complete session of establishment protocol session u, where $u \notin \mathfrak{L}(M_1)$, $M_i \xrightarrow{\boldsymbol{X}_i(u)} M_{i+1} \xrightarrow{\boldsymbol{X}_{i+1}(u)} M_{i+2}$, and*

$$\boldsymbol{X}_i(u)(L_i, R_i)(a) \parallel \boldsymbol{X}_{i+1}(u)(L_{i+1}, R_{i+1})(b)$$

and $a \neq b$ and $M_1 \sim M_1'$. Then there exists a trace

$$T' = M_1, \ldots, M_i, M_{i+1}', M_{i+2}', \ldots, M_n',$$

where

$$M_i \xrightarrow{\boldsymbol{X}_{i+1}(u)} M_{i+1}' \xrightarrow{\boldsymbol{X}_i(u)} M_{i+2}'$$

and

$$M_j \sim M_j'' \quad (i+2 \leq j \leq n)$$

and

$$(\!| T \cap \mathcal{Q}(u) |\!) \sim (\!| T' \cap \mathcal{Q}(u) |\!).$$

Proof: If the redux and contractum in $\mathbf{X}_i$ are all located at node a and the redux and contractum in $\mathbf{X}_{i+1}$ are all located at node b, then no terms consumed or produced by the one will be consumed by the other. It follows from inspection of the rules and the fact that the two operations are independent that $M_{i+2} \sim M_{i+2}'$. We can apply the Simulation lemma 5.4 to get $M_{i+3} \sim M_{i+3}', \ldots, M_n \sim M_n'$ and the result follows.

The only rule where where all node terms are not located on the same node is **Rule F.1.1**. From inspection of the establishment layer rules we see that if $\mathbf{X}_i$ is an instance of this rule, then the only rule that could have consumed a term in the redux is **Rule F.2.1**, but that violates our assumption of the rules being independent. Whence we conclude that the lemma holds. □

5.7 Noninterference

In this section, we formulate a noninterference theorem that says the following. Consider a trace $T = M_1, M_2, \ldots, M_n$ that records a complete session u. The trace can record the activities of many other establishment sessions or no other sessions. Suppose T' also records a complete session of establishment session u, where T' may have recorded the execution of possibly many other establishment sessions executing. If we restrict the two traces to $\mathcal{Q}(u)$, then, regardless of the activity performed by other session, the two traces are the same up to α-equivalence.

Theorem 5.9 (Noninterference) *Let $T = M_1, \ldots, M_n$ be a trace that records the complete execution of establishment session u, where $u \notin \mathfrak{L}(M_1)$. Let $T' = M'_1, \ldots, M'_l$ be a trace that records the complete execution of establishment session u, where $M_1 \sim M'_1$. Then the following relation holds true:*

$$(\!|T \cap \mathcal{Q}(u)|\!) \sim (\!|T' \cap \mathcal{Q}(u)|\!).$$

Proof: Suppose the execution of session u in T has the application of i session u rules in $\mathbf{X}_1(u), \ldots, \mathbf{X}_i(u)$.

Apply Observational Commutativity 5.7 transform T into

$$T'' = M''_1 \xrightarrow{\mathbf{X}_1(u)} M''_2 \xrightarrow{\mathbf{X}_2(u)} \cdots \xrightarrow{\mathbf{X}_i(u)} M''_{i+1} \longrightarrow \cdots \longrightarrow M''_n,$$

where $M_1 \sim M''_1$ and

$$(\!|T \cap \mathcal{Q}(u)|\!) \sim (\!|T'' \cap \mathcal{Q}(u)|\!).$$

So T'' has all of the u operations moved to the front.

There are operations within session u that may be independent and therefore may occur in different order in the traces T and T' while preserving the dependencies imposed by the sequential operation of the initiator and responder process as well as the messages. Apply Independence Within in a Session lemma 5.8 to rearrange T' so that the independent operations occur in the same order as in T while still preserving the desired relation.

As above, apply Observational Commutativity 5.7 as well as Independence in a Session 5.8 to transform T' into

$$T''' = M'''_1 \xrightarrow{\mathbf{X}_1(u)} M'''_2 \xrightarrow{\mathbf{X}_2(u)} \cdots \xrightarrow{\mathbf{X}_i(u)} M'''_{i+1} \longrightarrow \cdots \longrightarrow M'''_l,$$

where the all the session u operations are moved to the front, all the independent operations in T''' occur in the same order as in T and $M_1' \sim M_1'''$, and

$$(\!| T' \cap \mathcal{Q}(u) |\!) \sim (\!| T''' \cap \mathcal{Q}(u) |\!).$$

Since $M_1 \sim M_1' \sim M_1'' \sim M_1'''$, we can apply the Simulation Lemma 5.4 to conclude that $M_2'' \sim M_2''', \ldots, M_i'' \sim M_i'''$. Given that

$$(\!| M_1'', \ldots, M_i'', \ldots M_n'' \cap \mathcal{Q}(u) |\!) = (\!| M_1'', \ldots, M_i'' \cap \mathcal{Q}(u) |\!).$$

and

$$(\!| M_1''', \ldots, M_i''', \ldots M_l''' \cap \mathcal{Q}(u) |\!) = (\!| M_1''', \ldots, M_i''' \cap \mathcal{Q}(u) |\!)$$

we can conclude from the previous fact that

$$(\!| M_1'', \ldots, M_i'' \cap \mathcal{Q}(u) |\!) \sim (\!| M_1''', \ldots, M_i''' \cap \mathcal{Q}(u) |\!).$$

and the result follows. □

The noninterference theorem is dependent on the two traces T and T' being complete. In remains to show that this complete session is obtainable.

5.8 Progress

The progress theorem that states that if communication between two parties is possible, then it is possible to extend any other to complete the communication.

Theorem 5.10 (Progress) *Consider the trace $T = M_1, \ldots, M_n$ and among the active sessions in the trace is session u and $u \notin \mathfrak{L}(M_1)$, where*

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

Suppose there exists $N_1 \longrightarrow^* N_m$, where $N_1 \sim M_1$ in which session u is among the active sessions. Then there exists $N_m \longrightarrow^* N_l$ such that

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

where

$$(\!| M_1, \ldots, M_n \cap \mathcal{Q}(u) |\!) \sim (\!| N_1, \ldots, N_m, \ldots, N_l \cap \mathcal{Q}(u) |\!).$$

Proof: The trace T records a complete run of session u, but the trace $W = N_1$, ..., N_m records a run of session u that has yet to complete. Furthermore, there are operations that may be independent which may occur in different order in the two sequences while preserving the dependencies imposed by the sequential operation of the initiator and responder process as well as the messages. Some prefix of T records the application of the same rules as W, but with independent operations possibly recorded as executing in a different order. Applying the Independence in a Session lemma 5.8, W may be transformed into $W' = N'_1,\ldots,N'_m$, where the independent operations occur in the same order as they did in T and

$$(\!|W \cap \mathcal{Q}(u)|\!) \sim (\!|W' \cap \mathcal{Q}(u)|\!).$$

Applying the Noninterference Theorem 5.9, T can be transformed into T' where all the activity for sessions other than u begin after session u has completed execution. So $T' = M'_1, \ldots, M'_l, M'_{l+1}, \ldots, M'_n$, where $M_1 \sim M'_1$, and only session u operations appear in the prefix $M'_1, \ldots, M'_l$ and establishment session u is complete in this prefix. In addition, $(\!|T \cap \mathcal{Q}(u)|\!) \sim (\!|T' \cap \mathcal{Q}(u)|\!)$. By the same logic, $W' = N'_1,\ldots,N'_m$ can be transformed into $W'' = N''_1, \ldots, N''_j, N''_{j+1}, \ldots, N''_m$, where $N_1 \sim N'_1 \sim N''_1$ and only session u operations occur in $N''_1, \ldots, N''_j$. In addition, $(\!|W' \cap \mathcal{Q}(u)|\!) \sim (\!|W'' \cap \mathcal{Q}(u)|\!)$.

Since $M_1 \sim M'_1 \sim N_1 \sim N''_1$, the Simulation lemma 5.4 can be applied to conclude that $M'_2 \sim N''_2, \ldots M'_j \sim N''_j$. It remains to show a completion for W'', but we do have

$$M'_j \xrightarrow{\mathbf{X}_j(u)} M'_{j+1} \xrightarrow{\mathbf{X}_{j+1}(u)} \cdots \xrightarrow{\mathbf{X}_{l-1}(u)} M'_l.$$

Given the fact that $M'_j \sim N''_j$, we can apply the Simulation lemma 5.4 $l - j$ times to yielding

$$N''_1 \longrightarrow \cdots \longrightarrow N''_j \xrightarrow{\mathbf{X}_j(u)} N''_{j+1} \longrightarrow \cdots , N'_{l-1} \xrightarrow{\mathbf{X}_{l-1}(u)} N'_l,$$

where $M'_{j+1} \sim N''_{j+1}, \ldots, M'_l \sim N''_l$ and

$$(\!|N''_1, \ldots, N''_j, \ldots, N''_l \cap \mathcal{Q}(u)|\!) \sim (\!|M'_1, \ldots M'_l \cap \mathcal{Q}(u)|\!).$$

Given that there is no activity in session u recorded after M_l

$$(\!|M'_1, \ldots M'_l \cap \mathcal{Q}(u)|\!) = (\!|M'_1, \ldots M'_l, \ldots, M'_n \cap \mathcal{Q}(u)|\!).$$

Similarly

$$(\!|N''_1, \ldots, N''_l \cap \mathcal{Q}(u)|\!) = (\!|N''_1, \ldots, N''_l, \ldots, N''_{m+l-j} \cap \mathcal{Q}(u)|\!).$$

It follows that

$$(\!|M'_1, \ldots M'_i, \ldots, M'_n \cap \mathcal{Q}(u)|\!) \sim (\!|N''_1, \ldots, N''_j, \ldots, N''_i, N''_{j+1}, \ldots, N''_{m+(l-j)} \cap \mathcal{Q}(u)|\!).$$

Hence the theorem holds. □

5.9 Conclusion

Having proven a number of results, let us now return to a concrete example to illustrate the utility of the theory developed in this chapter. Consider two nodes a and b and assume that both gateway and discovery policies are 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

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

and

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

It follows from Independence Between Sessions (corollary 5.2) that operations performed in sessions u and v are independent. From the Observational Commutativity Theorem 5.7 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 5.9 informs us that regardless of the interleaving of the operations of the two sessions, they will terminate with their respective tunnels set up. Since no assumptions were made about the initial state in which establishment is run, any failures in the state of the tunnel complex can be addressed by simply rerunning the protocol with new a new session identifier.

This chapter focused on the functional correctness of tunnel establishment, which is a key building block of any tunnel-complex protocol. In the chapters that follow, our focus shifts to tunnel-complex protocols, in particular, discovery protocols. The techniques developed in this chapter continue to be useful when applied to functional properties of discovery protocols and analyzing denial of service.

Chapter 6

Discovery Protocols

Recall that discovery protocols are tunnel-complex protocols that discover security gateways on the dataflow path, deliver distributed credentials required to negotiate gateway traversal, and construct a tunnel complex, where the traffic flow in the ensuing virtual topology is governed by high-level polices at the gateways. Discovery protocols are intended to ease the burden of setting up tunnel complexes associated with defense-in-depth protection schemes. A diversity of security requirements entails the need for a variety of tunnel complexes. As we shall see later in this chapter, different discovery protocol designs can be employed to create a wide range of tunnel complexes. In this chapter, we apply the tunnel calculus to the study of discovery protocols and several case studies of discovery protocols are presented. The functional correctness of discovery protocols are characterized in terms of a completeness theorem, which we shall formulate and prove for each of our examples.

The remainder of this chapter is organized as follows. The next section provides an overview of discovery protocols. This includes a discussion of discovery policies, gateway policies, and credentials, which were all introduced in Chapter 4, but we now elaborate more on their role in discovery protocols. We also introduce a completeness criteria for discovery protocols that acts as a critical functional correctness property. This is followed by the presentation of a protocol that constructs a tunnel complex composed of concatenated tunnels. Finally, we study protocols that construct a tunnel complex composed of nested tunnels.

6.1 Overview of Discovery

In order to prepare the reader for the case studies that occupy the remainder of this chapter, we give a brief overview of our discovery protocols including a discussion of credentials, gateway policies, and discovery policies.

In the tunnel calculus, discovery protocols are invoked by writing a $\downarrow_{\mathrm{dis}(u,k)}$ term to the multiset. In order to ensure the uniqueness of the session identifier we enforce the following restrictions on the tunnel-calculus rules defining discovery protocols:

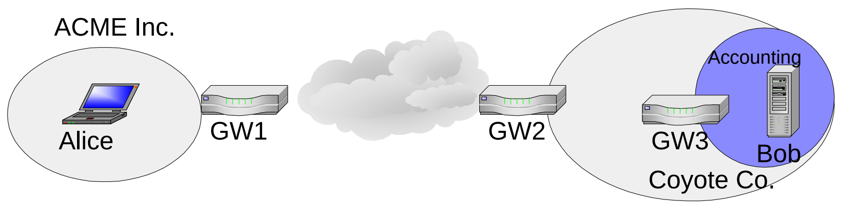

Figure 6.1: Example Topology

- The rule that invokes establishment has form

$$\ldots \longrightarrow \downarrow_{\text{dis}(u,k)} \ \ldots \ \text{new } k, u.$$

- The $\downarrow_{\text{dis}(u,k)}$ term contains no session identifiers other than u.
- The rules defining a discovery-protocol process only contain one session identifier variable, but a single rule may contain many instances of that session identifier variable.

These restrictions ensure that the session identifier is unique and that one session does not possess another's session identifier.

Although one can conceive of many different discovery protocols, all of the discovery protocols considered here have the same basic skeleton. A node s initiates discovery session u to establish communication with a node d. The initiating host s sends the distinguished packet $\mathsf{P}(s, d, \mathsf{C}(\text{Dis}(s, u)))$ toward d, containing the session identifier as well as the address s, which is the node with which the intercepting gateway will initiate establishment. The first gateway on the dataflow path, say $GW1$, intercepts this packet and the discovery protocol invokes the establishment layer to set up a tunnel with s. When this tunnel has been set up, $GW1$ releases the discovery packet. The address in the discovery packet will vary depending on the complex being created. For instance, if the gateway releases the packet $\mathsf{P}(s, d, \mathsf{C}(\text{Dis}(GW1, u)))$, then the next intercepting gateway will set up a tunnel to $GW1$; on the other hand, if the packet released had been $\mathsf{P}(s, d, \mathsf{C}(\text{Dis}(s, u)))$, then the next intercepting gateway will set up a tunnel to s that is nested inside of the tunnel between s and $GW1$. The process continues until the discovery process reaches the destination node d, at which point an end-to-end tunnel is established to secure communication between s and d. As demonstrated with the L3A protocol, additional acknowledgment messages may be needed depending on the discovery protocol in question.

Recall that each node a executing a discovery protocol is assumed to have a credential set Ξ^a that defines a relation with at least one other entity. For instance, hosts and gateways will have credentials defining the administrative domains to which they belong. In the situation illustrated in Figure 6.1, Alice and $GW1$ both belong to ACME, but gateway $GW1$ also contains a credential saying that ACME is a subcontractor of Coyote corporation. Gateways $GW2, GW3$, and Bob belong to Coyote Corp. In addition, both $GW3$ and Bob also belong to Coyote's accounting department. The credentials defining this relationship are given in the following table:

Node	Credential	Contents
Alice	Ξ^A	$K_A \Rightarrow K_{ACME}$
GW1	Ξ^{GW1}	$K_{GW1} \Rightarrow K_{ACME} \Rightarrow K_{CoyoteSub}$
GW2	Ξ^{GW2}	$K_{GW2} \Rightarrow K_{Coyote}$
GW3	Ξ^{GW3}	$K_{GW3} \Rightarrow K_{Acct} \Rightarrow K_{Coyote}$
Bob	Ξ^B	$K_B \Rightarrow K_{Acct} \Rightarrow K_{Coyote}$

These credentials will latter be used in examples illustrating how specific discovery protocols satisfy discovery polices and gateway polices. We make no assumptions as to how credentials are initialized at the nodes.

The fact that discovery protocols discover unknown gateways eases the administrative burden, but opens up a possible vulnerability because a hostile gateway could be placed on the dataflow path and either refuse to run the protocol, which will be noticed by the initiating principal, or execute the protocol as expected and monitor traffic flow, which may not be detected by the communicating parties. Discovery policies, introduced in Chapter 4, are designed to thwart such attacks under the assumption that trusted gateways behave honestly. Discovery policies define the administrative entities with which the discovery protocol is willing to communicate. Recall from Chapter 4 that discovery policies for node a are denoted as ϕ and have the form $\mathsf{Disc}\langle K_a | K_{b_1}, \ldots, K_{b_n}\rangle$, saying a is willing to communicate with $b_1, \ldots, b_n$. The discovery policy enforced for session u is denoted Φ^u and has the form $\mathsf{Discs}\{\phi \,@\, a_1, \ldots, \phi \,@\, a_n\}^u$. The node s initiating discovery session u is assumed to initialize the discovery policies Φ^u to its set of trusted entities $\phi \,@\, s$. Depending on the protocol, trusted gateways executing protocol session u can add their discovery polices to Φ^u, thus increasing the number of nodes that are allowed to participate in the protocol session. The discovery protocol is responsible for migrating the discovery policy for a session to each participating node as the protocol executes.

Consider the situation depicted in Figure 6.1. Assume that the discovery polices at each node are configured to allow communication as follows:

Node	Discovery Policy
Alice	ACME, Bob
GW1	ACME, Coyote
GW2	Coyote, Accounting
GW3	Coyote, Accounting

Suppose Alice initiates discovery to Bob. The discovery policy is initialized to $\Phi^u = \mathsf{Discs}\{\mathsf{Disc}\langle K_A | K_{\text{ACME}}, K_B\rangle\}$. If ACME's outer gateway $GW1$ intercepts the discovery packet and invokes establishment with Alice, then the establishment request message contains the credential set Ξ^{GW1}, which says $GW1$ belongs to ACME ($K_{GW1} \Rightarrow K_{\text{ACME}}$); hence, $\Xi^{GW1} \models_{K_A} \Phi^u$. Following our template for discovery protocols, $GW1$ releases the discovery packet, which gets intercepted by $GW2$, which contains credentials saying that it belongs to Coyote. The discovery polices ϕ at $GW1$ authorize communication with Coyote corporation, but Alice's do not. If the discovery protocol allows trusted principal $GW1$ to add its discovery polices to those of Alice ($\Phi^u \cup \phi\,@\,GW1$), then the protocol succeeds; otherwise, the protocol fails.

Consider the execution of discovery session u that sets up a tunnel complex to enable communication between s and d. Gateways on the dataflow path intercept the discovery packet and demand that the protocol demonstrate that it can satisfy the gateway's policy governing the flow of traffic between s and d ($\Theta_{s\leftrightarrow d}$) before establishment is allowed to set up the tunnel required to traverse the gateway. The discovery protocol is responsible for collecting distributed credentials and delivering them to the newly discovered gateways. Each session u has a designated credential set Ξ^u that is initialized to Ξ^s when the discovery protocol begins executing. The credential sets for each protocol session are only modified by the session to which they belong. We have seen in Chapter 4 that the establishment protocol sends Ξ^u in the establishment-reply message sent to the establishment initiator, which is typically the most recently discovered node. Included in Ξ^u is a credential that says that the initiator speaks for the responder. The establishment initiator executing in session u only proceeds if $\Xi^u \models \Theta_{s\leftrightarrow d}$. Depending on the protocol design, credentials may or not migrate from a previously discovered node to the most recently discovered node on the path.

Consider the example of the topology given in Figure 6.1, where all nodes are assumed to have the credentials given in the table above. Suppose the polices at the gateways are given as follows:

Gateway	Policy
GW1	$\mathsf{Pol}\langle K_{\text{ACME}}, \text{dom}(GW1) \leftrightarrow \text{dom}(GW2)\rangle$.
GW2	$\mathsf{Pol}\langle K_{\text{Coyote}}, K_{\text{CoyoteSub}}, \text{dom}(GW1) \leftrightarrow \text{dom}(GW2)\rangle$.
GW3	$\mathsf{Pol}\langle K_{\text{Alice}}, \text{dom}(GW1) \leftrightarrow \text{dom}(GW3)\rangle$.

If Alice initiates establishment session u to communicate with Bob, then Ξ^u is initialized to Ξ^A. Gateway $GW1$ intercepts the discovery packet and invokes establishment with Alice, which sends $GW1$ the credential set $\Xi^u = \Xi^A \cup \{K_{GW1} \Rightarrow K_A\}$. Since the credential set forms a chain saying that Alice belongs to ACME and the gateway's discovery policy governing this traffic flow says that communication is allowed by any host belonging to ACME, the credentials presented to $GW1$ by Alice satisfy $GW1's$ gateway policy. The discovery packet is next intercepted by $GW2$, which invokes establishment with a previously discovered node. The establishment responder is determined by the topology of the tunnel complex being constructed. The gateway policy at $GW2$ is satisfied if the protocol presents to $GW2$ the credential set located at $GW1$, which says that $GW1$ is a subcontractor to Coyote, but fails if the protocol only delivers Alice's credentials to $GW2$ because Alice's credentials fail to define a relationship to Coyote Corp. If the tunnel being set up flows between $GW1$ and $GW2$, it is easy to deliver $GW1'$s credentials to $GW2$, but if the tunnel being set up is between Alice and $GW2$, then the protocol must take additional action to migrate the credentials from $GW1$ to Alice so they can be presented to $GW2$ during establishment. The protocol designer must be cognizant that different designs deliver different credential sets to the gateways and the consequences that arise from different design decisions.

The discussion above leads us to view discovery protocols from the perspective of a credential delivery mechanism. From this viewpoint, functional correctness can be characterized as a property that says that if the credentials needed to satisfy a gateway's policies are initially located at specified nodes on the dataflow path, then the protocol delivers them to gateways on the dataflow path. As we have seen, different discovery protocol designs may restrict what nodes contribute to the credential set as the protocol executes. Hence, there is not a universal correctness criteria in this sense. Viewing our notion of correctness more formally, we express the concept of 'credentials located at specific node satisfying a gateway's policy' in terms of the semantic relation $\Xi \models \Theta$. We express the fact that a protocol delivers the credentials in terms of the rules of the tunnel calculus. Hence, the property says that if a semantic relation is satisfied, then an operational property should hold. Thus, the desired correctness property is a completeness theorem for the protocol. We now turn our attention to concrete examples of discovery protocols. For each of the discovery protocol case studies presented below, we shall formulate and prove a completeness theorem, which can be seen as characterizing the structure of the protocol.

6.2 Concatenated Discovery Protocol

The L3A protocol developed in Chapter 2 creates the tunnel complex depicted in Figure 2.10 with tunnels between the client and the NAS, the NAS and the server, and an end-to-end tunnel between the client and the server. This can be viewed

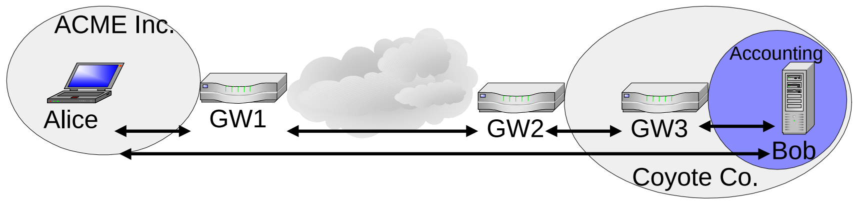

Figure 6.2: Concatenated Tunnel Complex

as a tunnel complex composed of two concatenated tunnels and an end-to-end tunnel nested inside of the other two tunnels. The host initiating the L3A protocol is assumed to know the address of both the network access server (NAS) as well as the server with which it is initiating communication. Recall that this configuration supports the enforcement of both the ingress and egress authenticated traversal property because all traffic arriving or exiting the NAS must do so in a tunnel. In this section, a discovery protocol is derived that can be seen as a generalization of the L3A protocol in that it discovers the gateways on the path and sets up a similar complex of concatenated tunnels while supporting authenticated traversal. The section concludes with the presentation and proof of a completeness theorem for this protocol.

Consider the situation depicted in Figure 6.2, where Alice, who works at ACME, wishes to communicate with Bob's server, located in the accounting department at Coyote Corporation. Alice is located in the administrative domain of ACME's corporate network gateway $GW1$ while Bob is located behind Coyote's corporate gateway $GW2$ and is also inside the administrative domain of the accounting department's gateway $GW3$. In order for Alice to communicate with Bob, she must traverse all three of these gateways. Building on our experience with L3A, a composition of concatenated tunnels can be set up to gateways enforcing authenticated traversal. The resulting tunnel complex is depicted in Figure 6.2.

Before embarking on the design of this protocol, recall that the analysis of L3A given in Section 2.5 revealed the need for acknowledgments to prevent the traffic setting up the end-to-end tunnel from getting caught in partially set up tunnels. Care must be taken to ensure that the problem does not reappear in the more general setting. Examine Figure 6.3, where discovery protocol messages are displayed as solid black lines and establishment messages are displayed as dotted black lines. Following the template sketched for discovery protocols in the previous section, node B releases a discovery packet that is intercepted by node A, which in turn initializes establishment with node B. Node B writes state for the $A \longrightarrow B$ association before sending the reply message back to A, after which, it writes state for the $B \longrightarrow A$ tunnel. Upon receiving the reply message, node A writes the state for both associations. At this point in the execution, node A knows that the association

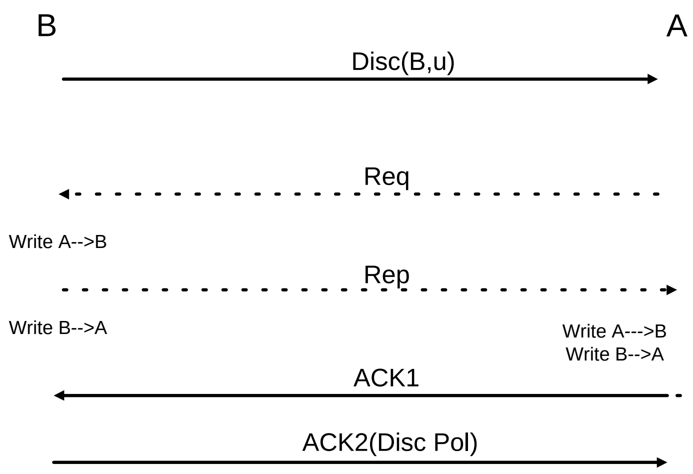

Figure 6.3: Acknowledgments in Concatenated Discovery

from $A \longrightarrow B$ is set up, but A has no knowledge of the state of the $B \longrightarrow A$ association at node B; hence, the end-to-end tunnel should not be set up until the destination node knows that the $B \longrightarrow A$ association has been set up at all the intermediate nodes on the path. As with L3A, we employ acknowledgment messages to prevent the problem from ever arising. After establishment at node A has terminated, it sends an acknowledgment to node B in the $A \longrightarrow B$ association. Upon termination of establishment at B, the discovery protocol waits for the arrival of the acknowledgment from A and responds in kind with an acknowledgment. The second acknowledgment serves the duel purpose of being both an acknowledgment and the vehicle for transmitting the discovery policy for the session to A. The acknowledgments could have been included in the establishment protocol, but, as we shall see, there are protocols that will not need all the acknowledgments required by the concatenated protocol and it seems best to eliminate superfluous messages.

A skeleton for our concatenated discovery protocol is given as follows. A node (starting with the node that initiates the protocol) sends out a discovery packet that gets intercepted by the next gateway on the dataflow path. Establishment is invoked at the newly discovered gateway to set up a pair of associations between itself and the node that last released the discovery packet. When the establishment initiator writes the term indicating that it has terminated, this node sends the establishment responder an acknowledgment ACK1 and receives an ACK2 acknowledgment in turn. The ACK2 acknowledgment contains the discovery policy Φ^u for the session. If the initiator is not the destination, then the discovery packet is released and the establishment responder is invoked to set up a tunnel to the next node on the path. If the most recently discovered node is the destination, then upon the termination of establishment and the receipt of the ACK2 message, establishment is invoked to

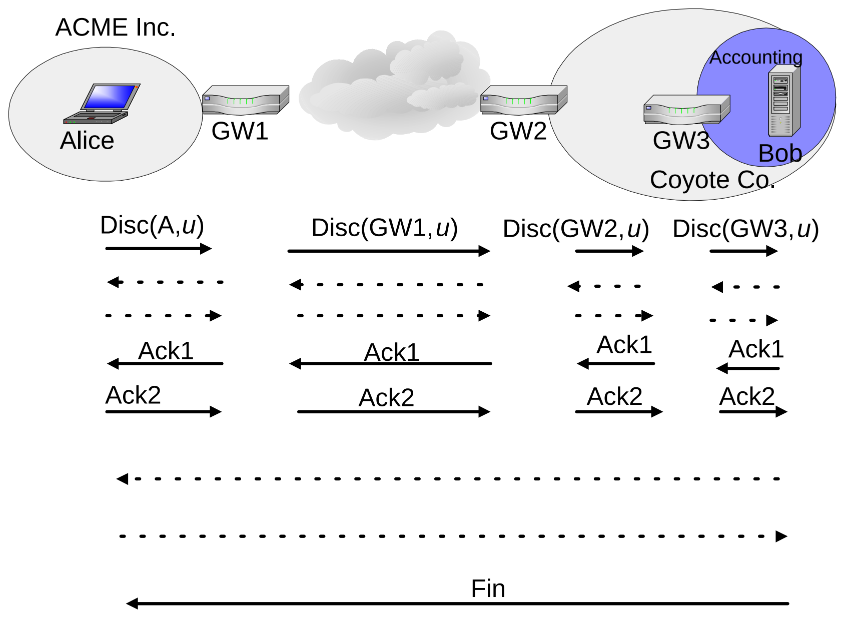

Figure 6.4: Concatenated Discovery Execution

set up the end-to-end tunnel. When this instance of establishment completes, a FIN message is sent to the host that initiated the discovery protocol.

Figure 6.4 illustrates the execution of the concatenated discovery protocol on the Alice-Bob example in Figure 6.2. The protocol messages are displayed as black lines and the establishment messages are displayed as dotted black lines. Alice releases a discovery packet $\mathsf{P}(A, B, \mathsf{C}(\text{Dis}(A, u)))$ destined for Bob. The packet is intercepted by gateway $GW1$, which invokes establishment with Alice. The gateway then sends Alice an acknowledgment message $\mathsf{P}(GW1, A, \text{ACK1})$ in the newly created association flowing from $GW1$ to Alice. Alice responds by sending an acknowledgment $\mathsf{P}(A, GW1, \text{ACK2}(\Phi^u))$ in the association flowing from Alice to gateway $GW1$. Upon receiving this message, $GW1$ releases the discovery packet $\mathsf{P}(A, B, \mathsf{C}(\text{Dis}(GW1, u)))$, where the address A is replaced by $GW1$. The process repeats until Bob has set up a tunnel with $GW3$, after which, Bob initiates establishment with Alice to set up the end-to-end tunnel. When this tunnel has been set up, Bob sends Alice a Fin message. We now examine, in turn, how the protocol handles discovery and gateway policies.

Suppose we execute session u of the concatenated discovery protocol using our example in Figure 6.4, where the discovery policies are as given above. The discovery policy for the session Φ^u is initialized at Alice to $\mathsf{Discs}\{\phi \,@\, A\}$, which says Alice is willing to communicate with anyone who can prove that they belong to ACME Inc or Bob. Gateway $GW1$ sends its credential saying $K_{GW1} \Rightarrow K_{ACME}$ to Alice, which satisfies Φ^u so the protocol is willing to communicate with $GW1$ and therefore continues execution. The protocol then passes Φ^u along to $GW1$ in the ACK2 acknowledgment, which adds its own discovery policy $\phi \mathbin{@} GW1$ to Φ^u, saying that $GW1$ is willing to communicate with any member of Acme or Coyote Corporation. The discovery packet is released and gets intercepted by gateway $GW2$. Gateway $GW2$ sends its credential set Ξ^{GW2}, saying that $GW2$ belongs to Coyote Corporation, to $GW1$ during establishment. Since these credentials satisfy the discovery policy Φ^u at node $GW1$, the protocol continues to execute, sending Φ^u to $GW2$ in the ACK2 acknowledgment, where $GW2$ adds its discovery policy $\phi \mathbin{@} GW2$ to Φ^u, saying that $GW2$ is willing to communicate with anyone belonging to ACME, Coyote Corporation, or Coyote's accounting department. The discovery packet is then intercepted by gateway $GW3$, which invokes establishment with $GW2$. During this process $GW3$ sends its credential set to $GW2$, saying that $GW3$ belongs to Coyote Corporation's accounting department. Since these credentials satisfy $\Phi^u \mathbin{@} GW2$, the protocol continues executing setting up the end-to-end tunnel. Note that the discovery policy used by Alice in setting up the end-to-end tunnel is $\Phi^u = \phi \mathbin{@} A$ because the Φ^u that gets delivered to Bob is never migrated back to Alice. Whence there is an implicit assumption that the initiating host will always include the final destination in the discovery policy.

We now show how the protocol delivers credentials to the gateways using the example given in Figure 6.4. Assume that the gateway polices and credentials at nodes are as given in the tables above. During the execution of establishment between Alice and $GW1$, Alice delivers the credential $K_{GW1} \Rightarrow K_A \Rightarrow K_{\text{Acme}}$ to $GW1$, which satisfies the policy at that gateway. In setting up the tunnel between $GW1$ and $GW2$, the establishment protocol delivers the credential set Ξ^u from $GW1$ to $GW2$ that is composed of the credential sets located at Alice (Ξ^A) and $GW1$ (Ξ^{GW1}) as well as credentials saying that $GW1$ speaks for Alice and that $GW2$ speaks for $GW1$. These credentials form the chain

$$K_{GW2} \Rightarrow K_{GW1} \Rightarrow K_A \Rightarrow K_{\text{Acme}} \Rightarrow K_{\text{CoyoteSub}}.$$

Since this credential set satisfies the policy at $GW2$, the protocol continues executing. The credential set delivered to $GW3$ is composed of the credentials delivered to $GW2$, $GW2'$s credential set Ξ^{GW2}, and a credential saying $GW3$ speaks for $GW2$. These credentials form the chain

$$K_{GW3} \Rightarrow K_{GW2} \Rightarrow K_{GW1} \Rightarrow K_A \Rightarrow K_{\text{Acme}} \Rightarrow K_{\text{CoyoteSub}},$$

which satisfies the applicable policy at $GW3$ and the protocol continues executing. Notice that the protocol, in essence, collected credentials as it executed and presents them to the next gateway on the path. This idea will be made precise when we formulate the completeness theorem.

6.2.1 Rules for Concatenated Discovery

The concatenated discovery protocol is composed of an initiator and responder processes. The initiator process runs only at the host that initiates discovery and the responder processes run at the remaining nodes. We now define both processes using the notation of the tunnel calculus.

The initiator sends out the first discovery packet and invokes the establishment responder. When the tunnel is set up with the first gateway on the path, the initiator awaits the arrival of the ACK1 acknowledgment message and responds with the ACK2 acknowledgment message containing the discovery policy Φ^u. The initiator then invokes the establishment responder once again to set up the end-to-end tunnel. Once this tunnel is set up, the initiator awaits the arrival of the Fin message indicating that the discovery protocol has completed. The initiator is composed of the following three rules.

Rule CD.1.1

$$\begin{array}{lll} \phi,\ \Xi^a & \vdash_s & \downarrow_{\mathrm{dis}(u,k_1)} \mathsf{D}(s,d) \longrightarrow \\ & & \downarrow_{\mathrm{sec}(u,k_2)} \mathsf{P}(s,d,\mathsf{C}(\mathrm{Dis}(s,u))), \\ & & \downarrow_{\mathrm{eresp}(u,k_3)}\ ,\quad \mathsf{Discs}\{\phi\}^u,\quad \Xi^u, \\ & & \langle s,d,u,k_1,k_2,k_3\rangle \\ & & \text{new } k_2,k_3 \\ & & \text{where } \Xi^u = \Xi^a. \end{array}$$

The discovery protocol session u is initiated at node s to communicate with node d by writing a $\downarrow_{\mathrm{dis}(u,k_1)} \mathsf{D}(s,d)$ term to the multiset. **Rule CD.1.1** sends a discovery packet toward d and invokes the establishment-responder processes, with session identifier set to u, to set up a tunnel with the newly discovered gateway. The discovery policy Φ^u is initialized to $\phi\,@\,s$, which is the discovery policy of node s; and the credential set for the session Ξ^u is initialized to Ξ^s, which is the credential set of node s.

Rule CD.1.2

$$\begin{array}{lll} \Phi^u & \vdash_s & \langle s,d,u,k_1,k_2,k_3\rangle,\quad \uparrow_{\mathrm{sec}(k_2)}\ , \\ & & \uparrow_{\mathrm{eresp}(k_3)} \mathbf{R}(a),\quad \Uparrow_{\mathrm{sec}(u)} \mathsf{P}(a,s,\mathrm{ACK1}) \longrightarrow \\ & & \downarrow_{\mathrm{sec}(u,k_4)} \mathsf{P}(s,a,\mathrm{ACK2}(\Phi^u)),\quad \downarrow_{\mathrm{eresp}(u,k_5)}\ , \\ & & \langle s,d,u,k_1,k_4,k_5\rangle \\ & & \text{new } k_4,k_5. \end{array}$$

If the discovery message has been sent, the establishment responder has completed set up of the tunnel with the first gateway on the path a, and an acknowledgment message has been received in the newly set up tunnel, then reply with an acknowledgment message that contains the discovery policy Φ^u that was initialized in the previous rule. The establishment responder is invoked again to set up the end-to-end tunnel between s and d.

Rule CD.1.3

$$\begin{aligned}\vdash_s \quad & \langle s,d,u,k_1,k_4,k_5\rangle,\ \ \uparrow_{\mathrm{sec}(k_4)}\ ,\ \Xi^u, \\ & \uparrow_{\mathrm{eresp}(k_5)} \mathbf{R}(d),\ \ \Uparrow_{\mathrm{sec}(u)} \mathsf{P}(d,s,Fin) \longrightarrow \uparrow_{\mathrm{dis}(k_1)}\ .\end{aligned}$$

If the acknowledgment for the ACK2 message has been written to the multiset and the establishment responder returns indicating that the tunnel between s and d has been set up and a Fin message is received from d, then write the acknowledgment that the concatenated discovery session u has successfully terminated. Note that the credential set for the session is removed to clean up the multiset upon termination.

The responder process executes the protocol on all nodes other than the node that initiated the protocol. Upon arrival of the discovery message, the responder invokes establishment to set up a tunnel with the node that last released the discovery packet. Once this tunnel is set up, the ACK1 acknowledgment is sent inside of the newly set up tunnel to the node that last released the discovery packet. The action taken upon receiving the ACK2 acknowledgment depends on whether or not the current node is the final destination. If the node is the final destination, then the protocol invokes establishment to set up the end-to-end tunnel. When the establishment process completes, a FIN message is sent to the host that initiated discovery. If the node is not the discovery destination, then the discovery packet is released with the current node's address as the last node to process the packet and the establishment responder is invoked to set up the next tunnel. When the establishment responder returns, indicating that the next tunnel has been set up, and an ACK1 acknowledgment message has been received, the protocol sends an ACK2 acknowledgment containing the discovery policy Φ^u. We now examine the ten rules that comprise the concatenated discovery responder in some detail.

Rule CD.2.1

$$\begin{aligned}\vdash_a \quad & \Uparrow_{\mathrm{sec}(u)} \mathsf{P}(s,d,\mathsf{C}(\mathrm{Dis}(b,u))) \longrightarrow \ \ \downarrow_{\mathrm{est}(u,k_1)} \mathsf{E}(b,(s,b),(d,a)),\ \ \langle a,b,s,d,u,k_1\rangle \\ & \text{new } k_1.\end{aligned}$$

Upon receiving a discovery message released by node b, the concatenated discovery protocol invokes the responder to set up a tunnel between a and b.

The filters passed to the establishment in **Rule CD.2.1** have the form (a, b),(c, d), which are read as the filter $a \longrightarrow c, b \longrightarrow d$. We view this as a disjoint address range. Establishment will install filters $(s, b) \longrightarrow (d, a)$, which matches traffic flowing from s to d and from b to a, and $(d, a) \longrightarrow (s, b)$, which matches traffic flowing from d to s and from a to b. This is necessary to ensure that both the end-to-end traffic as well as the acknowledgments travel in tunnels.

Rule CD.2.2

$$\begin{array}{ll} \vdash_a & \langle a, b, s, d, u, k_1\rangle \;\; \uparrow_{\mathrm{est}(k_1)} \;\; \longrightarrow \;\; \downarrow_{\mathrm{sec}(u,k_2)} \mathsf{P}(a, b, ACK1), \;\; \langle a, b, s, d, u, k_2\rangle \\ & \text{new } k_2. \end{array}$$

When establishment has terminated at the intercepting node it sends an acknowledgment message to b.

Rule CD.2.3

$$\begin{array}{ll} \vdash_a & \langle a, b, s, d, u, k_2\rangle, \;\; \Phi^u, \\ & \uparrow_{\mathrm{sec}(k_2)}, \;\; \Uparrow_{\mathrm{sec}(u)} \mathsf{P}(b, a, ACK2(\Phi^u)) \longrightarrow \;\; \downarrow_{\mathrm{est}(u,k_3)} \mathsf{E}(s, s, d), \;\; \langle a, b, s, d, u, k_3\rangle \\ & \text{new } k_3 \\ & \text{if } a = d. \end{array}$$

This rule will only execute if the intercepting node is the discovery packet's final destination. Upon receiving an acknowledgment message from b containing the discovery policy for session u, the protocol can be sure that the a–b tunnel has been set up. It then invokes establishment to set up the end-to-end tunnel.

Rule CD.2.4

$$\begin{array}{ll} \vdash_a & \langle a, b, s, d, u, k_3\rangle, \;\; \uparrow_{\mathrm{est}(k_3)} \;\; \longrightarrow \;\; \downarrow_{\mathrm{sec}(u,k_4)} \mathsf{P}(d, s, FIN), \;\; \langle a, b, s, d, u, k_4\rangle \\ & \text{new } k_4 \\ & \text{if } a = d. \end{array}$$

This rule will only execute if the intercepting node is the discovery packet's final destination. When the end-to-end tunnel has been set up, the protocol sends a FIN message to the host that initiated discovery.

Rule CD.2.5

$$\langle a, b, s, d, u, k_4\rangle, \quad \uparrow_{\mathrm{sec}(k_4)} \longrightarrow \cdot$$
if $a = d$.

This rule fires if the intercepting node is the discovery packet's final destination. If the secure layer has written a term indicating that the FIN message has been sent, the protocol run ends.

Rule CD.2.6

$$\begin{aligned}\phi \ \vdash_a \ & \langle a, b, s, d, u, k_2\rangle, \ \uparrow_{\mathrm{sec}(k_2)}, \ \Uparrow_{\mathrm{sec}(u)} \mathsf{P}(b, a, ACK2(\Phi^u)) \longrightarrow \\ & \downarrow_{\mathrm{sec}(u,k_5)} \mathsf{P}(s, d, \mathsf{C}(\mathrm{Dis}(a, u))), \ \Phi^u \cup \{\phi\}, \ \downarrow_{\mathrm{eresp}(u,k_6)}, \\ & \langle a, b, s, d, u, k_5, k_6\rangle\end{aligned}$$
new k_5, k_6
if $a \neq d$.

This rule only executes if the intercepting node is not the packet's final destination. Upon receiving an acknowledgment message containing the discovery policies for session u, this rule releases the discovery packet and invokes the establishment responder for session u. The discovery policy for the node a is added to the discovery policy for the session and written to the multiset.

Rule CD.2.7

$$\begin{aligned}\Phi^u \ \vdash_a \ & \langle a, b, s, d, u, k_5, k_6\rangle, \ \uparrow_{\mathrm{sec}(k_5)}, \\ & \uparrow_{\mathrm{eresp}(k_6)} \mathsf{R}(c), \ \Uparrow_{\mathrm{sec}(u)} \mathsf{P}(c, a, ACK1) \longrightarrow \\ & \downarrow_{\mathrm{sec}(u,k_7)} \mathsf{P}(a, c, ACK2(\Phi^u)), \ \langle a, b, s, d, u, k_7\rangle\end{aligned}$$
new k_7
if $a \neq d$.

This rule only executes if the intercepting node is not the packet's final destination. If the establishment responder has written a term indicating that a tunnel has been set up, the secure layer has acknowledged that the discovery message has been sent, and the secure layer has written a term indicating that an ACK1 acknowledgment has been received, then send the newly discovered node an acknowledgment message that includes the discovery polices for session u.

Rule CD.2.8

$$\begin{aligned}
\vdash_a \quad & \langle a, b, s, d, u, k_7 \rangle, \\
& \Uparrow_{\text{sec}(u)} \mathsf{P}(d, s, \mathsf{X}(\mathsf{Req}(s, d, u, \iota_d, \Xi^d, g))), \quad \Xi^u, \quad \uparrow_{\text{sec}(k_7)} \longrightarrow \\
& \downarrow_{\text{sec}(u,k_8)} \mathsf{P}(d, s, \mathsf{X}(\mathsf{Req}(s, d, u, \iota_d, \Xi^d, g))), \quad \langle a, b, s, d, u, k_8 \rangle \\
& \text{new } k_8 \\
& \text{if } a \neq d.
\end{aligned}$$

This rule only executes if the current node is not the final destination d. When the secure layer acknowledges that the ACK2 message has been sent and the establishment request for the end-to-end tunnel arrives, this rule relays that establishment message toward its destination. A copy of the credentials Ξ^u have already been passed along to the next node on the path so we can clean up by removing them from this node.

Rule CD.2.9

$$\begin{aligned}
\vdash_a \quad & \langle a, b, s, d, u, k_1, k_8 \rangle, \\
& \Uparrow_{\text{sec}(u)} \mathsf{P}(s, d, \mathsf{X}(\mathsf{Rep}(s, d, u, \iota_s, \Xi'^u, g))), \quad \uparrow_{\text{sec}(k_8)} \longrightarrow \\
& \downarrow_{\text{sec}(u,k_9)} \mathsf{P}(s, d, \mathsf{X}(\mathsf{Rep}(s, d, u, \iota_s, \Xi'^u, g))), \\
& \langle a, b, u, s, d, k_1, k_9 \rangle \\
& \text{new } k_9 \\
& \text{if } a \neq d.
\end{aligned}$$

This rule only executes if the current node is not the final destination d. If the secure layer has acknowledged that the establishment request for the end-to-end tunnel has been forwarded and the corresponding establishment reply message has arrived, then forward that message on to its destination.

Rule CD.2.10

$$\begin{aligned}
\vdash_a \quad & \langle a, b, u, s, d, k_9 \rangle, \quad \Xi^u, \quad \uparrow_{\text{sec}(k_9)} \longrightarrow \cdot \\
& \text{if } a \neq d.
\end{aligned}$$

This rule only executes if the current node is not the final destination d. If the secure layer has acknowledged that the establishment reply for the end-to-end tunnel has been forwarded, then remove the credential set that has been left behind and terminate.

6.2.2 Completeness Theorem for Concatenated Discovery

We now formulate and prove a completeness theorem for concatenated discovery protocols that says the following. Assume that if every node possessed every credential, then the discovery protocol would run successfully and that the discovery policies are always satisfied. Consider the trace of the concatenated discovery protocol. Suppose node a_i is the last node discovered by the protocol, then if the credentials located at $a_1, \ldots, a_{i-1}$ satisfy the gateway policy at a_i, the trace should record the authorization layer returning true at this node. There remains some work to do before we can formalize and prove the desired theorem.

The following proposition follows directly from the restrictions on the format of rules defining discovery protocols.

Proposition 6.1 *Let $T = M_1, \ldots, M_n$ be a trace of the execution of discovery protocol session u, where $u \notin \mathfrak{L}(M_1)$. If for some i $(1 \leq i \leq n)$ $t \in M_i$ is a node term where session identifiers $u \in t$ and $v \in t$, then $u = v$*

If the gateway policies or the credentials defining the entities to which gateways belong change during execution of the discovery, then our notion of completeness is problematic. Consequently, we need to qualify the theorem with an assertion that the theorem holds for a fixed set of gateway polices and gateway credentials. This is formalized as follows. Suppose $T = M_1, \ldots, M_n$ is a trace recording the execution of a discovery protocol, where $a_1, \ldots, a_m$ are the nodes on the dataflow path. We say that the *gateway polices and node credentials are fixed in* T if for all i $(1 \leq i \leq m)$ and for all j, k $(1 \leq j, k \leq n)$,

$$\begin{array}{l} \text{if } \Xi^{a_i} \in M_j \text{ and } (\Xi')^{a_i} \in M_k, \text{ then } \Xi^{a_i} = (\Xi')^{a_i} \\ \text{if } \Theta \mathbin{@} a_i \in M_j \text{ and } \Theta' \mathbin{@} a_i \in M_k, \text{ then } \Theta = \Theta'. \end{array}$$

Let $\mathfrak{G}(M)$ denote the multiset where the credentials at each node in M are distributed to each node in the network of M. This is formalized as follows. Denote the nodes in the network of M as $a_1, \ldots, a_n$. The multiset $\mathfrak{G}(M)$ is defined to be the same as M, but with each Ξ^{a_i} is replaced by.

$$\bigcup_{1 \leq k \leq n} \Xi^{a_k}.$$

It is assumed that when a discovery protocol is run in $\mathfrak{G}(M)$ that the authorization layer uses the credentials that reside at the node where authorization is invoked. It follows from proposition 6.1 and the noninterference and progress theorems of Chapter 5 that one session of a discovery protocol cannot affect the execution of another. Assume that the authorization layer always returns true when verifying a discovery policy. Suppose a discovery protocol always executes to completion if invoked in $\mathfrak{G}(M)$. This means that the protocol will always run to completion if the requisite credentials are delivered to the proper gateways on the dataflow path.

Given this machinery we can now formulate the completeness theorem for the concatenated discovery protocol.

Theorem 6.2 *Let $T = M_1, \ldots, M_n$ be a trace of the execution of the concatenated discovery protocol session u initiated at node a_1 $(= s)$ to communicate with node a_n $(= d)$ and the gateway policies and node credentials are assumed fixed in T. Assume $u \notin \mathfrak{L}(M_1)$ and that the forwarding tables indicate that the nodes on the dataflow path are $a_2, \ldots, a_{n-1}$. Assume that the concatenated protocol session u always runs successfully when initiated in $\mathfrak{G}(M_1)$ and that the authorization layer always returns true when checking a discovery policy in T. For each i $(2 \leq i \leq n)$ there exists a node term $\uparrow_{\mathrm{auth}(k)} \mathsf{GWPol}(u, \mathit{true}) \,@\, a_i$ such that the following statement holds: if*

$$\bigcup_{1 \leq l < i} (\{K_{l+1} \Rightarrow K_l\} \cup \Xi^{a_l}) \ \models_{a_i} \Theta_{a_1 \leftrightarrow a_n} \,@\, a_i$$

then

$$\uparrow_{\mathrm{auth}(k)} \mathsf{GWPol}(u, \mathit{true}) \,@\, a_i \in T$$

Proof: Proof is by induction on the number of gateways.

For the base case there are no intermediate gateways only source a_1 and destination a_2. Consider the execution of the concatenated discovery protocol. At node a_1 **Rule CD.1.1** releases the discovery packet and invokes the establishment responder. Upon receiving the discovery packet, node a_2 executes **Rule CD.2.1**, which invokes the establishment-layer initiator to set up a pair of associations between a_1 and a_2. The establishment-layer responder (**Rule E.2.2**) sends $\Xi^u = \Xi^{a_1} \cup \{K_{a_2} \Rightarrow K_{a_1}\}$ to a_2 in the establishment response message. Upon receiving this message, **Rule E.1.2** invokes the authorization layer. Since it was assumed that $(\Xi \,@\, a_1 \cup \{K_{a_2} \Rightarrow K_{a_1}\}) \ \models_{a_2} \Theta_{a_1 \leftrightarrow a_2} \,@\, a_2$, we can conclude that the authorization layer returns true and the base case is satisfied.

Suppose the statement is true for i gateways, we show it is true for $i+1$ gateways. It follows from the induction hypothesis that the discovery protocol ran to the point that it had successfully set up tunnels between a_j and a_{j+1}, where $1 \leq j < i$. It remains to show that the protocol successfully sets up the pair of associations between a_i and a_{i+1}. Gateway i is not the final destination so, **Rule CD.2.6** releases the discovery packet and invokes the establishment reply. Upon receiving this message, gateway a_{i+1} executes **Rule CD.2.1** that invokes the establishment-layer initiator to set up a pair of associations between a_i and a_{i+1}. The establishment-layer responder (**Rule E.2.2**) sends $\Xi^u = \Xi^u \cup \Xi^{a_i} \cup \{K_{a_{i+1}} \Rightarrow K_{a_i}\}$ to a_{i+1}. It follows from the induction hypothesis, that authorization succeeded at nodes $a_1, \ldots, a_i$ and consequently the credential sets from these nodes are passed to the next node on the

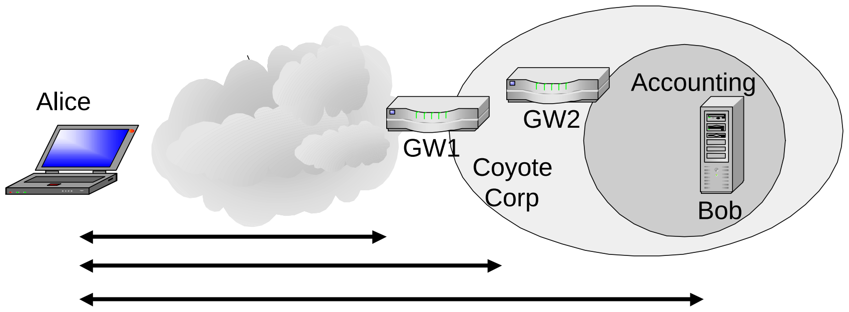

Figure 6.5: Nested Tunnel Complex

path as the protocol executes; so the credential set sent from a_i to a_{i+1} is

$$\bigcup_{1 \leq l \leq i} (\Xi \,@\, a_l \cup \{K_{a_{l+1}} \Rightarrow K_{a_l}\}).$$

Upon receiving the establishment response message, **Rule E.1.2** invokes the authorization layer. Since it was assumed that

$$\bigcup_{1 \leq l \leq i} (\Xi \,@\, a_l \cup \{K_{a_{l+1}} \Rightarrow K_{a_l}\}) \models_{a_{i+1}} \Theta_{a_1 \leftrightarrow a_n} \,@\, a_{i+1},$$

we can conclude that the authorization layer returns true.

The theorem follows from induction. □

6.3 Nested Tunnels Discovery Protocol

The concatenated discovery protocol presented in the previous section can be viewed as a generalization of the L3A protocol. The road-warrior example presented in Chapter 1 often employees a nested complex of tunnels to secure communication as illustrated in Figure 6.5, where each pair-wise tunnel has a one end anchored at a host. Recall that, although this configuration is commonly deployed, its setup is typically poorly coordinated, often involving multiple technologies, sometimes affecting performance. In this section, we investigate discovery protocols that create a nested tunnel complex. For the sake of simplicity, we assume that the host that initiates the discovery protocol is not behind a gateway.

Examine the nested topology depicted in Figure 6.5, there are tunnels between Alice and $GW1$, between Alice and $GW2$, nested in the Alice–$GW1$ tunnel, and between Alice and Bob, nested in the other two tunnels, but there is no tunnel between $GW1$ to $GW2$ and no tunnel between $GW2$ and Bob. This raises the question how $GW1$ should treat traffic flowing from $GW2$ to Alice and how $GW2$ should treat traffic flowing from Bob to Alice? The structural nature of the topology of this tunnel complex means that both gateways must allow the traffic to pass unchecked, hence neither $GW1$ nor $GW2$ can enforce the egress authenticated traversal property, although the ingress authenticated traversal property is enforced. Consequently, this topology can be subjected to cramming attacks originating from within an organization, although, this may be seen as less of a threat than an attack originating from the Internet and vigilant network administrators may be able to more readily deter cramming attacks originating from within their organization. In addition, nested protocols may have advantages over concatenated protocols, such as consuming fewer gateway resources, which may outweigh failing to enforce egress authenticated traversal. It is the very structure of the topology that precludes the egress authenticated traversal from being enforced and this weakness must be considered when choosing to deploy such a topology.

Suppose we apply our basic discovery protocol skeleton to the construction of the tunnel complex depicted in the example of Figure 6.5. Consider the state of the protocol once the tunnel between Alice and $GW1$ has been set up. Gateway $GW1'$s mechanism database needs to be configured so that traffic traveling in the association flowing from $GW2$ to Alice is allowed to pass unauthenticated and be placed in the association flowing from $GW1$ to Alice, but at this point in the execution, the protocol executing at gateway $GW1$ has no knowledge of gateway $GW2$. Our solution is to release the discovery packet and update the mechanism database when the establishment request message traveling from Alice to $GW2$ arrives at $GW1$.

As with the concatenated discovery protocol, we must consider the need for acknowledgments. Suppose we were to set up the tunnel complex illustrated in Figure 6.5 and Alice sends out a discovery message that gets intercepted by $GW1$. The establishment responder sends an establishment request message and upon arrival at Alice the protocol writes state for association flowing from $GW1$ to Alice and sends the reply back to $GW1$. Once the reply has been sent, the protocol writes state for the association flowing from Alice to $GW1$. Upon receiving the reply message, gateway $GW1$ writes state for both associations. At this point gateway $GW1$ knows that the association flowing from $GW1$ to Alice has been set up, but knows nothing of the status of the Alice to $GW1$ association at Alice. The protocol releases the discovery packet after establishment at $GW1$ has terminated; thus we can be assured that when the discovery packet arrives at $GW2$, state for both associations flowing between Alice and $GW1$ have been written at $GW1$, but the Alice to $GW1$ association may not be set up if Alice has not yet written state. So when $GW2$ invokes establishment to Alice, it can be assured that it will safely travel in the $GW1$ to Alice association. Since Alice will not invoke the establishment-responder process that handles establishment with $GW2$ until after establishment has written state for the Alice to $GW1$ association, the establishment reply message sent to $GW2$ will safely travel in the Alice to $GW1$ association. Consequently, no additional acknowledgments are needed for this protocol.

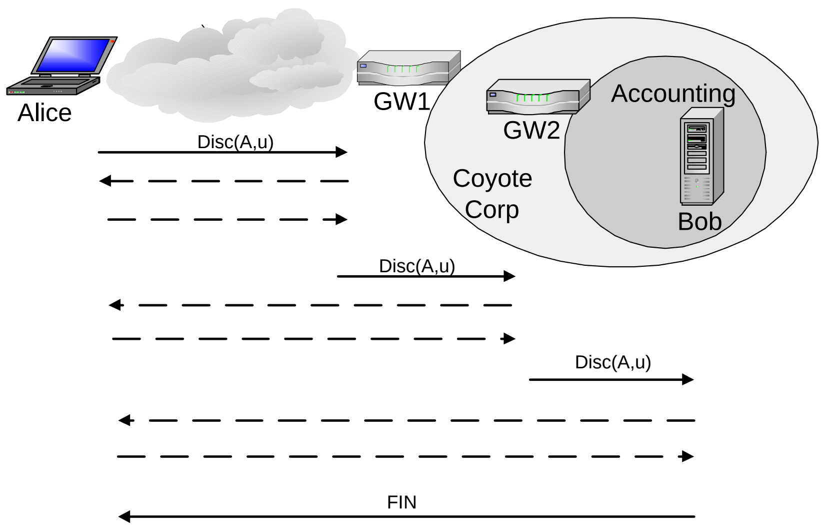

Figure 6.6: Nested Discovery Execution

Let us now illustrate how our nested discovery protocol works using the example from above. Alice sends out a discovery packet $\mathsf{P}(A, B, \mathsf{C}(Dis(A, u)))$ that gets intercepted by gateway $GW1$, which invokes establishment with Alice. The mechanism filters installed at both ends of the tunnel say that all traffic in session u flowing between Alice and any node in the administrative domain of $GW1$ should travel in the newly created associations flowing between Alice and $GW1$. When establishment at $GW1$ has terminated, the discovery packet is released and intercepted by $GW2$, which invokes establishment to set up the tunnel between Alice and $GW2$. When the establishment request message from $GW2$ to Alice arrives at $GW1$, the secure layer passes it up for processing. There is no establishment-responder processes waiting to process this message, instead, the discovery protocol writes entries in the mechanism database saying all traffic in session u arriving from $GW2$ will not be in a tunnel. The establishment request message is then sent on to Alice in the association flowing from $GW1$ to Alice. Note that the establishment response message will arrive at $GW1$ in the association flowing from Alice to $GW1$ and will hence be authenticated, but like the establishment request message, this message gets passed up and the discovery protocol relays it toward its destination. When establishment at $GW2$ has terminated, the discovery packet is released and intercepted by Bob, who invokes establishment with Alice to set up the end-to-end tunnel. Just as before, the establishment request message arriving at $GW2$ is passed up to the discovery protocol and the mechanism database is updated indicating that traffic from Bob can pass even though it is not authenticated in a tunnel. The establishment request is then sent on to Alice in the association flowing from $GW2$ to Alice. Note that this message arrives at $GW1$ traveling in the $GW2$ to Alice association so $GW1$ never sees the header and this message is simply relayed toward Alice without being passed up to the discovery layer. The establishment response message arrives at $GW1$ in the Alice to $GW1$ tunnel, so it is authenticated, the outer header is stripped off, and the message is relayed toward its destination and arrives at $GW2$ in the Alice to $GW2$ association so it too is authenticated, the header is stripped off, and seeing that it is an establishment message, it is passed up to the discovery layer that relays it to its destination and the establishment response message arrives at Bob in the clear. When establishment at Bob has terminated, Bob sends a Fin message to Alice indicating that the protocol has successfully terminated at Bob. This process is depicted in Figure 6.6, where the solid lines represent the discovery protocol messages and the dotted lines represent establishment messages.

This protocol has a different structure than the concatenated protocol in that each gateway on the path invokes establishment with the initiating host, but intermediate nodes do not communicate with other nodes and consequently discovery polices and credentials will not be gathered up and passed along as they did the concatenated protocol given above. So each gateway on the path must be in the discovery policy of the initiating host. Similarly, the initiating host's credentials must be able to satisfy the policies of all the gateways on the dataflow path.

6.3.1 Rules for Nested Discovery

The nested discovery protocol is composed of an initiator process that executes on hosts initiating discovery and a responder process that is executed on all other nodes. We now express the rules for the nested discovery protocol using the notation of the tunnel calculus.

The initiator sends out a discovery packet that will be intercepted by the first node on the path and invokes the establishment responder. Upon notification that the establishment responder has terminated, the initiator takes one of two actions. If establishment has not set up a tunnel with the final destination, then invoke the establishment responder again and repeat the process. If the establishment protocol has set up a tunnel with the final destination, then the initiator waits for the Fin message to arrive in the tunnels that have been set up and terminates. We now examine in detail the four rules that comprise the nested discovery protocol initiator.

Rule ND.1.1

$$\begin{array}{lll} \phi, \Xi^a & \vdash_s & \downarrow_{\mathrm{dis}(u,k_1)} \mathsf{D}(s,d) \longrightarrow \\ & & \downarrow_{\mathrm{sec}(u,k_2)} \mathsf{P}(s,d,\mathsf{C}(\mathrm{Dis}(s,u))) \\ & & \downarrow_{\mathrm{eresp}(u,k_3)}, \quad \mathsf{Discs}\{\phi\}^u, \quad \Xi^u, \\ & & \langle s,d,u,k_1,k_2,k_3\rangle \\ & & \text{new } k_2, k_3 \\ & & \text{where } \Xi^u = \Xi^a. \end{array}$$

The discovery protocol session u that constructs a nested tunnel complex to facilitate secure communication with node d is invoked at node s by writing a $\downarrow_{\mathrm{dis}(u,k_1)} \mathsf{D}(s,d)$ term to the multiset. **Rule ND-1.1** sends a discovery packet toward d and invokes the establishment-responder processes, with session identifier u, to set up a tunnel with the newly discovered gateway. The discovery policy Φ^u is initialized to $\phi \,@\, s$, which is the discovery policy of node s; and the credential set for the session Ξ^u is initialized to Ξ^s, the credential set of node s.

Rule ND.1.2

$$\vdash_s \quad \langle s,d,u,k_1,k_2,k_3\rangle, \quad \uparrow_{\mathrm{sec}(k_2)} \quad \longrightarrow \quad \langle s,d,u,k_1,k_3\rangle.$$

This rule simply processes the acknowledgment from the secure-processing layer indicating that the discovery message has been sent.

Rule ND.1.3

$$\begin{array}{ll} \vdash_s & \langle s,d,u,k_1,k_3\rangle, \quad \uparrow_{\mathrm{eresp}(k_3)} \mathbf{R}(c) \longrightarrow \quad \downarrow_{\mathrm{eresp}(u,k_3)}, \quad \langle s,d,u,k_1,k_3\rangle \\ & \text{new } k_3 \\ & \text{if } c \neq d. \end{array}$$

This rule is only executed if the tunnel has not been set up with the destination node d. The establishment responder writes a term indicating that a tunnel has been set up with a gateway c rather than with the host d. So the initiator invokes the establishment responder again as there is at least one more tunnel to set up in the complex.

Rule ND.1.4

$$\begin{array}{ll} \vdash_s & \langle s,d,u,k_1,k_3\rangle, \quad \uparrow_{\mathrm{eresp}(k_3)} \mathbf{R}(d), \quad \Uparrow_{\mathrm{sec}(u)} \mathsf{P}(d,s,\mathrm{Fin}), \\ & \mathsf{Mech}(e \longrightarrow s : u : \beta^i) \otimes \Pi^i, \quad \mathsf{Mech}(s \longrightarrow e : u : \beta^o) \otimes \Pi^o \longrightarrow \\ & \uparrow_{\mathrm{dis}(k_1)}, \quad \mathsf{Mech}(d \longrightarrow s : u : \beta^i) \otimes \Pi^i, \quad \mathsf{Mech}(s \longrightarrow d : u : \beta^i) \otimes \Pi^o. \end{array}$$

This rule is only executed if the tunnel has been set up with the destination node d. The establishment responder has written an acknowledgment indicating that the last tunnel in the complex has been set up with host d and host d has sent a Fin indicating that the tunnel is set up at d. The filters installed during establishment of the first tunnel are $s \longrightarrow \mathrm{dom}(GW1)$ and $\mathrm{dom}(GW1) \longrightarrow s$, which is a bit too permissive, whence we restrict the filters to $s \longrightarrow d$ and $d \longrightarrow s$.

The establishment responder runs as a daemon at all nodes on the dataflow path. Upon arrival of a discovery packet, the responder invokes the establishment layer to set up a tunnel with the host that initiated the protocol. Upon notification from the establishment layer that this tunnel has been set up, the protocol takes one of two actions. If the current node is not the final destination, then the protocol releases the discovery packet sending it toward its destination. The protocol then waits for the establishment reply message to arrive from the next node on the path and updates the mechanism database to 'open a hole' allowing traffic from this node to exit the gateway. The protocol then waits for the arrival of the establishment responder from the initiating host. If the current node is the final destination, then a Fin message is sent to the initiating host indicating that the protocol has terminated at the final node on the path. We now examine the six rules that comprise the establishment responder.

Rule ND.2.1

$\vdash_a \quad \Uparrow_{\mathrm{sec}(u)} \mathsf{P}(s, d, \mathsf{C}(\mathrm{Dis}(s, u))) \longrightarrow \quad \downarrow_{\mathrm{est}(u,k_1)} \mathsf{E}(s, s, \mathrm{dom}(a)), \quad \langle a, s, d, u, k_1 \rangle$

new k_1.

A discovery packet arrives, triggering a call to the establishment layer to set up a tunnel with the initiating host s. Note that the filters installed by establishment will be $s \longrightarrow \mathrm{dom}(a)$ because, as explained above, it has to allow traffic from nodes that have yet to be discovered.

Rule ND.2.2

$\vdash_a \quad \langle a, b, s, d, u, k_1 \rangle, \quad \uparrow_{\mathrm{est}(k_1)} \longrightarrow$

$\downarrow_{\mathrm{sec}(u,k_2)} \mathsf{P}(s, d, \mathsf{C}(\mathrm{Dis}(s, u))), \quad \langle a, s, d, u, k_2 \rangle$

new k_2

if $a \neq d$.

This rule is executed if the node a is not the final destination d. Upon notification that the establishment protocol has completed, release the discovery packet.

Rule ND.2.3

$$\begin{aligned}\vdash_a \quad & \langle a,s,d,u,k_2\rangle,\quad \Uparrow_{\mathrm{sec}(u)} \mathsf{P}(c,s,\mathsf{X}(\mathsf{Req}(s,e,u,\iota_c,\Xi^c,g))),\quad \uparrow_{\mathrm{sec}(k_2)} \quad \longrightarrow \\ & \downarrow_{\mathrm{sec}(u,k_3)} \mathsf{P}(c,s,\mathsf{X}(\mathsf{Req}(s,e,u,\iota_c,\Xi^c,g))), \\ & \mathsf{Mech}(c \to s : u : \mathsf{Bndl}[]) \otimes \Pi^i,\quad \mathsf{Mech}(s \to c : u : \mathsf{Bndl}[]) \otimes \Pi^o, \\ & \langle a,s,d,u,c,e,k_3\rangle \\ & \text{new } k_3 \\ & \text{if } a \neq d.\end{aligned}$$

This rule is executed if the node a is not the final destination d. When the establishment request arrives from the next node discovered, the secure layer by default passes it up for processing. Since there is no establishment responder waiting, the discovery layer examines the packet to find out the address of this node and updates the mechanism databases to allow the traffic from the newly discovered node to arrive, but not in a tunnel. Thus any node spoofing the newly discovered node can perform a cramming attack.

Rule ND.2.4

$$\begin{aligned}\vdash_a \quad & \langle a,s,d,u,c,e,k_3\rangle,\quad \Uparrow_{\mathrm{sec}(u)} \mathsf{P}(s,c,\mathsf{X}(\mathsf{Rep}(s,e,u,\iota_s,\Xi^u,g))),\quad \uparrow_{\mathrm{sec}(k_3)} \quad \longrightarrow \\ & \downarrow_{\mathrm{sec}(u,k_4)} \mathsf{P}(s,c,\mathsf{X}(\mathsf{Rep}(s,e,u,\iota_s,\Xi^u,g))),\quad \langle a,s,d,u,k_4\rangle \\ & \text{new } k_4 \\ & \text{if } a \neq d.\end{aligned}$$

This rule is only executed if the node a is not the final destination d. If the secure layer acknowledges that the establishment request has been relayed and an establishment reply has been received, then the establishment reply is relayed toward its destination.

Rule ND.2.5

$$\begin{aligned}\vdash_a \quad & \langle a,s,d,u,k_1,k_2\rangle,\quad \uparrow_{\mathrm{est}(k_1)} \quad \longrightarrow \quad \downarrow_{\mathrm{sec}(u,k_2)} \mathsf{P}(d,s,\mathrm{Fin}),\quad \langle a,s,d,u,k_2\rangle \\ & \text{new } k_2 \\ & \text{if } a = d.\end{aligned}$$

This rule is only executed if a is the final destination d. Upon notification that the tunnel between d and s has been set up, the protocol sends the FIN message to the initiating host s indicating that state for the $s \leftrightarrow d$ tunnel has been installed at d.

Rule ND.2.6

$$\vdash_a \quad \langle a, s, d, u, k_2 \rangle, \quad \uparrow_{\mathrm{sec}(k_2)} \; \longrightarrow \cdot.$$

This rule simply consumes the acknowledgment from the secure layer, indicating that either the establishment reply message originating at the next node on the path has been sent or the Fin message has been sent.

6.3.2 Completeness Theorem for Nested Discovery

The completeness theorem for this protocol differs from that of the concatenated protocol in that the node that initiated discovery must possess credentials to satisfy the policies of all the nodes on the dataflow path. The theorem is formally stated as follows.

Theorem 6.3 *Let $T = M_1, \ldots, M_n$ be a trace of the execution of the nested discovery protocol session u initiated at node a_1 $(= s)$ to communicate with node a_n $(= d)$ and the gateway policies and node credentials are assumed to be fixed in T. Assume $u \notin \mathfrak{L}(M_1)$ and that the forwarding tables indicate that the nodes on the dataflow path are $a_2, \ldots, a_{n-1}$ respectively. Assume that nested discovery protocol session u always runs successfully when initiated in $\mathfrak{G}(M_1)$ and that the authorization layer always returns true when checking a discovery policy in T. For each i $(2 \leq i \leq n)$ there exists a node term $\uparrow_{\mathrm{auth}(k)} \mathsf{GWPol}(u, true) \,@\, a_i$ such that the following statement holds: if*

$$\textit{if} \;\; (\{K_{a_i} \Rightarrow K_{a_1}\} \cup \Xi^{a_1}) \;\models_{a_i} \Theta_{a_1 \leftrightarrow a_n} \,@\, a_i, \;\; \textit{then} \uparrow_{\mathrm{auth}(k)} \mathsf{GWPol}(u, \textit{true}) \,@\, a_i \in T.$$

Proof: The proof proceeds by induction on the number of gateways.

For the base case, there are no intermediate gateways, only source a_1 and destination a_2. Consider the execution of the discovery protocol. At node a_1 **Rule ND.1.1** releases the discovery packet and invokes the establishment responder. Upon receiving the discovery packet, **Rule ND.2.1** invokes the establishment-layer initiator to set up a pair of associations between a_1 and a_2. The establishment-layer responder (**Rule E.2.2**) sends $\Xi^u = \Xi^{a_1} \cup \{K_{a_2} \Rightarrow K_{a_1}\}$ to a_2 in the establishment response message. Upon receiving this message, **Rule E.1.2** invokes the authorization layer. Since it was assumed that $\Xi^{a_1} \cup \{K_{a_2} \Rightarrow K_{a_1}\} \models_{a_2} \Theta_{a_1 \leftrightarrow a_2} \,@\, a_2$, we can conclude that the authorization layer returns true and the base case holds.

Assume the theorem is true for i gateways and show it holds for $i + 1$ gateways. From the induction hypothesis we know that tunnels were set up that authenticate the packet at the gateways $a_2, \ldots, a_i$. Gateway i is not the final destination so at a_i rule **Rule ND.2.2** executes, releasing the discovery packet. Upon receiving this message, gateway a_{i+1} executes **Rule ND.2.1**, which invokes the establishment-layer initiator to set up a pair of associations between a_1 and a_{i+1}. When the establishment request message arrives at a_i, **Rule ND.2.3** allows it to pass, sending it onto a_1 in the association flowing from a_i to a_1. When the establishment request message arrives at a_1, the establishment-layer responder (**Rule E.2.2**) sends $\Xi^{a_1} \cup \{K_{a_{i+1}} \Rightarrow K_{a_1}\}$ to a_{i+1}. Since it was assumed that

$$\{K_{a_{i+1}} \Rightarrow K_1\} \cup \Xi^{a_1} \models_{a_{i+1}} \Theta_{a_1 \leftrightarrow a_n} \mathbin{@} a_{i+1},$$

we can conclude that the authorization layer returns true and the theorem holds for $i+1$ gateways. The theorem follows from induction. □

Although the nested discovery protocol satisfies its completeness theorem, the correctness criteria is somewhat limiting in comparison to the completeness theorem for the concatenated protocol. For instance, suppose Alice does not posses the credentials to traverse GW2, but GW1 does, then the concatenated protocol will succeed while the nested protocol fails. We next consider how to modify our nested discovery protocol so that it satisfies the same completeness theorem as the concatenated protocol.

6.3.3 Modified Nested Discovery

The protocol presented above for constructing a nested tunnel complex fails in situations where the concatenated protocol succeeds because it fails to collect credentials as it executes in the fashion of our concatenated discovery protocol. In this section, we modify our nested tunnel protocol so that it satisfies the same completeness theorem as the concatenated protocol.

Recall that when the concatenated protocol executes, each newly discovered gateway invokes establishment to set up a tunnel with the previously discovered node and that the establishment response message passes session u's credentials along with that node's credentials and a new credential stating that the newly discovered node speaks for the establishment responder. Credentials are collected at each node and migrated to successive nodes on the path. During the exchange of acknowledgments, the node that had acted as the establishment responder sends the newly discovered node the cumulative discovery policies belonging to the host that initiated discovery as well as those of the previously discovered nodes. In the case of the nested discovery protocol given above, each node invokes establishment with the host that initiated discovery; hence, there is not the opportunity to 'collect' credentials and discovery policies as the protocol executes. To transform our nested gateway protocol into one that succeeds under the same circumstances as the concatenated discovery protocol, we modify the protocol so that credentials at each newly discovered gateway get migrated back to the initiating host.

We now illustrate our revised protocol using the example from above. Alice sends out a discovery packet $\mathsf{P}(A, B, \mathsf{C}(Dis(A, u)))$ that gets intercepted by gateway $GW1$,

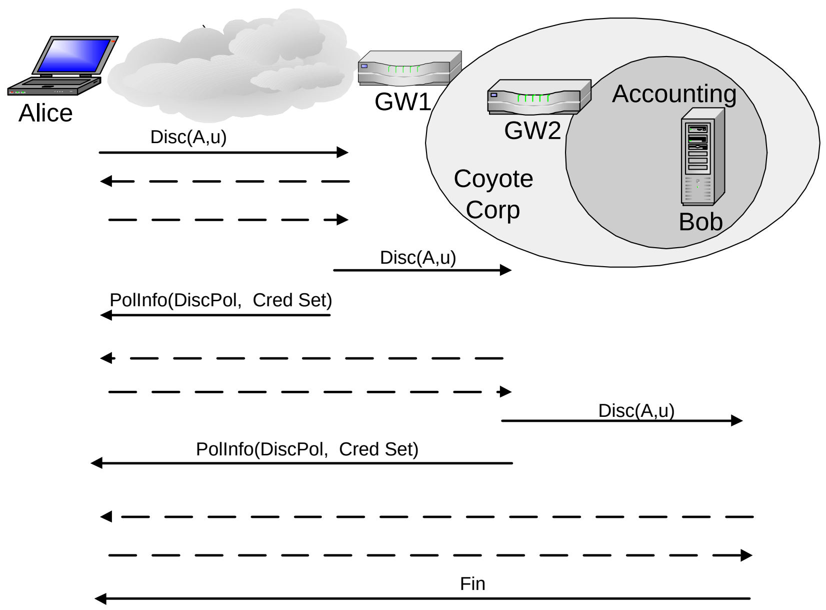

Figure 6.7: Modified Nested Discovery Execution

which invokes establishment with Alice. The mechanism filters at both ends of the tunnel say that all traffic belonging to session u flowing between Alice and any node in the administrative domain of $GW1$ is directed into the associations being set up. When establishment at $GW1$ has terminated, $GW1$ appends its credentials Ξ^{GW1} to the credential set Ξ^{u} obtained from Alice during establishment and sends a message to Alice containing $GW1'$s discovery policy Φ^{GW1} and the updated credential set Ξ^{u}. Recall that the tunnel flowing from GW1 to Alice is guaranteed to have been set up when the establishment layer writes the $\uparrow$ term so this message will safely travel in the GW1 to Alice association. Note that the credential set sent in this message is composed of Ξ^{A}, Ξ^{GW1}, and $\{K_{GW1} \Rightarrow K_A\}$. The discovery packet is released and gets intercepted by $GW2$. Gateway $GW2$ then invokes establishment to set up the tunnel between Alice and $GW2$. When the establishment request message from $GW2$ to Alice arrives at $GW1$, the secure layer passes it up for processing since it is an establishment message. There is no establishment-responder processes waiting to process this message, instead the discovery protocol writes entries in the mechanism database saying all traffic in session u arriving from $GW2$ will not be in a tunnel. The establishment request message is then sent on to Alice in the association flowing from $GW1$ to Alice. Note that the establishment response message will arrive at $GW1$ in the association flowing from Alice to $GW1$ and will hence be authenticated. When establishment at $GW2$ has terminated, $GW2$ appends its credentials Ξ^{GW2} to the credential set Ξ^{u} obtained from Alice during establishment and sends a message to Alice containing the discovery policy Φ^{GW2} and the updated credential set Ξ^u. In this case, the credential set that gets sent is composed of Ξ^A, Ξ^{GW1}, Ξ^{GW2} $K_{GW1} \Rightarrow K_A$, and $K_{GW2} \Rightarrow K_A$. The discovery packet is released and intercepted by Bob. From this point on the protocol execution is the same as with the original nested discovery protocol. The revised protocol is depicted in Figure 6.7, where the solid lines represent the discovery protocol messages and the dotted lines represent establishment messages.

6.3.4 Rules for Modified Nested Discovery

The rules for the modified nested discovery protocol are similar to those of the original protocol except that after a tunnel is setup between Alice and a gateway on the dataflow path, the newly discovered gateway sends its discovery policy as well as its credentials back to the initiating host in the recently setup tunnel.

The four rules that make up the modified nested discovery initiator are given as follows.

Rule MND.1.1

$$\begin{array}{lll} \phi, \Xi^a & \vdash_s & \downarrow_{\text{dis}(u,k_1)} \mathsf{D}(s,d) \longrightarrow \\ & & \downarrow_{\text{sec}(u,k_2)} \mathsf{P}(s,d,\mathsf{C}(\text{Dis}(s,u))), \\ & & \downarrow_{\text{eresp}(u,k_3)}, \quad \mathsf{Discs}\{\phi\}^u, \quad \Xi^u, \\ & & \langle s,d,u,k_1,k_2,k_3\rangle \\ & & \text{new } k_2, k_3 \\ & & \text{where } \Xi^u = \Xi^a. \end{array}$$

The discovery protocol session u is initiated at node s to communicate with node d by writing a $\downarrow_{\text{dis}(u,k_1)} \mathsf{D}(s,d)$ term to the multiset. **Rule MND.1.1** sends a discovery packet toward d and invokes the establishment responder processes, with session identifier u, to set up a tunnel with the newly discovered gateway. The discovery policy Φ^u is initialized to $\phi\,@\,s$, which is the discovery policy of node s; and the credential set for the session Ξ^u is initialized to Ξ^s, the credential set of node s.

Rule MND.1.2

$$\vdash_s \quad \langle s,d,u,k_1,k_2,k_3\rangle, \quad \uparrow_{\text{sec}(k_2)} \quad \longrightarrow \quad \langle s,d,u,k_1,k_3\rangle$$

This rule processes the acknowledgment from the secure processing layer that the discovery message has been sent.

Rule MND.1.3

$$\begin{aligned}\Phi^u \vdash_s \quad & \langle s, d, u, k_1, k_3\rangle,\ \Xi^u,\ \Uparrow_{\mathrm{sec}(u)} \mathsf{P}(c, s, \mathrm{PolInfo}(\phi', \Xi'^u))\ \uparrow_{\mathrm{eresp}(k_3)} \mathbf{R}(c) \longrightarrow \\ & \Phi^u \cup \phi',\ \ \Xi'^u,\ \downarrow_{\mathrm{eresp}(u,k_3)}\ ,\ \ \langle s, d, u, k_1, k_3\rangle \\ & \text{new } k_3 \\ & \text{if } c \neq d.\end{aligned}$$

The establishment responder has written an acknowledge met indicating that a tunnel has been set up with a gateway c rather than with the host d. The old credential set is removed. A message has arrived containing the discovery policy at c and the a new credential set Ξ'^u and the discovery policy from c, which is added to the discovery policy at s. The initiator also invokes the establishment responder again as there is at least one more tunnel to set up in the complex.

Rule MND.1.4

$$\begin{aligned}\vdash_s \quad & \langle s, d, u, k_1, k_3\rangle,\ \ \uparrow_{\mathrm{eresp}(k_3)} \mathbf{R}(d),\ \ \Uparrow_{\mathrm{sec}(u)} \mathsf{P}(d, s, \mathrm{Fin}), \\ & \mathsf{Mech}(e \longrightarrow s : u : \beta^i) \otimes \Pi^i,\ \ \mathsf{Mech}(s \longrightarrow e : u : \beta^o) \otimes \Pi^o \quad \longrightarrow \\ & \uparrow_{\mathrm{dis}(k_1)}\ ,\ \ \mathsf{Mech}(d \longrightarrow s : u : \beta^i) \otimes \Pi^i,\ \ \mathsf{Mech}(s \longrightarrow d : u : \beta^i) \otimes \Pi^o.\end{aligned}$$

The establishment responder has written an acknowledgment indicating that the last tunnel in the complex has been set up with host d and host d has sent a Fin indicating that the tunnel is set up at d. Note that the filters installed during establishment of the first tunnel are $s \longrightarrow \mathrm{dom}(GW1)$ and $\mathrm{dom}(GW1) \longrightarrow \downarrow_{\mathrm{dis}(k)}$, so we now restrict the filters to $s \longrightarrow d$ and $d \longrightarrow s$.

The six rules that comprise the modified nested discovery protocol are given as follows.

Rule MND.2.1

$$\begin{aligned}\vdash_a \quad & \Uparrow_{\mathrm{sec}(u)} \mathsf{P}(s, d, \mathsf{C}(\mathrm{Dis}(s, u))) \longrightarrow \quad \downarrow_{\mathrm{est}(u,k_1)} \mathsf{E}(s, s, \mathrm{dom}(a)),\ \ \langle a, s, d, u, k_1\rangle \\ & \text{new } k_1.\end{aligned}$$

A discovery packet arrives, triggering a call to the establishment layer to set up a tunnel with the initiating host s. Note that the filters installed by establishment will be $s \longrightarrow \mathrm{dom}(a)$ because, as explained above, it has to allow traffic from nodes that have yet to be discovered.

Rule MND.2.2

$$\begin{array}{lll} \phi^c, \Xi^u & \vdash_a & \langle a, s, d, u, k_1 \rangle, \quad \uparrow_{\text{est}(k_1)} \quad \longrightarrow \\ & & \downarrow_{\text{sec}(u,k_2)} \mathsf{P}(s, d, \mathsf{C}(\text{Dis}(s, u))), \\ & & \downarrow_{\text{sec}(u,k_3)} \mathsf{P}(c, s, \text{PolInfo}(\phi^c, \Xi^u \cup \Xi^a)), \\ & & \langle a, s, d, u, k_2, k_3 \rangle \\ & & \text{new } k_2, k_3 \\ & & \text{if } a \neq d. \end{array}$$

This rule is executed if the node a is not the final destination d. Upon notification that the establishment protocol has completed, the protocol sends a message to s containing the discovery policy at a as well as the credential set Ξ^u received during establishment joined with the credential set Ξ^a from gateway a. The discovery packet is sent on its way.

Rule MND.2.3

$$\begin{array}{ll} \vdash_a & \langle a, s, d, u, k_2, k_3 \rangle \\ & \Uparrow_{\text{sec}(u)} \mathsf{P}(c, s, \mathsf{X}(\mathsf{Req}(s, e, u, \iota_c, \Xi^c, g))) \ \uparrow_{\text{sec}(k_2)} , \ \uparrow_{\text{sec}(k_3)} \quad \longrightarrow \\ & \downarrow_{\text{sec}(u,k_4)} \mathsf{P}(c, s, \mathsf{X}(\mathsf{Req}(s, e, u, \iota_c, \Xi^c, g))), \\ & \mathsf{Mech}(c \rightarrow s : u : \mathsf{Bndl}[]) \otimes \Pi^i, \quad \mathsf{Mech}(s \rightarrow c : u : \mathsf{Bndl}[]) \otimes \Pi^o, \\ & \langle a, s, d, u, c, e, k_4 \rangle \\ & \text{new } k_4 \\ & \text{if } a \neq d. \end{array}$$

This rule is executed if the node a is not the final destination d. When the establishment request arrives from the next node discovered, the secure layer by default passes it up for processing. Since there is no establishment responder waiting, the discovery layer examines the packet to find out the address of this node and updates the mechanism databases to allow the traffic from the newly discovered node to arrive, but not in a tunnel. Thus any node spoofing the newly discovered node can perform a cramming attack.

Rule MND.2.4

$$\begin{array}{ll} \vdash_a & \langle a, s, d, u, c, e, k_4 \rangle, \\ & \Uparrow_{\mathrm{sec}(u)} \mathsf{P}(s, c, \mathsf{X}(\mathsf{Rep}(s, e, u, \iota_s, \Xi^u, g))), \quad \uparrow_{\mathrm{sec}(k_3)} \longrightarrow \\ & \downarrow_{\mathrm{sec}(u,k_5)} \mathsf{P}(s, c, \mathsf{X}(\mathsf{Rep}(s, e, u, \iota_s, \Xi^u, g)))) \\ & \langle a, s, d, u, k_4 \rangle \\ & \text{new } k_5 \\ & \text{if } a \neq d. \end{array}$$

This rule is only executed if the node a is not the final destination d. If the secure layer acknowledges that the establishment request has been relayed and a establishment reply has been received, then rely the establishment reply is relayed toward its destination.

Rule MND.2.5

$$\begin{array}{ll} \vdash_a & \langle a, s, d, u, k_1, k2 \rangle, \quad \uparrow_{\mathrm{est}(k_1)} \longrightarrow \quad \downarrow_{\mathrm{sec}(u,k_2)} \mathsf{P}(d, s, \mathrm{Fin}), \quad \langle a, s, d, u, k_2 \rangle \\ & \text{new } k_2 \\ & \text{if } a = d. \end{array}$$

This rule is only executed if a is the final destination d. Upon notification that the tunnel between d and s has been set up, the protocol sends the FIN message to the initiating host s indicating that state for the $s \leftrightarrow d$ tunnel has been installed at d.

Rule MD.2.6

$$\vdash_a \quad \langle a, s, d, u, k_3 \rangle, \quad \uparrow_{\mathrm{sec}(k_3)} \longrightarrow \cdot.$$

This rule simply consumes the acknowledgment from the secure indicating that either the establishment reply message originating at the next node on the path has been sent or the Fin message has been sent.

6.3.5 Completeness Theorem for Modified Nested Discovery

Our goal in developing the modified nested discovery protocol was to have a protocol that satisfied the same completeness theorem as the concatenated protocol. Recall that this theorem says the following. Assume that if every node possessed every credential, then the discovery protocol would run successfully and that the discovery policies are always satisfied. Suppose node a_i is the last node discovered by the protocol, then if the credentials located at $a_1, \ldots, a_{i-1}$ satisfy the gateway policy at a_i the trace should record the authorization layer returning true at this node.

Theorem 6.4 *Let $T = M_1, \ldots, M_n$ be a trace of the execution of the modified nested discovery protocol session u initiated at node a_1 $(= s)$ to communicate with node a_n $(= d)$ and the gateway policies and node credentials are assumed fixed in T. Assume $u \notin \mathfrak{L}(M_1)$ and that the forwarding tables indicate that the nodes on the dataflow path are $a_2, \ldots, a_{n-1}$. Assume that modified nested discovery protocol session u always runs successfully when initiated in $\mathfrak{G}(M_1)$ and that the authorization layer always returns true when checking a discovery policy in T. For each i $(2 \leq i \leq n)$ there exists a node term $\uparrow_{\text{auth}(k)} \mathsf{GWPol}(u, true) @ a_i$ such that the following statement holds: if*

$$\bigcup_{1 \leq l < i} (\{K_{l+1} \Rightarrow K_1\} \cup \Xi^{a_l}) \models_{a_i} \Theta_{a_1 \leftrightarrow a_n} @ a_i$$

then

$$\uparrow_{\text{auth}(k)} \mathsf{GWPol}(u, true) @ a_i \in T.$$

Proof: The proof proceeds by induction on the number of gateways.

For the base case there are no intermediate gateways only source a_1 and destination a_2. Consider the execution of the modified nested discovery protocol. At node a_1, **Rule MND.1.1** releases the discovery packet and invokes the responder process. Upon receiving the discovery packet, **Rule MND.2.1** invokes establishment to set up a pair of associations between a_1 and a_2. The establishment layer responder (**Rule E.2.2**) sends Ξ^u $(= \Xi^{a_1} \cup \{K_{a_2} \Rightarrow K_{a_1}\})$ to a_2 in the establishment response message. Upon receiving this message **Rule E.1.2** invokes the authorization layer. Since it was assumed that $(\Xi^{a_1} \cup \{K_{a_2} \Rightarrow K_{a_1}\}) \models_{a_2} \Theta_{a_1 \leftrightarrow a_2} @ a_2$, we can conclude that the authorization layer returns true.

Suppose the statement is true for i gateways we show it is true for $i+1$ gateways. It follows from the induction hypothesis that the discovery protocol ran to the point that it had successfully set up tunnels between a_1 and a_j, where $2 \leq j \leq i$. It remains to show that the protocol successfully sets up the pair of associations between a_1 and a_{i+1}. **Rule MND.2.2** executes at gateway a_i and releases the discovery packet. Upon receiving this message, node a_{i+1} executes **Rule MND.2.1** that invokes the establishment-layer initiator to set up a pair of associations between a_1 and a_{i+1}. The establishment-layer responder (**Rule E.2.2**) sends $\Xi^u = \Xi^u \cup \Xi^{a_i} \cup \{K_{a_{i+1}} \Rightarrow K_{a_1}\}$ to a_{i+1}. It follows from the induction hypothesis that the authorization succeeded at $a_2, \ldots, a_i$ hence the discovery polices and credential sets at these nodes have been migrated from these nodes to a_1. So the credential set sent from a_1 to a_{i+1} is

$$\bigcup_{1 \leq l \leq i} (\Xi^{a_l} \cup \{K_{a_{l+1}} \Rightarrow K_{a_1}\}).$$

Upon receiving the establishment response message, **Rule E.1.2** invokes the authorization layer. Since it was assumed that

$$\bigcup_{1 \leq l \leq i} (\Xi^{a_l} \cup \{K_{a_{l+1}} \Rightarrow K_{a_1}\}) \models_{a_{i+1}} \Theta_{a_1 \leftrightarrow a_n} @ a_{i+1},$$

we can conclude that the authorization layer returns true. The theorem follows from induction. □

6.4 Conclusion

In this chapter, we have introduced a class of tunnel-complex protocol called discovery protocols are that designed to aide in the construction of tunnel complexes by discovering gateways and setting up point-to-point tunnels that form the desired tunnel-complex. We have also introduced the role of discovery policies in safeguarding the process. In addition, we discussed the role of the discovery protocols in delivering credentials needed to satisfy the policies at the gateways. We developed concrete discovery protocols that set up a complex of concatenated tunnels and that set up a complex of nested tunnels. Completeness theorems that characterize the functional correctness of the protocols were introduced. In the case of the nested tunnel complex, we saw that a simple discovery protocol seemed 'weaker' than the concatenated protocol because it delivered fewer credentials to the gateways than did the concatenated tunnel protocol, which was reflected in their respective completeness theorems. A modified nested discovery protocol was presented that satisfied the same completeness theorem as the concatenated discovery protocol. This illustrates how slightly different protocols can have significantly different behavior. For simplicity's sake, we have assumed only one of these protocols will be active in the network at one time. Adding either an additional field to the discovery message or having distinguishing constructors would allow us it execute each of them simultaneously. We recognize that we could have developed a soundness theorem that is the dual to our completeness theorem and similar soundness and completeness theorems could have been developed for discovery polices, but we believe that our completeness theorem for credentials is the basic functional correctness criteria that characterizes discovery protocols. Discovery protocols are a nascent area of investigation, which lead us to concentrate our attention on relatively simple discovery protocols with the hope that the current effort can serve as a foundation for future work. In the next chapter, our attention shifts from functional correctness to DoS threats.

Chapter 7

Denial of Service Threats

Discovery protocols ease the burden of configuring tunnel complexes. Once set up, a tunnel-complex can protect nodes from DoS threats as we saw with L3A in Chapter 2. Having been thwarted from attacking one target, an adversary may choose to target the discovery protocols themselves. If the attacker can prevent the protocol from setting up the tunnel complex or coerce it into constructing an ill-formed tunnel through which communication cannot proceed, then service is denied. Another way to deny service is by overwhelming resources at the gateways, for instance, filling up databases that may be located in memory or consuming CPU resources. In this chapter, we analyze the vulnerability of our discovery protocols to DoS threats.

The remainder of this chapter is organized as follows. In the first section, we introduce a classification of threats and attacker capabilities that informs our analysis. In the second section, we introduce a version of the tunnel calculus establishment layer that had been modified to include a DoS protection scheme. In the third section, we use the tunnel calculus to model and analyze attacks targeted at tunnel establishment. In section four, we model and analyze attacks against discovery protocols. In section five, we apply a cost model to analyze the previously introduced attackers in terms of attacks targeting computational resources. In section six, we formulate and prove a theorem saying that a specific collection of attackers cannot succeed in exploiting an executing discovery protocol by forcing it to perform high-cost operations.

7.1 DoS Attacks

Examining the discovery protocols in Chapter 6, we see that gateways respond to the arrival of a discovery packet by invoking the establishment layer, which writes to the association and mechanism databases. In practice, the database structures are usually kept in memory for performance reasons. This exposes the system to attacks that exhaust these resources by filling the structures with bogus entries. An attack can also deny service by forcing a gateway to write state that breaks the tunnel complex or forces a discovery protocol into a state where the protocol cannot progress. The establishment protocol incorporates protections that place barriers in front of an attacker before state is written. These protections take the form of digital signatures and credential verification, but an attacker can exploit the computational expense of performing these operations and consume CPU resources resulting in degraded gateway performance. To facilitate our analysis, we designate classifications that distinguish attacks and attacker capabilities.

Throughout this chapter, we shall say nodes are *honest* if they execute the discovery protocol and tunnel calculus layers as defined. The credentials and keys stored on honest nodes are assumed to be secure. An attacker is assumed to only be able to affect an honest node by sending it messages. Only honest gateways are presumed to possess the proper credentials to traverse gateways. We shall not consider the case where an honest gateway is corrupted and uses its credentials for malicious purposes.

We classify attacks into two broad categories. *Logical attacks* attempt to deny service by breaking protocol execution or by writing bogus tunnel state to the association or mechanism databases. Although there may be a small amount of state associated with executing discovery and establishment, it is the databases that constitute the state of interest. In our rewriting logic, many terms may be written to a multiset, but only the databases will viewed as 'state' in this chapter. In particular, a failed attack may result in unconsumed terms being left in a multiset, but this does not constitute writing of state as it does not affect the databases. For instance, an attacker that forces the establishment responder into writing state with a different SPI than the establishment initiator will create a broken tunnel. Another way to break a tunnel complex and deny service would be to force the tunnel complex into creating an invalid nesting of tunnels. The second form of attacks are *resource-exhaustion attacks* that attempt to force a target to perform a large number of computationally costly operations in order to tax the CPU on the target and consequently deny service to honest users. An attack aimed at forcing a gateway to allocate state can also be resource-exhaustion attack in that it may fill a table to capacity, but from the point of view of our analysis, it is similar to a logical attack and so we treat it as such. An attack can often be viewed from both the perspective of a logical attack and a resource-exhaustion attack. For instance, suppose a protocol receives a message, performs one or more actions to verify the integrity of the message, and writes state. If the attacker can overcome the integrity checks and force the target to write state, then it succeeds from the view of a logical attacker. If, on the other hand, the integrity checks prove costly, the attacker can flood the node with these messages and exhaust computational resources.

We now classify the capabilities of the attackers under consideration. A Dolev-Yao attacker has control of the network and by definition can simply intercept any message, and therefore always succeeds in denying service. Such a strong attacker model is necessary for proving that key-establishment protocols preserve secrecy and integrity of cryptographic keys in the face of any possible attack, but in analyzing

	Logical Attacks	Exhaustion Attacks
Off-Path Attacker	Exposed	Exposed
	Spoofed	Spoofed
On-Path Attacker	Exposed	Exposed
	Spoofed	Spoofed

Table 7.1: Classification of Attacks

DoS attacks, we must simply admit the futility of preventing an attacker that can simply drop all messages. Instead, we assume that all attackers can synthesize protocol messages, that no attacker possesses the credentials authorizing communication, and that the public-key cryptography is secure. Attackers are classified as either *On-Path* attackers that are able to observe traffic on the network or *Off-Path* attackers that cannot observe traffic on the network, but can synthesize messages. We shall see that on-path attackers can succeed in situations where off-path attackers fail.

Another differentiating factor between attackers is that some attackers use spoofed addresses, signatures, and credentials where others use their actual address. An objective of a DoS attack is to force the target to perform high-cost operations at little cost to the attacker; so sending the target a spoofed address and signature makes sense. For instance, in the TCP SYN attack, an attacker spoofs the source address on a SYN packet and sends it to the target, which responds to the spoofed address. No further effort is required by the attacker. We classify such attackers as *spoofed attackers.* On the other hand, bots acting as DDoS attackers often do expose their address, which is done without exposing the address of the node that originally launched the attack. Since these nodes are victims as well, the original attacker is not affected if the bots consume resources while performing an attack, although it is better if the attack does not draw the attention of the user of the machine that has been compromised. We classify such attackers as *exposed attackers.*

Table 7.1 summarizes our classification of attacks. This table will guide us in the analysis performed later in this chapter. In particular, for each protocol message that can be exploited, we shall construct exposed and spoofed on-path attackers as well as exposed and spoofed off-path attackers. Each attack is analyzed from the perspective of both logical attacks and resource-exhaustion attacks.

7.2 Modified Establishment Protocol

Consider the discovery protocols defined in the previous chapter. Each gateway executes the discovery protocol responder and upon the arrival of a discovery message, the discovery-layer responder invokes the establishment-layer initiator to set up a tunnel with a node advertised in the discovery message. The node advertised in the discovery packet is presumed to be executing the establishment-layer responder. The first rule of the establishment initiator sends a signed establishment-request message to that advertised node. This opens the possibility that the attacker can force the gateway to perform a costly cryptographic operation by simply sending it a discovery message advertising a spoofed address much as the TCP SYN attack does. Protocols such as IKE have incorporated cookies to protect against this kind of attack. The idea is to require that a round trip be performed before committing state or performing costly operations. Puzzles are a variation on this theme that not only require a round trip, but demand that the other node perform a costly computational task, where the objective is to provide protection against an exposed attacker such as bots launching a DDoS attack. In this section, we present a modified version of the establishment layer, incorporating a DoS-protection scheme that is intended to protect gateways from DoS attacks. When a gateway receives a discovery message and invokes the establishment initiator, the establishment initiator now requires a round trip between the establishment initiator running at the newly discovered gateway and the establishment responder running at the node advertised in the discovery message. This exchange must be succeed before the establishment initiator performs a costly operation such as generating a digital signature. This is achieved through the addition of two messages to the establishment protocol defined in Chapter 4. Our modified establishment protocol works as follows. The establishment responder is invoked for a specific session and it awaits the arrival of a DoS-request message in that same session. This provides proof that a round trip has been made between the two nodes. When the establishment initiator is invoked, it sends a DoS-request message, containing the session identifier, to the node that is advertised in the discovery message. This node should be running the establishment-responder process, which processes the DoS-request message and sends a DoS-reply message back to the establishment initiator. Upon receiving this message, the establishment initiator verifies that it is valid, from which it concludes that a round trip has been performed between the two nodes. From this one can conclude that the discovery message intercepted by the gateway advertises a valid address and attacks similar to the TCP SYN attack are thwarted.

We do not define the details of the DoS-protection scheme, but specify that the DoS-request message has format $\mathsf{DoSReq}(u, f(a, K_a^{-1}))$ and the DoS-reply message has the format $\mathsf{DoSRep}(u, h(f(a, K_a^{-1})))$, where f and h are suitable functions. The DoS-reply message is verified by the DoSCheck proposition that verifies the DoS-reply message in a rule and is defined by the equation

$$\begin{cases} \text{DosCheck}(\mathsf{DoSRep}(u, h(f(a, K_a^{-1})))) & = & \text{true} \\ \text{Otherwise} & & \text{false.} \end{cases}$$

Recall that digital signatures are employed to ensure that the establishment messages originate from their purported origin. Also recall that the function $\text{Sign}(K_a^{-1})$ produces a signature of the message sent in the given rule and that the function

CheckSig(K_b, g) is defined by the equation

$$\begin{cases} \text{CheckSig}(K_a, Sign(K_a^{-1}, p)) & = & \text{true} \\ \text{Otherwise} & & \text{false.} \end{cases}$$

The four rules that comprise the modified establishment responder are defined as follows.

Rule ME.1.1

$$\begin{array}{ll} \vdash_a & \downarrow_{\text{est}(u,k_1)} \mathsf{E}(b, s, d) \longrightarrow \\ & \downarrow_{\text{sec}(u,k_1)} \mathsf{P}(a, b, \mathsf{X}(\mathsf{DoSReq}(u, f(a, K_a^{-1})))) \\ & \langle u, a, b, s, d, k_1, k_2 \rangle \\ & \text{new} \quad k_2. \end{array}$$

The discovery layer responder process invokes the establishment layer initiator at node a by writing a $\downarrow_{\text{est}(u,k)} \mathsf{E}(b, s, d)$ term, where b is the responder and s and d are the packet filters to be installed in the mechanism database.

Rule ME.1.2

$$\begin{array}{lll} \Xi^a & \vdash_a & \langle u, a, b, s, d, k_1, k_2 \rangle, \quad \uparrow_{\text{sec}(k_2)}, \\ & & \Uparrow_{\text{sec}(u)} \mathsf{P}(b, a, \mathsf{X}(\mathsf{DoSRep}(u, h(f(a, K_a^{-1}))))) \longrightarrow \\ & & \downarrow_{\text{sec}(u,k_3)} \mathsf{P}(a, b, \mathsf{X}(\mathsf{Req}(s, d, u, \iota_a, \Xi^a, g))), \\ & & \langle u, a, b, s, d, k_1, k_3, \iota_a \rangle \\ & & \text{if } \exists \mathsf{In}(b, \iota_x) \in \Sigma \text{ then } \iota_a = \iota_x \text{ else } \iota_a \text{ isnew} \\ & & \text{new} \quad k_3 \\ & & \text{where } g = \text{Sign}(K_a^{-1}) \\ & & \text{if DoSCheck().} \end{array}$$

This rule executed upon the arrival of a valid DoS-reply 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 a signed establishment-request message to node b. The semantic function sign() produces a signature of the message being sent using the private key of a.

Rule ME.1.3

$$\begin{aligned}\Theta \ \vdash_a \ & \langle u, a, b, s, d, k_1, k_3, \iota_a \rangle, \\ & \uparrow_{\text{sec}(k_3)}, \Uparrow_{\text{sec}(u)} \mathsf{P}(b, a, \mathsf{X}(\mathsf{Rep}(s, d, u, \iota_a, \iota_b, \Xi^u, g'))) \longrightarrow \\ & \downarrow_{\text{auth}(u,k_4)} \mathsf{Ai}(a, b, s, d, \Theta, \Xi^u), \\ & \langle u, a, b, s, d, k_1, k_4, \iota_a, \iota_b \rangle \\ & \text{new} \ \ k_4 \\ & \text{if} \ \ \text{CheckSig}(K_b, g').\end{aligned}$$

Upon receiving the establishment-response message, the initiator a verifies the signature and invokes the authorization layer to verify that the credential set Ξ^u satisfies the gateway policy $\Theta_{a \leftrightarrow b}$.

Rule ME.1.4

$$\begin{aligned}\vdash_a \ & \langle u, a, b, s, d, k_1, k_4, \iota_a, \iota_b \rangle, \\ & \Sigma, \Pi^i, \Pi^o, \uparrow_{\text{auth}(k_4)} \mathsf{GWPol}(u, \text{true}) \longrightarrow \\ & \quad \Sigma \cup \{\mathsf{Out}(b, \iota_b)\}, \ \mathsf{Mech}(d \rightarrow s : u : \mathsf{Bndl}[\mathsf{Out}(b, \iota_b)]) \otimes \Pi^o, \\ & \quad \Sigma \cup \{\mathsf{In}(b, \iota_a)\}, \ \mathsf{Mech}(s \rightarrow d : u : \mathsf{Bndl}[\mathsf{In}(b, \iota_a)]) \otimes \Pi^i, \ \uparrow_{\text{est}(k_1)} .\end{aligned}$$

If the authorization layer returns true, then update the association and mechanism databases for both associations and write the establishment acknowledgment term.

The rules that comprise the responder of the modified establishment layer are given as follows.

Rule ME.2.1

$$\begin{aligned}\downarrow_{\text{eresp}(u,k_1)} \quad \vdash_b \ & \Uparrow_{\text{sec}(u)} \mathsf{P}(a, b, \mathsf{X}(\mathsf{DoSReq}(u, f(a, K_a^{-1})))) \longrightarrow \\ & \downarrow_{\text{sec}(u,k_1)} \mathsf{P}(a, b, \mathsf{X}(\mathsf{DoSRep}(u, h(f(a, K_a^{-1}))))) \\ & \text{new} \ \ k_2 \ .\end{aligned}$$

This rule executes if the establishment responder has been invoked and a DoS request possessing possessing the session identifier u is received. The DoS reply is constructed and sent back to the establishment initiator. Observe that the $\downarrow_{\text{eresp}(u,k)}$ term is not consumed here in order to prevent an attacker from breaking protocol execution by simply sending a valid DoS-request message to the node. This term is only consumed in the next rule when a valid establishment-request message arrives. Although, we recognize that this issue is typically handled by timestamps, we would rather not introduce time into the picture a this point. So to facilitate analysis, we incorporated this into the rules.

Rule ME.2.2

$$\begin{array}{ll}\Phi^u \vdash_b & \downarrow_{\mathrm{eresp}(u,k_1)}\ ,\ \ \uparrow_{\mathrm{sec}(k_2)}\ ,\ \Uparrow_{\mathrm{sec}(u)}\ \mathsf{P}(a,b,\mathsf{X}(\mathsf{Req}(s,d,u,\iota_a,\Xi^a,g))) \longrightarrow \\ & \downarrow_{\mathrm{auth}(u,k_3)}\ \mathsf{Ar}(a,b,s,b,\Phi^u,\Xi^a), \\ & \langle u,a,b,s,d,\iota_a,k_1,k_3\rangle \\ & \text{new}\ \ k_3 \\ & \text{if CheckSig}(K_a^{-1},g).\end{array}$$

Upon the arrival of an establishment-request message, the signature is verified and the authorization layer is invoked to verify that the initiator's credential Ξ^a satisfies the discovery policy Φ^u.

Rule ME.2.3

$$\begin{array}{ll}\Xi^b,\ \Xi^u \vdash_b & \langle u,a,b,s,d,\iota_a,k_1,k_3\rangle, \\ & \uparrow_{\mathrm{auth}(k_3)}\ \mathsf{DisPol}(u,\text{true}),\ \Sigma,\ \Pi^i\ \longrightarrow \\ & \Sigma \cup \mathsf{In}(a,\iota_b), \\ & \mathsf{Mech}(d \rightarrow s : u : \mathsf{Bndl}[\mathsf{In}(a,\iota_b)]) \otimes \Pi^i, \\ & \downarrow_{\mathrm{sec}(u,k_4)}\ \mathsf{P}(b,a, \\ & \quad \mathsf{X}(\mathsf{Rep}(s,d,u,\iota_a,\iota_s,\Xi^u \cup \Xi^b \cup \{K_a \Rightarrow K_b\},g'))), \\ & \langle u,a,b,s,d,\iota_a,\iota_b,k_1,k_4\rangle \\ & \text{new}\ \ k_4 \\ & \text{if } \exists \mathsf{In}(a,\iota_x) \in \Sigma \text{ then } \iota_b = i_x \text{ else} \iota_b \text{ is new} \\ & \text{where } g = \text{Sign}(K_b^{-1}).\end{array}$$

Rule ME.2.3 only executes if the authorization layer verifies that the discovery policy is satisfied. If there is an existing association flowing from the initiator to the responder, then it gets reused. Otherwise, a new association is generated. 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 ME.2.4

$$\begin{array}{ll}\vdash_b & \langle u,a,b,s,d,\iota_a,\iota_b,k_1,k_4\rangle,\Sigma,\Pi^o,\uparrow_{\mathrm{sec}(k_4)}\ \longrightarrow \\ & \uparrow_{\mathrm{eresp}(k_1)}\ \mathsf{R}(a),\ \Sigma \cup \{\mathsf{Out}(a,\iota_a)\}, \\ & \mathsf{Mech}(s \rightarrow d : u : \mathsf{Bndl}[\mathsf{Out}(a,\iota_a)]) \otimes \Pi^o.\end{array}$$

Upon acknowledgment that the reply has been sent, entries are made in the association and mechanism databases for the association flowing from b to a.

Note that the discovery protocols presented in the previous chapter will require slight modification to accommodate the additional DoS messages. Everywhere there are rules for relaying establishment request and reply messages, there must be additional rules added for relaying the DoS request and reply messages as well.

7.3 Logical Attacks on Establishment Layer

In this section we analyze logical attacks on the modified establishment protocol presented in the previous section. We assume that attackers lack the perquisite credentials to satisfy gateway policies, otherwise, they would be honest users rather than attackers, but attackers can synthesize protocol messages and address them to any node. Consider the rules for the establishment protocol given above. If an attacker can construct establishment request messages that pattern match, possess a valid signature, and possess valid credentials, then the message is accepted by a node's establishment responder, whence the attacker has succeeded in forcing its target into writing state. Similarly, if an attacker can construct establishment-reply messages that pattern match, possess a valid signature, and possess valid credentials, then it too has succeeded in forcing its target into writing state. An attack on the establishment initiator may need to overcome the DoS protections that have been added. Rather than a general Dolev-Yao like attacker, we define a collection of attackers targeting the messages of the modified establishment protocol. For each establishment-layer message, we consider the following four archetypal attackers: on-path exposed, on-path spoofed, off-path exposed, and off-path spoofed. In this section, we shall evaluate the effectiveness of each of these as logical attacks.

Each of our attackers will be defined in a rule having a $\downarrow_{\mathrm{at}(k)} \mathsf{ATK}$ term on the left-hand side and possibly having parameters indicating the target of the attack.

A on-path attacker is able to observe messages being sent, but is not able to control the network. Recall that the tunnel calculus forwarding-layer rules defined in Chapter 4 does not model a communication medium. Instead, a message at one node is rewritten to another node in **Rule F.1.1** and **Rule F.2.1** consumes the message at the destination node. In this case, the multiset acts as the medium connecting the two nodes. This raises the question as to how we should model an attacker that can observe traffic on the network. One option would have been to add additional rules to the forwarding layer that mimicked a communication medium, but this is not actually necessary simply to model an attacker 'observing' a message. Instead, we view the application of **Rule F.1.1** as sending the message and allow attackers to 'observe' a message before it is consumed by **Rule F.2.1**. The on-path attacker uses the tunnel calculus $t \vdash$ convention, where t is a term representing a message before it is consumed by **Rule F.2.1**; so an attacker observers message t, but does not consume the term. On-path attackers typically take the form

$$\begin{array}{rll} \mathsf{P}(s,d,m)))\,@\,a & \vdash & \downarrow_{\mathrm{at}(k_1)} \mathsf{ATK}()\,@\,\mathsf{e} \longrightarrow \\ & & \downarrow_{\mathrm{sec}(u,k_2)}\, p\,@\,\mathsf{e} \\ & & \text{new} \;\; k_2. \end{array}$$

Both a term invoking the attacker $\downarrow_{\mathrm{at}(k_1)} \mathsf{ATK}()$ and a term indicating a message of a particular format has been sent to node a are in the multiset. The $\downarrow_{\mathrm{at}(k_1)} \mathsf{ATK}()$ is removed from the multiset and a term is rewritten sending a message from the attacker and the original message $\mathsf{P}(s,d,m)))\,@\,a$ also remains in the multiset. The decision to model on-path attackers in this fashion was primarily driven by the desire to preserve the basic forwarding layer defined in Chapter 4.

Attackers use the tunnel calculus new operator to spoof addresses and other information needed to create a message. Recall that values generated by the new operator are assumed to be unique. In general, the on-path exposed attackers spoof the least amount of information in their attack and off-path spoofed attackers spoof the most amount of information in an attack. To keep the presentation concise, we do not include rules that only process acknowledgments from the secure processing layer.

7.3.1 Attacks on DoS Request

The establishment responder is invoked during execution of a discovery protocol to set up the 'next' tunnel in the complex. The establishment-responder processes both a DoS-request message and an establishment-request message before writing state. We shall now investigate attacks targeted at the DoS-request message. The first attacker to be considered is an off-path spoofed attacker that spoofs its address, the session identifier, and the DoS-request value. The attacker then forms a DoS-request message and sends it towards the target address. Recall that when we spoof an address it means we have generated it using the tunnel calculus new operator which means that it is unique. We do not consider the case where an attacker may have guessed a valid address or simply used a valid address such as yahoo.com. The rule for our attacker is given as follows:

$$\begin{array}{ll} \vdash_{\mathsf{e}} & \downarrow_{\mathrm{at}(k_1)} \mathsf{ATK}(b) \longrightarrow \\ & \downarrow_{\mathrm{sec}(u,k_2)} \mathsf{P}(c,b,\mathsf{X}(\mathsf{DoSReq}(u,f(c,K_c^{-1})))) \\ & \text{new} \;\; u,\; c,\; f(c,K_c^{-1}),\; k_2. \end{array}$$

Off-Path spoofed DoS Request 7.1

Suppose node b is executing the establishment responder session v, then upon receiving the DoS-request message from the attacker, the multiset will contain terms

$$\begin{array}{l} \downarrow_{\mathrm{eresp}(v,k)} \ @\, b \\ \Uparrow_{\mathrm{sec}(u)} \mathsf{P}(c, b, \mathsf{X}(\mathsf{DoSReq}(u, f(c, K_c^{-1})))) \,@\, b \end{array}$$

Since the session identifiers do not match, the DoS-request message is not consumed and the protocol continues normal execution. Consequently, the attack cannot prevent progress or force the target to write state.

An off-path attacker willing to expose itself is identical to the spoofed case, but uses its own address. Since the attacker does not posses the information to form a valid DoS reply, this information is spoofed as well. The rule for our attacker is given as follows:

$$\begin{array}{ll} \vdash_{\mathsf{e}} & \downarrow_{\mathrm{at}(k_1)} \mathsf{ATK}(b) \longrightarrow \\ & \downarrow_{\mathrm{sec}(u,k_2)} \mathsf{P}(\mathsf{e}, b, \mathsf{X}(\mathsf{DoSReq}(u, f(\mathsf{e}, K_{\mathsf{e}}^{-1})))) \\ & \text{new} \ \ u,\ k_2,\ f(\mathsf{e}, K_{\mathsf{e}}^{-1}). \end{array}$$

Off-Path Exposed DoS Request 7.2

As with the previous case, this attack will not succeed because the attacker does not have access to the session identifier.

The off-path attackers failed because they did not possess the requisite session identifier. We now consider on-path attackers that can read discovery messages that have been sent. Since discovery messages contain both the session identifier and the address of the node running the establishment responder, the on-path attacker can obtain the information needed to form the DoS-request message by observing the discovery messages in the network. If the attacker only obtained a valid session identifier and has not also obtained an address where that session is running, then the attack would have the same outcome as the off-path attack because the message from the attacker will not contain the expected session identifier. Consequently, all of our on-path attackers are constructed to obtain both a session identifier and the address where that session is executing. The rule for our on-path spoofed attacker is given as follows:

$$\begin{array}{lll} \mathsf{P}(s, d, \mathsf{C}(\mathsf{D}(b, u))) \,@\, a & \vdash & \downarrow_{\mathrm{at}(k_1)} \mathsf{ATK}() \,@\, \mathsf{e} \longrightarrow \\ & & \downarrow_{\mathrm{sec}(u,k_2)} \mathsf{P}(c, b, \mathsf{X}(\mathsf{DoSReq}(u, f(c, K_c^{-1})))) \,@\, \mathsf{e} \\ & & \text{new}\ c,\ k_2. \end{array}$$

On-Path Spoofed DoS Request 7.3

Suppose b is executing an instance of session v, then, upon receiving the DoS request message from this attacker, the multiset will contain terms

$$\begin{array}{l} \downarrow_{\text{eresp}(v,k)} \\ \Uparrow_{\text{sec}(v)} \mathsf{P}(c, b, \mathsf{X}(\mathsf{DoSReq}(v, f(c, K_c^{-1})))). \end{array}$$

Since this is a valid DoS-request message, the attacker forces **Rule ME.2.1** to fire, forming a DoS-reply message and sending it to the spoofed addresses. Notice that no tunnel state is written, but forming the DoS-reply message may require some computational effort. An on-path attacker can send any number of these messages at the node and it will not 'break' the protocol because the state machine responds to any DoS-request message until a valid establishment request message is received.

Our on-path exposed attacker behaves similarly to the spoofed attacker, but uses its own address. The rule for our attacker is given as follows:

$$\begin{array}{lll} \mathsf{P}(s, d, \mathsf{C}(\mathsf{D}(b, u))) \,@\, a & \vdash & \downarrow_{\text{at}(k_1)} \mathsf{ATK}() \,@\, \mathsf{e} \longrightarrow \\ & & \downarrow_{\text{sec}(u,k_2)} \mathsf{P}(\mathsf{e}, b, \mathsf{X}(\mathsf{DoSReq}(u, f(\mathsf{e}, K_{\mathsf{e}}^{-1})))) \,@\, \mathsf{e} \\ & & \text{new } k_2. \end{array}$$

On-Path Exposed DoS Request 7.4

In this case, the attacker has sent the establishment responder the DoS-request message containing a valid session identifier as well as the attacker's address, and **Rule ME.2.1** responds by forming a DoS-reply message to the attacker. In and of itself, this attack does not break the protocol or force it to write state unless followed by a valid establishment request from the attacker that arrives before an establishment-request message arrives from an honest node.

7.3.2 Attacks on DoS Reply

The first action taken by the modified establishment initiator is to send the responder a DoS-request message. Upon receiving a DoS-reply message, the establishment initiator performs a verification of the message via a call to the DoSCheck command. **Rule M.1.2** will not execute unless a valid DoS-reply message is received. In this section, we examine attackers targeted at the DoS-reply message. These attackers generate and send DoS-reply messages with the goal of having them accepted by the establishment initiator as valid.

An off-path spoofed attack on the DoS-reply message cannot obtain the information in a DoS-request message and instead spoofs the required information and sends it to the establishment initiator. Our attacker is defined by the following rule:

$$\begin{aligned}\vdash_{\mathsf{e}}\ \ & \downarrow_{\mathrm{at}(k_1)} \mathsf{ATK}(a) \longrightarrow \\ & \downarrow_{\mathrm{sec}(u,k_2)} \mathsf{P}(c, a, \mathsf{X}(\mathsf{DoSRep}(u, h(f(a, K_{\mathsf{a}}^{-1}))))) \\ & \text{new}\ \ u,\ h(f(a, K_{\mathsf{a}}^{-1})),\ c,\ k_2.\end{aligned}$$

Off-Path Spoofed DoS Reply 7.5

Suppose a is executing an instance of session v, then upon receiving the DoS-reply message from the attacker, the multiset will contain terms

$$\begin{aligned}& \langle v, a, b, s, d, k_0, k_1\rangle \\ & \Uparrow_{\mathrm{sec}(u)} \mathsf{P}(c, b, \mathsf{X}(\mathsf{DoSRep}(u, h(f(a, K_{\mathsf{a}}^{-1}))))) \,@\, a.\end{aligned}$$

Since the source address and the session identifier are spoofed, the message will not pattern match. Hence **Rule ME.1.2** does not execute and consequently, this attacker neither interrupts progress nor forces its target to write state.

An off-path exposed attacker performs the same actions as the previous case, but uses its own address rather than a spoofed address. Our attacker is defined as follows:

$$\begin{aligned}\vdash_{\mathsf{e}}\ \ & \downarrow_{\mathrm{at}(k_1)} \mathsf{ATK}(a) \longrightarrow \\ & \downarrow_{\mathrm{sec}(u,k_2)} \mathsf{P}(b, \mathsf{e}, \mathsf{X}(\mathsf{DoSRep}(u, h(f(\mathsf{e}, K_{\mathsf{e}}^{-1}))))) \\ & \text{new}\ \ u,\ k_2,\ h(f(\mathsf{e}, K_{\mathsf{e}}^{-1})).\end{aligned}$$

Off-Path Exposed DoS Reply 7.6

As with the previous case, the attack has failed to produce a message containing the expected session identifier and expected source address.

If the initiator is awaiting a DoS reply, then it has sent a DoS-request message to a node. A on-path attacker can observe DoS-request messages in the network and use the information in the message to form a DoS reply. For on-path DoS-reply attacks, we define a spoofed attacker as one who is unwilling reveal its address and is also unwilling to generate a DoS-reply message. Instead, the attacker uses the address in the DoS-request message that it has observed. In the case where the DoS-protection mechanism would require this node to perform some computational activity, as do puzzles, it is understandable that an attacker would not wish to commit its own resources. Our attacker is defined as follows:

$$\begin{aligned}& \mathsf{P}(a, b, \mathsf{X}(\mathsf{DoSReq}(u, f(a, K_{\mathsf{a}}^{-1})))) \,@\, b \vdash \\ & \downarrow_{\mathrm{at}(k_1)} \mathsf{ATK}() \,@\, \mathsf{e} \longrightarrow \\ & \downarrow_{\mathrm{sec}(u,k_2)} \mathsf{P}(b, a, \mathsf{X}(\mathsf{DoSRep}(u, h(f(a, K_{\mathsf{a}}^{-1}))))) \,@\, \mathsf{e} \\ & \text{new}\ \ k_2, h(f(a, K_{\mathsf{a}}^{-1}))\end{aligned}$$

On-Path Spoofed DoS Reply 7.7

Upon receiving the DoS-reply message from the attacker, the establishment initiator's multiset has the terms

$$\begin{aligned}&\langle v, a, b, s, d, k_0, k_1\rangle \,@\, a\\ &\uparrow_{\mathrm{sec}(k_1)}\ @\, a\\ &\Uparrow_{\mathrm{sec}(v)}\ \mathsf{P}(b, a, \mathsf{X}(\mathsf{DoSRep}(v, h(f(a, K_a^{-1}))))) \,@\, a,\end{aligned}$$

but **Rule ME.1.2** will not execute since the DoS-reply message is invalid. Hence the attacker fails to prevent progress or force tunnel state to be written.

Our exposed on-path attacker performs the same actions as the spoofed attacker, but is willing to generate the DoS reply. Note that the attacker does not use its own address in the message because it knows that the target is expecting a specific address, instead, the phrase 'exposed' is used here to mean that the attacker is willing to compute a DoS reply using its own resources. The rule for our attacker is given as follows:

$$\begin{aligned}&\mathsf{P}(a, b, \mathsf{X}(\mathsf{DoSReq}(u, f(a, K_a^{-1})))) \,@\, b \vdash\\ &\downarrow_{\mathrm{at}(k_1)}\ \mathsf{ATK}() \longrightarrow\\ &\downarrow_{\mathrm{sec}(u,k_2)}\ \mathsf{P}(b, a, \mathsf{X}(\mathsf{DoSRep}(u, h(f(a, K_a^{-1}))))) \,@\, \mathsf{e}\\ &\text{new}\ \ k_2.\end{aligned}$$

On-Path Exposed DoS Reply 7.8

Upon receiving the DoS-reply message from the attacker, the establishment initiator's multiset has terms

$$\begin{aligned}&\langle v, a, b, s, d, k_0, k_1\rangle \,@\, a\\ &\uparrow_{\mathrm{sec}(k_1)}\ @\, a\\ &\Uparrow_{\mathrm{sec}(v)}\ \mathsf{P}(b, a, \mathsf{X}(\mathsf{DoSRep}(v, h(f(a, K_a^{-1}))))) \,@\, a.\end{aligned}$$

In this case, the attacker has generated a valid response and hence the establishment initiator would execute **Rule ME.1.2**, sending the establishment-request message containing a signature. If a DoS-reply message originating at an honest node arrives at this node at some later point in time, then it will be ignored. The net effect is that while the attacker has forced itself past the DoS protections, it does not, in and of itself, halt protocol progress or force tunnel state to be written. Note that a different variation of the protocol may handled this differently that would have allowed the protocol to recover. On the other hand, had we could have assumed that the stale sate would expire and the protocol rerun, but we have not modeled this behavior here.

7.3.3 Attacks on Establishment Request

If an attacker can get the establishment responder to accept his request message, then that node will write the state as indicated in the malicious message and hence could break the tunnel complex. A message arriving from an honest source at a later point in time will be ignored. The attacks considered here are aimed at the establishment-request message and assume that the establishment-responder processes is in a mode waiting for an establishment-request message to arrive possessing a specific session identifier.

Our off-path spoofed attacker generates all of the fields of the establishment-request message as well as spoofing a source address and session identifier and sends the result toward a designated target. Our attacker is defined in the following rule:

$$\begin{array}{ll} \vdash_{\mathsf{e}} & \downarrow_{\mathrm{at}(k_1)} \mathsf{ATK}(b) \longrightarrow \\ & \downarrow_{\mathrm{sec}(u,k_2)} \mathsf{P}(c, b, \mathsf{X}(\mathsf{Req}(s, d, u, \iota_c, \Xi^c, g))) \\ & \text{new} \;\; u,\; c,\; s,\; d,\; \iota_c,\; \Xi^c,\;\; k_2,\;\; g. \end{array}$$

Off-Path Spoofed Establishment Request 7.9

Suppose the establishment responder has been invoked in session v and sent a DoS reply. Upon receiving the establishment-request message from the attacker, the multiset will contain terms

$$\begin{array}{l} \downarrow_{\mathrm{eresp}(v,k)} \\ \Uparrow_{\mathrm{sec}(u)} \mathsf{P}(c, b, \mathsf{X}(\mathsf{Req}(s, d, u, \iota_c, \Xi^c, g))). \end{array}$$

Since $u \neq v$, the attacker's message will not be consumed by an execution of **Rule ME.2.1.** Hence, this attacker fails to interrupt progress or to force the target to write state.

Our off-path exposed attacker behaves similarly to the spoofed attacker above, but will use its own address and credentials. Our attacker is defined in the following rule.

$$\begin{array}{lll} \Xi^{\mathsf{e}} & \vdash_{\mathsf{e}} & \downarrow_{\mathrm{at}(k_1)} \mathsf{ATK}(b, s, d) \longrightarrow \downarrow_{\mathrm{sec}(u,k_2)} \mathsf{P}(\mathsf{e}, b, \mathsf{X}(\mathsf{Req}(s, d, u, \iota_{\mathsf{e}}, \Xi^{\mathsf{e}}, g))) \\ & & \text{new} \;\; u,\; s,\; d,\; \iota_{\mathsf{e}},\; k_2,\; g. \end{array}$$

Off-Path Exposed Establishment Request 7.10

The establishment responder reacts the same as it did with the spoofed attacker because no off-path attacker has the capability of obtaining the session identifier.

A on-path attacker can observe traffic and can therefore possesses the session identifier. Specifically, a on-path attacker can observe a DoS-reply message being sent by a node and send an establishment-request message. If successful, the attacker will force its target to write the wrong state for the tunnel.

Our on-path spoofed attacker observes the DoS-reply message and sends its establishment message to that node containing spoofed filter entries as well as spoofed credentials. Rather than use a spoofed reply address, the attacker can use the address in the DoS-reply message, which is the expected address. On the other hand, it uses spoofed credentials. Since the attacker is not using its own address, it cannot produce a valid signature so this too must be spoofed.

$$\begin{array}{l}
\mathsf{P}(b, a, \mathsf{X}(\mathsf{DoSRep}(u, h(f(a, K_a^{-1}))))) \mathbin{@} b \vdash \\
\downarrow_{\mathrm{at}(k_1)} \mathsf{ATK}() \mathbin{@} \mathsf{e} \longrightarrow \\
\downarrow_{\mathrm{sec}(u,k_2)} \mathsf{P}(a, b, \mathsf{X}(\mathsf{Req}(s, d, u, \iota_c, \Xi^a, g))) \mathbin{@} \mathsf{e} \\
\text{new} \;\; k_2,\; s,\; d,\; \iota_a,\; \Xi^a,\; g.
\end{array}$$

On-Path Spoofed Establishment Request 7.11

When the establishment request arrives at the target, the message possesses the expected session identifier, but fails the signature check and hence **Rule ME.2.2** is not executed and the attack fails. Hence the attacker fails to interrupt progress or to force the target to write state.

Our on-path exposed attacker is similar to the previous case except that it is willing sign the message, and use its credentials.

$$\begin{array}{l}
\Xi^{\mathsf{e}}, \mathsf{P}(b, a, \mathsf{X}(\mathsf{DoSRep}(u, h(f(a, K_a^{-1}))))) \mathbin{@} b \vdash \\
\downarrow_{\mathrm{at}(k_1)} \mathsf{ATK}() \mathbin{@} \mathsf{e} \longrightarrow \\
\downarrow_{\mathrm{sec}(u,k_2)} \mathsf{P}(c, b, \mathsf{X}(\mathsf{Req}(s, d, u, \iota_{\mathsf{e}}, \Xi^{\mathsf{e}}, g))) \mathbin{@} \mathsf{e} \\
\text{new} \;\; k_2,\; s,\; d \\
\text{where } g = \mathrm{Sign}(K_{\mathsf{e}}^{-1}).
\end{array}$$

On-Path Exposed Establishment Request 7.12

The establishment responder does not retain the address of the node that sent the DoS request so the attacker can send an establishment request with its own address as the source. When the attacker's establishment-request message arrives at the target, it possesses the correct session identifier as well as a valid signature so **Rule ME.2.2** is executed, invoking the authorization layer with the credentials from the attacker. Since we have assumed that the attacker does not possess credentials to satisfy the policies, we can conclude that the authorization layer returns false; therefore, **Rule ME.2.3** fails to execute and tunnel state is not written. On the other hand, the protocol is now in a state where it cannot respond to the arrival of an establishment-request message from an honest principal.

7.3.4 Attacks on Establishment Reply

The establishment reply message can be subject to logical attacks similar to the establishment request message. Suppose the establishment initiator has sent a DoS-request message and received a valid DoS-reply message and then sent an establishment-request message. If an attacker sends this node an establishment-request message that is accepted, it can cause the node to write state for an invalid tunnel.

An off-path spoofed attacker cannot view traffic and can only guess that a request message has been sent and the attacker cannot obtain the session identifier or the SPI that was sent from the initiator to the responder or any of the other information in the establishment-request message. Our attacker generates its advertised address, the fields in the establishment reply messages, and a signature; the attacker then forms a message and sends it toward its target. The rule for this attacker is given as follows:

$$\begin{aligned}\vdash_{\mathsf{e}}\ & \downarrow_{\mathrm{at}(k)} \mathsf{ATK}(a) \,@\, \mathsf{e} \longrightarrow \\ & \downarrow_{\mathrm{sec}(u,k')} \mathsf{P}(c, a, \mathsf{X}(\mathsf{Rep}(s, d, u, \iota, \iota', \Xi^u, g'))) \\ & \text{new}\ \ c,\ s, d,\ u,\ k'\ \iota, \iota',\ \Xi^u,\ g'.\end{aligned}$$

Off-Path Spoofed Establishment Reply 7.13

Upon receiving the attacker's message, the multiset has

$$\begin{aligned}&\langle v, a, b, s', d', k_1, k_2, \iota_a\rangle @ a \\ &\Uparrow_{\mathrm{sec}} (\mathsf{P}(c, a, \mathsf{X}(\mathsf{Rep}(s, d, u, \iota, \iota', \Xi^u, g'')))) \,@\, a,\end{aligned}$$

but the initiator is waiting on a message of the form

$$\Uparrow_{\mathrm{sec}} (\mathsf{P}(b, a, \mathsf{X}(\mathsf{Rep}(s', d', v, \iota_a, \iota_x, \Xi^v, g')))) \,@\, a.$$

The off-path attacker has no way of obtaining the session identifier v or the nonce ι_a nor does this message originate from the expected node; so **Rule ME.1.3** does not execute and the protocol ignores this message. Hence the protocol does not interfere with progress or cause state to be written.

Our off-path exposed attacker performs the same actions as the spoofed attacker, but uses its own address and credentials and produces a valid signature. The rule for this attacker is given as follows:

$$\begin{aligned}\Xi^{\mathsf{e}} \quad \vdash_{\mathsf{e}} \quad & \downarrow_{\text{at}(k)} \mathsf{ATK}(a) \,@\, \mathsf{e} \longrightarrow \\ & \downarrow_{\text{sec}(u,k')} \mathsf{P}(\mathsf{e}, a, \mathsf{X}(\mathsf{Rep}(s, d, u, \iota, \iota', \Xi^{\mathsf{e}}, g'))) \\ & \text{new} \;\; c,\; a,\; b,\; u,\; k',\; \iota,\; \iota' \\ & \text{where } g = \text{Sign}(K_{\mathsf{e}}^{-1}) \\ & \text{where } \Xi^{u} = \Xi^{\mathsf{e}}.\end{aligned}$$

Off-Path Exposed Establishment Reply 7.14

This attacker exposes its address, but is unsuccessful for the same reasons as the attacker unwilling to expose its address.

If the establishment initiator is to accept an establishment-reply message, then the message must possess information that was sent in the establishment request as well as a valid signature and valid credentials. An on-path attacker can obtain the information from the request message. The spoofed attacker will not expose itself by using its own credentials and will spoof the signature as well. The rule for this attacker is given as follows:

$$\begin{aligned}& \mathsf{P}(a, b, \mathsf{X}(\mathsf{Req}(s, d, u, \iota_a, \Xi^{a}, g))) \,@\, b \vdash \\ & \downarrow_{\text{at}(k)} \mathsf{ATK}() \,@\, \mathsf{e} \longrightarrow \\ & \downarrow_{\text{sec}(v,k')} \mathsf{P}(b, a, \mathsf{X}(\mathsf{Rep}(s, d, u, \iota_a, \iota_{\mathsf{e}}, \Xi^{u}, g'))) \,@\, \mathsf{e} \\ & \text{new} \;\; k',\; \iota_{\mathsf{e}},\; \Xi^{u},\; g'.\end{aligned}$$

On-Path Spoofed Establishment Reply 7.15

Upon arrival at the responder, the following terms are in the multiset

$$\begin{aligned}& \langle u, a, b, s, d, k_1, k_3, \iota_a \rangle @a \\ & \Uparrow_{\text{sec}(u)} \mathsf{P}(b, a, \mathsf{X}(\mathsf{Rep}(s, d, u, \iota_a, \iota_{\mathsf{e}}, \Xi^{u}, g'))) \,@\, a\end{aligned}$$

Rule ME.1.3 fails to execute because the message does not contain a valid signature. Hence the attacker fails to interrupt the progress of the protocol or to force state to be written.

The on-path exposed case is similar to the previous case, but is willing to use its credentials and sign the message. The rule for this attacker is given as follows:

$$\begin{aligned}& \Xi^{\mathsf{e}},\; \mathsf{P}(\mathsf{e}, b, \mathsf{X}(\mathsf{Req}(s, d, u, \iota_a, \Xi^{b}, g))) \,@\, b \vdash \\ & \downarrow_{\text{at}(k)} \mathsf{ATK}() \,@\, \mathsf{e} \longrightarrow \\ & \downarrow_{\text{sec}(u,k')} \mathsf{P}(b, a, \mathsf{X}(\mathsf{Rep}(s, d, u, \iota_a, \iota_{\mathsf{e}}, \Xi^{u}, g'))) \,@\, \mathsf{e} \\ & \text{new} \;\; k',\; \iota_{\mathsf{e}},\; \Xi^{u} \\ & \text{where } g' = \text{Sign}(K_{\mathsf{e}}^{-1}) \\ & \text{where } \Xi^{u} = \Xi^{\mathsf{e}}.\end{aligned}$$

On-Path Exposed Establishment Reply 7.16

When the attacker's message arrives at its target, it contains the correct session identifier, but not a valid signature since the attacker e is presumed not to be able to produce a valid signature for the establishment responder *b*; consequently, no state is written.

7.3.5 Composing Attacks on Establishment

Having studied attacks on each of the establishment-layer messages, we now consider how the attackers defined above can be composed to achieve the maximum effect. Observing that on-path-exposed attacks seem to make the most progress, we restrict our analysis to these attacks. The attacker rules and detailed analysis presented above are not reproduced here, but we do give a brief description of the attack and an analysis of its effectiveness. Attacks on the establishment responder and the establishment initiator are considered separately.

Our on-path exposed attack on the establishment-responder process is formed from the composition of the attackers On-Path Exposed DoS Request 7.4 and On-Path Exposed Establishment Request 7.12. First, On-Path Exposed DoS Request observes a discovery message and sends a DoS-request message to the node advertised in the discovery message. This node will be executing the establishment-responder process. Since the attacker possesses the session identifier, the responder node executes **Rule ME.2.1** and sends a DoS reply message. On-Path Exposed Establishment Request observes the DoS-reply message sent by the establishment-responder process in response to the previous attack message, but does not verify the DoS reply. The attacker then forms an establishment-request message and sends it to the establishment responder. This message possesses a valid signature, but does not possess credentials that will satisfy the discovery policy, and consequently the attack fails to force the target to write tunnel state. On the other hand, once the attacker has forced the target into this state, progress is halted for that particular session. Suppose a DoS request from an honest node arrives, then the protocol is in a state where it does not respond.

Our on-path exposed attack on the establishment initiator is formed from the composition of the attackers On-Path Exposed DoS Reply 7.8 and On-Path Exposed Establishment Reply 7.16. First, On-Path Exposed DoS Reply observes a DoS-request message and generates a valid DoS-reply message. On-Path Exposed Establishment Reply observes the establishment-request message sent by the establishment initiator in response to the first message sent by the attacker and forms an establishment-reply message. The attacker willing to expose itself in the sense that it signs messages and uses its own credential set, but the signature is not valid so the target does not write state. The attack has forced the target into a state where it is expecting a valid establishment-reply message, and if an honest node running establishment responder sends this message, then progress continues.

7.4 Attacks on Discovery Protocols

The attacks studied in the previous section are directed at the establishment layer, where the attackers sent only establishment-layer messages. The focus of this section are attacks that target the discovery layer. These attacks are targeted at gateways on the path running the discovery-responder process and aim to either force the target to write state or to prevent progress of a currently executing protocol. Our discovery layer attackers all have the same basic skeleton. The attacker sends a discovery message to a gateway. The discovery protocol reacts by sending a DoS-request message. The attacker will then send establishment-layer messages in order to force the target to write state or that interrupts the progress of an already executing discovery session. The discovery attackers under investigation have the range of capabilities defined in Table 7.1.

Several of our attackers make use of a modified version of the establishment responder atk_eresp that performs all the same actions as the legitimate attacker, except that it does not verify the signature on the establishment-request message. The attacker does not verify the credentials contained in the establishment-request message and it commits no state. We will not write out these rules since they are basically the same as the establishment responder minus the aforementioned signature verification and writing of state. An attacker executing this will generate a valid DoS-reply message as well as send an establishment-reply containing valid address, credentials, as well as a valid signature.

Our off-path spoofed attacker will generate an address and session identifier and send the target a discovery message with the spoofed address. The rule for this attacker is given as follows:

$$\begin{aligned} \vdash_{\mathsf{e}} \quad & \downarrow_{\mathrm{at}(k_1)} \mathsf{ATK}(a) \longrightarrow \downarrow_{\mathrm{sec}(u,k_2)} \mathsf{P}(b, a, \mathsf{C}(Dis(b,u))) \\ & \text{new } k_2,\ u,\ b. \end{aligned}$$

Off-Path Spoofed Discovery 7.17

The target gateway will respond by sending a DoS-request message to the spoofed address b. Since b is generated by the new operator, it is ensured to be unique and thus the message is sent to a nonexistent node, but even if the node did exist and were running a discovery session, no reply will ever be sent due to the fact that the session identifier is spoofed and hence distinct from any other in the system. Therefore, no state is written on the target.

Our off-path-exposed attacker on discovery is willing to expose itself and calls the attacker establishment responder atk_eresp. The rule for this attacker is given as follows:

$$\vdash_{\mathsf{e}} \ \downarrow_{\mathrm{at}(k_1)} \mathsf{ATK}(a) \longrightarrow \downarrow_{\mathrm{sec}(u,k_2)} \mathsf{P}(b, a, \mathsf{C}(Dis(\mathsf{e}, u))),\ \downarrow_{\mathrm{atk\_eresp}(u,k_3)}$$
$$\text{new } k_2,\ k_3,\ u,\ b.$$

Off-Path Exposed Discovery 7.18

Upon receiving the discovery message, the target will send the attacker a DoS-request message and the attacker sends back a valid DoS-reply to the target, which replies with an establishment request message. The attacker sends an establishment-reply message that possesses the correct information taken from the establishment-request message and it also has a valid signature, but the attacker cannot produce a valid credential set; hence, the target will not write state.

An on-path-spoofed attacker targeting the discovery layer could detect a discovery packet released at say node b and send another discovery packet to, say, node c. (Possibly using the forwarding table to direct the packet to the desired node.) Node c then responds as if it were the next node on the path. In the ideal case, the attacker sends it to a node that may have the correct credentials.

$$\mathsf{P}(s, d, \mathsf{C}(\mathsf{D}(b, u))) \mathbin{@} a \ \vdash_{\mathsf{e}} \ \downarrow_{\mathrm{at}(k_1)} \mathsf{ATK}(c) \longrightarrow \downarrow_{\mathrm{sec}(u,k_2)} \mathsf{P}(s, d, \mathsf{C}(\mathsf{D}(b, u)))$$
$$\text{new } k_2.$$

On-Path Spoofed Discovery 7.19

The attack detects a discovery packet in session u advertising node b and sends a discovery packet to its target c with those same values. Suppose node d is the next node on the path that will receive and process the discovery message. Hence, there will be two DoS-request messages (from c and d) that arrive at b. There is the possibility that node b will execute establishment with c or d or both. If the exchange between b and d terminates successfully before the establishment request from c arrives and the establishment responder is not invoked again so that establishment with c never begins, then the attack does no harm. If the establishment between b and c successfully terminates and establishment responder is not invoked again so that establishment with d never begins, then the attack creates a different tunnel complex that includes gateways that would otherwise have not been included in the complex, but if there is a path to the destination and these gateways satisfy the protocol session's discovery policy and that the credentials that are delivered are satisfactory to traverse the gateways on this new path, then the attack does no harm. If c does not possess credentials to satisfy the discovery policy at b, then progress may be interrupted. If like the nested protocols, b invokes establishment responder and one session terminates, then it could be the case that tunnels are set up between

b and c as well as between b and d. In this case, the attack succeeds in breaking the tunnel complex.

A on-path exposed attacker can obtain the session identifier of a discovery protocol session in progress and send a discovery message to a target on the path to force it to run establishment with it rather than the node advertised in a discovery message sent by an honest node. If this attack were successful, state would be written at the target for the session in question breaking the tunnel complex.

$$\begin{array}{lll}\mathsf{P}(s, d, \mathsf{C}(\mathsf{D}(b, u))) \,@\, a & \vdash_{\mathsf{e}} & \downarrow_{\mathrm{at}(k_1)} \mathsf{ATK}() \longrightarrow \\ & & \downarrow_{\mathrm{sec}(u,k_2)} \mathsf{P}(s, d, \mathsf{C}(\mathsf{D}(\mathsf{e}, u))),\ \downarrow_{\mathrm{atk\_eresp}(u,k_3)} \\ & & \text{new } k_2,\ k_3.\end{array}$$

On-Path Exposed Discovery 7.20

The discovery packet arrives at the target and establishment is invoked with the attacker. The attacker generates and sends a DoS-reply message that is valid. In response to receiving the establishment reply, the attacker generates and sends an establishment-response message that contains valid source address, session identifier, and signature, but the attacker does not contain valid credentials and the protocol fails to write state.

7.4.1 Logical Attacks on a Tunnel Complex

Suppose that a discovery protocol has successfully set up a tunnel complex. A logical attack against the tunnel complex could break the tunnel complex by coercing one or more nodes in a complex to alter their association or mechanism database entries. Consider a case where a tunnel has been set up between nodes a and b. Suppose node a has entries $\mathsf{In}(b, \iota_a)$ and $\mathsf{Out}(b, \iota_b)$ in the association database and entries

$$\mathsf{Mech}(s \longrightarrow d : u : \mathsf{Bndl}[\mathsf{In}(b, \iota_a)])$$

and

$$\mathsf{Mech}(d \longrightarrow s : u : \mathsf{Bndl}[\mathsf{Out}(b, \iota_a)]$$

in the inbound and outbound mechanism databases. If a new entry $\mathsf{In}(b, \iota_x)$ gets added to the association database, no problem will arise because the $\mathsf{In}(b, \iota_a)$ entry is still in the database and that is the association pointed to in the inbound mechanism database. Suppose an adversary has obtained the session identifier u and the filter entry $s \longrightarrow d$ and the attack aims to set up a a pair of associations between e and b. If the attack is successful, the inbound mechanism database entry becomes

$$\mathsf{Mech}(s \longrightarrow d : u : \mathsf{Bndl}[\mathsf{In}(b, \iota_a)\mathsf{In}(e, \iota_{a'})])$$

due to the assumption that a new entry is nested inside of existing associations. As a result, a packet arriving at a in the proper tunnel is dropped.

Exposed On-Path Discovery 7.20 can be employed in such an attack, but we have seen that they are destined to fail if the proper credentials are lacking.

7.5 Resource-Exhaustion Attacks

The focus of the previous two sections was whether or not a specific collection of attackers succeeds from the perspective of a logical attack. In the remainder of this chapter, we analyze the same attackers to determine whether they succeed as resource-exhaustion attacks. This will allow us to better judge the effectiveness of the DoS protections that were incorporated into the establishment protocol in Section 7.2. The analysis is performed using a cost model in the spirit of Meadows [89]. We note that JFK has been subjected to a similar analysis [111] and no claim is made as to the novelty of our analysis technique. Instead, the novelty lies in the application of the method to discovery protocols. The analysis performed here informs the development of the theory presented in the next section.

The cost incurred when executing a protocol is obtained by summing the costs of individual operations being performed. Therefore, in order to perform our analysis, it is necessary to define the cost of each operation. Before assigning specific costs to specific events, we briefly discuss the guiding principles behind the assignment. The high-cost operations of interest occur at the establishment and authorization layers. We will attach no cost to sending and receiving a message thus no cost is assigned to operations performed at the secure processing and forwarding layers. The tunnel calculus new operator is also assumed to be cost free. A small cost is assigned to pattern matching performed on messages at the discovery and establishment layer. Generating a cryptographic signature or performing signature verification typically consumes some CPU resources. The tunnel-calculus establishment layer employs asymmetric cryptography, which is usually less efficient than symmetric cryptography. With some asymmetric cryptosystems, signature generation can take much longer than signature verification. For instance, benchmarks for 1024-bit Rivest-Shamir-Adleman (RSA) [107] have measured $10.0ms$ for signing and $0.5ms$ for signature verification on a $1GHz$ processor [96]. Different cryptographic systems make different trade-offs, for instance, [96] measures Digital Signature Algorithm (DSA) [97] signature verification at $6.2ms$, signature generation at $5.1ms$, and signature generation with precomputation of message independent values at 3.0μs. In our cost model, we shall assume performance similar to RSA, with the cost of signature generation an order of magnitude greater than signature verification. Only negligible cost is attached to generating a discovery DoS-request message or verifying the DoS-reply message. The cost of DoS reply generation can vary. Cookies, for example, exact little cost on the node generating the DoS-reply message while cryptographic puzzles impose a high cost on the node generating the DoS reply. We shall consider both cases and designate the low-cost operation as Option-L and the high-cost option as Option-H. Credential verification is the most costly operation in our model due to the complexities involved. With this strategy in place, we assign specific cost values to specific operations.

Let the cost set be the set of naturals $\mathbf{N}$. We Define a cost function as follows:

$$\mathcal{G} : \text{Events} \longrightarrow \mathbf{N},$$

where the events in question are operations in the tunnel calculus.

$$\mathcal{G}(e) = \begin{cases} 10 & \text{if } e = \text{message pattern match values (session identifier, SPI, etc.)} \\ 20 & \text{if } e = \text{DoS Req Generation} \\ 20 & \text{if } e = \text{DoS Rep Generation (Option-L)} \\ 1000 & \text{if } e = \text{DoS Rep Generation (Option-H)} \\ 20 & \text{if } e = \text{DoS Rep Verification} \\ 2000 & \text{if } e = \text{Signature Generation} \\ 200 & \text{if } e = \text{Signature Verification} \\ 6000 & \text{if } e = \text{Credential Set Verification} \\ 0 & \text{otherwise} \end{cases}$$

The specific values were not chosen based on any quantitative analysis, but the process was guided by the strategy outlined above. Cheap operations have a cost less than 100, high-cost operations have a cost greater than 1000, while mid-rage operations have a cost that falls between 100 and 1000. Pattern matching, DoS request generation, DoS reply verification are all cheap, while the cost of DoS reply generation varies between low and high depending on which option the protocol uses. Signature generation cost is high, where verification is of medium cost. Credential set verification is assigned a cost that is three times that of signature generation because of the complexity of the operations performed by the authorization layer such as forming the credential chain and verifying many credentials.

An intruder cost function similarly maps the cost of the actions of an intruder to the their cost. Since our intruders perform the same action, we assume that the cost function is the same for the intruder.

7.5.1 Message Cost Functions

When a message arrives at a node, one or more *message verification operations,* often referred to as *message verification events,* are performed before a message is accepted. In most cases, verification events are those things that must be true before a rule fires. This may only involve performing pattern matching to verify that a message contains an expected session identifier or some other data. On the other hand, some rules also verify a DoS-reply message or verify a signature. Although it appears on the right-hand side of rules, the establishment layer invokes the authorization layer to verify that credentials satisfy policy and failure halts progress of the protocol.

Each of these activities is classified as a verification event. As a design philosophy, we prefer to perform the message acceptance checks that incur the smallest cost before those that involve a higher cost. For instance, when an establishment-request message arrives, it must pattern match, and if that is successful, the signature is verified, and if that is successful, the costly credential verification is performed.

For each establishment message and discovery message that is received, we define a *message processing cost function* $\mathcal{G}'$ that defines the cost of reaching and performing a particular verification event. Stating this formally, if verification events $V_1, \ldots, V_n$ are performed after the arrival of a message, then, for each verification event V_j, $\mathcal{G}'(V_j) = \mathcal{G}(V_1) + \cdots + \mathcal{G}(V_j)$.

We now compute the message processing cost for each verification event performed when processing a discovery message, DoS-request message, DoS-reply message, establishment-request message, and the establishment-reply message.

- Discovery. Pattern matching is employed to verify that the message is of the correct form.
  - $\mathcal{G}'$(ptrn-mtch) = 10.
- DoS request.(**ME.2.1**) Pattern matching is employed to verify that the message format and that it possesses the expected session identifier.
  - $\mathcal{G}'$(ptrn-mtch) = 10.
- DoS reply. (**ME.1.2**) Pattern matching is employed to verify the message format and session identifier. If successful, the DoS-reply message is verified.
  - $\mathcal{G}'$(ptrn-mtch) = 10.
  - $\mathcal{G}'$(DoS-check) = 30.
- Establishment request. (**ME.2.2**) Pattern matching verifies the message format and that the message has the correct session identifier. Signature verification is then performed. If the signature is correct, then the authorization layer is invoked to verify credentials.
  - $\mathcal{G}'$(ptrn-mtch) = 10.
  - $\mathcal{G}'$(sig-chk) = 210.
  - $\mathcal{G}'$(cred-chk) = 6210.
- Establishment reply. (**ME.1.3**) Pattern matching verifies the message format and that the message has the correct session identifier and contains the SPI value sent in the request. If this is successful, then signature verification is then performed. If the signature is correct, then the authorization layer is invoked to verify credentials.

- $\mathcal{G}'(\text{ptrn-mtch}) = 10$.
- $\mathcal{G}'(\text{sig-chk}) = 210$.
- $\mathcal{G}'(\text{cred-chk}) = 6210$.

The message acceptance cost measures the cost of a verifying that a message is valid. The protocol engagement cost function

$$\mathcal{E} : \text{Event} \longrightarrow \mathbf{N}$$

is defined as the sum of the cost of events performed at a node after the receipt of a message ($\Uparrow_{\text{sec}}$) until the next message is sent ($\downarrow_{\text{sec}(k)}$). This represents the cost of successfully accepting a message and the cost of any events that result from accepting the message. The protocol engagement cost for each of the messages considered above is given as follows.

- Discovery Message. Performs pattern matching and invokes establishment, which generates a DoS-request message.

$$\mathcal{E}(\text{Disc}) = 30.$$

- DoS Request. Pattern matching is performed to verify that the message possesses the correct session identifier and a DoS-reply message is generated and sent. Recall that there are two options for the cost of DoS reply generation and we consider both here.

$$\begin{aligned} \mathcal{E}(\text{DoS Request Option-L}) &= 30 \\ \mathcal{E}(\text{DoS Request Option-H}) &= 1010 \end{aligned}$$

- DoS Reply. Pattern match, verify the DoS-reply message, and generate an establishment request message, which requires a signature be generated.

$$\mathcal{E}(\text{Dos Reply}) = 2030.$$

- Establishment Request. Before the message is accepted, it must pass a pattern match, signature verification, and credential verification. Upon acceptance of this message, the establishment reply is generated and sent, which requires a digital signature be generated.

$$\mathcal{E}(\text{Establishment Request}) = 8210.$$

- Establishment Reply. The verification events are pattern matching, signature verification, and credential verification. Depending on the higher layer protocol, other actions may follow, but are not considered here, so the engagement cost is the same as the message acceptance cost.

$$\mathcal{E}(\text{Establishment Reply}) = 6210.$$

7.5.2 Evaluating the Cost of Attacks

Our objective is to evaluate DoS protections by comparing the cost incurred by an attacker against the cost incurred by their targets. To facilitate our analysis, we introduce a tolerance relation that will serve as a metric to judge the effectiveness of an attack. We then use this relation to analyze the DoS resistance of each establishment layer message. Finally, we examine each of our attackers, and compute the cost incurred by both the target and attacker and apply the tolerance relation to judge the success or failure of the attack.

The intruder cost function $\mathcal{A}$: Events $\longrightarrow \mathbf{N}$ maps an event to the cost borne by an intruder to force this event to succeed. We say that a protocol is fail-stop with respect to $\mathcal{A}(E)$ if an attacker sends a message that arrives before E, then the attacker should have to bear the cost of at least $\mathcal{A}(E)$ to force E to occur.

DoS resistance is defined in terms of a tolerance relation $\mathcal{T} \subset \mathbf{N}^* \times \mathbf{N}^*$ that is a subset of the protocol and attacker costs. The relation is defined as follows.

$$\mathcal{T}(x,y) = \begin{cases} 0 \leq x < 100 \text{ and } 0 \leq y \\ 100 \leq x < 1000 \text{ and } 100 \leq y \\ 1000 \leq x \text{ and } 1000 \leq y, \end{cases}$$

where x and y range over the naturals. This relation says that an attacker forcing its target to perform an operation having the message processing cost in the range of x is acceptable if the attacker incurs a cost in the range of y. The values chosen reflect the philosophy that a DoS attacker is successful if it can cause higher-cost operations to be performed at significantly less cost to itself. In order to determine whether a specific verification event is DoS resistant, we must perform the following analysis. Consider the verification events that follow the delivery of a secure layer message to the establishment or discovery layers. Suppose action E_1 precedes verification action E_2 in the execution of a rule, then we must verify that

$$(\mathcal{G}'(E_2), \mathcal{A}(E_1)) \in \mathcal{T}.$$

This means that the cost borne by the attacker in forcing the event E_1 to succeed in relation to the cost of executing E_2 at the target is within tolerance. If E is the final verification action for a message, then we must verify that

$$(\mathcal{E}(E), \mathcal{A}(E)) \in \mathcal{T}.$$

This means that the cost borne by the attacker to force the target to successfully perform E in relation to the cost incurred at the target in performing E, as well as successive operations until the next message is sent, is within tolerance. The idea behind the metric and our relation $\mathcal{T}$ is that if the cost to an attacker to force an operation to occur is at least the same magnitude as the cost borne by the target, then it is within tolerance. We now apply this metric to analyze the DoS resistance of the establishment layer.

The establishment initiator is the first to be analyzed. There are no verification events in **Rule ME.1.1**, but since this rule is usually invoked in response to the arrival of a discovery message, we consider the verification operations performed on this message. An attacker can generate and send a spoofed discovery message at no cost to itself, hence

$$(\mathcal{G}'(\text{ ptrn-match}) = 10, \mathcal{A}(\text{Disc msg del}) = 0) \in \mathcal{T}.$$

(Although a slight abuse of notation, we include the operation whose cost we are evaluating by say op = cost.) This is also a message-accept event, so we need to consider the protocol engagement cost that include the cost to generate the DoS-request message. In this case

$$(\mathcal{E}(\text{ptrn-match}) = 30, \mathcal{A}(\text{ptrn-match}) = 0) \in \mathcal{T}.$$

Rule ME.1.2 first performs pattern matching on the DoS-reply message and the preceding action is message delivery, which an attacker can induce at no cost to itself. Hence,

$$(\mathcal{G}'(\text{ptrn-match}) = 10, \mathcal{A}(\text{DoS rep msg del}) = 0) \in \mathcal{T}.$$

The next event verifies that the DoS-reply message is valid and if so, generates a signed message. An attacker able to obtain the correct session identifier can succeed at passing this check at no cost to itself. Therefore

$$(\mathcal{G}'(\text{verify DoS Rep}) = 30, \mathcal{A}(\text{ptrn-match}) = 0) \in \mathcal{T}.$$

This is a message-accept event so we must also consider the protocol engagement cost that includes the cost of generating the signature on the establishment request. Attackers are considered for both DoS-reply message generation costs. In the case of Option-L DoS reply generation, the attacker willing to generate DoS-reply message can force the target to perform a high-cost action at little cost to itself. Yielding

$$(\mathcal{E}(\text{verify DoS Rep}) = 2030, \mathcal{A}(\text{verify DoS Rep}) = 20) \notin \mathcal{T}$$

and hence is outside of the acceptable tolerance. In the case of Option-H DoS reply generation, the attacker cannot force the DoS-reply message generation without performing a high-cost operation itself. Hence,

$$(\mathcal{E}(\text{verify DoS Rep}) = 2030, \mathcal{A}(\text{verify DoS Rep}) = 1000) \in \mathcal{T}.$$

Since the attacker must perform a high-cost operation in order to force one, the relation is within tolerance. **Rule ME.1.3** first performs pattern matching, followed by signature verification, and finally credential verification. Since an attacker can force message delivery at no cost to itself,

$$(\mathcal{G}'(\text{ptrn-match}) = 10, \mathcal{A}(\text{est repl msg del}) = 0) \in \mathcal{T}.$$

A on-path attacker can obtain the information required to successfully pattern-match at no additional cost to itself. Hence,

$$(\mathcal{G}'(\text{sig-verify}) = 210, \mathcal{A}(\text{ptrn-match}) = 0) \notin \mathcal{T}.$$

A on-path attacker willing to generate a signature can force the target to verify the credentials, but only at a high cost to itself.

$$(\mathcal{G}'(\text{Cred. Verify}) = 6210, \mathcal{A}(\text{sig gen}) = 2000) \in \mathcal{T},$$

which is within tolerance. Since we have assumed that no attacker can possess the credentials necessary to satisfy policy, we assign $\mathcal{A}(\text{Cred Verify}) = \infty$. Given that there are no messages sent after the establishment reply is accepted,

$$(\mathcal{E}(\text{Cred Verify}) = 6210, \mathcal{A}(\text{Cred Verify}) = \infty) \in \mathcal{T}.$$

We now analyze the establishment-responder messages. **Rule ME.2.1** verifies that the session number is correct using pattern matching. The preceding action is the receipt of a message, which a on-path attacker can generate at no cost to itself.

$$(\mathcal{G}'(\text{ptrn-mtch}) = 10, \mathcal{A}(\text{DoS request msg del}) = 0) \in \mathcal{T}.$$

This is a message-accept event so we need to consider the protocol engagement values as well. Given that the next event is the generation of a DoS-reply message, we consider both the case of low-cost DoS reply generation and high-cost DoS reply generation. A on-path attacker can obtain the information to force the target to pattern match at no cost to itself. In the case of Option-L DoS reply generation, the relation is

$$(\mathcal{E}(\text{ptrn-mtch}) = 30, \mathcal{A}(\text{ptrn-match}) = 0) \in \mathcal{T},$$

which is within tolerance. In the case of Option-H DoS reply generation, the relation is

$$(\mathcal{E}(\text{ptrn-mtch}) = 1010, \mathcal{A}(\text{ptrn-match}) = 0) \notin \mathcal{T},$$

which is not within tolerance. We see that, although the high-cost operation is intended to protect the establishment initiator against attacker bots willing to cheaply generate DoS-reply messages, it opens up an avenue of attack against the establishment responder. In a traditional DoS analysis we are concerned the cost to one node, namely the cost borne by a server rather than the client initiating the protocol, but in our case, both nodes may be gateways, so we must be concerned with both the discovery initiator and discovery responder. **Rule ME.2.2** has three verification events. The first is a pattern match, the second is a signature verification, and the third is a credential verification. An attacker can force message delivery at no cost to itself. Hence,

$$(\mathcal{G}'(\text{ptrn-mtch}) = 10, \mathcal{A}(\text{estab req msg del}) = 0) \in \mathcal{T}.$$

A on-path attacker can, at no cost to itself, obtain the information to force the pattern match to succeed and consequently force the signature verification to occur. Hence,

$$(\mathcal{G}'(\text{Sig verify}) = 210, \mathcal{A}(\text{ptrn-match}) = 0) \notin \mathcal{T}.$$

A on-path attacker willing to generate a signature can force credential verification. Hence,

$$(\mathcal{G}'(\text{Cred. Verify}) = 6210, \mathcal{A}(\text{Sig Verify}) = 2000) \in \mathcal{T},$$

which is within tolerance because the attacker too must perform a high-cost operation. Since we have assumed that no attacker can possess the credentials necessary to satisfy policy, we assign $\mathcal{A}(\text{Cred Verify}) = \infty$. Given that no messages are sent after the establishment request is accepted,

$$(\mathcal{E}(\text{Cred Verify}) = 6210, \mathcal{A}(\text{Cred Verify}) = \infty) \in \mathcal{T}.$$

The tolerance relation gave us an indication of what attackers will succeed; our attention now turns to analyzing the specific attackers defined above. For each of our attackers, we shall compute the cost incurred by both an attacker and its target. The cost to the attacker is obtained by summing the cost of the operations performed by an attacker. We compute the cost to the target by summing the cost of the operations performed in response to the attacker's messages. The exact operations performed by the target are detailed in the analysis performed above. The results of our computations are given in tabular form, which makes it easy to see what attackers fall within our desired tolerance.

Table 7.2 gives the costs incurred by both attacker and target for each of the establishment layer attacks defined above. We provide a brief description of the actions performed by the attacker and target, from which it should be possible to reproduce the calculations that produced the values shown in each row of the table. Off-Path Spoofed DoS Request simply sends a spoofed message and the target only performs a pattern match. Off-Path Exposed DoS Request forms a DoS-request message, but the target only performs a pattern match. Both on-path DoS request attackers possess the session identifier and can therefore force the target to generate a DoS reply. We compute the cost for both the case where the cost to generate the DoS reply is negligible and the case where cost to generate this message is high. The two off-path DoS reply attackers lack the session identifier, therefore, the target only performs a pattern match. On-Path Spoofed DoS Reply sends a spoofed DoS reply and consequently forces the target to both pattern match and verify the DoS reply, but since DoS-reply verification fails, there is no signature generated. On-Path Exposed DoS Reply generates a valid DoS reply and forces the target to generate a signed establishment reply. We consider both the cases where the cost to generate the DoS reply is low cost and the case where the cost is high. Both of the off-path establishment request attackers simply generate messages at no cost to themselves and the target only performs pattern matching. On-Path Spoofed Establishment

Attack	Label	Cost to Attacker	Cost to Target
Off-Path Spoof DoS Req	7.1	0	10
Off-Path Exposed DoS Req	7.2	20	10
On-Path Spoof DoS Req Option-L	7.3	0	30
On-Path Spoof DoS Req Option-H	7.3	0	1010
On-Path Exposed DoS Req Option-L	7.4	20	30
On-Path Exposed DoS Req Option-H	7.4	20	1010
Off-Path Spoof DoS Rep	7.5	0	10
Off-Path Exposed DoS Rep	7.6	0	10
On-Path Spoof DoS Rep	7.7	0	30
On-Path Exposed DoS Rep Option-L	7.8	20	2030
On-Path Exposed DoS Rep Option-H	7.8	1000	2030
Off-Path Spoof Est Req	7.9	0	10
Off-Path Exposed Est Req	7.10	0	10
On-Path Spoof Est Req	7.11	0	210
On-Path Exposed Est Req	7.12	2000	6210
Off-Path Spoof Est Rep	7.13	0	10
Off-Path Exposed Est Rep	7.14	2000	10
On-Path Spoof Est Rep	7.15	0	210
On-Path Exposed Est Rep	7.16	2000	210

Table 7.2: Costs Incurred During Establishment Attacks

Request can obtain the session identifier and other information from the DoS-reply message and force the target to perform a signature verification. On-Path Exposed Establishment Request is willing to produce a signed message; hence, forcing the target to pattern match, perform a signature verification, and perform a credential check. Off-Path Spoofed Establishment Reply only forces the target to pattern match. Off-Path Exposed Establishment Attacker generates a signature, but the target only pattern matches because the attacker lacks the requisite session identifier. On-Path Spoofed Establishment Reply possesses the expected session identifier, but does not generate a valid signature; hence, the target does not verify credentials. Exposed Establishment Reply gets the session identifier and other information needed to pattern match by observing the establishment-request message, but the attacker cannot use its own address if it wants to make the most progress possible, but instead uses the address that is expected, but this means it cannot produce the expected signature and consequently the target pattern matches and verifies the signature, but does not verify credentials.

Examining table 7.2, we see that the following attackers succeed in the sense that the costs incurred by the attacker in comparison to those incurred by the target fall outside of our threshold.

- On-Path Spoofed DoS Request Option-H.
- On-Path Exposed DoS Request Option-H.
- On-Path Exposed DoS Reply Option-L.
- On-Path Spoofed Establishment Request.
- On-Path Spoofed Establishment Reply.

The fact that the most serious threats come from on-path attackers corresponds with the intuition of the capability of an intruder that can observe traffic.

Table 7.3 gives the costs incurred by both the attacker and target in our attacks on discovery. The computation of the costs are a somewhat more subtle than the in the previous table so we are more explicit here as to how we obtained the values given in each row of the table. Off-Path Spoofed Attacker simply sends a discovery message to the target at no cost to itself and the target pattern matches (10) and then generates and sends a DoS request (20). Off-Path Exposed Discovery participates in the protocol by sending a discovery message, generating a valid DoS reply message, as well as a signed establishment reply. Note that the successful attacker must pattern match the DoS request (10) and establishment request messages (10), and generate both the DoS-reply message (20 or 1000) and the establishment reply (2000). The target must pattern match the discovery message (10), DoS-reply message (10), and the establishment reply (10), generate the DoS request (20), generate an establishment request (2000), and verify the DoS reply (20), verify the signature

Attack	Label	Cost to Attacker	Cost to Target
Off-Path Spoof Disc	7.17	0	30
Off-Path Exposed Disc Option-L	7.18	2040	8270
Off-Path Exposed Disc Option-H	7.18	3020	8270
On-Path Spoof Disc	7.19	0	2060
On-Path Exposed Disc Option-L	7.20	2040	8270
On-Path Exposed Disc Option-H	7.20	3020	8270

Table 7.3: Costs Incurred During Discovery Attacks

on the establishment reply(200), and verify the credentials in the establishment reply (6000). On-Path Spoofed Attacker detects a node that is running a discovery session and sends the discovery message to its target using the session identifier and address advertised in the discovery message; this is done at no cost to the attacker. Since the node advertised in the attacker is running the establishment-responder process, when the target sends a DoS-request message it will be processed and a valid DoS reply sent back. The target then generates an establishment-request message with a signature. In this case, we assume that the target does not possess valid credentials to satisfy the discovery policy at the establishment responder so the target does never receives an establishment-reply message. Consequently, the cost incurred by the responder is the cost to perform two pattern matches (20), DoS request generation (20), DoS reply verification (20), and signature generation (2000) totals 2060. On-Path Exposed Discovery is carries out a complete exchange with the target. Hence the attacker must pattern match the DoS-request message (10) and establishment-request message (10), generate a DoS-reply message (20 or 1000) and generate a signature (2000). The target must pattern match discovery, DoS reply, and establishment reply messages (30), generate a DoS-request message (20), verify a DoS reply (20), sign an establishment-request message (2000), verify a signature (20), and verify credentials (6000). From the table, we see that only On-Path Spoofed Discovery falls outside of our original tolerance relation in that the attacker can force the target to perform a high-cost operation at little cost to itself. On the other hand, we see that only Off-Path Spoofed Discovery does not force the target to perform a high-cost operation. Yet this is really just a consequence of the fact that an attacker willing and able to send a valid DoS-reply message will negate the DoS protection. So we feel these results hold for most protocols using cookies as for DoS protections. Note that it may be of some interest to investigate other tolerance relations. For instance a tolerance relation in which an attacker is not allowed to force its target to perform a high-cost operation regardless of the cost incurred by the attacker.

7.6 Resource-Exhaustion Theorems

Earlier in this chapter, we introduced a modified version of the establishment protocol that incorporates DoS protection, the tunnel-calculus was used to model a collection of attackers, and we analyzed the effectiveness of the attacks from both the perspective logical attacks and resource-exhaustion attacks. Informed by the knowledge gained from these efforts, we formulate a theory that says a selected subset of attackers cannot exploit honest nodes executing discovery protocols by forcing them to execute high-cost operations.

The cost analysis performed above showed that several attacks can force an honest node executing our discovery protocols to perform high-cost operations. In particular, on-path attackers are able to obtain information by observing messages sent by executing protocols and can use this to thwart built in protections. In this section, we only consider the subset of attackers that do not succeed in forcing high-cost operations to be performed. Our choice is guided by the results presented in tables 7.2 and 7.3. In our current analysis, we do not consider attacks whose success depend on whether on not the cost of generating a DoS request operation is low or high. The specific attacks we shall consider are given as follows:

- Off-Path Spoofed Discovery 7.17.
- Off-Path Spoofed DoS Request 7.1.
- Off-Path Exposed DoS Request 7.2.
- Off-Path Spoofed DoS Reply 7.5.
- Off-Path Exposed DoS Reply 7.6.
- On-Path Spoofed DoS Reply 7.7.
- Off-Path Spoofed Establishment Request 7.9.
- Off-Path exposed Establishment Request 7.10.
- On-Path Spoofed Establishment Request 7.11.
- Off-Path Spoofed Establishment Reply 7.13.
- Off-Path Exposed Establishment Reply 7.14.
- On-Path Spoofed Establishment Reply 7.15.

- On-Path Exposed Establishment Reply 7.16.

Any mention of attacks or attackers in the remainder of this section refers to the attackers listed above.

High-cost operations are defined as those having cost greater or equal to 1000. The rules defining our three discovery protocols do not perform high-cost operations per se, but each discovery protocol invokes the establishment layer that does perform several high-cost operations. The specific establishment-layer rules that perform high-cost operations are:

ME.1.2 Generates a signature if the DoS-reply message is valid.

ME.1.3 Verifies credentials if the signature on the establishment reply is valid.

ME.2.1 Generates a DoS-reply message if Option-H is in effect.

ME.2.2 Verifies credentials if the signature is valid on the establishment request message.

ME.2.3 Generates a signature on the establishment reply if the credentials checked in **Rule ME.2.2** are valid.

Our objective is to show that our discovery protocols possess sufficient protections against the above attackers. So if a high-cost operation is performed, it was not due to the actions of one of the attackers.

The remainder of this section makes use of the following notational conventions. Let **Z** denote the set of labels on rules whose execution involve a high-cost action. Let $\mathcal{D}(u)$ denote the infinite set of all terms of the form $\downarrow_{\mathsf{dis}(u,k)} \mathsf{D}(s,d)$, which are employed to invoke discovery with session identifier u. The set $\mathcal{D}_C(u)$ is the set of terms denoting the invocation of the concatenated protocol. The set $\mathcal{D}_N(u)$ is the set of terms representing the invocation of the nested protocol. Let $\mathcal{R}(u)$ denote the infinite set of all terms of the form $\downarrow_{\mathsf{eresp}(u,k)}$. Note that the $\downarrow_{\mathsf{atk\_eresp}(u,k)}$ term used to invoke the attacker's establishment responder is distinct from terms in $\mathcal{R}(u)$.

The off-path attackers all generate a session identifier using the tunnel calculus new operator. The next three results combine to show that our off-path attackers cannot exploit an executing establishment responder. The first result says that an off-path attacker will generate session identifiers distinct from those of any executing discovery protocol. We then show that the attacker's session identifier must also be distinct from the session identifier of the establishment-responder processes executing in the network.

Proposition 7.1 *Suppose the trace $T = M_1, \ldots, M_n$ records the execution of possibly many discovery protocol sessions and trace T also records the execution of an attacker that generates a session identifier u using the new u operator, where $u \notin \mathfrak{L}(M_1)$. Then $\mathcal{D}(u) \cap T = \emptyset$. (That is, the trace does not record a discovery session invoked with the session identifier u.)*

Proof: Suppose the proposition is false. Then the trace records a $\downarrow_{\text{dis}(u,k)}$ term, where u was produced by an attacker using the new operator. Observe that none of the attackers under consideration rewrites a term of the form in $\mathcal{D}(u)$ so the term must have been produced by an honest node that invoked discovery with session identifier u. Recall that we assume that discovery protocols are always invoked with a rule of the form

$$\ldots \longrightarrow \downarrow_{\text{dis}(v,k)} \ \ldots \ \text{new } k, v.$$

This rule is not executed in response to any message sent by our attackers so session identifier u must have been generated by the new operator when discovery was invoked. Therefore, two distinct invocations of the new operator must have returned the same value, which contradicts the uniqueness of values generated by the new operator. Hence, the proposition must hold. □

Discovery protocols invoke the establishment-responder process by rewriting a $\downarrow_{\text{eresp}(u,k)}$ term. The discovery message advertises the node where the responder process is waiting to set up the next tunnel in the complex. This process awaits the arrival of a DoS-request message possessing session identifier u. An attacker possessing this session identifier will succeed in forcing its target to generate a DoS reply, which in our model may or may not be a high-cost operation. The next two results say that if the attackers under consideration spoof session identifier u, then the trace will not record any establishment-responder processes executing with that same session identifier. We consider the case of nested and concatenated protocols separately.

Lemma 7.2 *Assume that the only discovery protocol that nodes may execute are the nested discovery protocol and the modified nested discovery protocol. Suppose that the trace $T = M_1, \ldots, M_n$ records the execution of possibly many discovery protocol sessions as well as the execution of any of the attackers that generate their own session identifier using the tunnel calculus new operator. If the trace records an attacker that generates session identifier u, where $u \notin \mathfrak{L}(M_1)$, then $T \cap \mathcal{R}(u) = \emptyset$.*

Proof: Inspecting the rules of the discovery protocols as well as the attackers, we see that there are no $\downarrow_{\text{eresp}}$ terms in the attacker rules, but it remains to show that no action taken by the attackers could have resulted in establishment being invoked with the same session identifier as the attacker itself.

It follows from proposition 7.1 that $\mathcal{D}_N(u) \cap T = \emptyset$. Hence, the trace does not record any discovery protocol having the same session identifier as the attacker. Observing the rules defining both the nested and the modified nested discovery protocol, we see that the establishment-responder process is only invoked at the initiating host s and only after the discovery message is sent. The establishment responder is first invoked in **Rule ND.1.1/MND.1.1** using the same session identifier as the discovery session, which we know to be distinct from the attacker. The establishment responder is later repeatedly invoked in **Rule ND.1.3/MND.1.3** using the session identifier in the rule's resumption terms, which is the discovery session identifier, which we know is distinct from attacker's session identifier. Whence we can conclude that $T \cap \mathcal{R}(u) = \emptyset$. $\square$

The case of the concatenated discovery protocol is somewhat more complex because each gateway on the path invokes the establishment-responder process.

Lemma 7.3 *Assume that the only discovery protocol that nodes may execute is the concatenated discovery protocol. Suppose* $T = M_1, \ldots, M_n$ *records the execution of possibly many discovery protocol sessions as well as any of the attackers that generate their own session identifier using the tunnel calculus new operator. If the trace records an attacker that generates session identifier* u*, where* $u \notin \mathfrak{L}(M_1)$*, then* $T \cap \mathcal{R}(u) = \emptyset$*.*

Proof: Inspecting the rules that define the concatenated discovery protocol as well as the attackers, we see that there are no $\downarrow_{\mathrm{eresp}}$ terms in the attacker rules, but it remains to show that no action taken by the attackers could have resulted in establishment being invoked with the same session identifier as the attacker itself. It follows from proposition 7.1 that $\mathcal{D}_C(u) \cap T = \emptyset$. Hence, the trace does not record any discovery protocols invoked with the same session identifier as the attacker. We consider the concatenated discovery initiator and responder processes and show that they will not invoke the establishment responder using the attackers session identifier.

Assume that the trace records the execution of a concatenated discovery protocol that was invoked at a node by rewriting a $\downarrow_{\mathrm{dis}(v,k)}$ term. The establishment responder is invoked twice at this node during execution of the concatenated-discovery-initiator process. **Rule CD.1.1** invokes the establishment message using the session identifier in the discovery $\downarrow_{\mathrm{dis}(v,k)}$ term appearing on the left-hand side of the rule. It follows from proposition 7.1 that v is distinct from the session identifier of the attacker u. The establishment responder is again invoked in **Rule CD.1.2**, but this time using the session identifier located in the resumption term appearing on the left-hand side of the rule, but this is the session identifier for the discovery protocol not the attacker. Whence the result holds for the cases where establishment responder is invoked during execution of the concatenated discovery initiator.

The concatenated discovery protocol responder process runs as a daemon at each node and reacts to the arrival of a discovery message. During execution, the discovery-responder process sets up a tunnel with the node advertised in the discovery message and then releases the discovery packet and invokes the establishment responder, in **Rule CD.2.6**, in order to set up a tunnel with the next node on the path. If the concatenated discovery-responder process is executing session v, then the establishment-responder process will be invoked with the session identifier v contained in the resumption term on the left-hand side of the rule. On the other hand, we must consider the case where the discovery responder is executing in response to an attacker having sent a discovery message, advertising session u, and hence the discovery-responder process itself is executing in session u. Of the attackers under consideration, only Off-Path Spoofed Discovery exhibits this behavior. Recall that this attacker spoofs both the session identifier and the address advertised in the discovery message. The proof for this case is carried out by induction on the length of the trace. The base case of length zero is clear. Suppose the trace records $M_i \xrightarrow{\mathbf{CD.2.6}(u)} M_{i+1}$ invoking establishment responder for session u. Inspecting the concatenated discovery responder rules, we can infer that **Rule CD.2.1** must have executed and the establishment initiator must have successfully executed prior to executing **Rule CD.2.6**. Therefore, the trace must have recorded

$$M_1 \longrightarrow \cdots \longrightarrow M_j \xrightarrow{\mathbf{CD.2.1}(u)} M_{j+1} \longrightarrow \cdots \longrightarrow M_k \xrightarrow{\mathbf{ME.1.1}(u)} M_{k+1} \longrightarrow \cdots \longrightarrow M_l \xrightarrow{\mathbf{ME.1.4}(u)} M_{l+1} \longrightarrow \cdots \longrightarrow M_i \xrightarrow{\mathbf{CD.2.6}(u)} M_{i+1}.$$

Although we have assumed that the new operator always generates a unique value, in this case, it is useful to consider situations where the address generated by the new operator may be the same as an existing address. Therefore, we consider the following three cases:

- If the spoofed address in discovery message consumed in **Rule CD.2.1** does not correspond to an actual node in the network, no DoS-reply message will ever be received and consequently **Rule ME.1.4**(u) cannot have ever been executed. Therefore, **Rule CD.2.6** cannot have been executed.

- If the advertised address is an actual node in the network and not running the establishment responder, then it will not respond the DoS request so the DoS-reply message is never received and and consequently **Rule ME.1.4**(u) cannot have ever been executed. Therefore, **Rule CD.2.6** will have not been executed.

- If the advertised address is an actual node in the network and is running a responder processes, then for the DoS-request message to have been processed by the responder, both the DoS-request message and the establishment responder process must have the same session identifier. So the trace must have recorded some earlier invocation of the establishment responder with that session identifier, but it follows from the induction hypothesis that this did not happen. Therefore, **Rule CD.2.6** will have not been executed.

In each case, the targeted node never receives the DoS reply and consequently, progress halts and **Rule CD.2.6** is never executed.

Since the property holds for both the concatenated discovery initiator and the concatenated discovery responder, we can conclude that the result follows. $\square$

We are now in a position to prove that the attackers under consideration do not interfere with the execution of discovery protocols by forcing the execution of high-cost operations. The case of the discovery attacker is considered separately from attackers targeting the establishment-layer messages.

Recall that the discovery responder runs as a daemon at nodes on the dataflow path. Upon arrival of a discovery message advertising a session identifier and an address, the discovery responder invokes the establishment layer to set up a tunnel with the advertised address. As we have seen, discovery layer attackers exploit this behavior by sending discovery messages at their target. The next result says that Off-Path Spoofed Discovery 7.17 cannot force its target to perform high-cost operations.

Theorem 7.4 *Assume that the nodes in the network may execute the concatenated, nested, or modified nested discovery protocols. Assume that the only type of attacker allowed is Off-Path Spoofed Discovery 7.17. Suppose that trace*

$$T = M_1 \xrightarrow{\boldsymbol{X}_1} M_2 \xrightarrow{\boldsymbol{X}_2} M_3 \cdots \longrightarrow M_{n-1} \xrightarrow{\boldsymbol{X}_{n-1}} M_n$$

records the execution of discovery protocols as well as attackers. If an attacker generates session u, where $u \notin \mathfrak{L}(M_1)$, and the trace records the application of a rule $\boldsymbol{X}_i(u) \,@\, a$, where $1 \leq i < n$, then $\boldsymbol{X}_i \notin \boldsymbol{Z}$ (this is not a high-cost operation).

Proof: From inspection of the rules, we see that Off-Path Spoofed Discovery itself does not perform any high-cost operation, but it remains to show that the attacker cannot force its target to execute high-cost operations.

The attacker spoofs an address as well as the session identifier u. These values are advertised in a discovery message that is sent to an honest node that is executing a discovery protocol. The discovery packet at a gateway triggers the discovery protocol responder daemon to begin execution of protocol session u and invoke the establishment initiator. We must show that the attack could not have caused a high-cost operation to occur that would not have occurred otherwise.

Given that session identifier u was created by the attacker using the new operator, it follows from proposition 7.1 that $\mathcal{D}(u) \cap T = \emptyset$. So the session identifier is unique from that of any executing discovery protocol recorded by the trace. Lemmas 7.2 and 7.3 inform us that there is no establishment responder executing for process u.

The proof precedes by induction on the length l of the trace. The base case $l = 0$ is trivial. Consider the case where $l = i$. For the induction step assume $M_i \xrightarrow{\mathbf{X}_i(u)(a)} M_{i+1}$ is a high-cost operation. We examine each of the high-cost rule that a discovery protocol can execute and show that it could not have executed in response to actions taken by the attacker.

DoS reply generation option-H (Rule ME.2.1). This is an establishment responder rule. The attacker only sends a discovery message, which is processed by a discovery responder, which invokes the establishment initiator. Hence, the only way for the particular attacker in question to succeed is to send a discovery message to some node b with the payload $\mathsf{D}(a, u)$ so that the protocol reacts by running establishment with a. Yet the a is generated by the new operator and consequently guaranteed to be unique so this message would never arrive at an existing node and the consequently the **Rule ME.2.1**$(u) \mathbin{@} a$ would never be executed. For the sake of argument, let us assume that a is an existing node, then node b would have sent a DoS-request message to node a having session identifier u. Yet if follows from lemmas 7.2 and 7.3 that the session identifier of the establishment responder process executing at node a will be distinct from u and consequently **Rule ME.2.1**$(u) \mathbin{@} a$ would have never executed.

Credential verification (Rule ME.2.2). This is an establishment responder rule and would have only executed if the node that received the discovery message got to the point where it generated an establishment-request message with a valid signature and session identifier. So the trace must have recorded

$$M_l \overset{\mathbf{ME.1.2}(u)}{\longrightarrow} M_{l+1},$$

where $l < i$, yet this is a high-cost action and it follows from the induction hypothesis that this could not have happened.

Signature generation (Rule ME.2.3) This rule would only execute if

$$M_l \overset{\mathbf{ME.2.2}(u)}{\longrightarrow} M_{l+1},$$

where $l < i$, has been recorded earlier in the trace, yet this is a high-cost action and it follows from the induction hypothesis that this could not have happened.

Signature Generation (Rule ME.1.2). This rule only executes if a valid DoS reply has been received, which means that the node dispatching the DoS reply had processed the DoS-request message from session u. Yet we have already seen that this cannot be the case since the establishment-response process is expecting a message having a different session identifier.

Credential Verification (Rule ME.1.3). This rule would only execute if

$$M_l \overset{\mathbf{ME.1.2}(u)}{\longrightarrow} M_{l+1},$$

where $l < i$, has been recorded earlier in the trace, yet this is a high-cost action and it follows from the induction hypothesis that this could not have happened.

Having shown that the attacker cannot force either the initiator or the responder to perform a high-cost operation, the theorem holds. $\square$.

We now treat cases where the attacker targets a running discovery protocol's establishment layer messages. The following result says that if a high-cost operation occurred in the presence of an attacker, then it occurred absent the presence of an attacker. This means that the attacker did not force a high-cost operation to occur that would not have occurred otherwise.

Theorem 7.5 *Suppose the trace*

$$T = M_1 \xrightarrow{\boldsymbol{X}_1} M_2 \xrightarrow{\boldsymbol{X}_2} \cdots M_{n-1} \xrightarrow{\boldsymbol{X}_{n-1}} M_n$$

records the complete execution of session u of the nested, modified nested, or concatenated discovery protocol, where $u \notin \mathfrak{L}(M_1)$. Suppose

$$T' = M'_1 \xrightarrow{\boldsymbol{X}'_1} M'_2 \xrightarrow{\boldsymbol{X}'_2} \cdots M'_{m-1} \xrightarrow{\boldsymbol{X}'_{m-1}} M'_m$$

records the execution of protocol session u, where $M_1 \sim M'_1$ and $u \notin \mathfrak{L}(M'_1)$ and T' also records the following attackers:

7.1 *Off-Path Spoofed DoS Request.*

7.2 *Off-Path Exposed DoS Request.*

7.5 *Off-Path Spoofed DoS Reply.*

7.6 *Off-Path Exposed DoS Reply.*

7.7 *On-Path Spoofed DoS Reply.*

7.9 *Off-Path Spoofed Establishment Request.*

7.10 *Off-Path Exposed Establishment Request.*

7.11 *On-Path Spoofed Establishment Request.*

7.13 *Off-Path Spoofed Establishment Reply.*

7.14 *Off-Path Exposed Establishment Reply.*

7.15 *On-Path Spoofed Establishment Reply.*

If $M'_i \xrightarrow{\boldsymbol{X}'_i(\boldsymbol{u})(a)} M'_{i+1}$ is the first occurrence of an application of $\boldsymbol{X}'_i(u)$ in T', where $\boldsymbol{X}'_i \in \boldsymbol{Z}$ and a is an honest node, then $\exists j$ such $M_j \xrightarrow{\boldsymbol{X}'_i(\boldsymbol{u})(a)} M_{j+1}$ in T.

Proof: Since T is assumed to be a trace of a complete session, the execution of the discovery protocol terminates successfully. It follows from the noninterference and progress theorems that absent the presence of attackers, $T \cap \mathcal{Q}(u) \sim T' \cap \mathcal{Q}(u)$. If the theorem is false, then it must have been the action of an attacker that forces a high-cost operation to occur in T' that did not occur in T.

The proof is by induction on the length l of the trace T'. If $l = 0$, then the property holds. Consider the case of $M_i' \xrightarrow{\mathbf{X}_i'(u)(a)} M_{i+1}'$, where $\mathbf{X}_i' \in \mathbf{Z}$. We apply case analysis to each of the rules performing high-cost operations and analyze whether the attackers could have forced the rule in question to execute. The goal is to show that the attackers under consideration could not have induced the rule to execute and therefore it must have executed as part of a run of the protocol and consequently would occur in the trace without the attackers.

Rule ME.1.2 An attacker could have triggered this rule to execute if it sent a valid DoS-reply message to the node. Off-Path Spoofed DoS Reply 7.5 and Off-Path Exposed DoS Reply 7.6 both lack the expected session identifier so they could not have sent a message causing the rule to execute. On-Path Spoofed DoS Reply 7.7 does not generate a valid DoS-reply message, and therefore could not have caused the rule to execute.

There remains the possibility that Off-Path Spoofed DoS Request 7.1 and Off-Path Exposed DoS Request 7.2 could have previously forced a node to executing establishment responder session u to execute **Rule ME.2.1**, which sends valid DoS reply that triggered the **Rule ME.1.2** to execute. Yet it follows from lemmas 7.2 and 7.3 that these attackers lack the expected session identifier to succeed.

Consequently, the attackers did not cause this rule to execute.

Rule ME.2.1 Option-H In this case, we assume that DoS-reply message generation is a high-cost operation. The attacker would have to send a DoS-request message to a node running the establishment responder session u. The only attackers under consideration that could accomplish this are Off-Path Spoofed DoS Request 7.1 and Off-Path Exposed DoS Request 7.2, it follows from lemmas 7.2 and 7.3 that these attackers lack the expected session identifier to succeed, and therefore, could not have caused this rule to execute.

Consequently, the attackers cannot have forced this rule to execute.

Rule ME.1.3 This rule executes upon arrival of an establishment reply message that possesses a valid signature. Off-Path Spoofed Establishment Reply 7.13 and Off-Path Exposed Establishment Reply 7.14 cannot gain access to the session identifier u so neither of these attackers could have forced the node to execute the command. On-Path Spoofed Establishment Reply 7.15 cannot generate a valid signature, and therefore did not force the rule to execute.

Suppose one of the attackers under consideration had previously performed an action forcing some honest node running protocol session u to send a valid establishment-reply message. Hence the trace records $M'_l \xrightarrow{\mathbf{ME.2.3}(u)(a)} M'_{l+1}$, where $l < i$, which is a high-cost operation, but it follows from the induction hypothesis that this could not have happened.

Consequently, the attackers cannot have forced this rule to execute.

Rule ME.2.2 This rule executes upon arrival of an establishment-request message with a valid signature. Both the Off-Path Spoofed Establishment Request 7.9 and Off-Path Exposed Establishment Request 7.10 spoof the session identifier and send an establishment-request message to a node running establishment responder session u, but if follows from lemmas 7.2 and 7.3 that the attacker does not possess the expected session identifier and consequently does not succeed.

Suppose one of the attackers under consideration had previously performed an action forcing some honest node running protocol session u to send a valid establishment-request message. Hence the trace records $M'_l \xrightarrow{\mathbf{ME.1.2}(u)} M'_{l+1}$, where $l < i$, which is a high-cost operation, but it follows from the induction hypothesis that this could not have happened.

Consequently, the attackers cannot have caused this rule to execute.

Rule ME.2.3 This rule executes only if **Rule ME.2.2** has executed. Therefore, an attacker is only successful at forcing **Rule ME.2.3** to execute if it had previously been successful at forcing **Rule ME.2.2** to execute. Hence the trace records $M'_l \xrightarrow{\mathbf{ME.2.2}(u)} M'_{l+1}$, where $l < i$, which is a high-cost operation, but it follows from the induction hypothesis that this could not have happened.

Consequently, the attackers cannot have caused this rule to execute.

We have shown that for each high-cost operation appearing in a trace T' of session u, with the presence of attackers, that the attackers did not force the operation to occur. Hence, the application of a high-cost rule in T must have also been recorded in T'.

7.7 Conclusion

This chapter is a study of DoS threats to discovery protocols. A modified version of the establishment layer was given that incorporates a DoS protection scheme. This is intended to protect gateways on the dataflow path from attackers that would direct discovery messages at them in hopes of forcing the gateway to consume resources. The tunnel calculus was used to express a collection of attackers that attempt to exploit possible vulnerabilities in order to deny service to legitimate users. We analyzed the effectiveness of our attackers from the point of view of logical attacks. A cost-based analysis was applied to determine the effectiveness of the DoS protections against our attackers. This analysis highlighted those attacks which succeed in forcing its target to perform high-cost operations and those that fail. Finally, we consider only those attacks that the previous analysis had deemed to fail in forcing their target to execute a high-cost operation and prove several theorems that say that, indeed, these attackers fail in forcing a running discovery protocol to perform high-cost operations.

Chapter 8

Conclusion

In summary, this dissertation has developed a foundation for analyzing tunnel-complex protocols. The cornerstone of our treatment is the tunnel calculus, which is used to express tunnel-complex protocols and reason about their correctness. There are four primary components to our work: the formal definition of the tunnel calculus, its application to reason about deadlocks, its use in defining tunnel-complex protocols and proving their functional correctness, and its application to reasoning about DoS threats to discovery protocols.

The remainder of this chapter is structured as follows. The first section gives a brief overview and summary of the dissertation. The second section provides an assessment of the work. The third section discusses future work.

8.1 Overview

In Chapter 2, we expose the danger posed by cramming attacks and derive the L3A tunnel-complex protocol that constructs a tunnel complex that protects against the said attacks. Logical simulation showed how deadlocks can arise in tunnel-complex protocols and motivates the need for employing formal techniques to ensure their absence.

Chapter 3 provides an informal introduction to the tunnel calculus, where it is used to uncover deadlocks in a tunnel-establishment protocol. We introduce the notion of a session identifier as a means to prevent these deadlocks. In Chapter 4, we give a formal definition of the tunnel calculus that may be viewed as an operational semantics of a protocol stack. The tunnel calculus is composed of layers that model packet forwarding, security tunnels, authorization, and tunnel establishment. Tunnel-complex protocols are developed on top of these layers. The rewriting logic used to define the operational semantics gives rise to a trace theory that we apply to specifying and reasoning about functional correctness and DoS threats. In Chapter 5, we prove a noninterference theorem 5.9, which says one establishment session will not interfere with the messages sent another session. We also prove a progress theorem 5.10, which says if communication between two parties is possible, then it is possible to extend any other to complete the communication. These results depend on a simulation lemma 5.4 that tells us when two traces are semantically the same.

Our goal in creating the tunnel calculus was to develop a framework that could be used to express and reason about tunnel-complex protocols. In Chapter 6, the tunnel calculus is applied to the design of three discovery protocols, which discover security gateways on the dataflow path, deliver credentials needed to negotiate their traversal, and set up tunnels for a specific topology. We also introduce a notion of completeness for discovery protocols, which acts as a functional correctness property. Discovery protocol completeness says if the credentials needed to traverse the gateways are located at specified locations on the path, then the protocol will deliver these credentials to the gateways. Our first discovery protocol constructs a concatenated tunnel complex that can be viewed as a generalization of the tunnel complex set up by the L3A protocol. The second discovery protocol constructs a nested tunnel complex that can be viewed as a generalization of a typical road-warrior scenario. The completeness theorem for this protocol is far more restrictive with regards to the placement of credentials than the concatenated case. The third discovery protocol presented was a modification of the second that allows a more liberal placement of the credentials on the dataflow path so that its completeness theorem is essentially the same as for the concatenated case.

In Chapter 7, we apply the tunnel calculus to the study of DoS threats to discovery protocols. We distinguish between logical and resource-exhaustion attackers. Attacker capabilities are further classified as off-path, on-path, exposed, and spoofed. This classification forms the basis of the methodology we employ to analyze DoS threats. First, for each message that could be the target of attack, the tunnel calculus was used to construct attackers possessing each of the capabilities. Reasoning, in terms of rewrites performed on the multiset, about the execution of discovery protocols in the presence of each of the attackers allows us to evaluate whether the attacker would succeed from the point of view of a logical attack. A cost model applied to the same protocols and attackers allows us to evaluate their effectiveness from the point of view of a resource-exhaustion attack. Finally, we proved several theorems showing that our discovery protocols are resistant to resource-exhaustion attacks launched by a particular subset of our attackers.

8.2 Assessment

The tunnel calculus may seem too large and too complicated, especially in comparison to a formalism such as the $\lambda-$calculus, CSP, or the $\pi-$calculus. Consider the $\pi-$calculus, its basic model for communication corresponds to the tunnel calculus forwarding layer rather than the secure-processing layer. In order to augment the $\pi-$calculus with security tunnels (our secure processing layer), one must create structures that correspond to those in our model. Directly modeling persistent structures such as the association and mechanism databases is not easy in processes algebras and one is forced to choose between a model that is very convoluted or bolting on structures similar to those in the tunnel calculus. There are ample examples of key-exchange protocols expressed in variants of the $\pi-$calculus and these do not seem any simpler than expressing them in the tunnel-calculus, except that, in the tunnel calculus, establishment is a fundamental component of the network stack. We believe that using a 'simpler' formalism to model our protocol stack becomes an exercise in making rock-soup. Although one initially starts out with something that is arguably simpler, in order to construct a corresponding protocol stack, one eventually winds up with a system that is at least as complex as the tunnel calculus because the complexity is intrinsic in the subject being modeled.

Adding session identifiers to our model of security tunnels resolved many problems, but these may be seen as bound to the application that invoked the tunnel-complex protocol. Should not all applications on a host be able to use a tunnel complex that has been constructed to facilitate communication between it and another node without rerunning discovery? Can this be done without reengineering the tunnel calculus mechanism filter processing? The short answer to both questions is yes. We can add a rule that acts as an interface between the secure layer and higher-layer protocols. For instance,

$$\begin{array}{lll}\Pi^o & \vdash & \downarrow_{\mathrm{sec-inter}(k_1)} \mathsf{P}(b,c,y) \longrightarrow \downarrow_{\mathrm{sec}(u,k_2)} \mathsf{P}(b,c,y) \\ & & \text{where } \mathsf{Mech}(b \longrightarrow c : u : \beta^o) \in \Pi^o.\end{array}$$

This rule simply selects a session having the same traffic filter and sends a message in that tunnel complex. Once this interface has been constructed, rules for a higher-layer protocol can be written without any need to know about session identifiers.

The primary focus in this dissertation has been the functional correctness of tunnel-complex protocols. Namely, deadlocks at the establishment layer and completeness for discovery protocols. We have abstracted away questions of secrecy and integrity to the greatest degree possible. This stands in contrast to most studies of security protocols that focus on secrecy and integrity. The conditions that give rise to the deadlocks are due to the very nature of installing state that in turn can affect the traffic. In practice this is often avoided by having the mechanism filters installed before establishment is run and exempting the establishment traffic from the filters. This approach was rejected as we preferred to support both gateways enforcing ingress and egress authenticated traversal (all ingress/egress traffic traversing a gateway must be authenticated). This is a case where weaker security guarantees can eliminate functional errors and has led us to wonder whether strengthening security guarantees leads to a greater likelihood of functional errors in other protocols.

The discovery protocols developed in this dissertation set up relatively simple tunnel complexes. As a consequence, their completeness theorems were relatively straightforward. One could argue that for these relatively simple protocols, experimentation and informal techniques would suffice. Yet history has shown such techniques inadequate whenever concurrency is involved. In addition, one of the primary reasons for focusing on relatively simple discovery protocols was a succession of failed attempts to construct more sophisticated designs for which we were not able to formulate a completeness theorem. This led to the realization that we needed to master simple protocol designs before embarking on more sophisticated cases.

As it is currently designed, the tunnel calculus has proved sufficient to analyze functional correctness and DoS threats that were the focus of this dissertation, but in its current form, it would not be sufficient for analyzing secrecy and integrity properties of key establishment protocols. On the other hand, only minimal changes would be necessary to apply the tunnel calculus to analyzing the secrecy and integrity of messages traveling in the tunnels. In addition, the fact that, at the lowest level, communication is modeled as packet forwarding based on a forwarding table should make it possible to perform future studies on how discovery protocols interact with route changes.

We used the tunnel calculus to model a collection of DoS attackers and studied their interaction with the chosen target by examining rewrites performed by both. The processes was admittedly tedious, but rather mechanical, which suggests that if you have the attacker and the protocol, then the analysis could be mechanized. Probably the biggest downside to our specific approach was the need to write down so many rules to model the attackers. An alternative would be to design a more general attacker that would subsume the specific cases given above, but we would expect this to be a nontrivial task and best left for future work.

In our study of discovery protocols, we designed protocols that constructed a concatenated tunnel complex as well as protocols that constructed a nested tunnel complex. The nested protocols were somewhat limited in that they presumed that the host initiating the protocol is not behind a gateway. This is a consequence of our preference to have gateways enforce ingress authenticated traversal so that all traffic passing a gateway must be authenticated in a tunnel. Our investigations have revealed that the nested tunnel complex where each tunnel is rooted at one endpoint is not compatible with the authenticated traversal property because establishment traffic must be allowed to pass through the gateways protecting the host that anchors the tunnel complex unauthenticated by a tunnel. As mentioned in the next section, in the future, we hope to investigate protocols that set up other nested tunnel complexes that are compatible with gateways enforcing the ingress authenticated traversal property. We believe that an argument can be made favoring the concatenated tunnel-complex since it allows gateways to enforce the authenticated-traversal properties while being set up by a relatively simple protocol.

The analysis of DoS threats was purely theoretical no experimental validation has been performed. On the other hand, we do feel that the our models are sufficiently precise to enable a developer to implement the attacks. We speculate that the attacks against the discovery protocol would be the easiest to carry out in practice since the target need not be in a state waiting for a particular establishment-layer message. We also speculate that specific implementations will be vulnerable to logical attacks that were not considered here, but are specific to that implementation. Resource exhaustion of memory or storage resources depends to some extent on the implementation, and in our analysis, we only focused on the mechanism and association databases and it remains to see if that is sufficient for a given implementation. In the case of CPU exhaustion attacks, we believe that experimental evidence would support our theoretical analysis, but having precise numbers would allow us better gauge the utility of the analysis.

Most tunnel complexes found in practice are relatively simple. Yet managing even the hub-and-spoke topology has driven Cicso to build a tunnel-complex protocol DMVPN that automates its configuration. Similar concerns have motivated the development of the discovery protocol TED. So there is evidence that tunnel-complex protocols solve a recognized problem, but their future remains an open question. If gateways are deployed in a defense-in-depth and tunnels are used to authenticate traffic, then the complexity of managing them will likely lead to more sophisticated discovery protocols than currently exist. If this is indeed the case, then, based on our experience, formal analysis will be needed detect flaws and unexpected interactions that could arise in practice, but elude testing of prototype implementations. On the other hand, if defense in depth schemes composed of security gateways and security tunnels are not viewed as viable, then we would not expect to see the development of more sophisticated discovery protocols that this research supports.

We believe that the approach developed in this dissertation is applicable beyond the context of tunnel-complex protocols. For instance, web services and Voice Over IP (VOIP) are also characterized by many interacting protocol layers and there is great interest in analyzing both functional correctness and security properties for such protocols. Our technique for modeling the operational semantics of a protocol stack could prove quite useful in such domains. We also believe that our approach to studying DoS threats could be applied to VOIP and similar protocols that are current interest.

8.3 Future Work

There are a number of topics that we would like to pursue in the future. In its current form, the tunnel calculus is a piece of mathematics, but since the operational semantics is given in terms of a rewriting logic, an executable version built in the Maude framework, similar to the one built for L3A, would give the tunnel-complex protocol designer a valuable debugging tool that can be used to perform both logical simulations as well as model checking.

Additional discovery protocols are an obvious avenue for future work. The nested protocols presented in Chapter 6 assume one of the nodes is not located behind any gateway. This simplified the task of designing a discovery protocol that would satisfy the ingress authenticated traversal property. We would like to extend this work to a protocol where both end hosts are located behind multiple gateways. Protocols that perform more complex gateway negotiations also warrant further study. In addition, we would like to create implementations in order to gather performance measurements as we did with L3A.

In Chapter 7, we constructed a collection of DoS attackers that served as the object of our analysis. In future work, we would like to build upon the knowledge gained in this effort to construct a general attacker along the lines of a Dolev-Yao attacker. If successful, one should be able to instantiate the general model to each of our concrete attacks. In addition, some experimental validation of our theoretical analysis would be useful as discussed in the previous section.

Throughout this dissertation, we have assumed a network topology where each administrative domain has a single gateway and we also assumed each node has a fixed forwarding table (i.e no routing is performed). Relaxing these strictures would provide ample ground for future research. For instance, new theories may be needed to characterize the correctness of establishment protocols if messages can enter and exit different gateways. If a route changes, under what conditions does this break a tunnel-complex protocol? Under what conditions would a route change break a tunnel complex that has already been set up? Can route changes suddenly expose information that is believed to be protected? These and many other questions regarding the effects of route changes remain to be addressed.

Bibliography

[1] M. Abadi. On SDSI's Linked Local Name Spaces. *Journal of Computer Security*, 6(1-2):3–21, 1998.

[2] M. Abadi, B. Blanchet, and Cedric Fournet. Just Fast Keying in the Pi calculus. In D. Schmidt, editor, *European Symposium on Programming (ESOP)*, Lecture Notes in Computer Science 2618. Springer-Verlag, 2004.

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

[4] W. Aiello, S. Bellovin, M. Blaze, R. Caetti, J. Ioannidis, A. Keromytis, and O. Reingold. Just fast keying: Key agreement in a hostile internet. *ACM Transactions on Information System Security*, 7(2):242–273, 2004.

[5] G. Andrews. *Concurrent Programming: Principles and Practice.* Addison Wesley, 1991.

[6] T. Aura, P. Nikander, and J. Leiwo. DoS Resistant Authentication with Client Puzzles. In *Revised Papers from the 8th International Workshop on Security Protocols*, Lecture Notes in Computer Science 2133, pages 170–177. Springer-Verlag, 2001.

[7] M. Bellare, R. Canetti, and H. Krawczyk. A Modular Approach to the Design and Analysis of Authentication and Key Exchange Protocols. In *Proceedings of the 30th Annual ACM Symposium on the Theory of Computing (STOC'98)*, pages 419–423. ACM, 1998.

[8] M. Bellare and P. Rogaway. Entity authentication and key distribution. In D. Stinson, editor, *Proceedings of 13th Annual Advances in Cryptology (Crypto 93)*, Lecture Notes in Computer Science 773, pages 232–249. Springer-Verlag, 1994.

[9] B. Bencsath, I. Vajda, and L. Buttyan. A Game Based Analysis of the Client Puzzle Approach to Defend Against DoS Attacks. In *Proceedings of the IEEE Conference on Software, Telecommunications and Computer Networks (SoftCom 03)*. IEEE, 2003.

[10] D. J. Bernstein. SYN cookies.

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

[12] S. Bishop, M. Fairbairn, M. Norrish, P. Sewell, M. Smith, and K. Wansbrough. Rigorous specification and conformance testing techniques for network protocols, as applied to TCP, UDP, and Sockets. In *Proceedings of ACM Conference on Computer Communication (SIGCOMM 2005)*, pages 265–276. ACM, 2005.

[13] B. Blanchet. From Secrecy to Authenticity in Security Protocols. In M. Hermenegildo and G. Puebla, editor, *Proceeding of 9th International Static Analysis Symposium (SAS'02)*, Lecture Notes In Computer Science 2477, pages 342–359. Springer-Verlag, 2002.

[14] B. Blanchet. Automatic Proof of Strong Secrecy for Security Protocols. In *Proceedings of the 25th Annual IEEE Symposium on Security and Privacy (Oakland 04)*, pages 86–100. IEEE, 2004.

[15] M. Blaze, J. Feigenbaum, J. Ioannidis, and A. Keromytis. The Role of Trust Management in Distributed Systems Security. In *Secure Internet Programming*, Lecture Notes in Computer Science 1603. Springer-Verlag, 1999.

[16] M. Blaze, J. Feigenbaum, and J. Lacy. Decentralized Trust Management. In *Proceedings of Symposium on Security and Trust Management*, pages 164–173, 1996.

[17] M. Blaze, J. Feigenbaum, and M. Strauss. Compliance Checking in Policy Maker. In *Proceedings of Financial Cryptography*, Lecture Notes in Computer Science 1465, pages 254–274. Springer-Verlag, 1998.

[18] M. Blaze, J. Ioannidis, and A. Keromytis. Trust Management in IPsec. *ACM Transactions on Information and System Security*, 32:1–24, 2002.

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

[20] R. Branden, L. Zhang, S. Berson, S. Herzog, and S. Jamin. Resource ReSerVation Protocol (RSVP). RFC 2205, IETF, 1997.

[21] R. Canetti. Universally Composable Security: A New Paradigm for Cryptographic Protocols. In *Proceedings of the 42nd IEEE Symposium on Foundations of Computer Science*, pages 136–145. IEEE, 2001.

[22] R. Canetti. Security and Composition of Cryptographic Protocols: A Tutorial. *ACM SIGACT News* , 37(3,4), 2006.

[23] R. Canetti and H. Krawczyk. Analysis of Key-Exchange Protocols and Their Use for Building Secure Channels. In *Proceedings of Advances in Cryptology (Eurocrypt 01)*, Lecture Notes in Computer Science 2045, pages 453–474. Springer-Verlag, 2001.

[24] R. Canetti and H. Krawczyk. Security Analysis of IKE's Signature-based Key-Exchange Protocol. In *Proceedings of Advances in Cryptology - Crypto 2002*, Lecture Notes In Computer Science 2442, pages 127–142. Springer-Verlag, 2002.

[25] C.A.R. Hoare. *Communicating Sequential Processes.* Prentice-Hall, 1985.

[26] I. Cervesato. Typed MSR: Syntax and examples. In V.I. Gorodetski, V.A. Skormin, and L.J. Popyack, editors, *Workshop on Mathematical Methods, Models, and Architectures for Computer Network Security*, pages 159–176. Springer-Verlag, 2001. 159–176.

[27] K. Chandy and J. Misra. *Parallel Program Design.* Addison Wesley, 1988.

[28] Dynamic Multipoint VPN (DM VPN). Cisco White Paper.

[29] D. Clarke, J.-E. Elien, C. Ellison, M. Fredette, A. Morcos, and R. Rivest. Certificate Chain Discovery in SPKI/SDSI. *Journal of Computer Security*, 9(4):285–322, 2001.

[30] E. Clarke, E. Emerson, and A. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions of Programming Languages*, 8(2):244–263, 1986.

[31] E. Clarke, O. Grumberg, and D. Peled. *Model Checking.* MIT Press, 1999.

[32] M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, and J. Quesada. Maude: specification and programming in rewriting logic. *Theoretical Computer Science*, 285:187–243, 2002.

[33] A. Datta, A. Derek, J. Mitchell, and D. Pavlovic. A Derivation System and Compositional Logic for Security Protocols. *Journal of Computer Security*, 13:423–482, 2005.

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

[35] V. Dieker and G. Rozenberg, editors. *The Book of Traces.* World Scientific, 1994.

[36] T. Dierks and E. Rescorla. The TLS Protocol. RFC 4346, IETF, 2006. Obsoletes: 2246.

[37] W. Diffie and M. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, IT-22(6):644–654, 1976.

[38] W. Diffie, P. van Oorschot, and M. Wiener. Authentication and Authenticated Key Exchanges. *Designs, Codes, and Cryptography*, 2:107–125, 1992.

[39] E.W. Dijkstra. Solution of a Problem in Concurrent Programming. *Communication of the ACM*, 8(9):569, 1965.

[40] E.W. Dijkstra. Cooperating Sequential Processes. In F. Genuys, editor, *Programming Languages*, pages 43–112. Academic Press, 1968.

[41] E.W. Dijkstra. The Structure of the THE Multiprogramming System. *Communication of the ACM*, 11(5):341–346, 1968.

[42] E.W. Dijkstra. Hierarchical Ordering of Sequential Processes. *Acta Informatica*, 1:115–138, 1971.

[43] N. Durgin, J. Mitchell, and D. Pavlovic. A Compositional Logic for Proving Security Properties of Protocols. *Journal of Computer Security*, 11:677–721, 2004.

[44] C. Ellison. Spki requirements. RFC 2692, IETF, 1999.

[45] C. Ellison, B. Frantz, B. Lampson, R. Rivest, B. Thomas, and T. Ylonen. Simple Public Key Certificate. Technical report, IETF, 1999.

[46] C. Ellison, B. Frantz, B. Lampson, R. Rivest, B. Thomas, and T. Ylonen. SPKI Certificate Theory. RFC 2693, IETF, 1999.

[47] W.H.J Feijn and A.J.M. Van Gasteren. *On A Method of Multi-Programming* . Springer-Verlag, 1999.

[48] S. Fluhrer. Tunnel endpoint discovery. Internet Draft draft-fluhrer-ted-00.txt, IETF, 2001.

[49] X. Fu, D. Hogrefe, and C. Werner. Modeling Route Change in Soft State Signaling Protocols Using SDL: a Case of RSVP. In A.Prinz, R. Reed, and J. Reed, editors, *Proceedings of the 12th SDL Forum (SDL 2005)*, Lecture Notes in Computer Science 3530, pages 174–186. Springer-Verlag, 2005.

[50] Z. Fu, S.Wu, H. Haung, K. Loh, F. Gong, I. Baldine, and C. Xu. IPsec/VPN Security Policy: Correctness and Conflict Resolution. In M. Sloman, J. Lobo, and E. Lupu, editors, *IEEE Policy Workshop*, pages 39–56, 2001.

[51] Z. Fu and S. Wu. Automatic Generation of IPsec/VPN Security Policies in and Intra-Domain Environment. In O. Festor and A. Pras, editors, *International Workshop on Distributed Systems: Operations and Management (DSCM)*, Lecture Notes in Computer Science 1995, pages 279–290, Nancy, France, October 2001.

[52] A. E. Goodloe. *The Life and Times of Andrew I. Schein.* In Progress.

[53] A. E. Goodloe and C. A. Gunter. Reasoning About Concurrency for Security Tunnels. In *Proceedings of 20th IEEE Computer Security Foundations (CSF 07)*, pages 64–78. IEEE, 2007.

[54] A. E. Goodloe, M. Jacobs, G. Shah, and C. A. Gunter. Proceedings of the L3A: A protocol for layer three accounting. In *Proceedings of Secure Network Protocols (NPSec '05)*, Boston, MA, November 2005. IEEE.

[55] A.E. Goodloe, M.-O. Stehr, and C. A. Gunter. Formal prototyping in early stages of protocol design. In *Proceedings of the 4th Workshop on Issues in the Theory of Security (WITS '05)*, Long Beach, CA, January 2005. IFIP.

[56] L. Gordon, M. Loeb, W. Lucyshyn, and R. Richardson. CSI/FBI Computer Crime and Security Survey. Technical report, Computer Security Institute, 2006.

[57] D. Gries and S. Owicki. An Axiomatic Proof Technique for Parallel Programs. *Acta Informatica*, 6:319–340, 1976.

[58] C. A. Gunter and T. Jim. Design of an Application-Level Security Infrastructure. In *Proceedings of the DIMACS Workshop on Design and Formal Versification of Security Protocols*, 1997.

[59] C. A. Gunter and T. Jim. Policy-Directed Policy Certificate Retrieval. *Software: Practice and Experience*, 30(15):1609–1640, September 2000.

[60] C. A. Gunter and T. Jim. What is QCM. Technical report, University of Pennsylvania, 2000.

[61] J. Halpren and R. van de Meyden. A Logic for SDSI's Linked Local Name Space. *Journal of Computer Security*, 9(1):47–74, 2001.

[62] J. Halpren and R. van de Meyden. A Logical Reconstruction of SPKI. *Journal of Computer Security*, 11(4):581–614, 2003.

[63] K. Hamzeh, G. Pall, W. Verthein, J. Taarud, W. Little, and W. Little. Point-to-Point Tunneling Protocol (PPTP). RFC 2537, IETF, 1999.

[64] T. Hiller, P. Walsh, X. Chen, M. Munson, G. Dommety, S. Sivalingham, B. Lim, P. McCann, H. Shiino, B. Hirschman, S. Manning, R. Hsu, , M. Lipford, P. Calhoun, C. Lo, E. Jaques, E. Campbell, Y. Xu, S. Baba, T. Ayaki, T. Seki, and A. Hammed. CDMA2000 Wireless Data Requirements for AAA. RFC 3141, IETF, 2001.

[65] C.A.R Hoare. Monitors: An Operating System Structuring Concept. *Communication of the ACM*, 17(10):549–557, 1974.

[66] Evolution of Charging and Billing Models for GSM and Future Mobile Internet Services, 2000.

[67] Computing System Innovations. Security Systems Innovations. White Paper.

[68] J. Guttman and A. Herzog and F. Javier Thayer. Authentication and confidentiality via ipsec. In F. Cuppens and Y. Deswarte and D. Gollmann and M. Waidner , editor, *Proceedings of the 6th Annual European Symposium on Research in Computer Security (ESORICS)*, Lecture Notes in Computer Science 1895, pages 255–272. Springer-Verlag, 2000.

[69] S. Jha and T. Reps. Analysis of SPKI/SDSI Certificated Using Model Checking. In *Proceedings of 15th IEEE Computer Security Foundations Workshop (CSF 02)*, pages 129–144. IEEE, 2002.

[70] C. B. Jones. Tentative Steps Towards a Development Method for Interfering Programs. *ACM Transactions on Programming Languages and Systems*, 5(4):576–619, 1983.

[71] C. B. Jones. Accommodating Interference in the Formal Design of Concurrent Object-Based Programs. *Formal Methods in System Design*, 8(2), 1996.

[72] A. Juels and J. Brainard. Client Puzzles: A Cryptographic Countermeasure Against Connection Depletion Attacks. In *Proceedings of IEEE Network and Distributed System Security Symposium (NDSS) 1999*, pages 151–165. IEEE, 1999.

[73] GPRS Threats and Recommendations. Juniper Networks White Paper.

[74] Infrastructure Security Gateway for GPRS Networks. Juniper Networks White Paper.

[75] P. Kakkar, M. McDougall, C. A. Gunter, and T. Jim. Credential Distribution with Local Autonomy . In *The Second International Working Conference on Active Networks*, 2000.

[76] C. Kaufman. Internet Key Exchange (IKE V2) protocol. RFC 4306, IETF, 2005. Obsoletes: 2407, 2408, 2409.

[77] 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.

[78] S. Kent. IP Authentication Header (AH). RFC 4302, IETF, 2005. Obsoletes: 2402.

[79] S. Kent. IP Encapsulating Security Payload (ESP). RFC 2406, IETF, 2005. Obsoletes: 2406.

[80] S. Kent and K. Seo. Security architecture for the internet protocol. RFC 4301, IETF, 2005. Obsoletes: 2401.

[81] Guy Kewney. Official: Hackers Have Broken into GPRS Billing. newswireless.net, October 2003.

[82] M. Koutsopoulou, A. Kaloxylos, A. Alonistioti, L. Merakos, and K. Kawamura. Charging, Accounting, and Billing Management Schemes in Mobile Telecommunications Networks and the Internet. *IEEE Communications Surveys*, 6(1), 2004.

[83] S. Lafrance and J. Mullins. An Information Flow Method to Detect Denial of Service Vulnerabilities. *Journal of Universal Computer Science*, 9(11):1350–1369, 2003.

[84] L. Lamport. Time, Clocks, and the Ordering of Events in a Distributed System. *Communications of the ACM*, 21(7):558–565, July 1978.

[85] N. Li and J. Mitchell. Understanding SPKI/SDSI Using First Order Logic. In *IEEE Computer Security Workshop*, pages 89–103, 2003.

[86] J. Luciani, D. Katz, D. Piscitello, B. Cole, and N. Doraswamy. Next Hop Routing Protocol (NHRP). Technical report, IETF, 1998. RFC.

[87] A. Mahimkar and V. Shmatikov. Game-Based Analysis of Denial-of-Service Prevention Protocols. In *Proceedings of the 15th Annual IEEE Computer Security Foundations (CSF 05)*, pages 287–301. IEEE, 2005.

[88] D. McDonald, C. Metz, and B. Phan. PF_KEY Key Management API, Version 2. RFC 2367, IETF, 1998.

[89] C. Meadows. A Formal Framework and Evaluation Method for Network Denial of Service. In *Proceedings of the 12th Annual Computer Security Foundations Workshop (CSFW99)*, pages 4–13, 1999.

[90] C. Meadows and D. Pavlocic. Deriving, attacking, and defending the GDOI protocol. In *Proceedings of 10th Annual European Symposium on Research in Computer Security (ESORICS 04)*, Lecture Notes in Computer Science 3193, pages 53–72. Springer-Verlag, 2004.

[91] Catherine Meadows. The NRL protocol analyzer: An overview. *Journal of Logic Programming*, 19:1–19, 1994.

[92] Catherine Meadows. Analysis of the internet key exchange protocol using the NRL protocol analyzer. In *Proceeding of the IEEE Symposium on Security and Privacy (Oakland 99)*, pages 216–231. IEEE, 1999.

[93] A. J. Menezs, P. C. van Oorchot, and S.A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1996.

[94] R. Milner. *Communicating and Mobile Systems: the π-Calculus*. The Cambridge University Press, 1999.

[95] 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.

[96] D. Nicol, S. Smith, and M. Zhao. Evaluation of Efficient Security for BGP Route Announcements using Parallel Simulation. *Simulation and Practice and Theory Journal*, 12(3-4):187–216, 2004.

[97] National Institute of Standards and Technology (NIST). Digital Signature Standard (DSS). Technical report, 1994.

[98] L. Paulson. The Inductive Approach to Verifying Cryptographic Protocols. *J. Computer Security*, 6:85–128, 1998.

[99] L. Paulson. Inductive Analysis of the Internet Protocol TLS. *ACM Transactions on Computer and System Security*, 2(3):332–351, 1999.

[100] D. Pavlovic and C. Meadows. Deriving Secrecy in Key Establishment Protocols. In *Proceedings of 12th Annual European Symposium on Research in Computer Security (ESORICS 06)*, Lecture Notes in Computer Science 4189, pages 384–403. Springer-Verlag, 2006.

[101] Ariff Premji. Deploying Enhanced NAT Services in GPRS Networks: Mitigating Overbilling Attack. Juniper Networks White Paper.

[102] J. Reed, D. Jackson, B. Deianov, and G. Reed. Automated Formal Analysis of Networks: FDR Models of Arbitrary Topologies and Flow-Control Mechanisms. In *Fundamental Approaches to Software Engineering*, Lecture Notes in Computer Science 1382. Springer-Verlag, 1998.

[103] J. Reynolds. Syntactic control of interference. In *Proceeding of Fifth Symposium ACM on Principle of Programming Languages*, pages 39–46, 1978.

[104] J. Reynolds. Syntactic control of interference, part 2. In *Proceedings of the 16th International Colloquium on Automata, Languages and Programming*, volume 372 of *Lecture Notes in Computer Science*, pages 704–722. Springer-Verlag, 1989.

[105] C. Rigney. RADIUS Accounting. RFC 2866, IETF, 2000.

[106] R. Rivest and B. Lampson. SDSI - A Simple Distributed Security Infrastructure, 1996.

[107] R. Rivest, A. Shamir, and L. Adleman. A Method for Obtaining Digital Signatures in Public-Key Cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.

[108] P. Ryan and S. Schneider. *Modeling and Analysis of Security Protocols.* Addison-Wesley, 2001.

[109] F. Schneider. *On Concurrent Programming.* Springer-Verlag, 1997.

[110] M. Sherr, M. Greenwald, C. A. Gunter, S. Khanna, and S. Venkatesh. Mitigating DoS Attack Through Selective Bin Verification. In *Proceedings of the IEEE Workshop on Secure Network Protocols (NPsec '05)*, pages 7–12. IEEE, 2005.

[111] J. Smith, J.M. Gonzalez-Nieto, and C. Boyd. Modelling Denial of Service Attacks on JFK with Meadow's Cost-Based Framework. In *Proceedings of Fourth Australasian Information Security Workshop (AISW-NetSec 06)*, 2006.

[112] Solsoft Policy Server: Better Management for Network Security. Solsoft White Paper, 2003.

[113] P. Srisuresh and M. Holdrege. IP Network Address Translator (NAT) Terminology and Considerations. RFC 2663, IETF, 1999.

[114] Keith Stouffer, Joe Falco, and Karen Kent. Guide to Supervisory Control and Data Acquisition (SCADA) and Industrial Control Systems Security. Technical Report Special Publication 800-82, NIST, September 2006.

[115] U.S. Department of Health and Human Services. Security Standards for the Protection of Electronic Protected Health Information. Code of Federal Regulations, 2005. 45CFR164.308.

[116] T. Ylonen and C. Lonvick. The Secure Shell (SSH) Authentication Protocol. RFC 4252, IETF, 2006.

[117] T. Ylonen and C. Lonvick. The Secure Shell (SSH) Protocol Architecture. RFC 4251, IETF, 2006.