Security Goals: Packet Trajectories and Strand Spaces*

Joshua D. Guttman

The MITRE Corporation
Bedford, MA 01730 USA
guttman@mitre.org

Abstract. This material was presented in a series of lectures at FOSAD, a summer school on Foundations of Security Analysis and Design, at the University of Bologna Center at Bertinoro in September 2000. It has two main purposes.
The first purpose is to explain how to model and analyze two important security problems, and how to derive systematic solutions to them. One problem area is the "packet protection problem," concerning how to use the security services provided by routers—services such as packet filtering and the IP security protocols—to achieve useful protection in complex networks. The other problem area, the "Dolev-Yao" problem, concerns how to determine, given a cryptographic protocol, what authentication and confidentiality properties it achieves, assuming that the cryptographic primitives it uses are ideal.
Our secondary purpose is to argue in favor of an overall approach to modeling and then solving information security problems. We argue in favor of discovering security goals for specific domains by examining the threats and enforcement mechanisms available in those domains. Mathematical modeling allows us to develop algorithms and proof methods to ensure that the mechanisms achieve particular security goals. This leads to a systematic approach to trust management, often a more pressing information security problem than inventing new and improved security mechanisms.

1 Introduction

We summarize here a series of lectures at FOSAD, a summer school on Foundations of Security Analysis and Design, at the University of Bologna Center at Bertinoro, in September 2000.

1.1 The Purpose of These Lectures

This series of lectures has two main goals. The first goal is to explain how to model and analyze two important security problems, and how to design reliable solutions to them, namely:

* Work reported here was supported by the National Security Agency through US Army CECOM contract DAAB07-99-C-C201. Work was in collaboration with Amy L. Herzog, Jonathan C. Herzog, and F. Javier Thayer.

R. Focardi and R. Gorrieri (Eds.): FOSAD 2000, LNCS 2171, pp. 197–261, 2001.

1. How to use the security services provided by routers—services such as packet filtering and the IP security protocols—to achieve useful protection in complex networks; and
2. How to determine, given a cryptographic protocol, what authentication and confidentiality properties it achieves, assuming that the cryptographic primitives it uses are ideal.

We refer to these two problems as the packet protection problem and the Dolev-Yao [9] problem respectively.

The second goal is to argue in favor of an overall approach to modeling and solving information security problems, based around three ideas:

1. There does not exist any single property, which if we achieve it, will provide the information security we need in every situation.
   Instead, in each application area we must analyze the kinds of threat we face. From this, we can abstract a class of properties that capture the different meaningful protections in this type of situation. We call each property in the class a "security goal," and we regard the class itself as defining the range of security that may need to be provided in this application area.
   Different application areas will lead to different classes of security goals, formulated in terms of different modeling ideas.
2. Security goals need to be expressed using a simple, well-understood vocabulary of mathematical notions, and we will use the same vocabulary to model systems, the systems that we want to ensure will meet these security goals. In this series of lectures, the mathematical vocabulary we use will include boolean algebras and freely generated algebras of terms, as well as graphs. The graphs will include directed and undirected graphs, and at various stages we will need to partition either the nodes or the edges into distinct classes. Needless to say, this vocabulary is very familiar and very easy to reason about.
3. These mathematical notions themselves suggest algorithms and proof methods that are useful for *security management*. The mathematical notions lead to flexible and efficient ways to resolve problems, such as:
   (a) Does a given system meet a particular security goal?
   (b) Given a system, what security goals does it meet?
   (c) Given a security goal, how can we configure (or modify) a system to meet it?
   (d) Given a real-world system, how can we construct the abstraction that models it?
   (e) Given a real-world system, what automated tests (or manual analysis) will check whether a given abstraction models it faithfully? Whether a given security goal has been violated?
   (f) If two given systems each meet a security goal, does each continue to meet that security goal if they are combined in a particular way?

   These questions are the core of the crucial real-world problem of security management.

1.2 The Structure of These Lectures

We divide the remainder of our report into five sections.

Section 2 *Packet Trajectories*: Derived from [11,13]. Coauthors: A. Herzog and J. Thayer.
Contents: Introduce the packet protection problem. Define a class of security goals that filtering routers can achieve. Network model. Algorithms to determine whether given filtering behavior achieves a goal, and to assign filtering behavior that will achieve a given goal. Abstraction from router configuration files.

Section 3 *Strand Spaces and Protocol Security Goals*: Material derived from [47]. Coauthors: J. Herzog and J. Thayer. Also from [14,16]. Coauthor: J. Thayer.
Contents: Cryptographic protocols and the Dolev-Yao problem. Why cryptographic protocols fail: Unintended services. Powers of the penetrator. Strand space definitions. Authentication and secrecy goals.

Section 4 *Well-Behaved Bundles and Paths through Them*: Material derived from [16]. Coauthor: J. Thayer.
Contents: Bundle equivalence. Redundancies and redundancy elimination. Paths. The normal form lemma. Rising and falling paths, bridges, efficiency. Proof of the secrecy theorem.

Section 5 *Authentication Tests*: Material derived from [14,16]. Coauthor: J. Thayer.
Authentication tests: proving authentication and secrecy. Proofs of the authentication test results. Application to examples.

Section 6 *Protocol Independence via Disjoint Encryption*: Material derived from [15]. Coauthor: J. Thayer.
Contents: Multiprotocol strand spaces, independence. Disjoint encryption. Applications of protocol independence.

Copies of the papers cited above are available at URL

`http://www.ccs.neu.edu/home/guttman/`.

Acknowledgements I am grateful to my coauthors Amy L. Herzog, Jonathan C. Herzog, and F. Javier Thayer. Dan Klain made numerous suggestions. Sylvan Pinsky and Grant Wagner provided stimulus, support, and discussion.

2 The Packet Protection Problem

In this section, we explore one security mechanism, namely filtering routers (and firewalls of related kinds). They are widely used to protect enterprises or their parts from attacks that could be launched from outside them. Wide practical experience exists as to the types of packets (identified by the IP, TCP and UDP headers) that are most capable of causing harm to their recipients, and firewall

administrators configure their enforcement mechanisms to trade the danger of incoming datagrams off against the value of the network services that they provide and the level of trust in those that may have crafted the datagrams. We will describe a systematic method to ensure that security policies of this kind are faithfully implemented, despite the topological complexity of the networks. Most of this material is contained in [11] in a more systematic style, although the material on abstraction is previously unpublished.

We have carried out a similar study of how to use the IP security protocols (IPSEC) to achieve genuine confidentiality, authentication, and integrity. The IP security protocols apply cryptographic operations to individual datagrams, and they may be active either at the end systems (source and destination) exchanging the datagrams, or else at intermediate routers ("security gateways") that provide cryptographic services on behalf of nearby end systems. Indeed, IPSEC is currently used predominantly at security gateways, and there are large potential advantages of management in doing so. A systematic way to be sure of reaping those benefits is needed. That material is instead available in [13].

We start by considering the security goals that we would like to achieve. From those, we infer a way of modeling the goals (and the systems that should meet those goals) using simple mathematical notions. They in turn suggest algorithms to solve our security management problems.

2.1 Packet Filtering

The purpose of packet filtering is to prevent the delivery of packets that may harm the recipient, or occasionally to prevent the transmission of packets considered inappropriate (e.g. `http` requests to unwholesome servers). The filtering point must typically decide what action to take without examining the payload; only the headers are typically examined by the devices we are concerned with. Thus, aspects of the headers must be used as a clue to which packets ought to be discarded.

2.2 An Example

For instance, ICMP *destination unreachable* packets may be used to map an organization's network, to the extent of determining what IP addresses are in use and which routers are nearby. The attacker sends unsolicited ICMP *destination unreachable* messages with source IP address s into the organization. If the destination address d for that message is not in use, then a router r near the network on which d would lie sends back an outbound ICMP *destination unreachable* packet, with destination s. The source address for this packet is an IP address r for the router itself. In this way, the attacker learns which addresses are not in use (because an ICMP packet is returned); the rest are in use. He also learns which router r serves each missing IP address.

The scheme works because most firewalls permit ICMP *destination unreachable* packets to pass, because they are a useful part of the network infrastructure.

If an organization would like to prevent this sort of probing, then its administrators need to configure their routers (or firewall) to prevent it. They would prefer to allow as many ICMP packets as possible to pass around the network, so long as the attack is prevented. To prevent the attack with the least loss of service, the administrators would like to select a small number of filtering routers. Those routers will be the enforcement points. Each path should traverse an enforcement point, if that path leads from a network area where the attack might be launched to a network area whose contents should not be disclosed.

The enforcement point could discard inbound ICMP probes; indeed, it would suffice to discard inbound probes with destination addresses d that are not in use. The outbound replies are generated only for these addresses, so filtering these probes will ensure that no information is returned. Of course, this finer strategy can be used only if this set is known when the router configuration is chosen. Alternatively, the router could discard the outbound ICMP response, although in this case the packet source address is r, so that we cannot fine-tune the behavior depending on which addresses are in use.

2.3 Types of Information in Our Security Goals

There were two phases to the reasoning we have just described. One was to determine a potential kind of attack, and to infer from it a kind of packet that should be discarded, depending where that packet had come from. In the second phase, we looked for enforcement points, and attempted to assign filtering behavior to each enforcement point with the consequence that every harmful packet's path would be disrupted. We try to do so without disrupting the paths of too many other packets.

The security goal here is to prevent the delivery of certain packets, and the definition of *which* packets involves two different types of information. First, it concerns what the header of the packet says. In our example, we were concerned with the following IP header fields or protocol-specific header fields:

- The protocol header field in the IP header, which must indicate ICMP for this concern to be relevant;
- The ICMP-specific fields for message type and message code, which jointly indicate a *destination unreachable* message;
- The IP destination address, in case we wish to filter only inbound probes destined for addresses not in use.

In other examples, we would be concerned with the IP source address, or other protocols such as TCP or UDP, or with the protocol-specific fields for those protocols, such as source or destination port number, and the TCP `syn` and `ack` bits. All this information is contained in the datagrams, and available to any router inspecting it.

The other kind of information concerns the path that the packet has taken through the network. If the packet has never been on the public Internet, then we do not have to worry whether it is an attempt to map our networks. That is,

assuming that we trust our employees not to attack us, or we acknowledge that they have other ways to obtain mapping information if they intend to carry out an attack.

The path information is not contained in the packet itself (certainly not in any reliable form, given the ways that packets can be spoofed). Indeed, no individual router can observe the whole path, but different routers observe different small pieces of the path. The behavior of a router for a packet it receives is sensitive to which interface the router received it over; and the behavior of a router for a packet to be transmitted is sensitive to which interface it will be transmitted over. The access lists governing filtering are specified on an interface-by-interface basis. Thus, they are sensitive to a portion of the path. The art of achieving the desired network-wide security therefore requires us to do different filtering at different points in the network, thereby disrupting the paths that worry us. A typical example network, containing enough information to represent the paths of concern to us, may take the form shown in Figure 1.

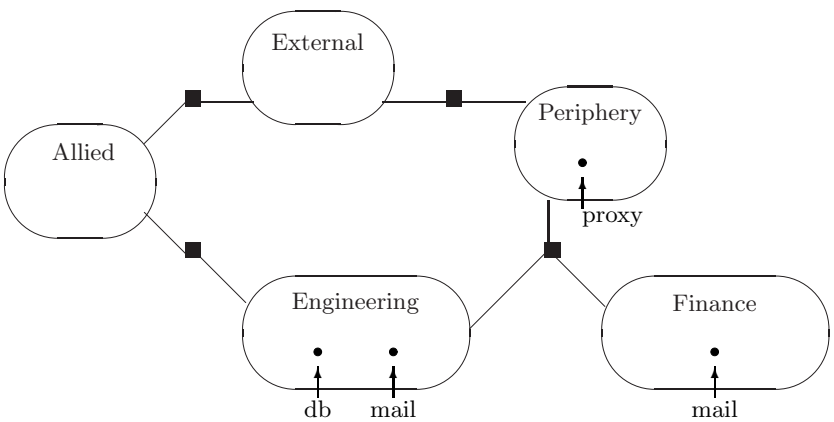


Fig. 1. Corporate protection example

The task of deciding on a security policy is simplified by the fact that information we care about is a limited portion of the path information: We care only whether the datagram was previously in a network region we do not trust, and whether the datagram has reached a region we would like to protect. A firewall policy can also enforce a confidentiality policy, thereby restricting the flow of packets from a vulnerable (internal) area to an untrusted (external) area. The strategy of filtering the outbound return ICMP packets in our example is a case of enforcing confidentiality (with regard to network information) via a firewall. Naturally, more inclusive kinds of confidentiality are likely to be harder to enforce reliably using a firewall.

2.4 The Logical Form of Our Security Goals

From our intuitive analysis, we can now stipulate the logical form of the security goals we would like to achieve. They integrate information about path with information about header in a specific way. A security goal always concerns two network areas a and a', with one (say, a) traversed earlier and the other a' traversed later. If a packet has followed any path from a to a', then its header fields should have some property φ. Thus, a "policy statement" for pair of areas a, a' is a statement of the form:

If p was in a and later reaches a',
then p's header fields satisfy φ.

Thus, for our firewall analysis, we have identified a class of security goals. Each goal describes a class of paths—those that traverse the areas a and a' in that order—and says which packets, defined in terms of header fields, are permitted to travel along those paths in the network. These firewall security goals are characterized by this logical form, namely an implication in which the antecedent mentions two areas in a particular order, and the consequent gives a constraint on the packets traversing those areas.

A *security policy*, in this context, will mean a security goal for every pair of areas. Given a graph in which the areas belong to a set A, a policy is a function $\Pi\colon A \times A \to B$, where B is some suitable collection of sets of packets.

In other contexts, other security goals will be needed. For instance, the security goals we can achieve using IPsec, as have discussed in [13], may take different logical forms or require different real world ingredients to be modeled.

2.5 Some Security Goals

We have already mentioned two candidate security goals, each a possible way to prevent the ICMP *destination unreachable* network mapping attack. First, we may prevent the attack by pruning the incoming probe. We assume that the attack is possible only if the probe packet has passed through the External area (in the terminology of Figure 1), and we assume that we are concerned only about an attacker mapping our internal networks, labelled Engineering and Financial. The Periphery network, being more exposed, may well be vulnerable to mapping in any case. Thus, one security goal would be:

If p was in External and later reaches Engineering,
then p's header fields satisfy φ,

where φ stipulates

Either $p.destination \notin$ Engineering,
or $p.protocol \neq$ ICMP,
or $p.message\_type \neq$ *destination unreachable*.

Another example security goal in this network (assuming that most TCP connections are proxied at the application level on the Proxy host on the Periphery network) may be to ensure that no TCP packet travels from External to Engineering unless it is an SMTP packet, and its destination is the email server Eng_mail. In this case, the antecedent of the policy statement is unchanged, since we are still concerned with what packets reach Engineering from External; the choice of φ is then:

$$\begin{array}{ll} & p.\mathit{destination} = \text{Eng\_mail}, \\ \textbf{and} & p.\mathit{protocol} = \text{TCP}, \\ \textbf{and} & p.\mathit{dest\_port} = 25. \end{array}$$

2.6 Security Modeling

Our analysis of the security goals we would like to achieve also suggests some modeling decisions. We must model the paths of packets as they pass from a network area across an interface to a router and then across another interface to some other network area. Thus, we may regard the system as forming a graph. The edges represent the interfaces, and since packets may flow in either direction over an interface we may regard the edges as undirected. The nodes are of two different kinds, namely the router at one end of an edge and the network area at the other end. Thus, we represent the system by a bipartite graph with undirected edges. A diagram like Figure 1 summarizes this view of the topology of the system.

Now, at each edge, for each direction, we have a filter that will be used to discard some packets that would otherwise flow in that direction. The filter divides all packets into two sets, those that are permitted to flow over the interface in that direction and those that will be discarded. Since different characteristics of packet headers will be used to make these decisions, but presumably not every set of packets will be relevant, we will regard the relevant sets of packets as forming a boolean algebra, always a vastly smaller boolean algebra than the boolean algebra of all sets of packet headers.

Since there are thirty-two bits of source address, thirty-two bits of destination address, and eight bits for selecting protocols, there are at least 2^{72} different IP headers, so at least $2^{2^{72}}$ different sets of IP headers. For ICMP there are sixteen bits of protocol specific message type and message code; for TCP there are sixteen bits of source port and sixteen bits of destination ports, and likewise for UDP. Thus, we have many more sets considering these protocol specific header fields. Naturally, many of these sets are ridiculously ragged, and play no role in any meaningful network protection. The "practically useful" sets are comparatively few and far between, and have smoother edges.

In our example above, there are just a few distinctions we need among IP addresses. The special hosts have IP addresses that must be distinguished from all others, since filters will need to permit packets through specially, if those hosts are source or destination. Then the workstations in a particular network area may need to be distinguished from those in any other area. Finally, for our

ICMP example, we might want to distinguish the unused IP addresses in the three internal corporate networks. This leads to a total of twelve relevantly different IP addresses, far less than the 2^{32} different addresses a priori. There is a big advantage to taking a special-purpose representation of the boolean algebra of relevant packet sets, rather than using a representation that could express all the addresses and all the sets of addresses, and all the sets of packet headers.

Thus, our modeling process leads us to two ideas, the idea of a undirected bipartite graph to represent the network, and the idea of a boolean algebra to represent the constraints, both the sets permitted at the various filtering points and also the φs used in the consequents of security goals. Let us see now what problems we can solve using these two modeling ideas.

2.7 Localization

The core issue with our packet filtering security policies concerns localization. Although our security properties are global properties about all paths from a to a', even when these areas are distant from each other, our enforcement mechanism consists of routers each of which must make its own decision. It has only local information about what interface a packet has been received over, and what interface it will be transmitted over. Localization is the problem of achieving global security goals using this local information.

Localization may be used in two different ways. First, we may know what security goals we want to achieve, and also the filtering that will be applied on each interface to packets flowing in each direction. Then the problem is to decide whether each security goal is enforced by those filters. When they are not satisfied, one would like to exhibit the undesirable packets and the paths by which they may reach their targets.

Alternatively, we may decide on the security goals, and wish to discover an assignment of filtering to interfaces that will achieve them. Each of these two problems is a matter of localization, because the problem is to interrelate security policies stated in terms of network-wide paths, with an enforcement mechanism that must act locally at specific interfaces.

We will use the word *posture* to mean a specification of filtering behavior. Filtering routers allow a different set of packets to be filtered at each interface, and in each direction of flow, into the router or out of the router. Thus, a filtering posture will mean an assignment of a filter to each interface/direction pair. A filter just means a set of packets, namely those permitted to traverse the interface in that direction. The sets of packets are the members of some boolean algebra that will be far smaller than the boolean algebra of all sets of packet headers.

2.8 Two Algorithms

We have mentioned two problems, namely

1. Given a security policy and a filtering posture, to determine whether the posture faithfully enforces the policy, and if not, to enumerate the counterexamples; and
2. Given a security policy, to construct a filtering posture guaranteed to enforce the policy.

We will describe an algorithm to resolve each problem, for which we need some auxiliary notions. Let $p = \langle \ell_0, \ldots, \ell_n \rangle$ be a path p. Here, each ℓ_i is either an area or a router in a bipartite graph such as the one in Figure 1, and p is a path only if ℓ_i and ℓ_{i+1} are adjacent for each i from 0 to $n-1$. We will write p_i for the i^{th} entry in p; we will write $|p|$ for the length of p; and we will write $p_{|p|}$ for the last entry in p. We use $p^\frown p'$ to mean the concatenation of p with p', which is well-defined only if $p_{|p|} = p'_0$, and is the path consisting of p followed by p', omitting the second occurrence of the common location at the end of p and beginning of p'.

We also assume to simplify notation that each router has at most one interface on any area. Thus, we regard a filtering posture as a function $f : R \times A \times D \to B$, where

R is the set of routers;
A is the set of areas;
D is the set containing the two directions *in* and *out*, meaning into the router and out of the router, respectively; and
B is a boolean algebra of sets of packets.

We adopt the convention that $f(r, a, d) = \emptyset$ when r lacks an interface on a. Let $\phi(\ell_i, \ell_j) = f(\ell_i, \ell_j, out)$ when ℓ_i is a router and ℓ_j is a network area, and $f(\ell_j, \ell_i, in)$ otherwise. Thus, in either case it is the set of packets allowed to traverse the interface from ℓ_i to ℓ_j.

We define $\mathcal{F}(p)$, the feasibility set for p, inductively:

1. $\mathcal{F}(\langle\rangle) = \mathcal{F}(\langle \ell_0 \rangle) = \top$, the top member of the boolean algebra;
2. $\mathcal{F}(p^\frown \langle \ell_i, \ell_{i+1} \rangle) = \mathcal{F}(p) \cap \phi(\ell_i, \ell_{i+1})$.

Thus, the feasibility set for a path p is the set of packets that survive all of the filters that are traversed while following p. Observe that if the range of a filtering posture f is in the boolean algebra B, then $\mathcal{F}(p)$ is also in B, for any p. Also, by the laws of boolean algebra, $\mathcal{F}(p^\frown p') = \mathcal{F}(p) \cap \mathcal{F}(p')$.

2.9 Posture Checking

Suppose given a posture f and a security policy $\Pi : A \times A \to B$, we would like to know whether f enforces Π. What does this mean?

It means that whatever path a packet may take from area a_0 to a_n, if that packet survives every filter it traverses, then it satisfies the policy constraint $\mathcal{F}(p) \subset \Pi(p_0, p_{|p|})$.

Note also that we may ignore any cyclic path $p = \langle \ell_0, \ldots, \ell_i, \ldots, \ell_i, \ldots \ell_n \rangle$, because

$$\begin{aligned}\mathcal{F}(p) &= \mathcal{F}(\langle \ell_0, \ldots, \ell_i \rangle) \cap \mathcal{F}(\langle \ell_i, \ldots, \ell_i \rangle) \cap \mathcal{F}(\langle \ell_i, \ldots, \ell_n \rangle) \\ &\subset \mathcal{F}(\langle \ell_0, \ldots, \ell_i \rangle) \cap \mathcal{F}(\langle \ell_i, \ldots, \ell_n \rangle) \\ &= \mathcal{F}(\langle \ell_0, \ldots, \ell_i \rangle^\frown \langle \ell_i, \ldots, \ell_n \rangle)\end{aligned}$$

So p cannot cause more violations than its cycle-free part $\langle \ell_0, ..., \ell_i \rangle^\frown \langle \ell_i, ..., \ell_n \rangle$.

Therefore, to find all violations, we need only enumerate, starting at each area a, each cycle-free path p leading to another area $a' = p_{|p|}$. We test $\mathcal{F}(p) \subset \Pi(a, a')$. If this is false, we report as violations all packets in the set-difference $\mathcal{F}(p) \setminus \Pi(a, a')$, noting that the path p permits them to pass. If no cycle-free path starting at any area a leads to violations, then f correctly enforces Π.

Doubtless, this algorithm could be made more efficient in several ways were it necessary to handle large graphs.

2.10 Posture Generation

Suppose next that we have a security policy Π that we want to enforce, and the task is to construct a filtering posture f that does so. The idea of this algorithm is this: We start with a simple but useful initial posture f_0, and we correct it successively to eliminate all violations.

The correction process enumerates all cycle-free paths p that start and end at distinct areas a_0 and a_n. The next-to-last entry in p is a router r, because the underlying graph is bipartite. Using the current approximation f_i, we calculate $\mathcal{F}(p)$. If $\mathcal{F}(p) \subset \Pi(a_0, a_n)$, then $f_{i+1} = f_i$. Otherwise, we correct the violations V, which are the packets in the set difference $V = \mathcal{F}(p) \setminus \Pi(a_0, a_n)$, those packets that may feasibly traverse the path p but are not permissible. The corrected posture f_{i+1} differs from f_i only for the arguments (r, a_n, out), where we have $f_{i+1}(r, a_n, out) = f_i(r, a_n, out) \setminus V$. Observe that this algorithm modifies only the outbound filters, not the inbound ones.

There are different strategies for constructing the initial posture f_0. A useful choice is the following.

1. $f_0(r, a_n, out) = \top$, the top member of the boolean algebra;
2. For any datagram δ, $\delta \in f_0(r, a, in)$ if $\delta.src \in a$, or else if, every a' adjacent to r, $\delta.src \notin a'$.

This choice of f_0 uses the inbound filters to protect against spoofing: If a datagram δ claims a source in some adjacent area a', but is observed entering r from some different adjacent area a, then δ must be lying. If it were really from a', it would reach r off its interface on area a'. This reasoning assumes that the packet has not been routed along on odd path because some network resources are currently down, but it matches an assumption that security-attuned network administrators frequently want to make.

This algorithm is not guaranteed to be ideal. The correction process discards packets only at the last step, immediately before they would finally cause a violation. It may be preferable instead, in a particular case, to discard the packet earlier, as it embarks on a path that will eventually lead to violations. Similarly, the choice of an anti-spoofing initial posture f_0 may not be ideal. For instance, a packet arriving at the External/Periphery router in Figure 1, inbound on its External interface, which claims to be from Finance, is doubtless lying, but our initial posture f_0 does not discard it. The reason is that Finance is not adjacent to this router. We know something about the way that packets from Finance should travel, but it would be hard to generalize that knowledge and incorporate it into our algorithm.

A reasonable implementation, such as the one described in [11] or its successor, the current Network Policy Tool (NPT) software available from the author, will combine this posture generation algorithm with the posture checking algorithm described previously. Then a human user can edit the posture output by the generation algorithm to make local improvements. The checking algorithm may then be used to ensure that these "improvements" have not in fact introduced violations. Alternatively, to understand why NPT has done something apparently unnecessarily complex, it is interesting to edit the output to the simpler, expected form. The checking algorithm will then show what would have gone wrong.

If one is given a posture f, it is also possible to calculate a security policy that describes what f permits; it is the policy Π_f inferred from f. It is defined by:

$$\Pi_f(a, a') = \bigcup_p \mathcal{F}(p)$$

where the union is taken over all p of the form $p = \langle a, \ldots, a' \rangle$.

2.11 Implementing Boolean Algebras

Any such piece of software needs to select a suitable data structure to represent the members of the boolean algebra. Different possibilities may be tried, although it is natural to regard each packet as a triple, consisting of a source IP address, a destination IP address, and "the rest."

The rest consists of information about the network service that the packet provides. It includes the packet's protocol field (possible values being TCP, UDP, ICMP, and so on), together with a variety of protocol specific fields. For instance, for UDP and TCP, the source port and destination port are important. If the destination port is a well-known value like 25, then this packet is headed toward a server (in this case an SMTP server); if its source port is 25, then it is headed from an SMTP server to a client.

For TCP, so are the `syn` and `ack` bits that say whether the packet belongs to the set-up phase of a connection, or to a connection that has already been negotiated. For ICMP, the message type and code are relevant.

Since each packet is characterized by the three dimensions of source, destination, and service, a boolean algebra of sets of packets may be regarded as consisting of regions of a three dimensional space. Although it is a large space, we are in fact interested only in rather coarse regions. For instance, looking at a diagram such as Figure 1, we can see that the number of relevantly different addresses is quite low, nine in fact:

- There are four specific systems mentioned, that will need special treatment in defining policy and implementing a suitable posture. They are `engineering db`, `engineering mail`, `finance mail`, and `periphery proxy`.
- Hosts other than these are treated uniformly according to the area in which they are located. Thus, we have `engineering other`, `finance other`, `periphery other`, `external any`, and `allied any`.

We also call these "relevantly different" addresses *abstract addresses*.

A similar consideration of the protocols and services that the network supports, and of the policy for permitting them from area to area, might lead to a short list of relevantly different services, for instance,

- ICMP, distinguishing *destination unreachable* messages from all others;
- TCP, distinguishing `ftp`, `smtp`, and `http` by their well-known ports;
- UDP, distinguishing messages to a particular port on the `engineering db` machine, on which a data base server is listening;
- all other protocols.

In addition, for the distinguished TCP and UDP services, one will want to treat packets differently depending whether the packet is traveling from client to server or *vice versa*; this doubles the number of distinguished TCP and UDP services. There is one additional service grouping together all others, the "undistinguished" services. On our example, we have thirteen relevantly different services:

- Two ICMP services;
- Seven $(= 3 \cdot 2 + 1)$ TCP services;
- Three $(= 1 \cdot 2 + 1)$ UDP services;
- One for the others.

We also call these relevantly different services *abstract services*.

All in all, there are $9 \cdot 9 \cdot 13 = 1053$ relevantly different packets in this example, leading to 2^{1053} sets in the boolean algebra. Naturally, far fewer sets will come up in any relevant computation. Observe that these "relevantly different packets," called "abstract packets" in [11], are the *atoms* of the boolean algebra, in the sense that any two atoms have a null intersection, and any set in the algebra is a union of atoms. Thus, an abstract packet is a triple, consisting of two abstract addresses (source and destination) together with an abstract service. Examples of these abstract packets, shown as triples of abstract source, destination, and service, would be:

```
(external any,  engineering any,  ICMP dest unreachable)
(engineering db,  allied any,  UDP db from server)
(engineering mail,  external any, TCP smtp to server)
```

Each of these represents a set of possible concrete (real) packets, namely all those with IP addresses matching the source and destination, and protocol specific header fields matching the service. It thus represents a cube in the three dimensional space of real datagrams, being the intersection of the sets of packets matching each of the three dimensions individually.[1] There are different ways to represent the algebra with these atoms. In NPT, we represent any set as a list of cubes, although we treat the source and destination dimensions specially. The lists are maintained in a form such that no source destination pair lies in the projection of two different cubes. For this reason, we described the lists as lists of colored rectangles, where the source-destination rectangles were always disjoint, and the permissible services for the packets with those sources and destinations were regarded as a coloring.

Wool and his colleagues [34] follow [11] in representing sets of packets as lists of disjoint colored rectangles, although in their work the relevantly different sources and destinations are not inferred at the start. Instead, they are constructed by examining real configuration files, and splitting IP addresses into ranges according to the access list lines contained in the configuration files.

One might alternatively dispense with lists of rectangles and instead represent the sets more directly, for instance using Binary Decision Diagrams (BDDs) [3]. In our next section, we will instead consider how BDDs can be used as an auxiliary structure to discover the set of atoms that naturally describe existing configuration files.

2.12 The Abstraction Problem

We have described how to use a boolean algebra in which the atoms are abstract packets as a way to represent problems about distributed packet filtering. But, how can we construct a boolean algebra that is faithful to the distinctions made in a particular set of configuration files? That is, we would like to take the configuration files for a given set of filtering routers, and deduce from them a set of abstract addresses and abstract services, such that the filtering posture determined by these configuration files can be represented in the resulting three dimensional boolean algebra. Indeed, we would prefer to choose the coarsest such boolean algebra, so that our abstract addresses and services make the minimum number of distinctions compatible with the files themselves. This is the problem addressed in [12], for which the binary decision diagram is an ideal data structure.

An abstract address is a set s of concrete IP addresses such that $i, j \in s$ implies that replacing i by j as the source or destination of a packet never transforms a packet rejected by a filter into a packet accepted by it, or *vice versa*. A set of IP addresses is an atom for a filtering posture is this holds for all of the filters specified by its configuration files.

[1] If there were some natural sense of adjacency for the points on the axes, this would amount to a finite union of cubes. Since we do not recognize any natural notion of adjacency, we collect together the matching intervals along any one axis, and regard the region as a single cube.

The problem is the more pressing because the router configurations of real organizations evolve over time as hosts and routers are added to the network or removed; as users clamor for additional services or administrators worry about new attacks; and as staff comes and goes with an increasingly hazy understanding of the contributions of their predecessors. Indeed, their decisions are typically documented only in the configuration file itself, with its low-level, procedural notation and inflexible syntax.

From our point of view, a configuration file such as for a Cisco router, contains *interface declarations* and *access lists*. An interface declaration may specify a particular access list to apply to packets arriving inbound over the interface or being transmitted outbound over the interface.

An access list is a list of lines. Each line specifies that certain matching packets should be accepted ("permitted") or discarded ("denied"). When a packet traverses the interface in the appropriate direction, the router examines each line in turn. If the first line that matches is a "deny" line, then the packet is discarded. If the first line that matches is a "permit" line, then the packet is permitted to pass. If no line matches, then the default action (with Cisco routers) is to discard the packet.

For instance, the lines in Figure 2 permit two hosts (at IP addresses 129.83.10.1 and 11.1) to talk to the network 129.83.114.∗. They also permit the other hosts on the networks 129.83.10.∗ and 129.83.11.∗ to talk to the network 129.83.115.∗. The asterisks are expressed using a netmask 0.0.0.255, meaning that the last octet is a wildcard.

```
!   (comments start with !)
!
! keyword   num action prot source                 destination
access-list 101 permit ip   host 129.83.10.1       129.83.114.0 0.0.0.255
access-list 101 permit ip   host 129.83.11.1       129.83.114.0 0.0.0.255
access-list 101 deny ip     host 129.83.10.1       any
access-list 101 deny ip     host 129.83.11.1       any
access-list 101 permit ip   129.83.10.0 0.0.0.255  129.83.115.0 0.0.0.255
access-list 101 permit ip   129.83.11.0 0.0.0.255  129.83.115.0 0.0.0.255
```

Fig. 2. A cisco-style access list

2.13 The Logic of Access Lists

Each line of an access list defines a set of sources φ_s, destinations φ_d, and service characteristics φ_v, and stipulates whether matching packets should be discarded or passed. A datagram δ matches a line if $\delta.src \in \varphi_s \wedge \delta.dst \in \varphi_d \wedge \delta.svc \in \varphi_v$.

At any stage in processing, a packet that has not yet been accepted or rejected is tested against the first remaining line of the list. If the line is a "permit" line,

the packet has two chances to be permitted: it may match the specification for the first line, or it may be permitted somehow later in the list. If the line is a "deny" line, the packet has to meet two tests to be permitted: it must not match the specification for the first line, and it must be permitted somehow later in the list. Since the default is to deny packets, the empty list corresponds to the null set of permissible packets. Thus, we have a recursive function η of the access list:

$$\begin{array}{lcl} \eta([\,]) & = & \emptyset \\ \eta((\text{permit}, \varphi_s, \varphi_d, \varphi_v) :: r) & = & (\varphi_s \cap \varphi_d \cap \varphi_v) \ \cup \ \eta(r) \\ \eta((\text{deny}, \varphi_s, \varphi_d, \varphi_v) :: r) & = & \eta(r) \ \setminus \ (\varphi_s \cap \varphi_d \cap \varphi_v) \end{array}$$

The function η allows us to transform a parser for the individual configuration file lines (emitting sets describing the matching conditions) into a parser that emits a set describing the meaning of the whole access list.

2.14 Binary Decision Diagrams

Again we face the question how to implement the boolean algebra of sets in this context, and for our current purpose of solving the abstraction problem, the binary decision diagram [3] fits our needs perfectly.

A binary decision diagram (BDD) is a finite directed acyclic graph with two sinks, labelled *true* and *false*. Each interior node has out-degree two, and is labeled with a propositional (bit-valued) variable. One out-arrow is associated with the variable taking the value *true* and the other with the variable taking the value *false*. One node is distinguished as the root.

A BDD n represents a propositional function; that is, it yields a truth value as a function of a number of truth-valued variables: to evaluate it for an assignment of values to the variables, we start at the root and follow a path determined as follows:

1. If we have reached a sink, its label is the result.
2. If the current node is labeled with variable v, and the assignment associates $v \mapsto b$, then we traverse the arrow associated with the value b.

Observe that any node n' accessible from a BDD n is itself a BDD. If α is an assignment of truth values to the variables, then we write $\alpha(n)$ for the truth value that results from evaluating n at α, i.e. traversing n as described above.

An example is shown in Figure 3, page 214. Each node represents a choice for the truth value of the variable shown at the right margin. The line marked + represents the result if that variable is true; the other line represents the result if that variable is false.

A BDD n is *ordered* if there exists an ordering $\prec$ on the variables such that $v \prec v'$ if any traversal starting at n encounters v before v'. A BDD n is *reduced* if, whenever n_0 and n_1 are accessible from n and $n_0 \neq n_1$, then there is some variable assignment α such that $\alpha(n_0) \neq \alpha(n_1)$. Thus, a BDD is reduced if its nodes obey the extensionality principle that different nodes represent different

propositional functions. Algorithms for maintaining BDDs in ordered, reduced form are found in [2].

In our case, we are interested in using BDDs to represent sets of packets. Since a packet is determined by source address, destination address, and protocol-specific information, the propositional variables are the thirty-two bits of the source address (for IP Version 4), the thirty-two bits of the destination address, and a number of bits representing the protocol-specific information (sixty-four bits is enough). Thus, any protocol header may be summarized, to the extent we need to model how filter routers treat it, as a sequence of 128 bits. Any filter (set of packets) may therefore be represented as a reduced, ordered binary decision diagram where each internal node is labeled with one of these 128 propositional variables.

An example BDD is shown in Figure 3; in this diagram, we have assumed that IP addresses are only three bits, and that protocol-specific information may be summarized in four bits. The nodes are thus labeled s_1, s_2, s_3, d_1, d_2, d_3, and p_1, p_2, p_3, p_4. The variables are grouped into three sections representing source information, destination information, and protocol information.

2.15 Finding Atoms in BDDs

Suppose that we are given a BDD like the one in Figure 3, in which the variables have been divided into sections. We would like to identify the sets of values in each section that are treated as atoms. For instance, we will identify the sets of IP addresses that are treated the same as packet source addresses, so that if two addresses i, j belong to the same atom, then a packet with source address i and the identical packet with source address j will necessarily receive the same treatment from the packet filter. Two concrete addresses belong to the same abstract addresses if they are treated the same as sources and also as destinations.

To find the source atoms, we will enumerate the *outcomes*, meaning those nodes that are not labeled by source variables, but lie at the end of an arc starting at a node labeled by a source variable. In our example (Figure 3), one of these nodes is enclosed in a small box. If two paths lead to the same outcome, then those paths can make no difference: in combination with any assignment to the remaining variables, either both paths evaluate to *true* or else both paths evaluate to *false*. For instance, let p_1 be the path that assigns T to s_1 and s_2, and F to s_3; let p_2 be the path that assigns F to s_1 and s_2, and T to s_3. Then both p_1 and p_2 lead to the boxed outcome. Therefore they must belong to the same source outcome.

On the other hand, if two paths through the source variables lead to different outcomes, then because the BDD is in reduced form, there must be at least one assignment to the remaining variables that leads to different outcomes. Therefore, the paths do not belong to the same source atom.

This way of thinking suggests some definitions and an algorithm. First, let us call a sequence of variable/value pairs a *path* if the variables occur in an order compatible with $\prec$. Some examples of paths involving only source variables are:

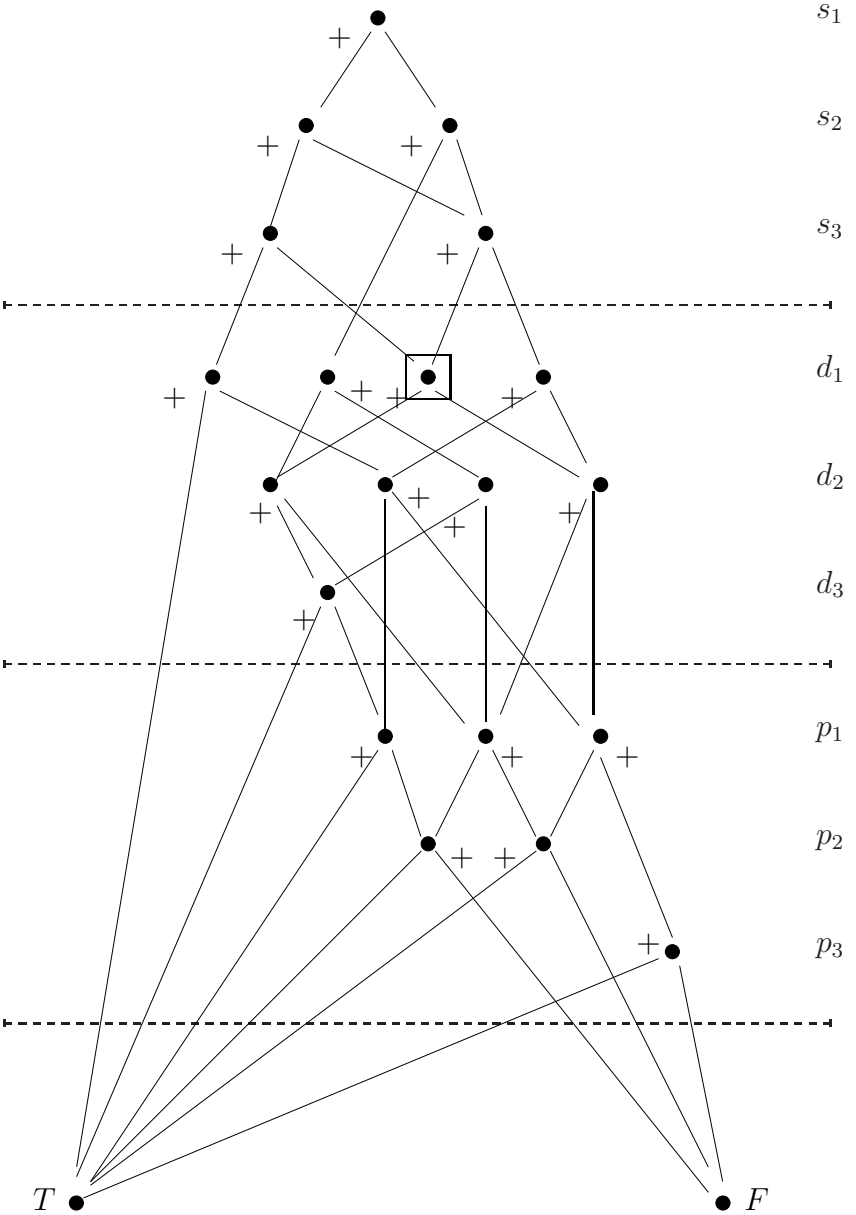


Fig. 3. An example BDD

$$(s_1, \mathit{true}), \quad (s_2, \mathit{true}), \quad (s_3, \mathit{false})$$
$$(s_1, \mathit{false}), \quad (s_2, \mathit{false}), \quad (s_3, \mathit{true})$$
$$(s_1, \mathit{false}), \quad (s_3, \mathit{true})$$

We say that a path p leads from n_0 to n_k, and write $n_0 \xrightarrow{p} n_k$, if there is a sequence of arcs leading from n_0 to n_k, and each node n_i for $i < k$ is labeled by a variable v_i mentioned in p, and the arc from n_i to n_{i+1} is labeled by the truth value paired with v_i in p. There may be several paths leading from n_0 to n_k. On the other hand, a particular path may not lead anywhere in a given BDD: the third path just mentioned does not lead anywhere in the BDD of Figure 3. In that BDD, each path involving s_1 and s_3 needs also to specify a value for s_2.

We may interpret each path as a formula in the obvious way, namely as a conjunction in which each variable in the path occurs negated if the associated value is *false* and unnegated if it is *true*. For convenience, we write the interpretation of a path p as $[\![p]\!]$. With this notation, we have the following interpretations for our example paths:

$$[\![(s_1, \mathit{true}), \quad (s_2, \mathit{true}), \quad (s_3, \mathit{false})]\!] \quad = \quad s_1 \wedge s_2 \wedge \neg s_3$$
$$[\![(s_1, \mathit{false}), \quad (s_2, \mathit{false}), \quad (s_3, \mathit{true})]\!] \quad = \quad \neg s_1 \wedge \neg s_2 \wedge s_3$$
$$[\![(s_1, \mathit{false}), \quad (s_3, \mathit{true})]\!] \quad = \quad \neg s_1 \wedge s_3$$

If s is a set of paths, we use $[\![s]\!]$ to mean the disjunction $\bigvee_{p \in s} [\![p]\!]$. Thus, for instance, if s contains the three paths mentioned above, then

$$[\![s]\!] \; = \; (s_1 \wedge s_2 \wedge \neg s_3) \vee (\neg s_2 \wedge \neg s_3) \vee (\neg s_1 \wedge s_3)$$

We also say that m is an *outcome* for n if for some path p, $n \xrightarrow{p} m$, and m does not lie in the same section as n, but if p' is any proper subpath of p and $n \xrightarrow{p'} n'$, then n' lies in the same section as n.

We can now see that each atom rooted at n_0 is of the form $[\![\{p\colon n \xrightarrow{p} m\}]\!]$ where m is an outcome for n. For if p, p' are two paths such that $n \xrightarrow{p} m$ and $n \xrightarrow{p'} m$, then they surely belong to the same atom. Since they re-join at m, no assignment to the remaining variables can cause p and p' to lead to different results. Moreover, if two paths lead to different outcomes, then by the extensionality principle for reduced BDDs, there must be some assignment to the remaining variables that separates them.

2.16 An Algorithm for Finding Atoms

The analysis we have just carried out motivates an algorithm for finding atoms in a BDD that has been divided into sections like Figure 3, where the sections consist of the source address variables, the destination address variables, and the protocol-specific variables. The two sinks lie below all three sections.

The algorithm maintains a hash table h that associates each outcome with the paths that lead to it. The algorithm has two phases. Phase one traverses

the BDD rooted at n_0, and phase two traverses the hash table. In phase one, we recursively descend the BDD. If the node n at the end of the current path p is an outcome for n_0, then:

If h contains an entry s for n, then the new entry for n in h is $\{p\} \cup s$;
Otherwise the initial entry for n in h is the singleton $\{p\}$.

If n is not yet an outcome, recursively consider both extensions of the path p.

In phase two, we walk the hash table h. Each entry associates an outcome node n with the set s of paths leading from n_0 to n. Each entry determines an atom with value $[\![s]\!]$. Since every path from n_0 must eventually leave the section, each path reaches some outcome, which entails that the union of all the atoms is exhaustive. The atoms are mutually exclusive because any path reaches at most one outcome. Thus, we have derived a partition of the set of source addresses.

As described, this algorithm discovers the source atoms, those sets of source addresses that are treated the same by a single filter, when appearing as IP packet sources. The destination atoms may be calculated in the same way, starting from each outcome n that arose in finding the source atoms. Each outcome n leads to a list of destination atoms, containing those destination addresses that are treated the same starting from n. The algorithm is executed once starting for each source outcome n, each time yielding some destination outcomes. Finally, service atoms may be discovered starting from each of these destination-atom outcomes; in calculating the service atoms, the remaining outcomes can only be the two sinks *true* and *false*.

2.17 Source and Destination Addresses, Multiple Filters

So far, we have computed a partition of the set of source addresses; a family of partitions of the set of destination addresses, starting from various source-atom outcomes; and a similar family of partitions of services. This calculation analyzes a single filter. Several filters may be defined for different interfaces (and opposite directions of traversal) within the same configuration file, and several configuration files for different routers may need to be analyzed. How do we put all of these pieces together? Let us concentrate on constructing the right abstract addresses, the treatment of abstract services being essentially the same.

In essence, we have a number of partitions $\mathcal{F}_i$ of the IP addresses. Each partition is a congruence with respect to one condition, such as the source addresses of packets passing through a particular filter, or the destination addresses, assuming that the source led to a particular outcome. We call each of these families a *family of pre-atoms*. We want to construct a single partition of the IP addresses which is a congruence with respect to all of the conditions. It will thus be the coarsest common refinement of the $\mathcal{F}_i$.

To do so we must split the pre-atoms of $\mathcal{F}_i$ wherever the pre-atoms of $\mathcal{F}_j$ overlap with them. Let $s_i \in \mathcal{F}_i$ and $s_j \in \mathcal{F}_j$, and define superimpose(s_i, s_j) to be either the tag None or the tag Some applied to a triple of sets:

– None, if $s_i \cap s_j = \emptyset$;

– Some$((s_i \cap s_j),\ (s_i \setminus s_j),\ (s_j \setminus s_i))$, otherwise.

To add a single pre-atom s_i to a list f representing a family of pre-atoms we recursively apply the following procedure:

– If f is the empty list [ ], then the result is the singleton list $[s_i]$.
– Otherwise, f is of the form $s_j :: rest$. If the result of superimposing s_i on s_j is None, recursively add s_i to *rest*, obtaining f', and return $s_j :: f'$.
– If the result of superimposing s_i on s_j is Some(c, s'_i, s'_j), then recursively add s'_i to *rest*, obtaining f', and return $c :: s'_j :: f'$.

Finally, if we have two families, represented as lists f_1 and f_2, then to combine them:

– If f_1 is the empty list [ ], then return f_2.
– Otherwise, f_1 is of the form $s_1 :: rest$. Recursively, combine *rest* with f_2, and then add the single pre-atom s_1 to the result (as defined above).

2.18 Atomizer Results

The Atomizer has been implemented as a rather large program written in OCaml, an implementation of ML developed at INRIA/Rocquencourt [25]. When run against a set of three unusually large configuration files in actual use, containing 1380 lines of access lists, it constructs about a hundred atoms. Computation takes 29 seconds on a 550MHz Pentium III with 700MB of store. The maximum process size is 58MB of store. The bulk of the space seems to be devoted to storing the BDDs, so that re-implementing that data structure in C (which is accessible from OCaml) would probably reduce the space used significantly.

When run against even larger but artificially generated configuration files, containing 5,575 lines of access lists, it completes in 20.5 seconds, having occupied 45MB of store.

The Atomizer generates abstract addresses and abstract services to be used in NPT, thus providing a method for analyzing actual router configurations to discover the security goals that they jointly achieve in use.

2.19 Packet Trajectories and Security Management

In this section, we have modeled the trajectories that packets may take through networks. We regard the networks as bipartite graphs, in which a node is either a router (used as an enforcement point) or a network area. Edges represent a router's interface on a network area. The packets that can survive all of the filters on a particular path form the *feasibility set* for that path. Given an implementation for boolean algebras and their fundamental operations, we can calculate feasibility sets, and use them to answer questions. We described an algorithm to determine whether a given posture meets a security policy. Another algorithm constructs a posture that will meet a security policy. We also described an algorithm that uses binary decision diagrams to determine (from a number of

actual configuration files) the atoms that will be needed in the boolean algebra for these computations.

Thus, this approach to modeling allows us to answer a number of different *security management* questions about our networks. In the introduction, we enumerated some crucial security management questions. Let us return to that list, and comment on the extent to which our methods, as described in this section, give answers to them, in the specific context of packet filtering.

1. Does a given system meet a particular security goal?
   If the system is described in terms of the abstract packets passed or discarded at the different filtering points, then NPT answers this question.
2. Given a system, what security goals does it meet?
   We use the Atomizer to construct a suitable boolean algebra and representations of the filters. Then, given any two areas a, a', we can calculate the union of the feasibility sets $\bigcup_p \mathcal{F}(p)$ for all p such that $p_0 = a$ and $p_{|p|} = a'$. This union is the most inclusive security policy enforced by these filters.
3. Given a security goal, how can we configure (or modify) a system to meet it?
   We have described how to use NPT to calculate a posture to enforce the policy. We have not, however, described an algorithm to construct a (procedural, Cisco-style) configuration file from a list-of-rectangles specification for a filter, or from a BDD representing it, although work has been done in this direction.
4. Given a real-world system, how can we construct the abstraction that models it?
   Network mapping tools may be used to construct the bipartite graph for the system, after which the Atomizer designs the necessary abstractions for the filters.
5. Given a real-world system, what automated tests (or manual analysis) will check whether a given abstraction models it faithfully? Whether a given security goal has been violated?
   Our methods can be used to discover what packets are expected to emerge from a given interface; by sniffing the network and sampling the headers, we can raise an alarm if an unexpected packet is found. See also [22], in which specifications are used to generate test cases systematically.
6. If two given systems each meet a security goal, does each continue to meet that security goal if they are combined in a particular way?
   NPT can be used to determine the security consequence of adding direct network connections between previously distant areas.

Thus, we have illustrated in some breadth how to specify and reason about net-wide security policies for filtering routers, providing an instance of the rigorous approach to security management.

3 Strand Spaces and Protocol Security Goals

In the remainder of this series of lectures, we will discuss cryptographic protocols. We would like to define a particular type of failure, and the class of security goals

that rule out these failures. We will explain our modeling for cryptographic protocols and their security goals in this section, and illustrate how to use the modeling ideas to detect flaws in protocols that have them. In Section 4 we will give a more rigorous treatment, leading to a simple method for proving that keys are not disclosed. Section 5 focuses on how to prove authentication goals. Finally, in Section 6, we will explain how to determine whether mixing two protocols will cause interactions that will undermine their (separate) security guarantees.

3.1 What is a Cryptographic Protocol?

A cryptographic protocol is a short, convention-bound sequence of messages. It uses cryptography to aim at security services such as authentication and key distribution (or key agreement) for session keys.

Despite their simplicity, cryptographic protocols are frequently wrong. Lowe estimates that about half the protocols published fail to achieve their goals in some respect [28]. Since this comment concerns only published, peer-reviewed protocols, one may imagine that the success rate for proprietary protocols would be lower. However, as a consequence of intense work on this problem, including apparently hundreds of published papers,[2] the quality of newer protocols such as SSH [50] and TLS [8] seems much better.

The problem is tricky because an attacker ("the penetrator") can be an active participant on the network itself. The attacker can start sessions with other principals, or wait for them to start sessions with each other or with the attacker (not realizing that the attacker is malicious). The penetrator can encrypt or decrypt using public keys, or legitimately obtained secret keys, or stolen keys, or keys extracted from messages. The penetrator can prevent the regular participants from receiving messages, and can substitute messages of his own.

3.2 The Dolev-Yao Problem

Because of this collection of potential tricks, it is difficult to see what the penetrator can accomplish. Indeed, the penetrator can sometimes undermine the goals of a protocol without any cryptanalysis. The attacks would succeed even if the protocol was implemented with ideal cryptography. The idea of studying protocol correctness without regard to cryptographic flaws is due to Dolev and Yao [9].

The terminology of correctness and flaws is, however, too crude. The question is really what security goals a cryptographic protocol can achieve. A "flaw" is a counterexample that shows that a protocol does not achieve some goal that we thought it should meet. A correct protocol is one that achieves specific goals that we find useful.

[2] See [6] for an extensive bibliography through the mid-nineties. Indeed, the present chapter has 50 citations, despite not citing large numbers of papers in the literature. For instance, we do not even cite [4], let alone the hordes of papers derived from it.

Thus, a more nuanced way to state the Dolev-Yao problem is, to determine, given a cryptographic protocol, what security goals it achieves, and to find counterexamples showing why it does not meet other goals. In doing so, one assumes that the cryptography in use is ideal, meaning that one cannot determine anything about plaintext from ciphertext, or anything about ciphertext from plaintext, without possessing the decryption or encryption key (respectively).

The problem is important, because cryptographic protocols are a central part of the infrastructure for secure communications and networking, and for distributed systems and electronic commerce. Their importance applies equally to military and civil systems. The Dolev-Yao approach of separating cryptographic issues from structural protocol flaws is valuable for two main reasons. First, it allows us to discover this class of flaws unencumbered by the complexity of cryptographic issues. Second, the same protocol can be implemented using a variety of different cryptographic algorithms; the Dolev-Yao approach tells us whether even the best adapted implementation can achieve its goals.

3.3 An Example: The Needham-Schroeder Public Key Protocol

The famous Needham-Schroeder article [37] introduced the idea of using cryptographic protocols to achieve authentication for networked systems. It discusses both secret-key and public-key protocols, and proposed one of each, both of which were problematic. Let us examine the public-key protocol. In this protocol (as presented here) each participant has the other's public key, guaranteed using some public-key infrastructure assumed reliable. The initiator A constructs a nonce—a randomly chosen bitstring—N_a and transmits it to the recipient B accompanied by A's name, encrypted in B's public key. B, if willing to have a conversation with A, replies with a nonce N_b of his own construction, accompanied by N_a and encrypted with A's public key. Finally, A replies with N_b, translated now into B's public key. This is summarized in Figure 4. We write

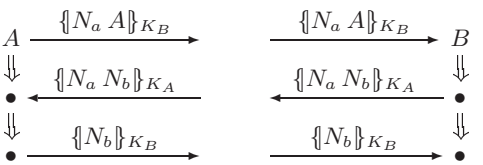


Fig. 4. The Needham-Schroeder protocol

$\{\!|t|\!\}_K$ to mean the encryption of t by K, and $t\,t'$ to mean the concatenation of t and t'. The protocol is intended to authenticate A and B to each other, and to assure them that they share the secret values N_a and N_b. These values may be combined (for instance, using exclusive-or) for a symmetric session key.

One of the most important parts of Figure 4 is the whitespace that separates the left side from the right side. The initiator A knows that on a particular occasion he sent $\{\!|N_a\ A|\!\}_{K_B}$ and then received $\{\!|N_a\ N_b|\!\}_{K_A}$, but he certainly does not know that B received the outgoing message and sent the reply. Otherwise, there would be no need for authentication. On the contrary, A needs to deduce from the way that the protocol is constructed, that if $\{\!|N_a\ N_b|\!\}_{K_A}$ came back, only B can have sent it, assuming that B's private key, K_B^{-1}, is uncompromised.

Protocol analysis is the art of inferring what others have done, knowing only what one has done oneself, and the laws of the protocol.

Unfortunately, in this protocol, we cannot simply ignore the whitespace. There is another way that things can fit together, if a malicious penetrator is present, shown in Figure 5. The attack is due to Lowe [27], and was discovered a decade and a half after the protocol was published. A has initiated a session

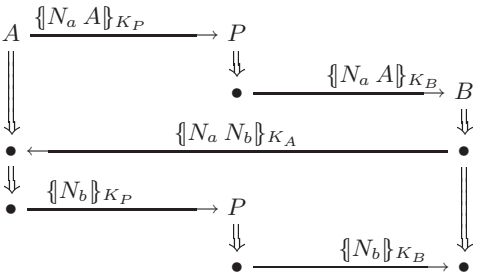


Fig. 5. A bundle: penetrated run of Needham-Schroeder

with a participant P who has decided to misbehave, possibly in response to this opportunity. As a consequence of A's bad luck, B is duped. B believes that he shares the confidential values N_a, N_b with A, whereas messages encrypted with the resulting session key will come from P.

A is not duped, because A intended to communicate with P and has done so. B is duped, because B believes that A is sending messages to B using the resulting session key, whereas P is doing so. Secrecy has failed, because the nonces and session key are known to P, and authentication has failed, because A has no run of the protocol intended for B, whereas B has a run apparently with A. How has this situation come about?

3.4 Unintended Services

The protocol itself imposes an obligation on A, when A starts a run as initiator, with responder P. A transmits a value N_a, and then undertakes to provide a service. If a value of the form $\{\!|N_a\ N|\!\}_{K_A}$ is presented (for any N), then A will perform a translation. A will translate N into the form $\{\!|N|\!\}_{K_P}$. Thus, N is

retransmitted in a form that P can read. We call this an unintended service, because the protocol requires initiators to perform it, despite its being potentially useful to penetrators.

Indeed, getting N in a readable form is particularly useful to a penetrator, and the reason for this becomes clear when we consider how the responder B gets his authentication guarantee. B can get a guarantee that A is executing his matching run only by receiving some message from B that only A can produce, and only in a matching run.

If we look at B's behavior, he receives only two messages. The first message is of the form $\{\!|N\, A|\!\}_{K_B}$. Since K_B is public, anyone can create a nonce N, concatenate it with A's name, and encode it with the public key. Therefore, from the point of view of its authenticating force, this message is mere junk. It contributes to no guarantee. Thus, any authenticating force is concentrated in the last message $\{\!|N_b|\!\}_{K_B}$.

Thus, we may examine a principal's incoming messages to determine which are junk. Discarding the junk messages, we are left with the other messages, those capable of providing authenticating force. We then examine the protocol to see what unintended services it provides, and whether these unintended services suffice to counterfeit the non-junk messages. This is a practical recipe for informal protocol analysis.

3.5 Another Example: An ISO Candidate

The protocol shown in Figure 6 was a candidate considered by an ISO committee as a *pure* authentication protocol. No shared secret is achieved. It was intended merely to assure each principal that the expected partner was initiating communications. In this protocol A uses his private key K_A^{-1} to sign nonces passed to B,

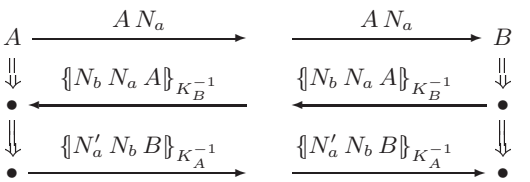


Fig. 6. An ISO candidate protocol

and conversely. Considering how A may authenticate itself to B, clearly the first, unsigned message is junk. Thus, the last message $\{\!|N'_a\, N_b\, B|\!\}_{K_A^{-1}}$ contains any authenticating force. Thus, we want to look for unintended services that would create a value of this form.

The possible services are the two transformations shown in Figure 7. In these services, any variables originating in incoming messages may be freely renamed.

Those originating on outgoing messages, by contrast, are created by the principal, and therefore cannot be manipulated by the penetrator. Now if A is active as a responder, and is presented with the values $B\, N_b$ in the incoming message, then the target term $\{\!|N'_a\, N_b\, B|\!\}_{K_A^{-1}}$ will be the result.

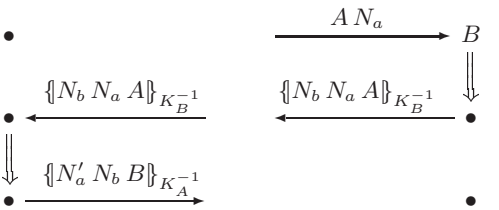


Fig. 7. Unintended services in the ISO candidate

The resulting attack is shown in Figure 8. Since it was discovered by the Canadian representatives to the committee, it is sometimes called the Canadian attack. To discover this attack, we discarded junk messages and focused on the

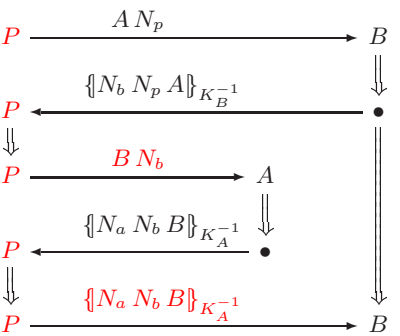


Fig. 8. The canadian attack

remaining non-junk target. The unintended services provided by the protocol then give us a recipe for generating our target message. In fact, it is easy to multiply examples in which this method of junk messages and unintended services lead us to attacks.

3.6 Types of Unintended Service

There are effectively four types of unintended service that a protocol can offer to the penetrator.

Signature In a signature service, a protocol entity is required to sign messages containing an incoming ingredient.
The transformation is $N_a \mapsto \{\!|\ N_a\ |\!\}_{K^{-1}}$. The Canadian attack of Section 3.5 is an example.

Encryption In an encryption service, a protocol entity is required to encrypt messages containing an incoming ingredient.
The transformation is $N_a \mapsto \{\!|\ N_a\ |\!\}_K$. It frequently occurs in protocols in which the ability to encrypt using a symmetric key is used as an authenticating characteristic.

Decryption In a decryption service, a protocol entity is required to decrypt messages containing an incoming ingredient.
The transformation is $\{\!|\ N_a\ |\!\}_K \mapsto N_a$. This one does not occur in nature as far as I know; presumably it is too obviously a problem.

Translation In a translation service, a protocol entity receives encrypted messages and transmits some ingredients encrypted under a different key.
The transformation is $\{\!|\ N_a\ |\!\}_K \mapsto \{\!|\ N_a\ |\!\}_{K'}$. Lowe's attack on the Needham-Schroeder protocol is an example.

3.7 The Dolev-Yao Problem Defined

The Dolev-Yao problem is the following challenge. The player is presented with a cryptographic protocol. The player must then state the secrecy properties and authentication properties that the protocol achieves. He must also give counter-examples to any secrecy or authentication properties that he believes it does not achieve.

In playing this game, the player is allowed to assume that cryptographic primitives will be chosen to be perfect. The primitives will never have collisions. The penetrator can infer nothing about plaintext, given a ciphertext, without using the key. Conversely, the penetrator can infer nothing about ciphertext, given a plaintext, without using the key. Moreover, the penetrator cannot learn a key, unless the key is contained in a message that the penetrator can decrypt.

We have already explained how to play the second part of the game, the part where counter-examples must be given. The next section is devoted to explaining how to find and prove the secrecy and authentication properties protocols achieve. The remainder of this section will be devoted to defining our model of protocol execution, called the strand space model, and to defining what secrecy and authentication properties mean in this model.

3.8 Strand Space Ideas

We very briefly summarize the ideas behind the strand space model [47,16]; see also the definitions given in Section 3.14, which is an appendix to Section 3.

A is the set of messages that can be sent between principals. We call elements of A *terms*. A is freely generated from two disjoint sets, T (representing texts such as nonces or names) and K (representing keys) by means of concatenation

and encryption. The concatenation of terms g and h is denoted $g\,h$, and the encryption of h using key K is denoted $\{\!|h|\!\}_K$. (See Section 3.14.)

A term t is a *subterm* of another term t', written $t \sqsubset t'$, if starting with t we can reach t' by repeatedly concatenating with arbitrary terms and encrypting with arbitrary keys. Hence, $K \not\sqsubset \{\!|t|\!\}_K$, except in case $K \sqsubset t$. The subterms of t are the values that are uttered when t is sent; in $\{\!|t|\!\}_K$, K is not uttered but used. (See Definition 6.)

A *strand* is a sequence of message transmissions and receptions, where transmission of a term t is represented as $+t$ and reception of term t is represented as $-t$. A strand element is called a *node*. If s is a strand, $\langle s, i\rangle$ is the i^{th} node on s. The relation $n \Rightarrow n'$ holds between nodes n and n' if $n = \langle s, i\rangle$ and $n' = \langle s, i+1\rangle$. Hence, $n \Rightarrow^+ n'$ means that $n = \langle s, i\rangle$ and $n' = \langle s, j\rangle$ for some $j > i$. Each column of nodes connected by vertical arrows $\Rightarrow$ in Figures 4–8 is a strand.

The relation $n \rightarrow n'$ represents inter-strand communication; it means that $\mathrm{term}(n_1) = +t$ and node $\mathrm{term}(n_2) = -t$. Inter-strand communication is shown by horizontal arrows $\rightarrow$ in Figures 5–8.

A *strand space* Σ is a set of strands. The two relations $\Rightarrow$ and $\rightarrow$ jointly impose a graph structure on the nodes of Σ. The vertices of this graph are the nodes, and the edges are the union of $\Rightarrow$ and $\rightarrow$.

We say that a term t *originates* at a node $n = \langle s, i\rangle$ if the sign of n is positive; $t \sqsubset \mathrm{term}(n)$; and $t \not\sqsubset \mathrm{term}(\langle s, i'\rangle)$ for every $i' < i$. Thus, n represents a message transmission that includes t, and it is the first node in s including t. For instance, A and N_p originate at the top left node of Figure 8. If a value originates on only one node in the strand space, we call it *uniquely originating*; uniquely originating values are desirable as nonces and session keys.

A bundle is a finite, causally well-founded collection of nodes and arrows of both kinds. In a bundle, when a strand receives a message m, there is a unique node transmitting m from which the message was immediately received. By contrast, when a strand transmits a message m, many strands (or none) may immediately receive m. (See Definition 3.) Figures 5 and 8 are examples of bundles; those examples happen to be undesirable.

Since a bundle $\mathcal{C}$ is an acyclic directed graph, the reflexive, transitive closure of the arrows ($\rightarrow$ together with $\Rightarrow$) form a partial order $\preceq_{\mathcal{C}}$. The statement $m \preceq_{\mathcal{C}} n$ means that there is a path from m to n traversing 0 or more arrows, in which both $\rightarrow$ and $\Rightarrow$ may appear. Because $\mathcal{C}$ is finite, $\preceq_{\mathcal{C}}$ is a well-ordering, meaning that every non-empty set of nodes contains a $\preceq_{\mathcal{C}}$-minimal element. This is an induction principle, used extensively in [47].

The height of a strand in a bundle is the number of nodes on the strand that are in the bundle. Authentication theorems generally assert that a strand has at least a given height in some bundle, meaning that the principal must have engaged in at least that many steps of its run.

A strand represents the local view of a participant in a run of a protocol. For a legitimate participant, it represents the messages that participant would

send or receive as part of one particular run of his side of the protocol. We call a strand representing a legitimate participant a *regular* strand.

3.9 The Powers of the Penetrator

For the penetrator, the strand represents an atomic deduction. More complex actions can be formed by connecting several penetrator strands. While regular principals are represented only by what they say and hear, the behavior of the penetrator is represented more explicitly, because the values he deduces are treated as if they had been said publicly.

We partition penetrator strands according to the operations they exemplify. E-strands encrypt when given a key and a plaintext; D-strands decrypt when given a decryption key and matching ciphertext; C-strands and S-strands concatenate and separate terms, respectively; K-strands emit keys from a set of known keys; and M-strands emit known atomic texts or guesses. A parameter to the model is the set $\mathsf{K}_{\mathcal{P}}$ of keys known initially to the penetrator. It represents an assumption about what keys the penetrator may emit on K-strands, as opposed to getting them indirectly, for instance by decrypting a message containing a new session key. (See Definition 8.)

As an example of how the penetrator can hook together several atomic strands to achieve a harmful compound effect, let us consider how the penetrator carries out his attack in Figure 5. To get from the incoming value $\{\!|N_a\ A|\!\}_{K_P}$ to the outgoing value $\{\!|N_a\ A|\!\}_{K_B}$, the penetrator must decrypt and then encrypt. To decrypt $\{\!|N_a\ A|\!\}_{K_P}$, the penetrator must apply his (private) decryption key K_P^{-1}. He obtains this directly, as it is assumed a member of the set $\mathsf{K}_{\mathcal{P}}$ that he possesses from the start. In the encryption that produces $\{\!|N_a\ A|\!\}_{K_B}$, the penetrator applies B's public key. We my also assume that it is a member of the set $\mathsf{K}_{\mathcal{P}}$, because it is public knowledge. Thus, we may diagram the penetrator's activity as shown in Figure 9. The lower column of penetrator activity in Figure 5 may be expanded similarly.

3.10 Representing Protocols

We turn now to the regular participants. Unlike the powers of the penetrator, which are the same regardless of the protocol, the behavior of the regular principals is dictated by the protocol. We may assume that we already have a diagram describing the protocol, in the style of Figure 4. To define a protocol, we take three steps. We will illustrate these steps starting with the Carlsen protocol [5], the diagram for which is shown in Figure 10.

Auxiliary Functions A public-key protocol typically requires a function associating a public key with each principal. A symmetric-key protocol typically requires a function associating a long-term key, shared with a key server, with each principal. We typically write such functions using subscripts, with K_A being the key associated with A, and refer to it as the "key-of" function. In the

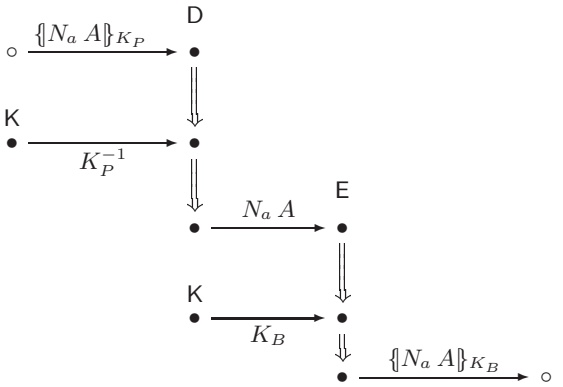


Fig. 9. Penetrator activity in figure 5

case of the Carlsen protocol, K_A represents the long-term key of A, shared with the key server.

If a protocol uses a key server as Carlsen's protocol does, then we will require that the session keys it chooses must be disjoint from the range of "key-of" function.

Occasionally a protocol requires a different function, such as a function associating a long-term shared secret with each pair of principals.

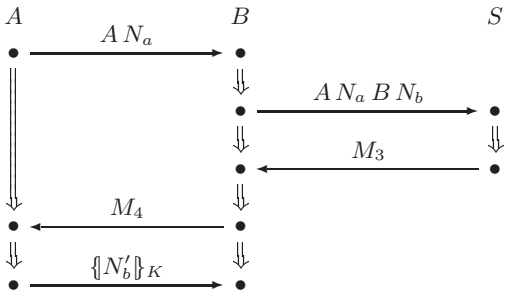
Parametric Strands Next we define the strands for the protocol. To do this, we start from the columns shown in the diagram. Each separate column represents a different role. In the Carlsen protocol, they are initiator, responder, and key server.

In some cases, there may be message components received a principal that it cannot check, because they are encrypted using a key that the recipient does not possess. These terms are simply forwarded to another principal. They will be represented simply by variables. The component (presumably) of the form $\{\!|N_a\ B\ K|\!\}_{K_A}$ cannot be checked when it is received by B. B can only forward it to A. We will represent this component using the variable H. Other messages are represented by terms of the obvious form.

By collecting the variables occurring in any term sent or received by that column, we find the parameters for that role. The parameters for the recipient are $A, B, N_a, N_b, K, N_b', H$, while those for the initiator are A, B, N_a, K, N_b'. The initiator never sees N_b, and never interprets an encrypted unit as H. The parameters for the key server are A, B, N_a, N_b, K, since it never sees N_b' and never interprets an encrypted unit as H. We write:

– CInit$[A, B, N_a, K, N_b']$ is the set of strands with trace (behavior):

$$+A\ N_a, \quad -\{\!|N_a\ B\ K|\!\}_{K_A}\ \{\!|N_a|\!\}_K\ N_b', \quad +\{\!|N_b'|\!\}_K$$



$$M_3 = \{\!|K\ N_b\ A|\!\}_{K_B}\ \{\!|N_a\ B\ K|\!\}_{K_A}$$

$$M_4 = \{\!|N_a\ B\ K|\!\}_{K_A}\ \{\!|N_a|\!\}_K\ N'_b$$

Fig. 10. Carlsen's protocol

This means that first a message is sent containing A and N_a, and then a message is received containing the nonce N_a again, encrypted in A's long-term key, and so on.

- CResp$[A, B, N_a, N_b, K, N'_b, H]$ is the set of strands with trace

$$-A\ N_a, \quad +A\ N_a\ B\ N_b, \quad -\{\!|K\ N_b\ A|\!\}_{K_B}\ H, \quad +H\ \{\!|N_a|\!\}_K\ N'_b, \quad -\{\!|N'_b|\!\}_K$$

- CServ$[A, B, N_a, N_b, K]$ is the set of strands with trace

$$-A\ N_a\ B\ N_b, \quad +\{\!|K\ N_b\ A|\!\}_{K_B}\ \{\!|N_a\ B\ K|\!\}_{K_A}$$

We write $*$ in particular argument positions to indicate a union. For instance,

$$\text{CInit}[A, B, *, *, *] = \bigcup_{N_a, K, N'_b} \text{CInit}[A, B, N_a, K, N'_b]$$

is the set of all initiator strands involving A and B, with any nonces and session keys. We also use $**$ to indicate that multiple adjacent arguments have been projected, writing e.g. CInit$[A, B, **]$ for CInit$[A, B, *, *, *]$.

The regular strands are generated by filling in the parameters with appropriate values. Instantiations may be limited to subtypes of the message algebra. For instance, the parameters A and B may be instantiated with any value that names a principal (perhaps these are IP addresses or X.509 Distinguished Names); N_a may be instantiated with any bitstring of length 128; and so on. We assume that there is some association D of variables with sets of terms in A. Very often it is convenient to assume some of these sets are disjoint, for instance the sets $D(N_a), D(K), D(A)$ of nonces, keys, and names (respectively).

Often, a strand space contains strands belonging to a parametric strand whenever the instantiation is type-correct. For instance, in the case of a parametric strand such as CServ$[A, B, N_a, N_b, K]$, if $a \in D(A)$, $b \in D(B)$, $n_a \in$

$D(N_a)$, $n_b \in D(N_b)$, and $k \in D(K)$, then CServ$[a, b, n_a, n_b, k]$ is non-empty. Sometimes it is convenient to stipulate that variables get distinct instantiations. For instance, in our formalization of the Needham-Schroeder-Lowe responder's guarantee (see below, Section 3.12 and [47,16]) we assume that the responder's nonce N_b is different from the initiator's nonce N_a. This may be encoded into the strand space itself by stipulating that NSLResp$[a, b, n_a, n_b]$ is non-empty whenever $a \in D(A)$, $b \in D(B)$, $n_a \in D(N_a)$, $n_b \in D(N_b)$, and $n_a \neq n_b$.

Restricted Values There are also some additional restrictions on the way that the values are used. For instance, in the case of the Carlsen protocol, the values N_a, N_b, and N'_b are nonces. This means that they are intended to originate uniquely. The session keys K are also intended to originate uniquely, as well as to be disjoint from the long-term keys. Correctness goals may also depend on the assumption that the player's long-term keys are not in $\mathsf{K}_{\mathcal{P}}$, the penetrator's initial knowledge, compromised through nefarious means or lucky guessing. This assumption is unavoidable, in that, when $K_A \in \mathsf{K}_{\mathcal{P}}$, the key server and B can have no assurance that A has done anything, or that session keys destined for A remain secret.

In this protocol, the assumption that $K_A \notin \mathsf{K}_{\mathcal{P}}$ is also an origination assumption. It amounts to the assumption that K_A will originate nowhere. It entails that K_A does not originate on a penetrator K-strand. No key originates on an initiator or responder strand in this protocol. And we have assumed that the session keys originating at the key server are disjoint from the long-term keys.

Origination restrictions are of course implemented using probabilistic mechanisms in reality, and some work has been done on quantifying the extent to which a particular implementation is reliable [17].

3.11 Unique Origination and Non-origination

The value N_a *originates* on the first node of any strand $s \in$ CInit$[*, *, N_a, *, *]$. What we mean by this is that a message is sent on that node, and the message contains N_a, and that this is the first node on the strand containing N_a. By contrast, N'_b does not originate on the third node of any $s \in$ CInit$[**, N'_b]$, because although a message is sent on that node and that message contains N'_b, the previous (receiving) node also contained N'_b.

We say that a term t originates uniquely in a bundle $\mathcal{C}$ if there is exactly one node n in $\mathcal{C}$ such that t originates on n. When there is no node on which t originates, we say that t is non-originating in $\mathcal{C}$.

In particular, K_A does not originate on $s \in$ CServ$[A, **]$, because although a term encrypted with K_A is sent on the second node of s, and K_A has not been used previously, K_A is not a subterm of the message. It contributes to *how* the term is constructed, but not what the term contains, and this is the intuition formalized in our definition of *subterm* (Definition 6).

Assumptions about unique origination and non-origination are a way of restricting what attacks on a protocol we are willing to worry about. For instance,

in Figure 11 we illustrate an "attack" on the Needham-Schroeder protocol in which the penetrator simply somehow knows what the responder's nonce N_b is. While this could conceivably happen, it is not a strategy likely to succeed for

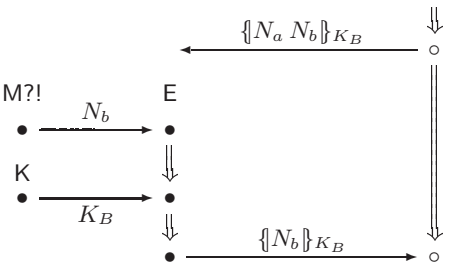


Fig. 11. An improbable attack: guessing a Needham-Schroeder nonce

the penetrator. Protocol design, therefore, does not need to concern itself systematically with how to prevent it.

A bundle describes what happens when a protocol runs, and in this we follow Robert Lowell's maxim, "Why not say what happened?" A bundle may contain some regular strands and some penetrator strands. The events of sending and receiving messages are connected in a causally well-founded way. We are concerned with a particular bundle only if it satisfies some unique origination conditions and some non-origination assumptions; otherwise, things may go wrong, but it is too implausible an attack.

3.12 What Is an Authentication Goal?

Strands are a suggestive formalism for understanding authentication and authentication failures. A strand represents one principal's experience of a protocol execution. The strand includes the information that the principal knows directly, namely that it sent certain messages and received other messages, in a particular order. Security goals for protocols concern what else must also have happened. Authentication goals are about what another regular principal must have done. Secrecy goals are inferences about what the penetrator cannot have done.

Let us return to the Needham-Schroeder protocol to consider what authentication goals are, in the light of the failure illustrated in Figure 5. One of the goals of the protocol is to assure the responder B,

> For every B-strand (apparently with A),
> there is an A-strand (apparently with B),
> and they agree on the nonces N_a, N_b.[3]

[3] It is sometimes said that this was not a goal of the protocol as originally described, but that the protocol was intended only to establish that A was active. That is,

The attack shows that there can be a B-strand, apparently with A, without any A-strand apparently with B.

This example is typical. An authentication result justifies a sound inference, and a counterexample to an authentication property shows that the inference is unsound. From B's local experience, a conclusion about A's behavior should follow. Naturally, there are assumptions that must be met for the inference to hold good, and protocol analysis is informative because it identifies exactly what the assumptions are.

When we consider the epistemology of authentication protocols, that is, the theory of knowledge that applies to them, there are four ingredients. First, there are the facts that a principal knows, which is to say the message sends and receives belonging to a single strand. Second, there are the conventions of the protocol, which dictate that the behavior of other regular participants will be described by the strands of the protocol. Third, there is the model of the penetrator embodied in Definition 8. Finally, there assumptions about origination: the unique origination of nonces and session keys, and the non-origination of long-term secrets.

From these elements, we try to infer what other events have occurred. The real world (the bundle) must contain events that causally explain what we saw. To find out what these events could be, we use the causal laws embodied in the definition of bundle (Definition 3). We may use these principles:

- What is heard was said, i.e. every negative node has a unique in-arrow in the bundle $\mathcal{C}$;
- Every principal starts at the beginning, i.e. if $n \in \mathcal{C}$, and $m \Rightarrow n$ precedes it on the same strand, then $m \in \mathcal{C}$;
- Causality is acyclic, and bundles are finite.

From the last of these, an induction principle follows. Namely, if $\preceq_{\mathcal{C}}$ is defined as the reflexive, transitive closure of the union of the two kinds of arrows, $\rightarrow$ and $\Rightarrow$, then every non-empty set S of nodes in $\mathcal{C}$ has a $preceq_{\mathcal{C}}$-minimal member. If S is a set of nodes at which the penetrator possesses some dangerous value, then a minimal member of S pinpoints how the penetrator learnt it. The maxim here, as in Watergate, is, "What did he know and when did he know it?"

To illustrate an authentication goal, let us switch now to a protocol that achieves its goals, such as the Needham-Schroeder-Lowe protocol [27] as shown in Figure 12. The only difference from the Needham-Schroeder protocol is in the second message $\{\!|N_a\, N_b\, B|\!\}_{K_A}$, in which the responder includes his name. This prevents the attack shown in Figure 5. We use NSLInit$[A, B, N_a, N_b]$ to refer to the set of all strands having the behavior shown in the left column of Figure 12,

there should be an A-strand with *some* partner. The protocol does in fact achieve this.

However, unless the goal of the protocol is the stronger assertion we have given, and the nonces are intended to become a shared secret between A and B, it is hard to see why the last message should be the encrypted value $\{\!|N_b|\!\}_{K_B}$. The plaintext message N_b would also achieve the weaker goal.

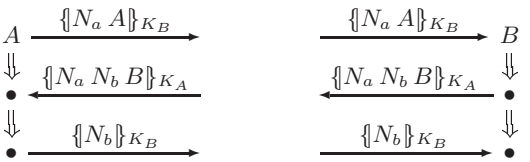


Fig. 12. Needham-Schroeder-Lowe protocol

while NSLResp$[A, B, N_a, N_b]$ refers to the set of all strands having the behavior shown in the right column.

In the revised Needham-Schroeder-Lowe, the responder B can in fact be sure that the initiator A wanted to execute a run with B. This means that in any bundle $\mathcal{C}$ containing a responder strand $s_r \in$ NSLResp$[A, B, N_a, N_b]$, there is an initiator strand $s_i \in$ NSLInit$[A, B, N_a, N_b]$ contained in $\mathcal{C}$ (subject to some origination assumptions). In fact [47], the assumptions needed are

- N_b is uniquely originating in $\mathcal{C}$ and $N_b \neq N_a$; and
- K_A^{-1} is non-originating in $\mathcal{C}$ (or alternatively, $K_A^{-1} \notin \mathsf{K}_{\mathcal{P}}$).

In the case of the initiator's guarantee, the situation is slightly different. Since the initiator sends the last message, the initiator can certainly never know whether the responder receives it. Thus, the only reasonable goal is to show that the first two nodes of the responder's strand are in the bundle $\mathcal{C}$. We express this by saying that the strand has $\mathcal{C}$-height at least 2 (see Definition 4). The initiator's guarantee states that if

- N_a is uniquely originating in $\mathcal{C}$; and
- K_A^{-1} and K_B^{-1} are non-originating in $\mathcal{C}$
  (or alternatively, $K_A^{-1}, K_B^{-1} \notin \mathsf{K}_{\mathcal{P}}$)

and $s_i \in$ NSLInit$[A, B, N_a, N_b]$ has $\mathcal{C}$-height 2,
then some $s_r \in$ NSLResp$[A, B, N_a, N_b]$ has $\mathcal{C}$-height 2.

It is an unexpected asymmetry of the Needham-Schroeder-Lowe protocol that the initiator's guarantee depends on both participants' private keys being uncompromised, while the responder's guarantee depends only on one private key being uncompromised.

In some cases, not all data values are authenticated between the principals. For instance, in the Carlsen protocol (Figure 10), the initiator never sees the responder's first nonce N_b. Thus, the conclusion of the initiator's guarantee can specify only that $s_r \in$ CResp$[A, B, N_a, *, K, \ldots]$.

We can now give the logical form of an authentication goal. Authentication goals always take the form: for all bundles $\mathcal{C}$ and all strands s, there exists a strand s' such that

If some origination assumptions hold,
and $s \in R$ has $\mathcal{C}$-height i,

then $s' \in R'$ and s' has $\mathcal{C}$-height j.

In this, R and R' are role predicates (or "asterisked" unions over role predicates), such as NSLInit$[A, B, N_a, N_b]$ and CResp$[A, B, N_a, *, K, N'_b, *]$. An origination assumption always concerns either a parameter X mentioned in R, or else K_X where X is a parameter mentioned in R.

Analyzing the authentication properties of a protocol means finding the right choices for R and R', for i and j, and the necessary origination assumptions.

Many different goals can be stated and proved (or refuted) within our framework. For instance, Gollmann has said that Lowe's attack does not undermine the claims made by Needham and Schroeder themselves, because they were working in a context where they assumed that all the principals with whom one might want to talk are trustworthy. Of course, in a world of open networks and widespread electronic commerce, this would not be a reasonable assumption, but such a world was remote in 1978 when their article was published.

Their authentication goal may be stated as follows. Let us say that X is an *interlocutor* in a bundle $\mathcal{C}$ if there exists a strand $s \in \mathcal{C}$ such that $s \in$ NSInit$[*, X, **]$ or $s \in$ NSResp$[X, **]$. Then the intended Needham-Schroeder authentication result simply has the additional assumption that for every interlocutor X, K_X^{-1} is non-originating in $\mathcal{C}$. The responder's authentication goal is sound with this additional assumption.

3.13 What Is a Secrecy Goal?

Secrecy goals are loosely dual to authentication goals. While authentication goals assert that j nodes of some strand $s' \in R'$ have happened in the bundle $\mathcal{C}$, secrecy goals say that nodes of a certain form do not occur in the bundle. Like authentication goals, they may depend on assumptions about origination. For instance, in the Needham-Schroeder-Lowe protocol, we want to ensure that there is no node in the bundle (even a penetrator node) where the message is N_a or N_b. The result takes the form: For all bundles $\mathcal{C}$, strands s_i, s_r, and nodes $n \in \mathcal{C}$

If $s_i \in$ NSLInit$[A, B, N_a, N_b]$ and $s_r \in$ NSLResp$[A, B, N_a, N_b]$ have $\mathcal{C}$-height at least 1 and 2 respectively,
and N_a and N_b are uniquely originating in $\mathcal{C}$, and
and K_A^{-1} and K_B^{-1} are non-originating in $\mathcal{C}$,
then term$(n) \neq N_a$ and term$(n) \neq N_b$.

Naturally, if we prove that N_a and N_b are not said in public in any bundle $\mathcal{C}$, then it follows that the penetrator cannot derive them from what he sees. If he could derive them, then there would exist a bundle in which he also (perhaps imprudently) utters them.

Secrecy goals always take the form: for all bundles $\mathcal{C}$ and all strands s

If some origination assumptions hold,
and $s \in R$ has $\mathcal{C}$-height i,
then there is no node $n \in \mathcal{C}$ such that term$(n) = t$.

Again, the origination assumptions concern parameters to R or values K_X where X is a parameter to R. The term t is a parameter to R, or a term constructed from parameters to R.

We can call the role R in a secrecy or authentication goal the *core* role of the goal, since the principal playing role R receives the assurance that the peer is active, or that the secret has not been disclosed. Thus, the origination assumptions always concern parameters to the core role.

Summary of this Section In this section, we have studied cryptographic protocols. We started by explaining the Dolev-Yao problem. We illustrated how to find flaws in protocols, even assuming that the cryptography by which they are implemented is perfect. One important insight for finding flaws is that we may ignore junk messages; the other one is that we want to examine the protocol to find the unintended services that may allow the penetrator to construct the non-junk messages that are intended to provide authenticating force to the regular principals. We then described how to formalize protocol behaviors using strand spaces; possible executions are bundles. Finally, we defined the logical forms of authentication and secrecy goals. We have thus illustrated two of the themes mentioned in the introduction, namely modeling security problems using simple mathematical materials, and defining specific classes of security properties to formalize real-world security goals.

3.14 Appendix: Strand Space Definitions

In this appendix to Section 3, we will define the basic strand space notions. This material is derived from [47,16]. The definitions of unique origination and non-origination (Definition 2, Clause 8) have been relativized to a bundle here, however. We also stipulate that a strand space has all the penetrator strands that it can (Definition 8).

Strand Spaces Consider a set A, the elements of which are the possible messages that can be exchanged between principals in a protocol. We will refer to the elements of A as *terms*. We assume that a *subterm* relation is defined on A. $t_0 \sqsubset t_1$ means t_0 is a subterm of t_1. We constrain the set A further below in Section 3.14, and define a subterm relation there.

In a protocol, principals can either send or receive terms. We represent transmission of a term as the occurrence of that term with positive sign, and reception of a term as its occurrence with negative sign.

Definition 1. *A* signed term *is a pair* $\langle \sigma, a \rangle$ *with* $a \in \mathsf{A}$ *and* σ *one of the symbols* $+, -$. *We will write a signed term as* $+t$ *or* $-t$. $(\pm\mathsf{A})^*$ *is the set of finite sequences of signed terms. We will denote a typical element of* $(\pm\mathsf{A})^*$ *by* $\langle \langle \sigma_1, a_1 \rangle, \ldots, \langle \sigma_n, a_n \rangle \rangle$.

A strand space *over* A *is a set* Σ *with a trace mapping* $\mathrm{tr} : \Sigma \to (\pm\mathsf{A})^*$.

By abuse of language, we will still treat signed terms as ordinary terms. For instance, we shall refer to subterms of signed terms. We will usually represent a strand space by its underlying set of strands Σ.

Definition 2. *Fix a strand space Σ.*

1. *A* node *is a pair $\langle s, i\rangle$, with $s \in \Sigma$ and i an integer satisfying $1 \leq i \leq$ length$(tr(s))$. The set of nodes is denoted by $\mathcal{N}$. We will say the node $\langle s, i\rangle$ belongs to the strand s. Clearly, every node belongs to a unique strand.*
2. *If $n = \langle s, i\rangle \in \mathcal{N}$ then index$(n) = i$ and strand$(n) = s$. Define term(n) to be $(tr(s))_i$, i.e. the ith signed term in the trace of s. Similarly, uns_term(n) is $((tr(s))_i)_2$, i.e. the unsigned part of the ith signed term in the trace of s.*
3. *There is an edge $n_1 \rightarrow n_2$ if and only if term$(n_1) = +a$ and term$(n_2) = -a$ for some $a \in \mathsf{A}$. Intuitively, the edge means that node n_1 sends the message a, which is received by n_2, recording a potential causal link between those strands.*
4. *When $n_1 = \langle s, i\rangle$ and $n_2 = \langle s, i{+}1\rangle$ are members of $\mathcal{N}$, there is an edge $n_1 \Rightarrow n_2$. Intuitively, the edge expresses that n_1 is an immediate causal predecessor of n_2 on the strand s. We write $n' \Rightarrow^+ n$ to mean that n' precedes n (not necessarily immediately) on the same strand.*
5. *An unsigned term t* occurs in *$n \in \mathcal{N}$ iff $t \sqsubset$ term(n).*
6. *Suppose I is a set of unsigned terms. The node $n \in \mathcal{N}$ is an* entry point *for I iff term$(n) = +t$ for some $t \in I$, and whenever $n' \Rightarrow^+ n$, term$(n') \notin I$.*
7. *An unsigned term t* originates *on $n \in \mathcal{N}$ iff n is an entry point for the set $I = \{t' : t \sqsubset t'\}$.*
8. *An unsigned term t is* uniquely originating in *a set of nodes $S \subset \mathcal{N}$ iff there is a unique $n \in S$ such that t originates on n.*
9. *An unsigned term t is* non-originating in *a set of nodes $S \subset \mathcal{N}$ iff there is no $n \in S$ such that t originates on n.*

If a term t originates uniquely in a suitable set of nodes, then it can play the role of a nonce or session key, assuming that everything that the penetrator does in some scenario is in that set of nodes.

$\mathcal{N}$ together with both sets of edges $n_1 \rightarrow n_2$ and $n_1 \Rightarrow n_2$ is a directed graph $\langle \mathcal{N}, (\rightarrow \cup \Rightarrow)\rangle$.

Bundles and Causal Precedence A *bundle* is a finite subgraph of $\langle \mathcal{N}, (\rightarrow \cup \Rightarrow)\rangle$, for which we can regard the edges as expressing the causal dependencies of the nodes.

Definition 3. *Suppose $\rightarrow_{\mathcal{C}} \subset \rightarrow$; suppose $\Rightarrow_{\mathcal{C}} \subset \Rightarrow$; and suppose $\mathcal{C} = \langle \mathcal{N}_{\mathcal{C}}, (\rightarrow_{\mathcal{C}} \cup \Rightarrow_{\mathcal{C}})\rangle$ is a subgraph of $\langle \mathcal{N}, (\rightarrow \cup \Rightarrow)\rangle$. $\mathcal{C}$ is a bundle if:*

1. *$\mathcal{N}_{\mathcal{C}}$ and $\rightarrow_{\mathcal{C}} \cup \Rightarrow_{\mathcal{C}}$ are finite.*
2. *If $n_2 \in \mathcal{N}_{\mathcal{C}}$ and term(n_2) is negative, then there is a unique n_1 such that $n_1 \rightarrow_{\mathcal{C}} n_2$.*
3. *If $n_2 \in \mathcal{N}_{\mathcal{C}}$ and $n_1 \Rightarrow n_2$ then $n_1 \Rightarrow_{\mathcal{C}} n_2$.*

4. $\mathcal{C}$ is acyclic.

In conditions 2 and 3, it follows that $n_1 \in \mathcal{N}_\mathcal{C}$, because $\mathcal{C}$ is a graph.

For our purposes, it does not matter whether communication is regarded as a synchronizing event or as an asynchronous activity. The definition of bundle formalizes a process communication model with three properties:

- A strand (process) may send and receive messages, but not both at the same time;
- When a strand receives a message t, there is a unique node transmitting t from which the message was immediately received;
- If a strand transmits a message t, many strands may immediately receive t.

Definition 4. *A node n is in a bundle $\mathcal{C} = \langle \mathcal{N}_\mathcal{C}, \rightarrow_\mathcal{C} \cup \Rightarrow_\mathcal{C} \rangle$, written $n \in \mathcal{C}$, if $n \in \mathcal{N}_\mathcal{C}$; a strand s is in $\mathcal{C}$ if all of its nodes are in $\mathcal{N}_\mathcal{C}$.*

*If $\mathcal{C}$ is a bundle, then the $\mathcal{C}$-*height *of a strand s is the largest i such that $\langle s, i \rangle \in \mathcal{C}$. $\mathcal{C}$-trace$(s) = \langle tr(s)(1), \ldots, tr(s)(m) \rangle$, where $m = \mathcal{C}$-height(s).*

We say that $s \in \mathcal{C}$ if the $\mathcal{C}$-height of s equals length(s).

Definition 5. *If $\mathcal{S}$ is a set of edges, i.e. $\mathcal{S} \subset \rightarrow \cup \Rightarrow$, then $\prec_\mathcal{S}$ is the transitive closure of $\mathcal{S}$, and $\preceq_\mathcal{S}$ is the reflexive, transitive closure of $\mathcal{S}$.*

The relations $\prec_\mathcal{S}$ and $\preceq_\mathcal{S}$ are each subsets of $\mathcal{N}_\mathcal{S} \times \mathcal{N}_\mathcal{S}$, where $\mathcal{N}_\mathcal{S}$ is the set of nodes incident with any edge in $\mathcal{S}$.

Proposition 1 *Suppose $\mathcal{C}$ is a bundle. Then $\preceq_\mathcal{C}$ is a partial order, i.e. a reflexive, antisymmetric, transitive relation. Every non-empty subset of the nodes in $\mathcal{C}$ has $\preceq_\mathcal{C}$-minimal members.*

We regard $\preceq_\mathcal{C}$ as expressing causal precedence, because $n \prec_\mathcal{S} n'$ holds only when n's occurrence causally contributes to the occurrence of n'. When a bundle $\mathcal{C}$ is understood, we will simply write $\preceq$. Similarly, "minimal" will mean $\preceq_\mathcal{C}$-minimal.

Terms, Encryption, and Freeness Assumptions We will now specialize the set of terms A. In particular we will assume given:

- A set $\mathsf{T} \subseteq \mathsf{A}$ of texts (representing the atomic messages).
- A set $\mathsf{K} \subseteq \mathsf{A}$ of cryptographic keys disjoint from T, equipped with a unary operator $\mathbf{inv} : \mathsf{K} \rightarrow \mathsf{K}$. We assume that $\mathbf{inv}$ is an inverse mapping each member of a key pair for an asymmetric cryptosystem to the other, and each symmetric key to itself.
- Two binary operators $\mathbf{encr} : \mathsf{K} \times \mathsf{A} \rightarrow \mathsf{A}$ and $\mathbf{join} : \mathsf{A} \times \mathsf{A} \rightarrow \mathsf{A}$.

We follow custom and write $\mathbf{inv}(K)$ as K^{-1}, $\mathbf{encr}(K, m)$ as $\{\!|m|\!\}_K$, and $\mathbf{join}(a, b)$ as $a\, b$. If $\mathfrak{K}$ is a set of keys, $\mathfrak{K}^{-1}$ denotes the set of inverses of elements of $\mathfrak{K}$. We assume, like many others (e.g. [29,32,39]), that A is freely generated, which is crucial for the results in this paper.

Axiom 1 A *is freely generated from* T *and* K *by* **encr** *and* **join**.

Definition 6. *The subterm relation* $\sqsubset$ *is defined inductively, as the smallest relation such that* $a \sqsubset a$; $a \sqsubset \{\!|g|\!\}_K$ *if* $a \sqsubset g$; *and* $a \sqsubset g\,h$ *if* $a \sqsubset g$ *or* $a \sqsubset h$.

By this definition, for $K \in \mathsf{K}$, we have $K \sqsubset \{\!|g|\!\}_K$ only if $K \sqsubset g$ already.

Definition 7. *1. If* $\mathfrak{K} \subset \mathsf{K}$, *then* $t_0 \sqsubset_{\mathfrak{K}} t$ *if* t *is in the smallest set containing* t_0 *and closed under encryption with* $K \in \mathfrak{K}$ *and concatenation with arbitrary terms* t_1.
2. A term t *is* simple *if it is not of the form* $g\,h$.
3. A term t_0 *is a* component *of* t *if* t_0 *is simple and* $t_0 \sqsubset_\emptyset t$.

Penetrator Strands The atomic actions available to the penetrator are encoded in a set of *penetrator traces*. They summarize his ability to discard messages, generate well known messages, piece messages together, and apply cryptographic operations using keys that become available to him. A protocol attack typically requires hooking together several of these atomic actions.

The actions available to the penetrator are relative to the set of keys that the penetrator knows initially. We encode this in a parameter, the set of penetrator keys $\mathsf{K}_\mathcal{P}$.

Definition 8. *A* penetrator trace *relative to* $\mathsf{K}_\mathcal{P}$ *is one of the following:*

$\mathbf{M}_t$ *Text message:* $\langle +t \rangle$ *where* $t \in \mathsf{T}$.
$\mathbf{K}_K$ *Key:* $\langle +K \rangle$ *where* $K \in \mathsf{K}_\mathcal{P}$.
$\mathbf{C}_{g,h}$ *Concatenation:* $\langle -g,\ -h,\ +g\,h \rangle$
$\mathbf{S}_{g,h}$ *Separation:* $\langle -g\,h,\ +g,\ +h \rangle$
$\mathbf{E}_{h,K}$ *Encryption:* $\langle -K,\ -h,\ +\{\!|h|\!\}_K \rangle$.
$\mathbf{D}_{h,K}$ *Decryption:* $\langle -K^{-1},\ -\{\!|h|\!\}_K,\ +h \rangle$.

$\mathcal{P}_\Sigma$ *is the set of all strands* $s \in \Sigma$ *such that* $\mathrm{tr}(s)$ *is a penetrator trace.*

We assume that $\mathcal{P}_\Sigma$ contains instances of a penetrator strand type, whenever the instantiation is type-correct, and (in the case of a K-strand) the key $K \in \mathsf{K}_\mathcal{P}$.

A strand $s \in \Sigma$ is a *penetrator strand* if it belongs to $\mathcal{P}_\Sigma$, and a node is a *penetrator node* if the strand it lies on is a penetrator strand. Otherwise we will call it a *non-penetrator* or *regular* strand or node. A node n is M, C, etc. node if n lies on a penetrator strand with a trace of kind M, C, etc.

4 Paths and Well-Behaved Bundles

In this section we will study the notion of bundle, focusing on a particular equivalence relation on them, and on the paths that messages and their constituents take through bundles. In certain "well-behaved" bundles, the paths are especially predictable, and in fact every bundle has an equivalent well-behaved bundle. This section will illustrate the advantages of the strand space model over closely related alternatives [40,43], at least as a heuristic for discovering results about cryptographic protocols.

4.1 Bundle Equivalence

Definition 9. *Bundles $\mathcal{C}, \mathcal{C}'$ on a strand space Σ are equivalent iff*

1. *they have the same regular nodes;*
2. *for all a, a originates uniquely and on a regular node in $\mathcal{C}$ if and only if a originates uniquely and on a regular node in $\mathcal{C}'$;*
3. *for all a, a is non-originating in $\mathcal{C}$ iff a is non-originating in $\mathcal{C}'$.*

A set ϕ of bundles is invariant under bundle equivalences *if whenever bundles $\mathcal{C}$ and $\mathcal{C}'$ are equivalent, $\mathcal{C} \in \phi \Rightarrow \mathcal{C}' \in \phi$.*

The penetrator's behavior may differ arbitrarily between two bundles that are equivalent in this sense, and the orderings $\preceq_{\mathcal{C}}$ and $\preceq_{\mathcal{C}'}$ may also differ freely.

Authentication goals as defined in Section 3.12 are invariant under bundle equivalences in this sense (see also [30,47,49]). As such, it always concerns what nodes, representing regular activity of the protocol, must be present in bundles. Penetrator activity may or may not be present.

Secrecy properties may also be expressed in a form that is invariant under bundle equivalences. We say (temporarily) that a value t is uncompromised in $\mathcal{C}$ if for every $\mathcal{C}'$ equivalent to $\mathcal{C}$, there is no node $n \in \mathcal{C}'$ such that $\mathrm{term}(n) = t$. In this form, a value is uncompromised if the penetrator cannot extract it in explicit form without further cooperation of regular strands. When stated in this form, the assertion that a value is uncompromised is invariant under bundle equivalences.

4.2 Redundancies

Some portions of the penetrator behavior in a bundle may be redundant in the sense that they may be excised locally. Removing them leads to a simpler yet equivalent bundle. We identify here two kinds of redundancy. First are cased where the penetrator encrypts a value with a key K, and then decrypts with the corresponding decryption key K^{-1}. This is illustrated in the upper portion of Figure 13, and may be eliminated by omitting nodes and adding the dotted arrow shown in the lower portion. Something very similar arises if the penetrator concatenates two values and then promptly separates the concatenated unit into its two immediate subterms. Since these operations introduce no cycles, the resulting graph is a bundle. They remove only penetrator nodes, so the new bundle is equivalent to the original one. Finally, since each bundle is finite, the process of removing redundancies must finally terminate with an equivalent bundle containing no redundancies.

As a consequence, we may assume that the penetrator's behavior always follows a specific pattern:

First take messages apart;
then put messages together;
finally deliver the resulting messages to a regular principal.

In order to formalize this pattern, we introduce the notion of a path.

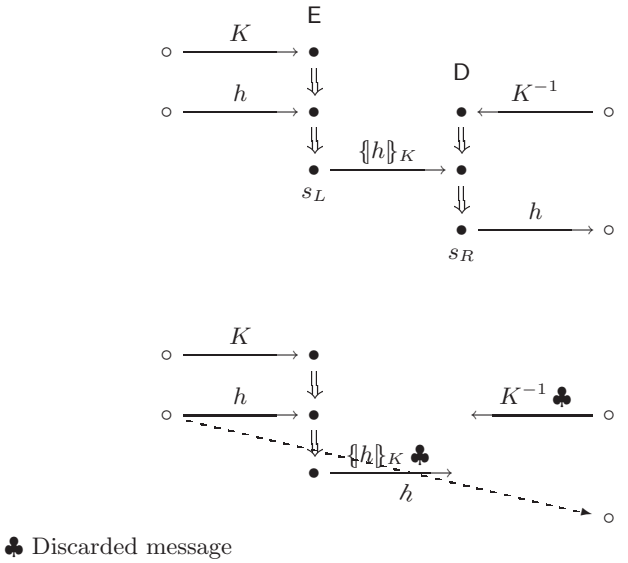


Fig. 13. E-D redundancies, and how to eliminate them

4.3 Paths

$m \Rightarrow^+ n$ means that m, n are nodes on the same strand with n occuring after m (Definition 2, Clause 4). The notation $m \longmapsto n$ means:

Either $m \Rightarrow^+ n$ with term(m) negative and term(n) positive,
or else $m \rightarrow n$.

A *path* p through $\mathcal{C}$ is any finite sequence of nodes and edges $n_1 \longmapsto n_2 \longmapsto \cdots \longmapsto n_k$. We refer to the ith node of the path p as p_i. The length of p will be $|p|$, and we will write $\ell(p)$ to mean $p_{|p|}$, i.e. the last node in p. A penetrator path is one in which all nodes other than possibly the first or the last node are penetrator nodes. As an example of a penetrator path, in which the first and last nodes are in fact regular, consider again the partial bundle shown in Figure 14. The path $\pi =$

$$\pi_1 \rightarrow \pi_2 \Rightarrow^+ \pi_3 \rightarrow \pi_4 \Rightarrow^+ \pi_5 \rightarrow \pi_6$$

is a path that traverses penetrator nodes, connecting A's first transmission $\{\!|N_a\, A|\!\}_{K_P}$ to B's first reception $\{\!|N_a\, A|\!\}_{K_B}$. In contrast to π, the path $\psi =$

$$\psi_1 \rightarrow \psi_2 \Rightarrow^+ \pi_5 \rightarrow \pi_6$$

starts on a penetrator node and ends on a regular node. Observe that by our conventions, ψ_3 and ψ_4 are well-defined (and equal to π_5 and π_6 respectively).

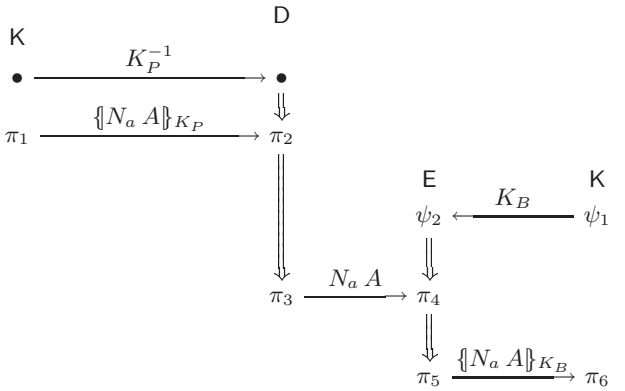


Fig. 14. Penetrator strands for Lowe's attack on Needham-Schroeder

4.4 Constructive and Destructive Edges, Normal Bundles

Definition 10. *$A \Rightarrow^+$-edge is* constructive *if it is part of a E or C strand. It is* destructive *if it is part of a D or if it is part of a S strand.*

A penetrator node n is initial *if it is a K or M node.*

Any penetrator path that begins at a regular node contains only constructive and destructive $\Rightarrow^+$-edges, because initial nodes can occur only at the beginning of a path.

Proposition 2 *In a bundle, a constructive edge immediately followed by a destructive edge has one of the following two forms:*

1. *Part of a $\mathsf{E}_{h,K}$ immediately followed by part of a $\mathsf{D}_{h,K}$ strand for some h, K.*
2. *Part of a $\mathsf{C}_{g,h}$ immediately followed by part of a $\mathsf{S}_{g,h}$ strand for some g, h.*

This result requires the freeness of the message algebra.

Proposition 3 (Penetrator Normal Form Lemma) *If the bundle $\mathcal{C}$ has no redundancies of type C-S and E-D, then for any penetrator path of $\mathcal{C}$, every destructive edge precedes every constructive edge.*

Every bundle is equivalent to a bundle with no redundancies of type C-S and E-D.

We call a bundle *normal* if it has no redundancies of type C-S and E-D, by analogy with Prawitz's normal deductions [42]. Many others have noted related properties, including [7,40,20].

We may also assume another property of the bundle $\mathcal{C}$.

Definition 11. *$\mathcal{C}$ is* directed *if for every node $n \in \mathcal{C}$, there is a regular node $m \in \mathcal{C}$ such that $n \preceq_{\mathcal{C}} m$.*

Every bundle $\mathcal{C}$ is equivalent to some directed bundle $\mathcal{C}'$. Define the graph $\mathcal{C}'$ to contain those nodes n of $\mathcal{C}$ such that $n \preceq_{\mathcal{C}} m$ for some regular node m; the arrows of $\mathcal{C}'$ are those arrows of $\mathcal{C}$ that connect nodes in $\mathcal{C}'$. $\mathcal{C}'$ is easily seen to be a bundle by enumerating the clauses in Definition 3. Moreover, $\mathcal{C}'$ is equivalent to $\mathcal{C}$, just by the reflexiveness of $\preceq_{\mathcal{C}}$.

4.5 Rising and Falling Paths

We will call a path p *rising* if $\mathrm{term}(p_i) \sqsubset \mathrm{term}(p_{i+1})$ whenever $1 \leq i$ and $i+1 \leq |p|$. This means that each term is a subterm of the next, and ultimately the first is a subterm of the last. Moving in the other direction, we will call a path p *falling* if $\mathrm{term}(p_{i+1}) \sqsubset \mathrm{term}(p_i)$ whenever $1 \leq i$ and $i+1 \leq |p|$. In this case, each term includes the next, so that the last is a subterm of the first.

A destructive $\Rightarrow$-edge may not be part of a falling path, because the path may traverse the key edge of a D-strand, as shown in left diagram of Figure 15. K^{-1} will typically have no relation to h. As we see in the right side, a E-strand

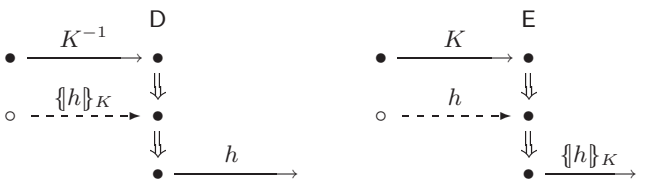


Fig. 15. Key edges into D and E-strands

is similar, since with our definitions, $K \not\sqsubset \{\!|h|\!\}_K$ unless by chance $K \sqsubset h$. However, as long as a path p does not traverse a key edge, it will be falling while traversing destructive penetrator strands and rising while traversing constructive penetrator strands.

Falling Paths, Rising Paths, and Subterms Falling paths have a property we need to explain; rising paths have a dual property. As we follow a falling path, we traverse a number of S-strands, which select a subterm from a concatenation, and a number of D-strands which select a plaintext from an encryption. If the encryption is of the form $\{\!|h|\!\}_K$, then the key used on this D-strand is K^{-1}.

Suppose that p is a falling path and $\mathfrak{K}$ is a set of keys, and suppose that p traverses a D-strand only when the key used is of the form K^{-1} for some $K \in \mathfrak{K}$. So the ciphertext is always of the form $\{\!|h|\!\}_K$ with $K \in \mathfrak{K}$. That means that the term p_1 at the start of p is of the form

$$\cdots \{\!| \cdots \mathrm{term}(\ell(p)) \cdots |\!\}_K \cdots$$

in the sense that it can be constructed from $\mathrm{term}(\ell(p))$ by concatenating (with anything) and encrypting using only keys $K \in \mathfrak{K}$. This is what in Definition 7 we wrote $\mathrm{term}(\ell(p)) \sqsubset_{\mathfrak{K}} p_1$.

The case of a rising path is similar except that we may omit the inverses. Suppose that p is a rising path and $\mathfrak{K}$ is a set of keys, and suppose that p traverses an E-strand only when the key used is some $K \in \mathfrak{K}$. Then $p_1 \sqsubset_{\mathfrak{K}} \mathrm{term}(\ell(p))$.

4.6 A Catalog of Penetrator Paths

This suggests that we separate penetrator activity into paths which do not traverse key edges; we end a path p at the node before it traverses a key edge. In this case, $\mathrm{term}(\ell(p)) = K$ for some key K. The ciphertext is of the form $\{\!|h|\!\}_K$ if we stopped before an E-strand, and of the form $\{\!|h|\!\}_{K^{-1}}$ if we stopped before a D-strand.

In cataloging penetrator paths, we may assume that the bundle is normal, since otherwise there is an equivalent bundle that is. We may also assume that the bundle is directed. From this, we may infer that a penetrator path terminates only when it reaches either a key edge or a regular node.

Thus, the purpose of a penetrator path is always either:

- To make a key available for a D or E-strand, or
- To construct some message to deliver to a regular node.

The first type of path terminates before a key edge, and the second terminates at a regular node.

In our catalog of paths p that never traverse a key edge, we will also distinguish possibilities depending whether p begins on a penetrator node or on a regular node.

1. p begins on a penetrator node and ends before a key edge. Then $\mathrm{term}(\ell(p)) = K$, and since p_1 is an initial penetrator node, it must be a K node with $|p| = 1$.
2. p begins on a regular node and ends before a key edge. So $\mathrm{term}(\ell(p)) = K$. Because p never traverses a key edge and ends with an atomic term, p is falling. So $K \sqsubset \mathrm{term}(p_1)$. In other words, the penetrator has extracted a key from a message sent by a regular principal. By our remark at the beginning of Section 4.6, if every occurrence of K in $\mathrm{term}(p_1)$ is encrypted using some other key K_1, then K_1^{-1} is a key edge joining p at some D-strand. There must be an earlier path p' furnishing this key K_1^{-1}.
3. p begins on a penetrator node and ends at a regular node. Then p_1 is either a K node or a M node, and in either case $\mathrm{term}(p_1)$ is atomic. Therefore p is a rising path and $\mathrm{term}(p_1) \sqsubset \mathrm{term}(\ell(p))$. In this path, the penetrator makes up a value, and after possibly combining it with other ingredients, delivers it to a regular participant at $\ell(p)$.
4. p begins at a positive regular node and ends at a negative regular node. In this case, p consists of a falling portion followed by a rising portion, either (or both) of which can be trivial in the sense that it consists of a single node and no arrows.

Thus, there is a falling path q ending at a positive node $\ell(q)$, and a rising path q' beginning at a negative node q'_1, where $\mathrm{term}(\ell(q)) = \mathrm{term}(q'_1)$. We call this common term the *path bridge term* of p, writing it as $\mathrm{pbt}(p)$. The whole path p is of the form $q \to q'$.

We know that $\mathrm{pbt}(p) \sqsubset \mathrm{term}(p_1)$, and $\mathrm{pbt}(p) \sqsubset \mathrm{term}(\ell(p))$. So the effect of the whole path p is to extract the term $\mathrm{pbt}(p)$ via the falling part q, and compose it with other ingredients in the rising part q', delivering the result to a regular participant.

This is a complete listing of what the penetrator can achieve by means of any path.

4.7 New Components and Efficient Bundles

According to Definition 7, a component of t is a subterm t_0 such that t_0 is either an atom or an encryption, and such that there are no encryptions hiding t_0 in t. Thus, given a term t, we generate the set of its components by repeatedly separating concatenations, stopping whenever we reach an atom or an encryption. Components are important in cryptographic protocols, because the penetrator can always undo concatenations and redo them in whatever form is desired. Only the cryptographic work required to change components can provide authentication or confidentiality. We write $\boxed{t_0} \sqsubset t$ to mean that t_0 is a component of t.

A term t_0 is a new component of a node n if $\boxed{t_0} \sqsubset \mathrm{term}(n)$, and whenever $m \Rightarrow^+ n$ it is not the case that $\boxed{t_0} \sqsubset \mathrm{term}(m)$. That is, it should not have been a component of an earlier node on the same strand. We are interested in the new components of a node, because they summarize what cryptographic work has been done at that point on the strand.

To simplify reasoning about bundles, it is convenient to assume that when a penetrator gets a component from the regular participants, he gets it from the earliest point possible. We call such a bundle *efficient*.

Definition 12. *A bundle is* efficient *if and only if, for every node m and negative penetrator node n, if every component of n is a component of m, then there is no regular node m' such that $m \prec m' \prec n$.*

We call a bundle of this kind efficient because the penetrator does the most with what he has rather than making use of additional regular nodes.

All of the bundles we have shown in earlier figures are efficient. Whenever the penetrator node handles a term, there is no earlier node that has all the same components, and a regular node has been traversed in between. However, in the case of the nonsensical variant of the Needham-Schroeder protocol shown in Figure 16, the edge marked ♣ would need to be removed, and replaced with the dashed diagonal. The negative penetrator node n must not receive its term from the third initiator node, when it can be obtained directly from the first initiator node.

Every bundle $\mathcal{C}$ may be replaced by an equivalent efficient bundle $\mathcal{C}'$, and $\mathcal{C}'$ will be normal or directed assuming that $\mathcal{C}$ was.

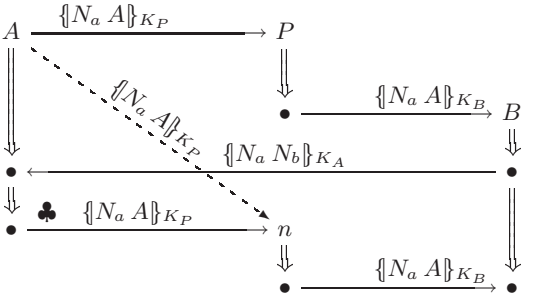


Fig. 16. An inefficient bundle for a fictitious protocol

4.8 Penetrable Keys

We may use the ideas we have developed to give an easy way to determine whether the secrecy of a key is preserved by a protocol. Let us suppose that $\mathcal{C}$ is a bundle in which some key K is disclosed, meaning that there exists a node $n \in \mathcal{C}$ such that $\boxed{K} \sqsubset \text{term}(n)$. We may assume that $\mathcal{C}$ is normal, directed, and efficient. By the bundle induction principle, we may assume that n has been chosen minimal in the ordering $\preceq_{\mathcal{C}}$ with component K. We want to determine when a key is penetrable.

There are only three cases:

- n may be a K node, in which case $K \in \mathsf{K}_{\mathcal{P}}$;
- n may be a regular node, in which case (by minimality), K is a new component of n;
- n lies on a penetrator path p at node p_i with $1 < i$. We may assume that p traverses no key edges.

In the last case, the path $\langle p_1, \ldots, p_i \rangle$ is falling, as it ends with an atomic value, so p_1 is a regular node. We know from Section 4.5 that $K \sqsubset_{\mathfrak{K}} \text{term}(p_1)$, where $\mathfrak{K}$ contains K_1 whenever K_1^{-1} was used in a D-strand along p.

So a collection of previously penetrable keys, namely the K_1^{-1}s for $K_1 \in \mathfrak{K}$, suffice to extract K from some component $\boxed{t_0} \sqsubset \text{term}(p_1)$. By the efficiency of $\mathcal{C}$, t_0 is a *new* component of the regular node p_1.

Letting $\mathfrak{K} = \emptyset$, this also covers the second case. Therefore, for every penetrable key K, either:

1. $K \in \mathsf{K}_{\mathcal{P}}$, or
2. There is a regular node $m \in \mathcal{C}$ and a new component t_0 of m such that $K \sqsubset_{\mathfrak{K}} t_0$, where for every $K \in \mathfrak{K}$, K_1^{-1} is already penetrable.

So every key that the penetrator learns, he either starts off knowing in $\mathsf{K}_{\mathcal{P}}$, or else some regular participant puts K into a new component, where it is protected only by keys that the penetrator can already learn to undo. In this construction,

we are primarily interested in paths of type 2 in our list of possible types of penetrator path in Section 4.6.

We may therefore define $\mathsf{P}(\mathcal{C})$ to be the smallest set of keys such that $\mathsf{K}_{\mathcal{P}} \subset \mathsf{P}(\mathcal{C})$ and closed under Clause 2 as just given. We have just proved that if $K = \text{term}(n)$ for any $n \in \mathcal{C}$, then $K \in \mathsf{P}(\mathcal{C})$.

Why is this useful? Because we can define a set of *safe* keys such that it is very easy to see when a key is safe, and the safe keys are disjoint from the penetrable keys.

4.9 Safe Keys

Let $\mathsf{S}_0(\mathcal{C})$ be the set of keys K such that:

- $K \notin \mathsf{K}_{\mathcal{P}}$, and
- for every positive regular node $n \in \mathcal{C}$ and every new component $\boxed{t_0} \sqsubset \text{term}(n)$, $K \not\sqsubset t_0$.

These keys are patently safe. No regular principal will ever utter them in a component, unless given that component earlier. So, no one is ever the first to spill the beans.

Let $\mathsf{S}_{i+1}(\mathcal{C})$ be the set of keys K such that:

- $K \notin \mathsf{K}_{\mathcal{P}}$, and
- for every positive regular node $n \in \mathcal{C}$ and every new component $\boxed{t_0} \sqsubset \text{term}(n)$, every occurrence of K in t_0 lies within an encryption using some key K_0 where $K_0^{-1} \in \mathsf{S}_i(\mathcal{C})$:

$$\cdots \{\!|\ \cdots K \cdots \ |\!\}_{K_0} \cdots$$

These keys are derivatively safe, since they can never be penetrated unless the penetrator gets K_0^{-1}, which is already known to be safe.

In practice, protocol secrecy goals frequently amount to showing that certain keys are in either S_0 or S_1. Larger values of i seem rarely to occur in these protocols. Showing that a private key or a long-term symmetric key is in S_0 typically reduces to checking that it is assumed not to be in $\mathsf{K}_{\mathcal{P}}$, because protocols generally avoid emitting terms containing these keys.

For instance, in the Needham-Schroeder protocol, if n is a regular node, then $K \not\sqsubset \text{term}(n)$. Hence, $\mathsf{S}_0 = \mathsf{K} \setminus \mathsf{K}_{\mathcal{P}}$, which says that any key not initially known to the penetrator is permanently safe.

Many protocols expect session keys to be generated by a key server, which sends them encrypted in the long-term keys of two principals, and no principal ever re-encrypts a session key under a new key. In a particular session, a session key K may be sent encrypted with long-term keys not in $\mathsf{K}_{\mathcal{P}}$ (or, if they are asymmetric, their inverses are not in $\mathsf{K}_{\mathcal{P}}$). If the server never re-sends the same session key K in a different session, then $K \in \mathsf{S}_1$. This method is an easy way to establish secrecy.

5 Proving Authentication

We focus now on protocols in which a regular participant authenticates its peer by sending a fresh value a (typically a nonce N), expecting to receive it back in a cryptographically altered form. If only the intended peer can perform the right cryptographic operation, then this pattern will authenticate the peer. The treatment of authentication tests in this section differs from that in [16], and is indebted to [41].

Consider some arbitrary bundle $\mathcal{C}$.

5.1 The Outgoing Authentication Test

Let us say that $n_0 \Rightarrow^+ n_1$ is an *outgoing test edge* for a if

- a originates uniquely on n_0;
- There is only one component $t_0 = \boxed{\{\!|h|\!\}_K} \sqsubset \mathrm{term}(n_0)$ such that $a \sqsubset t_0$;
- $t_0 \not\sqsubset \mathrm{term}(n_1)$ but $a \sqsubset \mathrm{term}(n_1)$; and
- $K^{-1} \notin \mathsf{P}$.

Consider the set $S = \{m \in \mathcal{C} \colon a \sqsubset \mathrm{term}(m) \wedge t_0 \not\sqsubset \mathrm{term}(m)\}$. S is non-empty, because $n_1 \in S$. So by the bundle induction principle, S has minimal members m_1.

We claim that no such m_1 is a penetrator node. Clearly such an m_1 is positive, since if it were negative, it must receive its message from another node with the same property. If m_1 is an initial penetrator node, this contradicts the assumption that a originates uniquely at n_0. Thus, if it is a penetrator node at all, m_1 lies on an edge $m_0 \Rightarrow^+ m_1$ where $t_0 \sqsubset \mathrm{term}(m_0)$ but $t_0 \not\sqsubset \mathrm{term}(m_1)$. Since $t_0 = \{\!|h|\!\}_K$, $m_0 \Rightarrow^+ m_1$ lies on a D-strand, with key edge K^{-1}. But this contradicts the assumption that $K^{-1} \notin \mathsf{P}$. Therefore, every minimal member of S is regular.

Let us call a regular edge $m_0 \Rightarrow^+ m_1$ a *transforming edge for* $n_0 \Rightarrow^+ n_1$ if $t_0 \sqsubset \mathrm{term}(m_0)$ and m_1 is a minimal member of S.

The outgoing test authentication principle states that if $\mathcal{C}$ contains an outgoing test edge, then it also contains a (regular) transforming edge for it.

The meaning of this assertion is illustrated in Figure 17. The two bulleted nodes in the figure represent m_0 and m_1.

The Outgoing Test in Needham-Schroeder We may illustrate the outgoing authentication tests by Needham-Schroeder (see Figure 4). Assume that $\mathcal{C}$ is a bundle, and the $\mathcal{C}$-height of $s_r \in \mathrm{NSResp}[A, B, N_a, N_b]$ is 3, which means that all three nodes of s_r belong to $\mathcal{C}$. Assume that $K_A^{-1} \notin \mathsf{K}_{\mathcal{P}}$. Finally, assume that N_b originates uniquely, and $N_b \neq N_a$ (which together mean that N_b originates uniquely at the second node of s_r).

Hence, the edge from the seocnd node of s_r to its third node is an outgoing test edge for N_b. By the Outgoing Authentication Test principle, there exist regular nodes $m_0, m_1 \in \mathcal{C}$ such that $m_0 \Rightarrow^+ m_1$ is a transforming edge for it. So

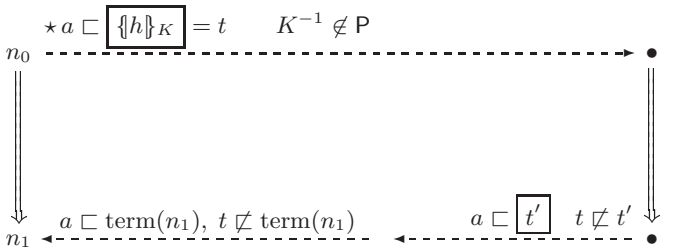


Fig. 17. Authentication provided by an outgoing test

$\{|N_a\, N_b|\}_{K_A} \sqsubset (m_0)$. The only negative regular node containing a subterm of this form is the second node of an initiator strand s_i for $s_i \in \mathrm{NSInit}[A, B', N_a, N_b]$ and some responder B'. Thus, the transforming edge $m_0 \Rightarrow^+ m_1$ must be the edge from the second node of s_i to its third node, and s_i has $\mathcal{C}$-height 3.

Unfortunately, we have not proved that $s_i \in \mathrm{NSInit}[A, B, N_a, N_b]$ for the expected responder B, rather than some other responder B'. And Figure 5 is a counterexample in which $B' = P \neq B$. Hence we have uncovered a limitation in the authentication achieved by Needham-Schroeder, first noted by Lowe [26,27], which led Lowe to amend the protocol to contain the responder's name B in the second message $\{|N_a\, N_b\, B|\}_{K_A}$.

The Outgoing Test in Needham-Schroeder-Lowe Consider now the corrected Needham-Schroeder-Lowe protocol as shown in Figure 12. As before, assume that $\mathcal{C}$ is a bundle, and the $\mathcal{C}$-height of $s_r \in \mathrm{NSResp}[A, B, N_a, N_b]$ is 3. Assume again that $K_A^{-1} \notin \mathsf{K}_{\mathcal{P}}$; that N_b originates uniquely; and that $N_b \neq N_a$.

We again infer that there exist regular nodes $m_0, m_1 \in \mathcal{C}$ such that $m_0 \Rightarrow^+ m_1$ is a transforming edge for it. So $\{|N_a\, N_b\, B|\}_{K_A} \sqsubset (m_0)$. The only negative regular node containing a subterm of this form is the second node of an initiator strand s_i for $s_i \in \mathrm{NSInit}[A, B, N_a, N_b]$. Hence, the desired s_i has $\mathcal{C}$-height 3.

5.2 The Incoming Authentication Test

Let us say that $n_0 \Rightarrow^+ n_1$ is an *incoming test edge* for a if

- a originates uniquely on n_0, and $t_1 = \{|h|\}_K \not\sqsubset \mathrm{term}(n_0)$;
- $a \sqsubset t_1$, and $t_1 \sqsubset \mathrm{term}(n_1)$; and
- $K \notin \mathsf{P}$.

We call a regular edge $m_0 \Rightarrow^+ m_1$ a *transforming edge* for $n_0 \Rightarrow^+ n_1$ if m_0 contains a as a subterm and m_1 is a minimal node such that $t_1 \sqsubset \mathrm{term}(m_1)$.

As before, only a regular edge can have this property. This assertion is illustrated in Figure 18 using the same conventions as in Figure 17. Carlsen's

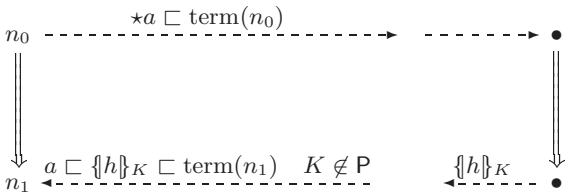


Fig. 18. Authentication provided by an incoming test

protocol (see Figure 10) is designed around incoming authentication tests. So is the Neuman-Stubblebine protocol, as we will illustrate in Section 5.4.

5.3 The Unsolicited Test

One other authentication test is important especially in the case of a key server, but in some other instances too. This is the unsolicited test. If $\boxed{\{\!|h|\!\}_K} \sqsubset \mathrm{term}(n)$ and $K \notin \mathsf{P}$, then we may infer that there is a node m such that:

- m is regular;
- $\{\!|h|\!\}_K$ originates at m.

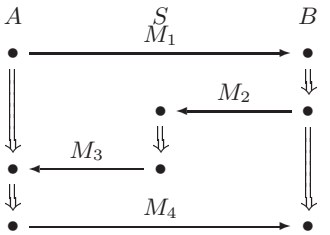
This is valid because $\{\!|h|\!\}_K$ certainly originates at some node m, and m cannot be a penetrator node: If it were, it would be the positive node of a D-strand. And then the preceding key node would have the term K, contrary to the assumption $K \notin \mathsf{P}$.

5.4 Neuman-Stubblebine

The Neuman-Stubblebine protocol [38] contains two sub-protocols. We will call the first sub-protocol the authentication protocol and the second sub-protocol the re-authentication protocol. In the authentication sub-protocol, a key distribution center generates a session key for an initiator (a network client) and a responder (a network server); the message exchange is shown in Figure 19. This session key is embedded in encrypted form in a re-usable ticket of the form $\{\!|A\, K\, T|\!\}_{K_B}$. In the re-authentication protocol the client presents the same ticket again to the network server to use the same key for another session. The value T is an expiration date, after which the network server should no longer accept the ticket, although we will not bother to model this aspect of the behavior.

We consider the authentication protocol alone first. Strands of the form shown in the columns labelled A, B, and S in Figure 19 will be called

- $\mathrm{Init}[A, B, N_a, N_b, t_b, K, H]$,
- $\mathrm{Resp}[A, B, N_a, N_b, t_b, K]$, and
- $\mathrm{Serv}[A, B, N_a, N_b, t_b, K]$,



$M_1 = A\, N_a$
$M_2 = B\, \{\!|A\, N_a\, t_b|\!\}_{K_B}\, N_b$
$M_3 = \{\!|B\, N_a\, K\, t_b|\!\}_{K_A}\, \{\!|A\, K\, t_b|\!\}_{K_B}\, N_b$
$M_4 = \{\!|A\, K\, t_b|\!\}_{K_B}\, \{\!|N_b|\!\}_K$

Fig. 19. Neuman-Stubblebine part I (Authentication)

respectively.

We define LT to be the set of long-term keys, i.e. the range of the injective function K_A for $A \in \mathsf{T}_{\text{name}}$. All long-terms keys are symmetrical: $K \in \mathsf{LT}$ implies $K = K^{-1}$.

We likewise assume that the key server generates keys in a reasonable way, meaning that that $\text{Serv}[**, K] \cap \mathcal{C} = \emptyset$ unless:

- $K \notin \mathsf{K}_{\mathcal{P}}$;
- $K = K^{-1}$;
- K is uniquely originating in $\mathcal{C}$;
- $K \notin \mathsf{LT}$.

Because of the unique origination assumption, it follows that the cardinality $|\text{Serv}[**, K] \cap \mathcal{C}| \leq 1$ for every K. We say that $\mathcal{C}$ *has a reasonable server* when these conditions are met.

The overall strategy for showing the responder's guarantee, assuming given a bundle $\mathcal{C}$ such that $\mathcal{C}$ has a reasonable server and $\mathcal{C}$ contains a strand $s_r \in \text{Resp}[A, B, N_a, N_b, t_b, K]$ with $K_A, K_B \notin \mathsf{K}_{\mathcal{P}}$, is the following:

1. Observe that $\mathsf{LT} \subset \mathsf{S}_0 \cup \mathsf{K}_{\mathcal{P}}$, as the protocol never transmits a long-term key. For any key K', if $\text{Serv}[A, B, *, *, *, K'] \neq \emptyset$, then K' is transmitted protected by K_A and K_B, but it will never be transmitted with any different protection (with a reasonable key server). Since $K_A, K_B \in \mathsf{S}_0$, $K' \in \mathsf{S}_1$ whenever $\text{Serv}[A, B, *, *, *, K'] \neq \emptyset$.
2. $\{\!|A\, K\, t_b|\!\}_{K_B}$ is an unsolicited test, originating on a regular strand. This can only be a server strand $s_s \in \text{Serv}[A, B, *, *, t_b, K]$. Therefore, $K \in \mathsf{S}_1$.
3. $M_2 \Rightarrow M_4$ is an incoming test for N_b in $\{\!|N_b|\!\}_K$. Hence, there is a regular transforming edge producing $\{\!|N_b|\!\}_K$. This can lie only on the second and third nodes of an initiator strand $s_i \in \text{Init}[A', B', N'_a, N_b, t'_b, K, *]$.

4. The first and second nodes of s_i form an incoming test for N'_a. Therefore, there is a regular transforming edge producing $\{\!|B'\ N'_a\ K\ t'_b|\!\}_{K_{A'}}$. This can only be $s'_s \in \mathrm{Serv}[A', B', *, *, t'_b, K]$.
5. By the assumption that $\mathcal{C}$ has a reasonable key server, K is uniquely originating in $\mathcal{C}$. Therefore, $s'_s = s_s$, and $A' = A$, $B' = B$, $t'_b = t_b$. Thus, $s_i \in \mathrm{Init}[A, B, *, N_b, t_b, K, *]$.

The initiator's guarantee is simpler to establish. The edge $M_1 \Rightarrow M_3$ on an initiator strand is an incoming test for N_a in $\{\!|B\ N_a\ K\ t_b|\!\}_{K_A}$. It shows there is a server strand $s_s \in \mathrm{Serv}[A, B, N_a, *, t_b, K]$. The first node of s_s is an unsolicited test, showing the existence of a responder strand $s_r \in \mathrm{Resp}[A, B, N_a, *, t_b, *]$.

In the re-authentication sub-protocol, the key distribution center no longer needs to be involved; the initiator again presents the same ticket to the responder, as shown in Figure 20. In this diagram, the first arrow inbound to the initiator

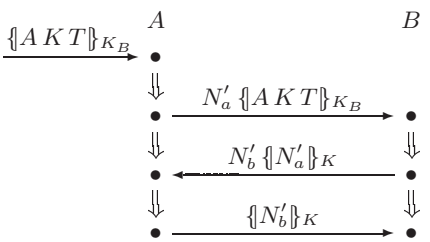


Fig. 20. Neuman-Stubblebine, part II (Re-authentication)

strand does not represent a real message; it represents the state stored in the initiator that preserves the ticket for later re-use.

In the presence of this additional sub-protocol, step 3 in the responder's guarantee can no longer be completed. There must certainly still be a transforming edge producing $\{\!|N_b|\!\}_K$, but this edge may lie either on an initiator strand for Part I of the protocol, or on (conceivably) either type of strand for Part II. By contrast, the initiator's guarantee for Part I is unaffected, because we have not added any strand with a transforming edge producing a term of the form $\{\!|B\ N_a\ K\ t_b|\!\}_{K_A}$.

This example illustrates the need for a systematic way to understand protocol mixing, as for instance the mixing of part I and part II of Neuman-Stubblebine. We undertake that task in the next section.

6 Protocol Independence via Disjoint Encryption

Whether a cryptographic protocol achieves a security goal depends on what cannot happen. To authenticate a regular principal engaging in a protocol run, we

must observe a pattern of messages that can only be constructed by that principal in that run, regardless of how the penetrator combines his own actions with those of principals engaging in other runs, as codified in the Dolev-Yao model [9]. When several cryptographic protocols are combined, the penetrator has new opportunities to obtain the messages which ought to authenticate principals to their peers. The penetrator has more unintended services to draw on.

Indeed, because protocol mixing has shown itself to be a significant cause of protocol failure, and makes protocol analysis more difficult [6,10,23,35,46,48], it has been identified [36] as a key problem in applying formal methods to cryptographic protocols.

Moreover, in practice, different protocols using cryptography are usually combined. A key distribution protocol is useful only if the session key it delivers is used for encryption. That later use may involve constructing messages similar to messages used in the key distribution protocol itself. Does this make replay attacks possible? Does the use of a key undermine the guarantees provided by the protocol distributing that key? Or conversely, can the penetrator manipulate messages from the key distribution protocol to spoof the later use of the key?

There are other reasons why protocol mixture is prevalent. Many recent protocols have large numbers of different options, and therefore have large numbers of different sub-protocols [33,18,8,35]. Each of these protocols may be easy to analyze on its own. But the same principal is required to be able to engage in any sub-protocol. Can the penetrator manipulate this willingness for his own purposes?

When protocols are mixed together, and we want to appraise whether the security of one is affected by the others, we will refer to the protocol under study as the *primary* protocol. We will refer to the others as *secondary* protocols.

6.1 Avoiding Conflict

Common sense suggests a rule of thumb when protocols are to be mixed together. This rule is that if the primary protocol uses a particular form of encrypted message as a test to authenticate a peer [14], then the secondary protocols should not construct a message of that form. If the primary protocol uses a particular form of encrypted component to protect some private value, then the secondary protocol should not receive messages of that form and retransmit their contents in other (potentially less secure) forms. Putting these two ideas together, the sets of encrypted messages that the different protocols manipulate should be disjoint.

In the case of Neuman-Stubblebine, for instance, the ticket $\{\!|A\,K\,T|\!\}_{K_B}$ originates on the primary protocol; it is stored by the initiator for use in the secondary protocol; and it is then manipulated and transformed by the responder in the secondary protocol. This violates the disjointness we would like to maintain.

One way to arrange for disjoint encryption is to give each protocol some distinguishing value, such as a number; that number may then be included as part of each plaintext before encipherment. Then no principal can mistake a value as

belonging to the wrong protocol; an encrypted value bearing a different protocol's number must not be transformed. Another way to achieve disjoint encryption is to ensure that different protocols never use the same key, although this may be expensive or difficult to arrange. Although the Abadi-Needham paper on prudent engineering practice for cryptographic protocols [1] does not discuss mixing different protocols, this rule—to try to achieve disjoint encryption—is in the same spirit as those it proposes.

In this section, we will prove that, properly formalized, it suffices. If two protocols have disjoint encryption, then the first protocol is *independent* of the second. By this we mean that if the primary protocol achieves a security goal (whether an authentication goal or a secrecy goal as defined in Sections 3.12–3.13) when the protocol is executed in isolation, then it still achieves the same security goal when executed in combination with the secondary protocol.

One of the advantages of our approach is that the result works for all secrecy and authentication goals; in this it continues a trend visible from several recent papers [31,21,45,44,19]. We have an additional reason for including this material here: It is a good example of the power of the machinery of paths and well-behaved bundles developed in Section 4.

6.2 Multiprotocol Strand Spaces

To represent multiple protocols [46], we select some regular strands as being runs of the primary protocol; we call these strands *primary strands*.

Definition 13. *A* multiprotocol strand space *is a strand space* (Σ, tr) *together with a distinguished subset of the regular strands* $\Sigma_1 \subset \Sigma \setminus \mathcal{P}_\Sigma$ *called the set of* primary *strands.*

Σ_2 denotes the set of all other regular strands, called *secondary strands*. A node is primary or secondary if the strand it lies on is. From the point of view of a particular analysis, the secondary strands represent runs of other protocols, different from the primary one under analysis.

The notion of *equivalence* needed for our purposes in this section concentrates on primary nodes.

Definition 14. *Two bundles* $\mathcal{C}, \mathcal{C}'$ *in the multiprotocol strand space* (Σ, tr, Σ_1) *are equivalent if and only if*

1. *they have the same primary nodes, meaning* $\mathcal{C} \cap \Sigma_1 = \mathcal{C}' \cap \Sigma_1$*;*
2. *for all* a*,* a *originates uniquely and on a primary node in* $\mathcal{C}$ *if and only if* a *originates uniquely and on a primary node in* $\mathcal{C}'$*;*
3. *for all* a*,* a *is non-originating in* $\mathcal{C}$ *iff* a *is non-originating in* $\mathcal{C}'$*.*

Since this is a more liberal notion of equivalence (it requires fewer nodes and unique origination facts to be unchanged), any existence assertion about equivalent bundles from Section 4 remains true in the present context.

We will also jettison the strand space parameter $\mathsf{K}_\mathcal{P}$ for our current purposes, and express our assumptions about safe keys purely in terms of non-origination.

For our present purposes, this has the advantage of not distinguishing whether a key is disclosed through a K-strand or through a secondary strand. The effect for us is now the same.

6.3 Linking Paths

From Section 4, we also know that every bundle $\mathcal{C}$ is equivalent to a directed bundle $\mathcal{C}'$ (Definition 11), and this remains true with our new definition of equivalent, assuming that in the definition of directed bundles we make the same substitution of "primary" for "regular:"

Definition 15. *$\mathcal{C}$ is* directed *if for every node $n \in \mathcal{C}$, there is a primary node $m \in \mathcal{C} \cap \Sigma_1$ such that $n \preceq_{\mathcal{C}} m$.*

Suppose we can show that given any bundle $\mathcal{C}$ involving both protocols, we can find an equivalent bundle in which no path leads from a secondary node to a primary node. Then there is also an equivalent $\mathcal{C}'$ in which there are no secondary nodes at all. Therefore, if $\mathcal{C}$ is a counterexample to some authentication goal, $\mathcal{C}'$ is a counterexample in which the secondary protocol does not occur at all. This will establish protocol independence for authentication goals.

Let us say that a penetrator path p is an *inbound linking path* if $p_0 \in \Sigma_2$ and $\ell(p) \in \Sigma_1$. We thus take the point of view of the primary path, and regard p as linking the secondary node to a primary node. One of the crucial steps in showing protocol independence is showing that we can unlink inbound linking paths.

A penetrator path p is an *outbound linking path* if $p_0 \in \Sigma_1$ and $\ell(p) \in \Sigma_2$. We need not unlink these, but we need to ensure that they do not deliver terms to the secondary protocol from which the secondary protocol will extract a secret.

6.4 Bridges

If $\mathcal{C}$ is a normal bundle and p is a penetrator path through $\mathcal{C}$, then all destructive edges precede constructive edges in p. The edge that separates the destructive portion of a path from the constructive portion is of special interest. We call it a bridge.

Definition 16. *A* bridge *in a bundle $\mathcal{C}$ is a message transmission edge $m \rightarrow n$ embedded in a subgraph of one the types shown in Figure 21.*

If $m \rightarrow n$ is a bridge, then its bridge term *is term(m), which equals term(n).*

A bridge is simple *iff its bridge term is simple, that is, is not of the form $g\,h$.*

Any edge between regular nodes is an external bridge. The source m of a bridge $m \rightarrow n$ is never on a constructive penetrator strand, and the target n is never on a destructive penetrator strand.

A term is simple if it is an atom or an encryption, not a concatenation (see Definition 7 in Section 3.14).

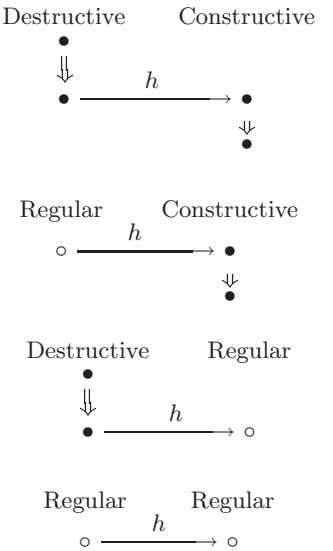
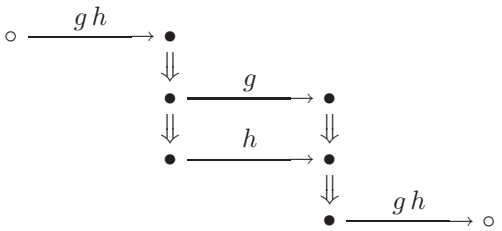


Fig. 21. Four types of bridge: internal, entry, exit, and external bridges

Proposition 4 *Suppose that $\mathcal{C}$ is a normal bundle, and p is any penetrator path in $\mathcal{C}$. Then p traverses exactly one bridge.*

Any bundle $\mathcal{C}$ can be replaced by an equivalent bundle $\mathcal{C}'$ in which the bridge term for every path is simple. Moreover if $\mathcal{C}$ is normal or efficient, so is $\mathcal{C}'$.

The proof of the second assertion consists of adding S-strands to separate any concatenated bridge term $g\,h$ and C-strands to reconstruct $g\,h$ after the bridges of the two new sub-paths.



Since every path p has a unique bridge, we can write $\mathrm{pbt}(p)$ for the bridge term occurring on the bridge.

If a path includes penetrator nodes and regular nodes, then it never traverses the same component before and after a regular node:

Proposition 5 *Suppose that $\mathcal{C}$ is normal, efficient, and has simple bridges, and p is a path through $\mathcal{C}$ that traverses no key edges. If $i < j < k$, p_j is a negative regular node, and p_i, p_k are penetrator nodes with simple terms, then $term(p_i) \neq term(p_k)$.*

Suppose moreover that p_k is the first penetrator node after p_j such that p_k has a simple term. Then $p_j \Rightarrow^+ p_{j+1}$ produces a new component $\boxed{t_0} \sqsubset term(p_{j+1})$, and $term(p_k) = t_0$.

The result follows routinely from efficiency.

6.5 Disjoint Encryption

The simplest way to state the disjoint encryption assumption would be to require that the two protocols not use the same ciphertext as a part of any message. That would mean that if $n_1 \in \Sigma_1$ and $n_2 \in \Sigma_2$, and if $\{\!|h|\!\}_K \sqsubset \mathrm{term}(n_1)$, then $\{\!|h|\!\}_K \not\sqsubset \mathrm{term}(n_2)$.

However, this simple version is unnecessarily restrictive. The secondary protocol would be unable to accept public-key certificates generated in the primary protocol, which is intuitively harmless because the contents are public in any case. The secondary protocol would also be unable to re-use symmetric-key tickets such as those generated by the Kerberos Key Distribution Center [24,38]. These are also intuitively harmless, so long as the secondary protocol does not extract private values from within them, or repackage their private contents, potentially insecurely. Hence, we allow these harmless exceptions to the requirement that no encrypted term be used by both protocols.

Definition 17. *$\{\!|h|\!\}_K$ is a* shared encryption *if there exist $n_1 \in \Sigma_1$ and $n_2 \in \Sigma_2$ such that $\{\!|h|\!\}_K \sqsubset term(n_1)$ and $\{\!|h|\!\}_K \sqsubset term(n_2)$. It is an* outbound shared encryption *if this holds with n_1 positive and n_2 negative. It is an* inbound shared encryption *if this holds with n_1 negative and n_2 positive.*

We want to restrict but not prohibit shared encryptions, and we will do so in slightly different ways for inbound and outbound shared encryptions.

Definition 18. (Disjoint Outbound Encryption) *Σ has* disjoint outbound encryption *if and only if, for every outbound shared encryption $\{\!|h|\!\}_K$, for every atom $a \sqsubset \{\!|h|\!\}_K$, and for every $n_2 \Rightarrow^+ n_2' \in \Sigma_2$,*

if *n_2 is negative and $\{\!|h|\!\}_K \sqsubset term(n_2)$,*
and *n_2' is positive and t_0 is a new component of n_2',*
then *$a \not\sqsubset t_0$.*

That is, no secondary strand manipulates a into a new component.

This definition has the important property that values originating uniquely on primary nodes cannot "zigzag" to a secondary node, before being disclosed to the penetrator.

Proposition 6 (No Zigzags) *Let Σ have disjoint outbound encryption, and let $\mathcal{C}$ be a normal, efficient bundle with simple bridges in Σ. Suppose p is a path such that $term(\ell(p)) = a$ (where $a \in \mathsf{K} \cup \mathsf{T}$), $a \sqsubset term(p_i)$ for all $1 \leq i \leq |p|$, and $p_k \in \Sigma_2$. Then $p_j \notin \Sigma_1$ for any $j < k$.*

Proof. Suppose otherwise. We may assume that j, k are chosen so that j is the largest index such that $p_j \in \Sigma_1$ and there is some later $p_k \in \Sigma_2$, and k is the smallest value $> j$ such that $p_k \in \Sigma_2$. So the path $\tilde{p} = p_j \longmapsto \cdots \longmapsto p_k$ is a penetrator path.

If $\tilde{p}$ traverses a key edge (see Section 4.6) at $p_i \rightarrow p_{i+1}$, then $\mathrm{term}(p_i) = a$ is a key. Therefore $p_i \prec p_k \prec \ell(p)$, contradicting efficiency.

Therefore $\tilde{p}$ begins at a positive regular (primary) node, ends at a negative regular (secondary) node, and never traverses a key edge. It is of type 4 in the catalog of Section 4.6. Therefore it has a bridge term $\mathrm{pbt}(\tilde{p})$ such that $\mathrm{pbt}(\tilde{p}) \sqsubset \mathrm{term}(p_j)$ and $\mathrm{pbt}(\tilde{p}) \sqsubset \mathrm{term}(p_k)$. Since $a \sqsubset \mathrm{pbt}(\tilde{p})$, either a is itself a component of $\mathrm{pbt}(\tilde{p})$, or else $a \sqsubset \boxed{\{\!|h|\!\}_K} \sqsubset \mathrm{pbt}(\tilde{p})$. If a is a component of $\mathrm{pbt}(\tilde{p})$, then we have a contradiction to efficiency as before.

Otherwise, $\{\!|h|\!\}_K$ is an outbound shared encryption. By Proposition 5, $p_k \Rightarrow^+ p_{k+1}$ produces a new component t_0, and $a \sqsubset t_0$. But this contradicts the definition of outbound disjoint encryption. □

As a consequence, we may infer that there is never a failure of secrecy where any secondary node touches the secret value.

The condition on inbound shared encryptions is that they should never occur in new components created on secondary nodes.

Definition 19. (Disjoint Encryption) *Σ has* disjoint inbound encryption *if, for every inbound shared encryption $\{\!|h|\!\}_K$ and $n_2 \Rightarrow^+ n_2' \in \Sigma_2$, if $\boxed{t_0} \sqsubset \mathit{term}(n_2')$ is a new component, then $\{\!|h|\!\}_K \not\sqsubset t_0$.*

Σ has disjoint encryption *if it has both disjoint inbound encryption and disjoint outbound encryption.*

6.6 The Protocol Independence Theorem

Definition 20. *Σ_1 is independent of Σ_2 if for every bundle $\mathcal{C}$ in Σ, there is a bundle $\mathcal{C}'$ in Σ that is equivalent to $\mathcal{C}$ such that $\mathcal{C}'$ is disjoint from Σ_2.*

Proposition 7 (Protocol Independence) *If Σ has disjoint encryption, then Σ_1 is independent of Σ_2.*

Proof. We may assume that $\mathcal{C}$ is normal, efficient, and has simple bridges. We want to show that we can remove any inbound linking paths in $\mathcal{C}$.

Let p be an inbound linking path. Suppose first that p traverses an atomic value $a \in \mathsf{T} \cup \mathsf{K}$. This may either be the key edge into a D or E strand, or it may be the bridge of p. In any case, let a be the first atomic value on p. By Proposition 6, a does not originate uniquely on a primary node. Therefore, there is an equivalent bundle $\mathcal{C}'$ in which a is produced by an initial penetrator strand (a K-strand or a M-strand).

Suppose next that p never traverses an atomic value. Then in particular it never traverses a key edge into a D or E strand. Thus, the path bridge term $\mathrm{pbt}(p) \sqsubset \mathrm{term}(p_1)$ and $\mathrm{pbt}(p) \sqsubset \mathrm{term}(\ell(p))$. Since $\mathrm{pbt}(p)$ is not atomic but it is simple, it is of the form $\{\!|h|\!\}_K$. Therefore, by disjoint inbound encryption, it

does not occur in a new component of p_1. But by Proposition 5, this contradicts efficiency.

Therefore we may remove any inbound linking path p in a normal, efficient bundle with simple bridges. It follows that there is a $\mathcal{C}'$ equivalent to $\mathcal{C}$ such that $\mathcal{C}' \cap \Sigma_2 = \emptyset$. □

An easy consequence of this theorem shows that if the primary and secondary protocols share no keys whatever, then we have independence.

Corollary 1 *For $i = 1$ and 2, let $\mathfrak{K}_i$ be the set of K such that $K \sqsubset term(n)$ for any $n \in \Sigma_i$ or $\{\!|h|\!\}_K \sqsubset term(n)$ for any h and any $n \in \Sigma_i$.*

If $\mathfrak{K}_1 \cap \mathfrak{K}_2 = \emptyset$, then Σ_1 is independent of Σ_2.

If Σ_1 and Σ_2 involve the activity of different principals, and the keys for the protocols are chosen in an unpredictable way from a large set, then the keys they use will in practice never overlap. Therefore, Σ_1 is independent of Σ_2. The same holds when the same principals may participate in both protocols, but they choose keys independently for each protocol.

Similarly, suppose each ciphertext created in Σ_1 or Σ_2 contains a distinguishing values such as different protocol numbers. If Σ_1 never accepts a ciphertext containing Σ_2's value, then we have disjoint inbound encryption. If Σ_2 never extracts a subterm from a ciphertext containing Σ_1's value, then we have disjoint outbound encryption. Together, they suffice for protocol independence.

6.7 An Application of Protocol Independence

Let us return to the Neuman-Stubblebine protocol, as described in Section 5.4 and summarized in Figures 19 and 20.

We regard the re-authentication protocol as the secondary protocol; the presence of the re-authentication protocol should not undermine any security guarantee offered by the primary protocol. However, terms of the form $\{\!|N|\!\}_K$ are constructed as new components on secondary strands, and accepted on primary strands. Hence the corresponding multiprotocol strand space does not have disjoint inbound encryption. Indeed, the penetrator can use a session of the re-authentication protocol to complete a responder strand in a bundle with no initiator [46].

For this reason, we amend (see [46]) the re-authentication protocol to the form shown in Figure 22. To apply our independence theorem, we check that the corresponding strand space Σ has disjoint encryption. But that is trivial, because tickets $\{\!|A\, K\, T|\!\}_{K_{BS}}$ are the only common encrypted subterms of primary and secondary nodes. The outbound property holds because no private subterm of a ticket is uttered in a new component of a secondary node. The inbound property holds because no new component of a secondary node contains a ticket.

Therefore, if $\mathcal{C}$ is a counterexample to some security property, we may deform $\mathcal{C}$ into an equivalent standard bundle $\mathcal{C}'$, in which there are no secondary nodes. $\mathcal{C}'$ is still a counterexample, assuming that the security property is invariant

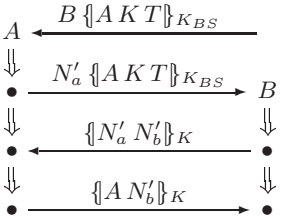


Fig. 22. Neuman-Stubblebine, part II modified (Re-authentication)

under bundle equivalences, as authentication and secrecy properties are. Thus, if the primary protocol fails to meet the security goal, that is independent of the presence of the secondary protocol: the corrected Neuman-Stubblebine re-authentication protocol is entirely guiltless in this affair.

6.8 Conclusion

In this report, we have focused on two information security problems. One is the packet protection problem, where we have studied filtering routers and firewall-oriented security goals that they are capable of achieving. The other is the Dolev-Yao problem, where we have studied how to achieve authentication and confidentiality goals in the presence of an active penetrator.

In both areas, we applied essentially the same method. We identified a class of security goals that capture important real-world security services that people need to achieve. We introduced simple mathematical modeling notions, such as directed graphs, boolean algebras, and freely generated algebras. In terms of this vocabulary, we were able to formalize the security goals and develop proof techniques and algorithms to determine what postures or protocols achieve the goals.

We regard these two problems as instances of foundational work in *security management*. Although the phrase sounds prosaic, the problems it covers are fundamental in a world where many mechanisms and systems cooperate to achieve our security objectives. Knowing that they jointly achieve something meaningful is difficult. Yet the problems have enough structure to repay mathematical abstraction, and the abstractions tell us, systematically, how to marshal the mechanisms to achieve practical protection.

References

1. Martín Abadi and Roger Needham. Prudent engineering practice for cryptographic protocols. In *Proceedings, 1994 IEEE Symposium on Research in Security and Privacy*, pages 122–136. IEEE, IEEE Computer Society Press, 1994. 252

2. Karl S. Brace, Richard L. Rudell, and Randal E. Bryant. Efficient implementation of a BDD package. In *27th ACM/IEEE Design Automation Conference*, pages 40–45, 1990. 213
3. Randal E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, C-35(8):677–691, August 1986. 210, 212
4. Michael Burrows, Martín Abadi, and Roger Needham. A logic of authentication. *Proceedings of the Royal Society*, Series A, 426(1871):233–271, December 1989. Also appeared as SRC Research Report 39 and, in a shortened form, in ACM Transactions on Computer Systems 8, 1 (February 1990), 18-36. 219
5. Ulf Carlsen. Optimal privacy and authentication on a portable communications system. *Operating Systems Review*, 28(3):16–23, 1994. 226
6. John Clark and Jeremy Jacob. A survey of authentication protocol literature: Version 1.0. University of York, Department of Computer Science, November 1997. 219, 251
7. Edmund Clarke, Somesh Jha, and Will Marrero. Using state space exploration and a natural deduction style message derivation engine to verify security protocols. In *Proceedings, IFIP Working Conference on Programming Concepts and Methods (*Procomet*)*, 1998. 240
8. T. Dierks and C. Allen. The TLS protocol. RFC 2246, January 1999. 219, 251
9. D. Dolev and A. Yao. On the security of public-key protocols. *IEEE Transactions on Information Theory*, 29:198–208, 1983. 198, 219, 251
10. Li Gong and Paul Syverson. Fail-stop protocols: An approach to designing secure protocols. In *5th International Working Conference on Dependable Computing for Critical Applications*, pages 44–55, September 1995. 251
11. Joshua D. Guttman. Filtering postures: Local enforcement for global policies. In *Proceedings, 1997 IEEE Symposium on Security and Privacy*, pages 120–29. IEEE Computer Society Press, May 1997. 199, 200, 208, 209, 210
12. Joshua D. Guttman. Packet filters and their atoms. Lecture at University of Pennsylvania, host C. Gunter, April 1999. 210
13. Joshua D. Guttman, Amy L. Herzog, and F. Javier Thayer. Authentication and confidentiality via IPsec. In D. Gollman, editor, *ESORICS 2000: European Symposium on Research in Computer Security*, LNCS. Springer Verlag, 2000. 199, 200, 203
14. Joshua D. Guttman and F. Javier Thayer Fábrega. Authentication tests. In *Proceedings, 2000 IEEE Symposium on Security and Privacy*. May, IEEE Computer Society Press, 2000. 199, 251
15. Joshua D. Guttman and F. Javier Thayer Fábrega. Protocol independence through disjoint encryption. In *Proceedings, 13th Computer Security Foundations Workshop*. IEEE Computer Society Press, July 2000. 199
16. Joshua D. Guttman and F. Javier Thayer Fábrega. Authentication tests and the structure of bundles. *Theoretical Computer Science*, 2001. To appear. 199, 224, 229, 234, 246
17. Joshua D. Guttman, F. Javier Thayer Fábrega, and Lenore D. Zuck. Faithful to the cryptography. Submitted for publication. Available at `http://cs.nyu.edu/zuck`. 229
18. D. Harkins and D. Carrel. *The Internet Key Exchange (IKE)*. IETF Network Working Group RFC 2409, November 1998. 251
19. James Heather, Gavin Lowe, and Steve Schneider. How to prevent type flaw attacks on security protocols. In *Proceedings, 13th Computer Security Foundations Workshop*. IEEE Computer Society Press, July 2000. 252

20. James Heather and Steve Schneider. Toward automatic verification of authentication protocols on an unbounded network. In *Proceedings, 13th Computer Security Foundations Workshop*. IEEE Computer Society Press, July 2000. 240
21. Mei Lin Hui and Gavin Lowe. Safe simplifying transformations for security protocols. In *12th Computer Security Foundations Workshop Proceedings*, pages 32–43. IEEE Computer Society Press, June 1999. 252
22. Jan Jürjens and Guido Wimmel. Specification-based testing of firewalls. Submitted for publication, 2001. 218
23. John Kelsey, Bruce Schneier, and David Wagner. Protocol interactions and the chosen protocol attack. In *Security Protocols, International Workshop April 1997 Proceedings*, pages 91–104. Springer-Verlag, 1998. 251
24. J. Kohl and C. Neuman. The Kerberos network authentication service (v5). RFC 1510, September 1993. 255
25. Xavier Leroy, Damien Doligez, Jacques Garrigue, Didier Rémy, and Jérôme Vouillon. *The Objective Caml System*. INRIA, `http://caml.inria.fr/`, 2000. Version 3.00. 217
26. Gavin Lowe. An attack on the Needham-Schroeder public key authentication protocol. *Information Processing Letters*, 56(3):131–136, November 1995. 247
27. Gavin Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using FDR. In *Proceeedings of* TACAS, volume 1055 of *Lecture Notes in Computer Science*, pages 147–166. Springer Verlag, 1996. 221, 231, 247
28. Gavin Lowe. Some new attacks upon security protocols. In *Proceedings of the Computer Security Foundations Workshop IX*. IEEE Computer Society Press, 1996. 219
29. Gavin Lowe. Casper: A compiler for the analysis of security protocols. In *10th Computer Security Foundations Workshop Proceedings*, pages 18–30. IEEE Computer Society Press, 1997. 236
30. Gavin Lowe. A hierarchy of authentication specifications. In *10th Computer Security Foundations Workshop Proceedings*, pages 31–43. IEEE Computer Society Press, 1997. 238
31. Gavin Lowe. Toward a completeness result for model checking of security protocols. In *11th Computer Security Foundations Workshop Proceedings*, pages 96–105. IEEE Computer Society Press, 1998. 252
32. Will Marrero, Edmund Clarke, and Somesh Jha. A model checker for authentication protocols. In Cathy Meadows and Hilary Orman, editors, *Proceedings of the DIMACS Workshop on Design and Verification of Security Protocols*. DIMACS, Rutgers University, September 1997. 236
33. D. Maughan, M. Schertler, M. Schneider, and J. Turner. *Internet Security Association and Key Management Protocol (ISAKMP)*. IETF Network Working Group RFC 2408, November 1998. 251
34. Alain Mayer, Avishai Wool, and Elisha Ziskind. Fang: A firewall analysis engine. In *Proceedings, IEEE Symposium on Security and Privacy*, 2000. 210
35. Catherine Meadows. Analysis of the Internet Key Exchange protocol using the NRL protocol analyzer. In *Proceedings, 1999 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, May 1999. 251
36. Catherine Meadows. Open issues in formal methods for cryptographic protocol analysis. In *DISCEX Workshop*. DARPA, January 2000. 251
37. Roger Needham and Michael Schroeder. Using encryption for authentication in large networks of computers. *Communications of the ACM*, 21(12), December 1978. 220

38. B. Clifford Neuman and Stuart G. Stubblebine. A note on the use of timestamps as nonces. *Operating Systems Review*, 27(2):10–14, April 1993. 248, 255
39. Lawrence C. Paulson. Proving properties of security protocols by induction. In *10th IEEE Computer Security Foundations Workshop*, pages 70–83. IEEE Computer Society Press, 1997. 236
40. Lawrence C. Paulson. The inductive approach to verifying cryptographic protocols. *Journal of Computer Security*, 1998. Also Report 443, Cambridge University Computer Lab. 237, 240
41. Adrian Perrig and Dawn Xiaodong Song. Looking for diamonds in the desert: Extending automatic protocol generation to three-party authentication and key agreement protocols. In *Proceedings of the 13th IEEE Computer Security Foundations Workshop*. IEEE Computer Society Press, July 2000. 246
42. Dag Prawitz. *Natural Deduction: A Proof-Theoretic Study*. Almqvist and Wiksel, Stockholm, 1965. 240
43. Steve Schneider. Verifying authentication protocols with CSP. In *Proceedings of the 10th IEEE Computer Security Foundations Workshop*, pages 3–17. IEEE Computer Society Press, 1997. 237
44. Scott Stoller. A bound on attacks on authentication protocols. Available at http://www.cs.indiana.edu/~stoller/, July 1999. 252
45. Scott Stoller. A reduction for automated verification of authentication protocols. In *Workshop on Formal Methods and Security Protocols*, July 1999. Available at http://www.cs.indiana.edu/~stoller/. 252
46. F. Javier Thayer Fábrega, Jonathan C. Herzog, and Joshua D. Guttman. Mixed strand spaces. In *Proceedings of the 12th IEEE Computer Security Foundations Workshop*. IEEE Computer Society Press, June 1999. 251, 252, 257
47. F. Javier Thayer Fábrega, Jonathan C. Herzog, and Joshua D. Guttman. Strand spaces: Proving security protocols correct. *Journal of Computer Security*, 7(2/3):191–230, 1999. 199, 224, 225, 229, 232, 234, 238
48. D. Wagner and B. Schneier. Analysis of the SSL 3.0 protocol. In *Proceedings, Second USENIX Workshop on Electronic Commerce*, pages 29–40, 1996. Available at http://www.counterpane.com/ssl.html. 251
49. Thomas Y. C. Woo and Simon S. Lam. Verifying authentication protocols: Methodology and example. In *Proc. Int. Conference on Network Protocols*, October 1993. 238
50. T. Ylonen, T. Kivinen, and M. Saarinen. SSH prototcol architecture. Internet draft, November 1997. Also named draft-ietf-secsh-architecture-01.txt. 219