holding packets at the destination until they can be delivered in order, and retransmitting lost packets. 5. session layer. The OSI session layer adds extra functions to the reliable pair-wise communication provided by the transport layer. Most network architectures do not have or need the functionality in this layer, and it is not of concern to security, so for the purposes of this book we can ignore it. 6. presentation layer. This layer encodes application data into a canonical (system-independent) format and decodes it into a system-dependent format at the receiving end. 7. application layer. This is where the applications that use the network, such as web surfing, file transfer, and electronic mail, reside. A layer communicates with the equivalent layer in a different node. In order to get data to a peer layer, though, the layer at the transmitting node gives the data to the layer below it (on the same node), which adds a header containing additional information if necessary, and that layer in turn gives it to the layer below. As the packet is received by the destination node, each layer reads and strips off its own header, so that the packet received by layer n looks to that layer just like it did when it was sent down to layer n−1 for transmission. This seven-layer model is the basis for a lot of the terminology in networking, and a good first step towards understanding networks, but today's network protocols do not neatly fit this model. Throughout the book we sometimes use the OSI terminology by discussing things such as encryption at layer 2 vs. layer 3 vs. layer 4, or use the terms data link layer, or transport layer. 1.1.2 IP, UDP, and TCP Today the most common protocols are the ones standardized by the IETF (Internet Engineering Task Force). All the IETF documents are on-line and freely available from the web site www.ietf.org. The protocols are specified in documents known as RFCs. (RFC is an abbreviation for "Request for Comments", but the time to comment is when the documents are in the more preliminary "internet draft" stage. Nobody wants to hear your comments on RFCs.) The IETF's protocol suite is usually referred to as the "TCP/IP suite", after the most common layer 3 (IP) and layer 4 (TCP) protocols at the time the suite was being nicknamed. IP (Internet Protocol), the layer 3 protocol, is defined in RFC 791. Its job is to deliver data across a network. To get a letter mailed with the postal service, you put it in an envelope that specifies the source and desti- 1.1.3 PRIMER ON NETWORKI PRIMER ON NETWORKING 3 nation address. Similarly, the IP layer adds an envelope (a header) to the data that specifies the source and destination addresses. But the IP address only specifies the destination machine. There might be multiple processes at the destination machine all communicating across the network, so it's necessary to also specify which process should receive the data. This is similar to putting an apartment number on the envelope in addition to the street address. IP doesn't identify the processes, but instead has a 1-octet field that specifies which protocol should receive the packet, and the rest of the information necessary to identify the destination process is contained in the layer 4 header, in the PORT fields. The two most important layer 4 protocols in the IETF suite are TCP (Transmission Control Protocol, defined in RFC 793) and UDP (User Datagram Protocol, defined in RFC 768). TCP sends an unlimited size stream of data, reliably (either all data is delivered to the other end without loss, duplication, or misordering, or the connection is terminated). UDP sends limited-sized individual chunks, with best-effort service. Both TCP and UDP have fields for SOURCE PORT and DESTINATION PORT, which specify the process to whom the data belongs. TCP additionally has sequence numbers and acknowledgments to ensure the data arrives reliably. Some port numbers are "well-known", i.e., permanently assigned to a particular service, whereas others are dynamically assigned. Being able to contact someone at a well-known port makes it easy to establish communication. In contrast, if Alice and Bob were going to attempt to communicate by going to public telephones wherever they happened to be, they'd never be able to communicate, since neither one would know what number to call. But if one of them were listening at a well-known telephone number, then the other could call from

anywhere. This is very similar to the use of well-known ports. To communicate with a particular service, say the telnet service, at some machine at IP address x, you'd know that telnet uses TCP, and is always assigned to port 23. So in the IP header, you'd specify x as the destination address, and 6 (which means TCP) as the protocol type. In the TCP header, you'd specify port 23 as the destination port. Your process would be at a dynamically assigned port, but the recipient process at node x would know which port to reply to by copying the port from the source port in the received TCP header. This will all become much more relevant when we discuss firewalls in Chapter 20 Firewalls, and how they can distinguish telnet packets (which firewall administrators would usually like to block) from, say, email packets (which firewall administrators would usually like to allow). 1.1.3 Directory Service Having a telephone line into your house means you can access any phone in the world, if you know the telephone number. The same thing is true, more or less, in computer networks. If you know the network layer address of a node on the network, you should be able to communicate with that node. (This isn't always true because of security gateways, which we'll discuss in Chapter 20 Firewalls.) 4 INTRODUCTION 1.1.3 But how do you find out another node's network layer address? Network layer addresses are not the kind of things that people will be able to remember, or type. People instead will want to access something using a name such as File-Server-3. This is a similar problem to finding someone's telephone number. Typically you start out by knowing the name of the person or service you want to talk to, and then look the name up in a telephone book. In a computer network there is a service which stores information about a name, including its network layer address. Anything that needs to be found is listed in the service. Anything that needs to find something searches the service. We call such a service a directory, though some people like to reserve the term "directory" for something in which you search based on an attribute (e.g., "find all the people who live on Main Street") rather than look up something based on knowing its name. Those people would call a simple service in which you look up information (rather than do complex searches) a naming service. We see no reason to make that distinction. It might be nice to search for all items that match a certain attribute, but usually the name will be known, and the attributes of that name will be fetched. Rather than keeping all names in one directory, the directory service is typically structured as a tree of directories. Usually a name is hierarchical, so that the directory in which the name can be found is obvious from the name. For example, an Internet name looks like radia@east.sun.com. The top level consists of pointers to the directories com for commercial enterprises, edu for educational institutions, gov for U.S. government, and various country names. Under com, there are various company names. Having multiple directories rather than keeping all names in one directory serves two purposes. One is to prevent the directory from getting unreasonably large. The other reason is to reduce name collisions (more than one object with the same name). For instance, when you're looking up a telephone number for your friend John Smith, it's bad enough trying to figure out which John Smith is the one you want if you know which town he lives in and the telephone company has separate directories for each town, but imagine if the telephone company didn't have separate books for each town and simply had a list of names and telephone numbers! Ideally, with a hierarchy of directories, name collisions could be prevented. Once a company hired one Radia Perlman, they just wouldn't hire another. I2 think that's reasonable, but someone with a name like John Smith might start having problems finding a company that could hire him. Now why did you name your baby John? Every Tom, Dick, and Harry is named John. —Sam Goldwyn For electronic mail addresses, conflicts must be prevented. Typically, companies let the first John Smith use the name John@companyname for his email address, and then perhaps the next one will be Smith@companyname, and the next one JSmith@companyname, and the next one has to start using middle initials. But for

directories of names, there is usually no way to avoid name 1.1.4 PRIMER ON NETWORKING 5 collisions within a directory. In other words, both John Smiths will use the same name within the company. Then, just like with a telephone book and multiple John Smiths, you have to do the best you can to figure out which one you want based on various attributes (such as in the telephone directory, using the street address). And just like in "real life," there will be lots of confusion where one John Smith gets messages intended for a different John Smith. The directory service is very important to security. It is assumed to be widely available and convenient to access—otherwise large-scale networking really is too inconvenient to be practical. The directory service is a convenient place to put information, such as a user's public cryptographic key. But the directory service, although convenient, is not likely to be very secure. An intruder might tamper with the information. The magic of cryptography will help us detect such tampering so that it will not be necessary to physically secure all locations that store directory service information. If the information is tampered with, good guys will detect this. It might prevent good guys from accessing the network, since they won't be able to find information they can trust, but it will not allow bad guys unauthorized access. 1.1.4 Replicated Services Sometimes it is convenient to have two or more computers performing the same function. One reason is performance. A single server might become overloaded, or might not be sufficiently close to all users on a large network. Another reason is availability. If the service is replicated, it does not matter if some of the replicas are down or unavailable. When someone wants to access the service provided, it doesn't matter which of the computers they reach. Often the user can't even tell whether there's a single copy of the service or there are replicas. What are the security issues with a replicated service? You'd want the user to have the same authentication information regardless of which replica was authenticating the user. If authentication information is stored at each replica, then coordinating the databases, for example after a change password command, can be tricky. And if the identical exchange will work with any of the replicas, then having an eavesdropper repeat the authentication handshake with a different replica might be a security problem. 1.1.5 Packet Switching A really naive assumption would be that if people wanted computer A to talk to computer B, they'd string a wire between A and B. This doesn't work if networks get large, either in number of nodes (n2 wires) or physical distance (it takes a lot of wire to connect each of 10000 nodes in North America with each of 10000 nodes in Asia). So in a network, messages do not go directly from sender to recipient, but rather have to be forwarded by various computers along the way. These 6 INTRODUCTION 1.1.6 message forwarders are referred to as packet switches, routers, gateways, bridges, and probably lots of other names as well. A message is generally broken into smaller chunks as it is sent through the network. There are various reasons for this. • Messages from various sources can be interleaved on the same link. You wouldn't want your message to have to wait until someone else finished sending a huge message, so messages are sent a small chunk at a time. If the link is in the process of sending the huge message when your little single-chunk message arrives, your message only has to wait until the link finishes sending a chunk of the large message. • Error recovery is done on the chunk. If you find out that one little chunk got mangled in transmission, only that chunk needs to be retransmitted. • Buffer management in the routers is simpler if the size of packets has a reasonable upper limit. 1.1.6 Network Components The network is a collection of packet switches (usually called routers) and links. A link can either be a wire between two computers or a multi-access link such as a LAN (local area network). A multi-access link has interesting security implications. Whatever is transmitted on the link can be seen by all the other nodes on that link. Multi-access links with this property include Ethernet (also known as CSMA/CD), token rings, and packet radio networks. Connected to the backbone of the network are various types of nodes. A common

categorization of the nodes is into clients, which are workstations that allow humans to access the resources on the network, and servers, which are typically dedicated machines that provide services such as file storage and printing. It should be possible to deploy a new service and have users be able to conveniently find the service. Users should be able to access the network from various locations, such as a public workstation a company makes available for visitors. If a person has a dedicated workstation located in one location, such as an office, it should be possible with a minimum of configuration for the user to plug the workstation into the network. Historically, another method for users to access a network is through a dumb terminal. A dumb terminal is not a general-purpose computer and does not have the compute power to do cryptographic operations. Usually a dumb terminal hooks directly into a host machine, or into a terminal server which relays the terminal's keystrokes via a network protocol across the network to the host machine (the machine the user logs into). Very few dumb terminals remain today, but their legacy lives on in the form of software-based terminal emulators implemented in most PCs and workstations. Even though these devices are capable of complex calculations, for backward compatibility, they don't do them. 1.1.7 PRIMER ON PRIMER ON NETWORKING 7 1.1.7 Destinations: Ultimate and Next-Hop A network is something to which multiple systems can attach. We draw it as a cloud since, from the point of view of the systems connected to it, exactly what goes on inside is not relevant. If two systems are on the same cloud, one can send a message to the other by attaching a header that contains a source address and a destination address, much like putting a letter into an envelope for delivery by the postal service. Figure 1-2. A Network But how do you connect to the network? With a point-to-point link to a packet switch inside the network, things are reasonably simple. If A wants to send a message to B, A will put A as source address and B as destination address and send the message on the point-to-point link. But what if A is connected on a LAN? In that case, in order to transmit the packet through the network, A has to specify which of its neighbors should receive the message. For example: Figure 1-3. Network Connections If A wants to send a message to D it has to know (somehow—if you care how, you can read my2 book [PERL99]) that the appropriate neighbor for forwarding the packet is R2. So when A transmits the message there are two destinations: R2 as the next recipient and D as the ultimate recipient. A reasonably simple way of thinking about this is that the data link layer worries about transmission across a single link. The data link header has a source address and a destination address which indicate the transmitter on that link and the receiver on that link. The network layer worries about transmission across a multi-hop network. It has a header that carries the original source and ultimate destination. The data link header is removed each time a message is received, and a new data link header is tacked onto the message when it is forwarded to the next hop. A B C D E R1 A B R2 R3 R4 R5 R6 R7 D token ring 8 INTRODUCTION 1.1.8 When A transmits the packet, the network header has source A, destination D. The data link header has source A, destination R2. R2 forwards the packet to R5. Since R2 is connected to R5 with a point-to-point link, the data link header will not have addresses. But when R5 forwards the packet to R6 across the LAN, the network layer header will (still) be source A, destination D. The data link header will be source R5, destination R6. When R6 forwards it (across the token ring LAN) the network header is still the same, and the data link header has source R6, destination D. Most likely A's data link address will look different from its network layer address, so it's a bit sloppy to say source A in both the data link header and network header. But this is all irrelevant to security. Fascinating in its own right, but irrelevant to this book. The network layer header can be thought of as an envelope for the message. The data link header is an outer envelope. We've described the case of two envelopes—a network header inside a data link header. The world can be even more complicated than this. In fact, the "data link layer" might be a multi-hop network with multi-hop

networks inside it as well. So a message might wind up with several envelopes. Again this is fascinating stuff but irrelevant to this book. 1.1.8 Address Structure What do addresses look like? In terms of security, the main issue is how difficult it is to forge a source address, and how easy it is to arrange for the network to deliver packets to you when they are addressed to someone other than you. For instance, think of a letter as having a source address (the return address, it's called in paper mail) and a destination address. It's easy to send a letter to anyone and put President, White House, USA as the source address. It's harder to arrange to receive mail sent to President, White House, USA if you are not the U.S. President, especially if you don't live in the White House, and most likely more difficult the further you live from the address you'd like to impersonate. Network addresses are usually hierarchical, just like a postal address. If we think of the address as specifying country/state/city/person, then in general it will be easier to arrange to receive someone else's messages if you reside in the same city (for instance by bribing a postal employee), and most difficult if they're in a different country. Forging source addresses is easy in most network layers today. Routers can be built more defensively and do a sanity check on the source address, based on where they receive the packet from. After some highly publicized denial of service attacks, where vandals overwhelmed victim sites with nuisance traffic, many routers are now deployed with this feature of checking source addresses and discarding traffic received from an unexpected direction. It's not a perfect solution, though. As typically implemented, it requires extra configuration (so the routers will know what source addresses to expect from which directions), somewhat violates my2 philosophy (as a layer 3 specialist) that routers should be self-configuring and adapt to topological changes, and slows down the router because it has to make an extra check when forwarding a packet. 1.2 ACTIVE VS. PASSIVE ATTACKS 9 1.2 ACTIVE VS. PASSIVE ATTACKS A passive attack is one in which the intruder eavesdrops but does not modify the message stream in any way. An active attack is one in which the intruder may transmit messages, replay old messages, modify messages in transit, or delete selected messages from the wire. A typical active attack is one in which an intruder impersonates one end of the conversation, or acts as a man-in-the-middle (see §6.4.1 The Bucket Brigade/Man-in-the-Middle Attack). 1.3 LAYERS AND CRYPTOGRAPHY Encryption and integrity protection are sometimes done on the original message or on each chunk of the message, and if on each chunk, it might be done end-to-end or hop-by-hop. There are interesting tradeoffs and implications of these choices. If done on the original message, it can be protected while being stored, and the infrastructure does not need to even know whether the data it is moving is cryptographically protected. This means that the location where the cryptographically protected message is kept, and the infrastructure for transmitting the message, need not be trusted. Encryption hop-by-hop can foil traffic analysis, i.e., it hides from eavesdroppers the information about which parties are communicating. Thus it is useful even if encryption is being done at other layers. If done hop-by-hop, the packet switches must be trusted, because by definition of hopby-hop, the packet switches will see the plaintext. If done end-to-end as the data is being transmitted, if individual chunks are separately encrypted and integrity protected, then the data that arrives intact can be used, whereas if there's only a single integrity check for the entire message, then any corruption or loss will require retransmitting the entire thing, since (by definition of cryptographically protecting the data as a whole instead of individual chunks) there will be no way to know where the loss/corruption occurred. 1.4 AUTHORIZATION Network security basically attempts to answer two questions: "who are you?" and "should you be doing that?" Authentication proves who you are. Authorization defines what you're allowed to do. Typically the way a server decides whether someone should have access to a resource is by first authenticating the user, and then consulting a database associated with the resource that

indicates 10 INTRODUCTION 1.5 who is allowed to do what with that resource. For instance, the database associated with a file might say that Alice can read it and Bob and Carol can both read and write it. This database is often referred to as an ACL (access control list). Another model of authorization is known as the capability model. Instead of listing, with each resource, the set of authorized users and their rights (e.g., read, write, execute), you would have a database that listed, for each user, everything she was allowed to do. If there were only a single resource, then the ACL model and the capability model would be basically the same, since in both cases there would be a database that lists all the authorized users and what rights each has to that resource. But in a world in which there are many resources, not all under control of one organization, it would be difficult to have a central database listing what each user was allowed to do (for instance, all the files that user is allowed to read), and it would have scaling problems if there were many resources each user was allowed to access. Some people worry that ACLs don't scale well if there are many users allowed access to each resource. But the concept of a group answers that concern. A very basic form of group implemented in some systems is that each user is a member of one group, and someone with special privileges assigns users to groups. There is a special group known as "world", which includes everyone. Alice would be allowed to read a file if her name was listed on the ACL with read access, or if her group was listed on the ACL with read access, or if "world" was given read access. Extensions to the idea of groups that might be useful: • allow a user to be in multiple groups (researchers, security experts, U.S. citizens) • allow anyone (not just someone with special privileges) to create a group. Allow anyone to name that group on an ACL they are authorized to administer. • allow a group for which the user explicitly invokes his membership. This type of group is known as a role. The main difference between what people think of as a role and what people think of as a group is that the user always has all the rights of all the groups he is a member of, but only has the rights of the role he has explicitly invoked. Some people would claim that if the user is allowed to assert multiple roles, he can have only one of them active at any time. We discuss ways of implementing very flexible notions of groups in §13.8.3 Groups. 1.5 TEMPEST One security concern is having intruders tap into a wire, giving them the ability to eavesdrop and possibly modify or inject messages. Another security concern is electronic emanation, whereby through the magic of physics, the movement of electrons can be measured from a surprising dis- 1.6 KEY ESCROW FOR LAW ENFORCEMENT 11 tance away. This means that intruders can sometimes eavesdrop without even needing to physically access the link. The U.S. military Tempest program measures how far away an intruder must be before eavesdropping is impossible. That distance is known as the device's control zone. The control zone is the region that must be physically guarded to keep out intruders that might be attempting to eavesdrop. A well-shielded device will have a smaller control zone. I1 remember being told in 1979 of a tape drive that had a control zone over two miles. Unfortunately, most control zone information is classified, and I2 couldn't get me1 to be very specific about them, other than that they're usually expressed in metric. Since it is necessary to keep intruders away from the control zone, it's certainly better to have something with a control zone on the order of a few inches rather than a few miles (oh yeah, kilometers). CIA eavesdroppers could not intercept the radio transmissions used by Somali warlord Mohammed Farah Aidid; his radios, intelligence officials explained, were too "low tech." —Douglas Waller & Evan Thomas, Newsweek, October 10, 1994, page 32 1.6 KEY ESCROW FOR LAW ENFORCEMENT Law enforcement would like to preserve its ability to wiretap otherwise secure communication. (Also, sometimes companies want to be able to read all data of their employees, either to enforce company policies, or to ensure data is not lost when an employee forgets a password or leaves the company.) In order for the government to ensure it can always wiretap, it must

prevent use of encryption, break the codes used for encryption (as it did in a military context during World War II), or somehow learn everyone's cryptographic keys. The Clipper proposal was proposed in the mid-90's and attempted the third option. It allows the government to reconstruct your key (only upon court order and with legitimate cause of course). This is made possible through the use of a device known as the Clipper chip. A lot about Clipper was classified by the government as secret (and classified by a lot of other people as evil). We describe the basic technical design of Clipper in §21.9 Clipper. Although the Clipper proposal appears to have been a failure, and the government appears to have for the moment at least given up on attempting to control cryptography, the Clipper design was fascinating, and is worth learning about. The simple concept is that encryption is done with a special chip (the Clipper chip). Each chip manufactured has a unique key, and the government keeps a record of the serial number/encryption key correspondence of every chip manufactured. Because not all people have complete trust in the government, rather than keeping the key in one place, each key is broken into two quantities which must be $\oplus$'d in order to obtain the actual key. Each piece is 12 INTRODUCTION 1.6 completely useless without the other. Since each piece is kept with a separate government agency, it would require two U.S. government agencies to cooperate in order to cheat and obtain the key for your Clipper chip without a valid court order. The government assures us, and evidence of past experience supports its claim, that cooperation between U.S. government agencies is unlikely. The Clipper proposal was always controversial, starting with its name (which violated someone's trademark on something unrelated). Why would anyone use Clipper when alternative methods should be cheaper and more secure? The reason alternatives would be cheaper is that enforcing the ability of the U.S. government to wiretap adds a lot of complexity over a design that simply encrypts the data. Proponents of Clipper gave several answers to this question: • The government would buy a lot of Clipper chips, bringing the cost down because of volume production, so Clipper would wind up being the most cost-effective solution. • Encryption technology is only useful if both parties have compatible equipment. Since the U.S. government would use Clipper, to talk securely to the U.S. government, you would have to use Clipper. So any other mechanism would have to be implemented in addition to Clipper. • Again, since encryption technology is only useful if both parties have compatible equipment, if Clipper took over enough market share, it would essentially own the market (just like VHS, a technically inferior standard supposedly, beat out Beta in the VCR marketplace). Since Clipper would be one of the earliest standards, it might take over the marketplace before any other standards have an opportunity to become entrenched. The argument was that most people wouldn't care that Clipper enables wiretapping, because they'll assume they have nothing to fear from the U.S. government wiretapping them. • The government claimed that the cryptographic algorithm in Clipper was stronger than you could get from a commercial source. Civil libertarians feared Clipper was a first step towards outlawing untappable cryptography. Clipper proponents say it was not. It's true that outlawing alternatives was not part of the Clipper proposal. However, there have been independent efforts to outlaw cryptography. Those efforts have been thwarted in part with the argument that industry needs security. But if Clipper were deployed, that argument would have gone away. Clipper was designed for telephones, fax, and other low-speed applications, and in some sense is not relevant to computer networking. Many people regard it, however, as a first step and a model for taking the same approach for computer networks. The Clipper proposal was a commercial failure, and export controls are currently relaxed. However, the technical aspects of such designs are fascinating, laws can change at any time, and export controls have created other fascinating and arcane designs that we will describe throughout the book, for instance, §25.13 Exportability. 1.7 KEY ESCROW FOR CARELESS USERS 13 1.7 KEY ESCROW

FOR CARELESS USERS It is prudent to keep your key in a safe place so that when you misplace your own key you can retrieve a copy of the key rather than conceding that all your encrypted files are irretrievably lost. It would be a security risk to have all users' keys stored unencrypted somewhere. The database of keys could be stored encrypted with a key known to the server that was storing the database, but this would mean that someone who had access to that machine could access all the user keys. Another possibility is to encrypt the key in a way that can only be reconstructed with the cooperation of several independent machines. This is feasible, and we'll discuss it more in §21.9.1 Key Escrow. Some applications don't require recoverable keys. An example of such an application is login. If a user loses the key required for login, the user can be issued a new key. A user may therefore want different keys for different uses, where only some of the keys are escrowed. For applications that do require recoverable keys, protection from compromise can be traded off against protection from loss. 1.8 VIRUSES, WORMS, TROJAN HORSES Lions and tigers and bears, oh my! —Dorothy (in the movie The Wizard of Oz) People like to categorize different types of malicious software and assign them cute biological terms (if one is inclined to think of worms as cute). We don't think it's terribly important to distinguish between these things, but will define some of the terms that seem to be infecting the literature. • Trojan horse—instructions hidden inside an otherwise useful program that do bad things. Usually the term Trojan horse is used when the malicious instructions are installed at the time the program is written (and the term virus is used if the instructions get added to the program later). • virus—a set of instructions that, when executed, inserts copies of itself into other programs. More recently, the term has been applied to instructions in email messages that, when executed, cause the malicious code to be sent in email to other users. • worm—a program that replicates itself by installing copies of itself on other machines across a network. 14 INTRODUCTION 1.8.1 • trapdoor—an undocumented entry point intentionally written into a program, often for debugging purposes, which can be exploited as a security flaw. • logic bomb—malicious instructions that trigger on some event in the future, such as a particular time occurring. • zombie—malicious instructions installed on a system that can be remotely triggered to carry out some attack with less traceability because the attack comes from another victim. Often the attacker installs large numbers of zombies in order to be able to generate large bursts of network traffic. We do not think it's useful to take these categories seriously. As with most categorizations (e.g., plants vs. animals, intelligence vs. instinct), there are things that don't fit neatly within these categories. So we'll refer to all kinds of malicious software generically as digital pests. 1.8.1 Where Do They Come From? Where do these nasties come from? Except for trapdoors, which may be intentionally installed to facilitate troubleshooting, they are written by bad guys with nothing better to do with their lives than annoy people. How could an implementer get away with writing a digital pest into a program? Wouldn't someone notice by looking at the program? One of the most famous results in computer science is that it is provably impossible to be able to tell what an arbitrary program will do by looking at it*, so certainly it would be impossible to tell, in general, whether the program had any unpleasant side effects besides its intended purpose. But that's not the real problem. The real problem is that nobody looks. Often when you buy a program you do not have access to the source code, and even if you did, you probably wouldn't bother reading it all, or reading it very carefully. Many programs that run have never been reviewed by anybody. A major advantage offered by the "open source" movement (where all software is made available in source code format) is that even if you don't review it carefully, there is a better chance that someone else will. What does a virus look like? A virus can be installed in just about any program by doing the following: • replace any instruction, say the instruction at location x, by a jump to some free place in memory, say location y; then • write the virus program starting at

location y; then * This is known in the literature as the halting problem, which states that it is impossible in general to tell whether a given program will halt or not. In fact it is impossible in general to discern any nontrivial property of a program. 1.8.1 VIRUSES, WORMS, TROJAN HORSES 15 • place the instruction that was originally at location x at the end of the virus program, followed by a jump to x+1. Besides doing whatever damage the virus program does, it might replicate itself by looking for any executable files in any directory and infecting them. Once an infected program is run, the virus is executed again, to do more damage and to replicate itself to more programs. Most viruses spread silently until some triggering event causes them to wake up and do their dastardly deeds. If they did their dastardly deeds all the time, they wouldn't spread as far. How does a digital pest originally appear on your computer? All it takes is running a single infected program. A program posted on a bulletin board might certainly be infected. But even a program bought legitimately might conceivably have a digital pest. It might have been planted by a disgruntled employee or a saboteur who had broken into the computers of the company and installed the pest into the software before it was shipped. There have been cases where commercial programs were infected because some employee ran a program gotten from a friend or a bulletin board. Often at holiday times people send email messages with attached programs and instructions to run them. While this used to require extracting the email message to a file and possibly processing it first, modern email systems make it very convenient to run such things... often just by clicking on an icon in the message. Often the result is some sort of cute holiday-specific thing, like displaying a picture of a turkey or playing a Christmas carol. It could certainly also contain a virus. Few people will scan the program before running it, especially if the message arrives from a friend. And if you were to run such a program and it did something cute, you might very well forward it to a friend, not realizing that in addition to doing the cute thing it might very well have installed a virus that will start destroying your directory a week after the virus is first run. A good example of a Christmas card email message is a program written by Ian Phillipps, which was a winner of the 1988 International Obfuscated C Code Contest. It is delightful as a Christmas card. It does nothing other than its intended purpose (I1 have analyzed the thing carefully and I2 have complete faith in me1), but we doubt many people would take the time to understand this program before running it (see Figure 1-4). Sometimes you don't realize you're running a program. PostScript is a complete programming language. It is possible to embed a Trojan horse into a PostScript file that infects files with viruses and does other damage. Someone could send you a file and tell you to display it. You wouldn't realize you were running a program by merely displaying the file. And if there was any hope of scanning a C program to find suspicious instructions, there are very few people who could scan a PostScript file and look for suspicious PostScript commands. PostScript is, for all practical purposes, a write-only language. As mail systems get more clever in their attempt to make things more convenient for users, the risk becomes greater. If you receive a PostScript file and you are running a non-clever mail program, you will see delightful crud like Figure 1-5 16 INTRODUCTION 1.8.1 If you wanted to display the file, you'd have to extract the mail message and send it to the printer, or input it into a special program that displays PostScript on your screen. However, a clever mail program might look at the message, recognize that it was PostScript, and automatically run the PostScript code to display the message. Although this is convenient, it is dangerous. There are various other clever features being added to mail programs. Some mail programs allow the sender to send a program along with the mail message. Usually the mail message will arrive and display some sort of icon. If the receiver clicks on the icon, the transmitted program is executed. Someone, to illustrate this point, sent such a mail message. It displayed only as a button /* Have yourself an obfuscated Christmas! */ #include main(t,_,a) char *a; { return!0<t?t<0?t<-72?main(_,t,

"@n'+,#'/*{}w+/w#cdnr/+,{}r/*de}+,/*{*+,/w{%+,/w#q#n+,/#{l,+,/n{n+,/+#n+,/#\
;#q#n+,/+k#;*+,/'r :'d*'3,}{w+K w'K:'+}e#';dq#'l \
q#'+d'K#!/+k#;q#'r}eKK#}w'r}eKK{nl]'/#;#q#n')(}#}w'){)(nl]'/+#n';d}rw' i;# \ ){nl]!/n{n#'; r{#w'r
nc{nl]'/#{l,+'K {rw' iK{;[{nl]'/w#q#n'wk nw' \ iwk{KK{nl]!/w{%'l##w#' i; :{nl]'/*{q#'ld;r'}{nlwb!/*de}'c
\ ;;{nl'-{}rw]'/+,}##'*}#nc,',#nw]'/+kd'+e}+;#'rdq#w! nr'/ ') }+}{rl#'{n' ')# \ }'+}##(!!/'') :t<-
50?_==*a?putchar(31[a]):main(-65,_,a+1):main((*a=='/')+t,_,a+1) :0< confidential < secret < top secret. • A set of zero or more categories (also known as compartments), which describe kinds of information. For instance, the name CRYPTO might mean information about cryptographic algorithms, INTEL might mean information about military intelligence, COMSEC might mean information about communications security, or NUCLEAR might mean information about types of families. Documents (or computer files) are marked with a security label saying how sensitive the information is, and people are issued security clearances according to how trustworthy they are perceived to be and what information they have demonstrated a “need to know.” A clearance might therefore be (SECRET;{COMSEC,CRYPTO}), which would indicate someone who was allowed to know information classified unclassified, confidential, or secret (but not top secret) dealing with cryptographic algorithms or communications security. Given two security labels, (X,S1) and (Y,S2), (X,S1) is defined as being “at least as sensitive as” (Y,S2) iff X ≥ Y and S2 ⊆ S1. For example, (TOP SECRET, {CRYPTO,COMSEC}) > (SECRET, {CRYPTO}) where “>” means “more sensitive than”. It is possible for two labels to be incomparable in the sense that neither is more sensitive than the other. For example, neither of the following are comparable to each other: (TOP SECRET, {CRYPTO,COMSEC}) (SECRET, {NUCLEAR,CRYPTO}) 1.9.3 Mandatory Access Control Rules Every person, process, and piece of information has a security label. A person cannot run a process with a label higher than the person’s label, but may run one with a lower label. Information is only allowed to be read by a process that has at least as high a rating as the information. The terminology 24 INTRODUCTION 1.9.4 used for having a process read something with a higher rating than the process is read-up. Read-up is illegal and must be prevented. A process cannot write a piece of information with a rating lower than the process’s rating. The terminology used for a process writing something with a lower rating than the process is write-down. Write-down is illegal and must be prevented. The rules are: • A human can only run a process that has a security label below or equal to that of the human’s label. • A process can only read information marked with a security label below or equal to that of the process. • A process can only write information marked with a security label above or equal to that of the process. Note that if a process writes information marked with a security label above that of the process, the process can’t subsequently read that information. The prevention of read-up and write-down is the central idea behind mandatory access controls. The concepts of confinement within a security perimeter and a generalized hierarchy of security classes were given a mathematical basis by Bell and La Padula in 1973 [BELL74]. There is significant complexity associated with the details of actually making them work. There has been significant subsequent research on more complex models that capture both the trustworthiness and the confidentiality of data and programs. 1.9.4 Covert Channels A covert channel is a method for a Trojan horse to circumvent the automatic confinement of information within a security perimeter. Let’s assume an operating system has enforced the rules in the previous section. Let’s assume also that a bad guy has successfully tricked someone with a TOP SECRET clearance into running a program with a Trojan horse. The program has access to some sensitive data, and wants to pass the data to the bad guy. We’re assuming the operating system prevents the process from doing this straightforwardly, but there are diabolical methods that theoretically could be employed to get information out. The Trojan horse program cannot directly pass data, but all it needs is for there to be anything it can do that can be

detected by something outside the security perimeter. As long as information can be passed one bit at a time, anything can be transmitted, given enough time. One kind of covert channel is a timing channel. The Trojan horse program alternately loops and waits, in cycles of, say, one minute per bit. When the next bit is a 1, the program loops for one minute. When the next bit is a 0, the program waits for a minute. The bad guy's program running on the same computer but without access to the sensitive data constantly tests the loading of the sys- 1.9.4 THE MULTI-LEVEL MODEL OF SECURITY 25 tem. If the system is sluggish, its conspirator inside the perimeter is looping, and therefore transmitting a 1. Otherwise, the conspirator is waiting, and therefore transmitting a 0. This assumes those two processes are the only ones running on the machine. What happens if there are other processes running and stopping at seemingly random times (from the point of view of the program trying to read the covert channel)? That introduces noise into the channel. But communications people can deal with a noisy channel; it just lowers the potential bandwidth, depending on the signal to noise ratio. Another kind of covert channel called a storage channel involves the use of shared resources other than processor cycles. For instance, suppose there were a queue of finite size, say the print queue. The Trojan horse program could fill the queue to transmit a 1, and delete some jobs to transmit a 0. The covert channel reader would attempt to print something and note whether the request was accepted. Other possible shared resources that might be exploited for passing information include physical memory, disk space, network ports, and I/O buffers. Yet another example depends on how clever the operating system is about not divulging information in error messages. For instance, suppose the operating system says file does not exist when a file really does not exist, but says insufficient privilege for requested operation when the file does exist, but inside a security perimeter off limits to the process requesting to read the file. Then the Trojan horse can alternately create and delete a file of some name known to the other process. The conspirator process periodically attempts to read the file and uses the information about which error message it gets to determine the setting of the next bit of information. There is no general way to prevent all covert channels. Instead, people imagine all the different ways they can think of, and specifically attempt to plug those holes. For instance, the timing channel can be eliminated by giving each security level a fixed percentage of the processor cycles. This is wasteful and impractical in general, because there can be an immense number of distinct classifications (in our model of (one of four levels, {categories}), the number of possible security perimeters is $4 \cdot 2^n$, where n is the number of categories). Most covert channels have very low bandwidth. In many cases, instead of attempting to eliminate a covert channel, it is more practical to introduce enough noise into the system so that the bandwidth becomes too low to be useful to an enemy. It's also possible to look for jobs that appear to be attempting to exploit covert channels (a job that alternately submitted enough print jobs to fill the queue and then deleted them would be suspicious indeed if someone knew to watch). If the bandwidth is low and the secret data is large, and knowing only a small subset of the secret data is not of much use to an enemy, the threat is minimized. How much secret data must be leaked before serious damage is done can vary considerably. For example, assume there is a file with 100 megabytes of secret data. The file has been transmitted, encrypted, on an insecure network. The enemy therefore has the ciphertext, but the cryptographic algorithm used makes it impossible for the enemy to decrypt the data without knowing the key. A Trojan horse with access to the file and a covert channel with a bandwidth of 1 bit every 10 seconds would require 250 years to leak the data (by which time it's hard to believe the divulging of the 26 INTRODUCTION 1.9.5 information could be damaging to anyone). However, if the Trojan horse had access to the 56-bit key, it could leak that information across the covert channel in less than 10 minutes. That information would allow the enemy to decrypt the 100-megabyte file. For this reason, many secure systems go to

great pains to keep cryptographic keys out of the hands of the programs that use them. 1.9.5 The Orange Book The National Computer Security Center (NCSC), an agency of the U.S. government, has published an official standard called “Trusted Computer System Evaluation Criteria”, universally known as “the Orange Book” (guess what color the cover is). The Orange Book defines a series of ratings a computer system can have based on its security features and the care that went into its design, documentation, and testing. This rating system is intended to give government agencies and commercial enterprises an objective assessment of a system’s security and to goad computer manufacturers into placing more emphasis on security. The official categories are D, C1, C2, B1, B2, B3, and A1, which range from least secure to most secure. In reality, of course, there is no way to place all the possible properties in a linear scale. Different threats are more or less important in different environments. The authors of the Orange Book made an attempt to linearize these concerns given their priorities. But the results can be misleading. An otherwise A1 system that is missing some single feature might have a D rating. Systems not designed with the Orange Book in mind are likely to get low ratings even if they are in fact very secure. The other problem with the Orange Book rating scheme is that the designers focused on the security priorities of military security people—keeping data secret. A rating of B1 or better requires implementation of multi-level security and mandatory access controls. In the commercial world, data integrity is at least as important as data confidentiality. Mandatory access controls, even if available, are not suitable for most commercial environments because they make some of the most common operations, such as having a highly privileged user send mail to an unprivileged user, very cumbersome. Mandatory access controls do not by themselves protect the system from infection by viruses. Mandatory access controls allow write-up, so if some unprivileged account became infected by having someone carelessly run, say, a game program loaded from a bulletin board, the virus could spread to more secure areas. Ironically, if it was a very secure area that first got infected, the mandatory access control features would prevent the infection from spreading to the less secure environments. The following is a summary of what properties a system must have to qualify for each rating. 1.9.5 THE MULTI-LEVEL MODEL OF SECURITY 27 D – Minimal Protection. This simply means the system did not qualify for any of the higher ratings; it might actually be very secure. No system is ever going to brag about the fact that it was awarded a D rating. C1 – Discretionary Security Protection. The requirements at this level correspond roughly to what one might expect from a classic timesharing system. It requires • The operating system must prevent unprivileged user programs from overwriting critical portions of its memory. (Note that many PC operating systems do not satisfy this condition.) • Resources must be protected with access controls. Those access controls need not be sophisticated; classic owner/group/world controls would be sufficient. • The system must authenticate users by a password or some similar mechanism, and the password database must be protected so that it cannot be accessed by unauthorized users. There are additional requirements around testing and documentation, which become more detailed at each successive rating. C2 – Controlled Access Protection. This level corresponds roughly to a timesharing system where security is an important concern but users are responsible for their own fates; an example might be a commercial timesharing system. The additional requirements (over those required for C1) for a C2 rating are • access control at a per user granularity—It must be possible to permit access to any selected subset of the user community, probably via ACLs. An ACL is a data structure attached to a resource that specifies the resource’s authorized users. C2 does not explicitly require ACLs, but they are the most convenient way to provide the granularity of protection that C2 requires. • clearing of allocated memory—The operating system must ensure that freshly allocated disk space and memory does not contain “left-over” data deleted by some previous

user. It can do that by writing to the space or by requiring processes to write to the space before they can read it. • auditing—The operating system must be capable of recording security-relevant events, including authentication and object access. The audit log must be protected from tampering and must record date, time, user, object, and event. Auditing must be selective based on user and object. It is reasonable to expect that C2-rateable systems will become ubiquitous, since they contain features that are commonly desired and do not represent an unacceptable overhead. It is somewhat surprising that such systems are not the norm. 28 INTRODUCTION 1.9.5 B1 – Labeled Security Protection. Additional requirements at this level are essentially those required to implement Mandatory Access Controls for secrecy (not integrity) except that little attention is given to covert channels. Requirements for B1 above those for C2 include • Security Labels: Sensitivity labels must be maintained for all users, processes, and files, and read-up and write-down must be prevented by the operating system. • Attached devices must either themselves be labeled as accepting only a single level of information, or they must accept and know how to process security labels. • Attached printers must have a mechanism for ensuring that there is a human-readable sensitivity label printed on the top and bottom of each page corresponding to the sensitivity label of the information being printed. The operating system must enforce this correspondence. B2 – Structured Protection. Beyond B1, there are few new features introduced; rather, the operating system must be structured to greater levels of assurance that it behaves correctly (i.e., has no bugs). Additional requirements for B2 include • trusted path to user—There must be some mechanism to allow a user at a terminal to reliably distinguish between talking to the legitimate operating system and talking to a Trojan horse password-capturing program. • security level changes—A terminal user must be notified when any process started by that user changes its security level. • security kernel—The operating system must be structured so that only a minimal portion of it is security-sensitive, i.e., that bugs in the bulk of the O/S cannot cause sensitive data to leak. This is typically done by running the bulk of the O/S in the processor's user mode and having a secure-kernel mini-O/S which enforces the mandatory access controls. • Covert channels must be identified and their bandwidth estimated, but there is no requirement that they be eliminated. • Strict procedures must be used in the maintenance of the security-sensitive portion of the operating system. For instance, anyone modifying any portion must document what they changed, when they changed it, and why, and some set of other people should compare the updated section with the previous version. B3 – Security Domains. Additional requirements for B3 mostly involve greater assurance that the operating system will not have bugs that might allow something to circumvent mandatory access controls. Additional requirements include 1.9.6 THE MULTI-LEVEL MODEL OF SECURITY 29 • ACLs must be able to explicitly deny access to named individuals even if they are members of groups that are otherwise allowed access. It is only at this level that ACLs must be able to separately enforce modes of access (i.e., read vs. write) to a file. • active audit—There must be mechanisms to detect selected audited events or thresholds of audited events and immediately trigger notification of a security administrator. • secure crashing—The system must ensure that the crashing and restarting of the system introduces no security policy violations. A1 – Verified Design. There are no additional features in an A1 system over a B3 system. Rather, there are formal procedures for the analysis of the design of the system and more rigorous controls on its implementation. 1.9.6 Successors to the Orange Book Gee, I wish we had one of them Doomsday machines —Turgidson, in Dr. Strangelove Governments are rarely willing to adopt one another’s ideas, especially if they didn’t contribute. They would rather develop their own. The publication of the Orange Book in 1983 set off a series of efforts in various countries to come up with their own standards and classifications. These efforts eventually merged in 1990

into a single non-U.S. standard called ITSEC, followed by a reconciliation with the U.S. and the development of a single worldwide standard called the Common Criteria in 1994. Version 2.1 of the Common Criteria became an international standard in 1999. The details of the various rating systems are all different, so passing the bureaucratic hurdles to qualify for a rating under one system would not be much of a head start toward getting an equivalent rating in another (much as different countries don't recognize each other's credentials for practicing medicine). The following table is an oversimplification that we hope won't be too offensive to the advocates of these rating systems, but it does allow the novice to judge what is being claimed about a system. In a partial acknowledgment of the multifaceted nature of security, many of the systems gave two ratings: one for the features provided and one for the degree of assurance that the system implements those features correctly. In practice, like the artistic and technical merit scores at the Olympics, the two scores tend to be closely correlated. 30 INTRODUCTION 1.10 1.10 LEGAL ISSUES The legal aspects of cryptography are fascinating, but the picture changes quickly, and we are certainly not experts in law. Although it pains us to say it, if you're going to do anything involving cryptography, talk to a lawyer. 1.10.1 Patents One legal issue that affects the choice of security mechanisms is patents. Most cryptographic techniques are covered by patents and historically this has slowed their deployment. One of the important criteria for selection of the AES algorithm (see §3.5 Advanced Encryption Standard (AES)) was that it be royalty free. The most popular public key algorithm is RSA (see §6.3 RSA). RSA was developed at MIT, and under the terms of MIT's funding at the time there are no license fees for U.S. government use. It was only patented in the U.S., and licensing was controlled by one company which claimed that the Hellman-Merkle patent also covered RSA, and that patent is international. Interpretation of patent rights varies by country, so the legal issues were complex. At any rate, the last patent on RSA ran out on September 20, 2000. There were many parties on that day. To avoid large licensing fees, many standards attempted to use DSS (see §6.5 Digital Signature Standard (DSS)) instead of RSA. Although in most respects DSS is technically inferior to RSA, when first announced it was advertised that DSS would be freely licensable, i.e., it would not be necessary to reach agreement with the RSA licensing company. But the company claimed Hellman-Merkle covered all public key cryptography, and strengthened its position by acquiring rights TCSEC (Orange Book) German (Green Book) British CLEF ITSEC Common Criteria D Q0 E0 EAL 0 EAL 1 C1 Q1/F1 L1 C1/E1 EAL 2 C2 Q2/F2 L2 C2/E2 CAPP/EAL 3 B1 Q3/F3 L3 B1/E3 CSPP/EAL 4 B2 Q4/F4 L4 B2/E4 EAL 5 B3 Q5/F5 L5 B3/E5 EAL 6 A1 Q6/F5 L6 B3/E6 EAL 7 1.10.2 LEGAL ISSUES 31 to a patent by Schnorr that was closely related to DSS. Until the patents expired, the situation was murky. "I don't know what you mean by your way," said the Queen: "all the ways about here belong to me..." —Through the Looking Glass Some of the relevant patents, luckily all expired, are: • Diffie-Hellman: Patent #4,200,770, issued 1980. This covers the Diffie-Hellman key exchange described in §6.4 Diffie-Hellman. • Hellman-Merkle: Patent #4,218,582, issued 1980. This is claimed to cover all public key systems. There is some controversy over whether this patent should be valid. The specific public key mechanisms described in the patent (knapsack systems) were subsequently broken. • Rivest-Shamir-Adleman: Patent #4,405,829, issued 1983. This covers the RSA algorithm described in §6.3 RSA. • Hellman-Pohlig: Patent #4,424,414, issued 1984. This is related to the Diffie-Hellman key exchange. 1.10.2 Export Controls Mary had a little key (It's all she could export) And all the email that she sent Was opened at the Fort. —Ron Rivest The U.S. government used to impose severe restrictions on export of encryption. This caused much bitterness in the computer industry and led to some fascinating technical designs so that domestic products, which were legally allowed to use strong encryption, could use strong encryption where possible, and yet interoperate with exportable products that were not allowed

to use strong encryption. We describe such designs in this book (see §25.13 Exportability). Luckily, export controls seem to be pretty much lifted. For historical reasons, and in case they become relevant again (we sure hope not), we couldn't bear to delete the following section. It was written for the first edition of this book, and was timely in 1995 when that edition was published: The U.S. government considers encryption to be a dangerous technology, like germ warfare and nuclear weapons. If a U.S. corporation would like to sell to other countries (and the proceeds 32 INTRODUCTION 1.10.2 are not going to be funding the Contras), it needs export approval. The export control laws around encryption are not clear, and their interpretation changes over time. The general principle is that the U.S. government does not want you to give out technology that would make it more difficult for them to spy. Sometimes companies get so discouraged that they leave encryption out of their products altogether. Sometimes they generate products that, when sold overseas, have the encryption mechanisms removed. It is usually possible to get export approval for encryption if the key lengths are short enough for the government to brute-force check all possible keys to decrypt a message. So sometimes companies just use short keys, or sometimes they have the capability of varying the key length, and they fix the key length to be shorter when a system is sold outside the U.S. Even if you aren't in the business of selling software abroad, you can run afoul of export controls. If you install encryption software on your laptop and take it along with you on an international trip, you may be breaking the law. If you distribute encryption software within the U.S. without adequate warnings, you are doing your customers a disservice. And the legality of posting encryption software on a public network is questionable. PART 1 CRYPTOGRAPHY This page is intentionally left blank 35 2 INTRODUCTION TO CRYPTOGRAPHY 2.1 WHAT IS CRYPTOGRAPHY? The word cryptography comes from the Greek words κρυπτο (hidden or secret) and γραφη (writing). So, cryptography is the art of secret writing. More generally, people think of cryptography as the art of mangling information into apparent unintelligibility in a manner allowing a secret method of unmangling. The basic service provided by cryptography is the ability to send information between participants in a way that prevents others from reading it. In this book we will concentrate on the kind of cryptography that is based on representing information as numbers and mathematically manipulating those numbers. This kind of cryptography can provide other services, such as • integrity checking—reassuring the recipient of a message that the message has not been altered since it was generated by a legitimate source • authentication—verifying someone's (or something's) identity But back to the traditional use of cryptography. A message in its original form is known as plaintext or cleartext. The mangled information is known as ciphertext. The process for producing ciphertext from plaintext is known as encryption. The reverse of encryption is called decryption. While cryptographers invent clever secret codes, cryptanalysts attempt to break these codes. These two disciplines constantly try to keep ahead of each other. Ultimately, the success of the cryptographers rests on the Fundamental Tenet of Cryptography If lots of smart people have failed to solve a problem, then it probably won't be solved (soon). plaintext ciphertext plaintext encryption decryption 36 INTRODUCTION TO CRYPTOGRAPHY 2.1.1 Cryptographic systems tend to involve both an algorithm and a secret value. The secret value is known as the key. The reason for having a key in addition to an algorithm is that it is difficult to keep devising new algorithms that will allow reversible scrambling of information, and it is difficult to quickly explain a newly devised algorithm to the person with whom you'd like to start communicating securely. With a good cryptographic scheme it is perfectly OK to have everyone, including the bad guys (and the cryptanalysts) know the algorithm because knowledge of the algorithm without the key does not help unmangle the information. The concept of a key is analogous to the combination for a combination lock. Although the concept of a combination lock is well known (you dial in the secret numbers in the

correct sequence and the lock opens), you can't open a combination lock easily without knowing the combination. 2.1.1 Computational Difficulty It is important for cryptographic algorithms to be reasonably efficient for the good guys to compute. The good guys are the ones with knowledge of the keys.* Cryptographic algorithms are not impossible to break without the key. A bad guy can simply try all possible keys until one works. The security of a cryptographic scheme depends on how much work it is for the bad guy to break it. If the best possible scheme will take 10 million years to break using all of the computers in the world, then it can be considered reasonably secure. Going back to the combination lock example, a typical combination might consist of three numbers, each a number between 1 and 40. Let's say it takes 10 seconds to dial in a combination. That's reasonably convenient for the good guy. How much work is it for the bad guy? There are 403 possible combinations, which is 64000. At 10 seconds per try, it would take a week to try all combinations, though on average it would only take half that long (even though the right number is always the last one you try!). Often a scheme can be made more secure by making the key longer. In the combination lock analogy, making the key longer would consist of requiring four numbers to be dialed in. This would make a little more work for the good guy. It might now take 13 seconds to dial in the combination. But the bad guy has 40 times as many combinations to try, at 13 seconds each, so it would take a year to try all combinations. (And if it took that long, he might want to stop to eat or sleep). With cryptography, computers can be used to exhaustively try keys. Computers are a lot faster than people, and they don't get tired, so thousands or millions of keys can be tried per second. Also, lots of keys can be tried in parallel if you have multiple computers, so time can be saved by spending money on more computers. Sometimes a cryptographic algorithm has a variable-length key. It can be made more secure by increasing the length of the key. Increasing the length of the key by one bit makes the good guy's *We're using the terms good guys for the cryptographers, and bad guys for the cryptanalysts. This is a convenient shorthand and not a moral judgment—in any given situation, which side you consider good or bad depends on your point of view. 2.1.2 WHAT IS CRYPTOGRAPHY? 37 job just a little bit harder, but makes the bad guy's job up to twice as hard (because the number of possible keys doubles). Some cryptographic algorithms have a fixed-length key, but a similar algorithm with a longer key can be devised if necessary. If computers get 1000 times faster, so that the bad guy's job becomes reasonably practical, making the key 10 bits longer will make the bad guy's job as hard as it was before the advance in computer speed. However, it will be much easier for the good guys (because their computer speed increase far outweighs the increment in key length). So the faster computers get, the better life gets for the good guys. Keep in mind that breaking the cryptographic scheme is often only one way of getting what you want. For instance, a bolt cutter works no matter how many digits are in the combination. You can get further with a kind word and a gun than you can with a kind word alone. —Willy Sutton, bank robber 2.1.2 To Publish or Not to Publish Some people believe that keeping a cryptographic algorithm as secret as possible will enhance its security. Others argue that publishing the algorithm, so that it is widely known, will enhance its security. On the one hand, it would seem that keeping the algorithm secret must be more secure—it makes for more work for the cryptanalyst to try to figure out what the algorithm is. The argument for publishing the algorithm is that the bad guys will probably find out about it eventually anyway, so it's better to tell a lot of nonmalicious people about the algorithm so that in case there are weaknesses, a good guy will discover them rather than a bad guy. A good guy who discovers a weakness will warn people that the system has a weakness. Publication provides an enormous amount of free consulting from the academic community as cryptanalysts look for weaknesses so they can publish papers about them. A bad guy who discovers a weakness will exploit it for doing bad-guy things like embezzling money or

stealing trade secrets. It is difficult to keep the algorithm secret because if an algorithm is to be widely used, it is highly likely that determined attackers will manage to learn the algorithm by reverse engineering whatever implementation is distributed, or just because the more people who know something the more likely it is for the information to leak to the wrong places. In the past, “good” cryptosystems were not economically feasible, so keeping the algorithms secret was needed extra protection. We believe (we hope?) today’s algorithms are sufficiently secure that this is not necessary. Common practice today is for most commercial cryptosystems to be published and for military cryptosystems to be kept secret. If a commercial algorithm is unpublished today, it’s probably for trade secret reasons or because this makes it easier to get export approval rather than to enhance its security. We suspect the military ciphers are unpublished mainly to keep good cryptographic methods out of the hands of the enemy rather than to keep them from cryptanalyzing our codes. 38 INTRODUCTION TO CRYPTOGRAPHY 2.1.3 2.1.3 Secret Codes We use the terms secret code and cipher interchangeably to mean any method of encrypting data. Some people draw a subtle distinction between these terms that we don’t find useful. The earliest documented cipher is attributed to Julius Caesar. The way the Caesar cipher would work if the message were in English is as follows. Substitute for each letter of the message, the letter which is 3 letters later in the alphabet (and wrap around to A from Z). Thus an A would become a D, and so forth. For instance, DOZEN would become GRCHQ. Once you figure out what’s going on, it is very easy to read messages encrypted this way (unless, of course, the original message was in Greek). A slight enhancement to the Caesar cipher was distributed as a premium with Ovaltine in the 1940s as Captain Midnight Secret Decoder rings. (There were times when this might have been a violation of export controls for distributing cryptographic hardware!) The variant is to pick a secret number n between 1 and 25, instead of always using 3. Substitute for each letter of the message, the letter which is n higher (and wrap around to A from Z of course). Thus if the secret number was 1, an A would become a B, and so forth. For instance HAL would become IBM. If the secret number was 25, then IBM would become HAL. Regardless of the value of n, since there are only 26 possible ns to try, it is still very easy to break this cipher if you know it’s being used and you can recognize a message once it’s decrypted. The next type of cryptographic system developed is known as a monoalphabetic cipher, which consists of an arbitrary mapping of one letter to another letter. There are 26! possible pairings of letters, which is approximately 4×1026. [Remember, n!, which reads “n factorial”, means n(n−1)(n−2)⋯1.] This might seem secure, because to try all possibilities, if it took 1 microsecond to try each one, would take about 10 trillion years. However, by statistical analysis of language (knowing that certain letters and letter combinations are more common than others), it turns out to be fairly easy to break. For instance, many daily newspapers have a daily cryptogram, which is a monoalphabetic cipher, and can be broken by people who enjoy that sort of thing during their subway ride to work. An example is Cf lqr’xs xsnyctm n eqxxqgsy iqul qf wdcp eqqh, erl lqrx qgt iqul! Computers have made much more complex cryptographic schemes both necessary and possible. Necessary because computers can try keys at a rate that would exhaust an army of clerks; and possible because computers can execute the complex algorithms quickly and without errors. 2.2 BREAKING AN ENCRYPTION SCHEME 39 2.2 BREAKING AN ENCRYPTION SCHEME What do we mean when we speak of a bad guy Fred breaking an encryption scheme? The three basic attacks are known as ciphertext only, known plaintext, and chosen plaintext. 2.2.1 Ciphertext Only In a ciphertext only attack, Fred has seen (and presumably stored) some ciphertext that he can analyze at leisure. Typically it is not difficult for a bad guy to obtain ciphertext. (If a bad guy can’t access the encrypted data, then there would have been no need to encrypt the data in the first place!) How can Fred figure out the plaintext if all he can see is the ciphertext? One possible strategy is to search through all the

keys. Fred tries the decrypt operation with each key in turn. It is essential for this attack that Fred be able to recognize when he has succeeded. For instance, if the message was English text, then it is highly unlikely that a decryption operation with an incorrect key could produce something that looked like intelligible text. Because it is important for Fred to be able to differentiate plaintext from gibberish, this attack is sometimes known as a recognizable plaintext attack. It is also essential that Fred have enough ciphertext. For instance, using the example of a monoalphabetic cipher, if the only ciphertext available to Fred were XYZ, then there is not enough information. There are many possible letter substitutions that would lead to a legal three-letter English word. There is no way for Fred to know whether the plaintext corresponding to XYZ is THE or CAT or HAT. As a matter of fact, in the following sentence, any of the words could be the plaintext for XYZ: The hot cat was sad but you may now sit and use her big red pen. [Don't worry—we've found a lovely sanatorium for the coauthor who wrote that. —the other coauthors] Often it isn't necessary to search through a lot of keys. For instance, the authentication scheme Kerberos (see §11.4 Logging Into the Network) assigns to user Alice a DES key derived from Alice's password according to a straightforward, published algorithm. If Alice chooses her password unwisely (say a word in the dictionary), then Fred does not need to search through all 256 possible DES keys—instead he only needs to try the derived keys of the 10000 or so common English words. A cryptographic algorithm has to be secure against a ciphertext only attack because of the accessibility of the ciphertext to cryptanalysts. But in many cases cryptanalysts can obtain additional information, so it is important to design cryptographic systems to withstand the next two attacks as well. 40 INTRODUCTION TO CRYPTOGRAPHY 2.2.2 2.2.2 Known Plaintext Sometimes life is easier for the attacker. Suppose Fred has somehow obtained some ⟨plaintext, ciphertext⟩ pairs. How might he have obtained these? One possibility is that secret data might not remain secret forever. For instance, the data might consist of specifying the next city to be attacked. Once the attack occurs, the plaintext to the previous day's ciphertext is now known. With a monoalphabetic cipher, a small amount of known plaintext would be a bonanza. From it, the attacker would learn the mappings of a substantial fraction of the most common letters (every letter that was used in the plaintext Fred obtained). Some cryptographic schemes might be good enough to be secure against ciphertext only attacks but not good enough against known plaintext attacks. In these cases, it becomes important to design the systems that use such a cryptographic algorithm to minimize the possibility that a bad guy will ever be able to obtain ⟨plaintext, ciphertext⟩ pairs. 2.2.3 Chosen Plaintext On rare occasions, life may be easier still for the attacker. In a chosen plaintext attack, Fred can choose any plaintext he wants, and get the system to tell him what the corresponding ciphertext is. How could such a thing be possible? Suppose the telegraph company offered a service in which they encrypt and transmit messages for you. Suppose Fred had eavesdropped on Alice's encrypted message. Now he'd like to break the telegraph company's encryption scheme so that he can decrypt Alice's message. He can obtain the corresponding ciphertext to any message he chooses by paying the telegraph company to send the message for him, encrypted. For instance, if Fred knew they were using a monoalphabetic cipher, he might send the message The quick brown fox jumps over the lazy dog. knowing that he would thereby get all the letters of the alphabet encrypted and then be able to decrypt with certainty any encrypted message. It is possible that a cryptosystem secure against ciphertext only and known plaintext attacks might still be susceptible to chosen plaintext attacks. For instance, if Fred knows that Alice's message is either Surrender or Fight on, then no matter how wonderful an encryption scheme the telegraph company is using, all he has to do is send the two messages and see which one looks like the encrypted data he saw when Alice's message was transmitted. A cryptosystem should resist all three sorts of attacks. That way its users don't need to worry

about whether there are any opportunities for attackers to know or choose plaintext. Like wearing both a belt and suspenders, many systems that use cryptographic algorithms will also go out of their way to prevent any chance of chosen plaintext attacks. 2.3 TYPES OF CRYPTOGRAPHIC FUNCTIONS 41 2.3 TYPES OF CRYPTOGRAPHIC FUNCTIONS There are three kinds of cryptographic functions: hash functions, secret key functions, and public key functions. We will describe what each kind is, and what it is useful for. Public key cryptography involves the use of two keys. Secret key cryptography involves the use of one key. Hash functions involve the use of zero keys! Try to imagine what that could possibly mean, and what use it could possibly have—an algorithm everyone knows with no secret key, and yet it has uses in security. Since secret key cryptography is probably the most intuitive, we'll describe that first. 2.4 SECRET KEY CRYPTOGRAPHY Secret key cryptography involves the use of a single key. Given a message (called plaintext) and the key, encryption produces unintelligible data (called an IRS Publication—no! no! that was just a finger slip, we meant to say "ciphertext"), which is about the same length as the plaintext was. Decryption is the reverse of encryption, and uses the same key as encryption. Secret key cryptography is sometimes referred to as conventional cryptography or symmetric cryptography. The Captain Midnight code and the monoalphabetic cipher are both examples of secret key algorithms, though both are easy to break. In this chapter we describe the functionality of cryptographic algorithms, but not the details of particular algorithms. In Chapter 3 Secret Key Cryptography we describe the details of some popular secret key cryptographic algorithms. 2.4.1 Security Uses of Secret Key Cryptography The next few sections describe the types of things one might do with secret key cryptography. plaintext ciphertext encryption ciphertext plaintext decryption key 42 INTRODUCTION TO CRYPTOGRAPHY 2.4.2 2.4.2 Transmitting Over an Insecure Channel It is often impossible to prevent eavesdropping when transmitting information. For instance, a telephone conversation can be tapped, a letter can be intercepted, and a message transmitted on a LAN can be received by unauthorized stations. If you and I agree on a shared secret (a key), then by using secret key cryptography we can send messages to one another on a medium that can be tapped, without worrying about eavesdroppers. All we need to do is have the sender encrypt the messages and the receiver decrypt them using the shared secret. An eavesdropper will only see unintelligible data. This is the classic use of cryptography. 2.4.3 Secure Storage on Insecure Media If I have information I want to preserve but which I want to ensure no one else can look at, I have to be able to store the media where I am sure no one can get it. Between clever thieves and court orders, there are very few places that are truly secure, and none of these is convenient. If I invent a key and encrypt the information using the key, I can store it anywhere and it is safe so long as I can remember the key. Of course, forgetting the key makes the data irrevocably lost, so this must be used with great care. 2.4.4 Authentication In spy movies, when two agents who don't know each other must rendezvous, they are each given a password or pass phrase that they can use to recognize one another. This has the problem that anyone overhearing their conversation or initiating one falsely can gain information useful for replaying later and impersonating the person to whom they are talking. The term strong authentication means that someone can prove knowledge of a secret without revealing it. Strong authentication is possible with cryptography. Strong authentication is particularly useful when two computers are trying to communicate over an insecure network (since few people can execute cryptographic algorithms in their heads). Suppose Alice and Bob share a key KAB and they want to verify they are speaking to each other. They each pick a random number, which is known as a challenge. Alice picks rA. Bob picks rB. The value x encrypted with the key KAB is known as the response to the challenge x. How Alice and Bob use challenges and responses to authenticate each other is shown in Figure 2-1. If someone, say Fred, were impersonating Alice,

he could get Bob to encrypt a value for him (though Fred wouldn't be able to tell if the person he was talking to was really Bob), but this information would not be useful later in impersonating Bob to the real Alice because the real Alice 2.4.5 SECRET KEY CRYPTOGRAPHY 43 would pick a different challenge. If Alice and Bob complete this exchange, they have each proven to the other that they know KAB without revealing it to an impostor or an eavesdropper. Note that in this particular protocol, there is the opportunity for Fred to obtain some ⟨chosen plaintext, ciphertext⟩ pairs, since he can claim to be Bob and ask Alice to encrypt a challenge for him. For this reason, it is essential that challenges be chosen from a large enough space, say 264 values, so that there is no significant chance of using the same one twice. That is the general idea of a cryptographic authentication algorithm, though this particular algorithm has a subtle problem that would prevent it from being useful in most computer-to-computer cases. (We would have preferred not bringing that up, but felt we needed to say that so as not to alarm people who already know this stuff and who would realize the protocol was not secure. Protocol flaws such as this, and methods of fixing them, are discussed in Chapter 9 Security Handshake Pitfalls.) 2.4.5 Integrity Check A secret key scheme can be used to generate a fixed-length cryptographic checksum associated with a message. This is a rather nonintuitive use of secret key technology. What is a checksum? An ordinary (noncryptographic) checksum protects against accidental corruption of a message. The original derivation of the term checksum comes from the operation of breaking a message into fixed-length blocks (for instance, 32-bit words) and adding them up. The sum is sent along with the message. The receiver similarly breaks up the message, repeats the addition, and checks the sum. If the message had been garbled en route, the sum will not match the sum sent and the message is rejected, unless, of course, there were two or more errors in the transmission that canceled one another. It turns out this is not terribly unlikely, given that if flaky hardware turns a bit off somewhere, it is likely to turn a corresponding bit on somewhere else. To protect against such "regular" flaws in hardware, more complex checksums called CRCs were devised. But these still only protect against faulty hardware and not an intelligent attacker. Since CRC algorithms are published, an attacker who wanted to change a message could do so, compute the CRC on the new message, and send that along. Alice Bob rA rA encrypted with KAB rB rB encrypted with KAB Figure 2-1. Challenge–response authentication with shared secret 44 INTRODUCTION TO CRYPTOGRAPHY 2.5 To provide protection against malicious changes to a message, a secret checksum algorithm is required, such that an attacker not knowing the algorithm can't compute the right checksum for the message to be accepted as authentic. As with encryption algorithms, it's better to have a common (known) algorithm and a secret key. This is what a cryptographic checksum does. Given a key and a message, the algorithm produces a fixed-length message authentication code (MAC) that can be sent with the message. A MAC is often called a MIC (message integrity code). We prefer the term MIC, and MIC is used in standards such as PEM (see Chapter 18 PEM & S/MIME), but the term MAC seems to have become more popular. If anyone were to modify the message, and they didn't know the key, they would have to guess a MAC and the chance of getting it right depends on the length. A typical MAC is at least 48 bits long, so the chance of getting away with a forged message is only one in 280 trillion (or about the chance of going to Las Vegas with a dime and letting it ride on red at the roulette table until you have enough to pay off the U.S. national debt). Such message integrity codes have been in use to protect the integrity of large interbank electronic funds transfers for quite some time. The messages are not kept secret from an eavesdropper, but their integrity is ensured. 2.5 PUBLIC KEY CRYPTOGRAPHY Public key cryptography is sometimes also referred to as asymmetric cryptography. Public key cryptography is a relatively new field, invented in 1975 [DIFF76b] (at least that's the first published record—it is rumored that NSA or similar