Names, suppose a cross-certificate from A/B/C to A/B/D/X indicates how many levels up from the issuer the least common ancestor is (in this case, one level up, since the parent is A/B and is also an ancestor of A/B/D/X), followed by the name down from there. So the cross-certificate from A/B/C to A/B/D/X would contain "../D/X" (go up from my name one level, and then go down to D/X). In what case of namespace reorganization would this scheme not require the cross-certificate to be reissued, whereas with an absolute name it would? In what case would the absolute name certificate remain valid whereas the scheme proposed in this problem would need to be reissued? 3. What name constraints would you put into PKIX certificates to build the anarchy model? 4. Compare the following schemes for obtaining Bob's public key, in terms of bandwidth and computation efficiency, security, flexibility, and any other criteria you can think of: downloading Bob's key from the node located at a particular IP address (via an unauthenticated interaction), looking up Bob's key in a directory via an unauthenticated interaction, having an authenticated conversation to the directory, having the directory sign the information you request, storing and retrieving certificates from the directory, having no directory but having each principal responsible for keeping its own certificate and sending it to someone who needs to talk to it. 5. Why must a CRL be reissued periodically, even when no new certificates have been revoked? 6. If there is a revocation mechanism, why do certificates need an expiration date? 7. Compare revocation schemes of verifiers downloading complete CRLs, clients obtaining non-revocation certificates, and verifiers checking individual validity status. Consider in overhead (on bandwidth, verifiers, clients, the OLRS) and revocation timeliness. Consider factors such as how many clients a verifier serves and how many services a client visits. 8. Why is it important in a good-list revocation scheme to keep hashes of the valid certificates, rather than just their serial numbers? 9. Show how (using policies and policy mapping as described in §13.3.11 Policies in Certificates), it is possible for the only legal path to contain a loop. Show a path that requires going around a loop twice in order to meet the policy requirements. 364 PKI (PUBLIC KEY INFRASTRUCTURE) 13.9 10. Discuss mechanisms for support of groups, and compare them for ability to support large numbers of groups and members, cross-organizational groups, dynamically changing groups, and quick revocation and group joining. 365 14 REAL-TIME COMMUNICATION SECURITY A real-time protocol is one in which the parties negotiate interactively to authenticate each other and establish a session key, in contrast to a protocol such as email in which one party prepares a message that can later be decrypted and authenticated by the intended recipient. Standards for realtime public-key-based security protocols include IPsec, SSL/TLS, and SSH. In this chapter we cover the problems and solutions generically, rather than diving into the idiosyncracies of particular protocols, because the actual protocols (especially the ones defined by committee) are gratuitously complex. The subsequent chapters deal with the specifics of IPsec and SSL/TLS. At a minimum, the protocols provide mutual authentication between Alice and Bob and establish a session key for cryptographic protection of the subsequent conversation. The conversation protected with that session key is known as a security association. Other features such protocols might provide, and which are explained in this chapter, are perfect forward secrecy, clogging protection, escrow-foilage, and endpoint identifier hiding. This chapter also discusses the differences between protocols such as IPsec, which is said to be "implemented at layer 3", and protocols such as SSL/TLS and SSH, which are said to be "implemented at layer 4". 14.1 WHAT LAYER? Definition of a layer n protocol: anything defined by a committee whose charter is to define a layer n protocol. —Radia Perlman (in Interconnections: Bridges, Routers, Switches, and Internetworking Protocols) SSL/TLS and SSH are said to be "implemented at layer 4", whereas IPsec is said to be "implemented at layer 3". What does it mean that a real-time communication security protocol is imple- 366 REAL-TIME COMMUNICATION SECURITY

14.1 mented at layer 3 vs. layer 4, and what are the implications? (For a definition of the layers, see §1.5.1 OSI Reference Model.) For brevity, since for the principles discussed in this chapter, SSL/TLS and SSH have the same properties, in this chapter we'll simply say SSL when we mean any of the protocols SSL, TLS, or SSH. Many IP stacks are implemented so that layer 4 (e.g., TCP) and below are implemented in the operating system, and anything above is implemented in a user process. The philosophy of SSL is that it is easier to deploy something if you don't have to change the operating system, so these protocols act "above TCP". It requires the applications to interface to SSL instead of TCP. The name Secure Sockets Layer comes from the most popular API to TCP, which is called "sockets". The API to SSL is a superset of the API to TCP sockets. Modifying an application to work on top of SSL requires minimal changes. Of course, the security benefits are limited if the richer API is not used. Although the applications have to be modified (albeit minimally), the operating system, which includes TCP and the layers below it, does not need to be modified. "Transport Layer Security" is somewhat of a misnomer because rather than being at layer 4, these protocols tend to act like a layer on top of layer 4. In contrast, the philosophy behind IPsec is that implementing security within the operating system automatically causes all applications to be protected without the applications having to be modified. There is a problem in operating above TCP. Since TCP will not be participating in the cryptography, it will have no way of noticing if malicious data is inserted into the packet stream, as long applications applications SSL TCP TCP OS IPsec IP IP lower layers lower layers SSL/TLS or SSH operate above TCP. OS doesn't change. Applications do. IPsec is within the OS. OS changes. Applications and API to TCP are unchanged. Figure 14-1. Implementing at layer 3 vs. layer 4 14.1 WHAT LAYER? 367 as the bogus data passes the (noncryptograpic) TCP checksum. TCP will acknowledge such data and send it up to SSL. SSL will discard it because the integrity check will indicate the data is bogus, but there is no way for SSL to tell TCP to accept the real data at this point. When the real data arrives, TCP will assume it is duplicate data and discard it, since it will have the same sequence numbers as the bogus data. SSL has no choice but to close the connection, since it can no longer provide the service that the API claims it offers, namely a lossless stream of data. An attacker can launch a successful denial-of-service attack by inserting a single packet into the data stream. IPsec's approach of cryptographically protecting each packet independently can better protect against such an attack. In contrast, IPsec's constraint of not modifying the applications winds up with its own serious problem. With the current commonly used API, IP tells the application only what IP address it is talking to, not what user is on the other end. So IPsec, with the current API, cannot tell the application anything more than which IP address is on the other end, even though IPsec is capable of authenticating an individual user. Most applications need to distinguish between users. If IPsec has authenticated the user, it could in theory tell the application the user's name, but that would require changing the API and the application. Implementing IPsec without changing the application has the same effect as putting firewalls between the two systems and implementing IPsec between the firewalls. It accomplishes the following: • It causes the traffic on the path between the communicating parties to be encrypted, hiding it from eavesdroppers. • As with firewalls, IPsec can access a policy database similar to what a firewall can access. For instance, it can specify which IP addresses are allowed to talk to which other IP addresses, or which TCP ports should be allowed or excluded. This is true whether the endpoint of the IPsec connection is a firewall or an endnode (see §23.1 Packet Filters). • Some applications do authentication based on IP addresses (see §9.2 Address-Based Authentication). The API informs the application of the IP address from which information was received. Before IPsec, the source IP address was assumed based solely on the value of the SOURCE ADDRESS field in the IP header. With IPsec, address-based authentication becomes much more secure because one of the types of endpoint identifiers IPsec can authenticate is an IP address. What IPsec (with the current API and an unmodified application) does not accomplish for the application is authentication of anything other than IP addresses. Most principals would have some identity such as a name, and be allowed to access the network from a variety of IP addresses. In these cases, the most likely scenario for using IPsec is that IPsec would do its highly secure and expensive authentication, authenticating based on the user's public key and establishing a security association to the user's name, but would have no way of telling the application who is on the other 368 REAL-TIME COMMUNICATION SECURITY 14.2 side. The application would have to depend on existing mechanisms, most likely a name and password, to determine which user it is talking to. IPsec is still of value in this scenario in which the (unmodified) application authenticates the user based on name and password, since the name and password will now be encrypted when transmitted. To take full advantage of IPsec, applications will have to change. The API has to change in order to pass identities other than IP addresses, and the applications have to change to make use of this information. So the best solution is one where both the operating system and the applications are modified. This illustrates why security would best be done by being designed in from the start, rather than being added in with the least modification to an existing implementation. One other advantage of the packet-by-packet cryptographic handling of IPsec is that it is easier to build an outboard device that does IPsec processing. To implement an SSL-type protocol that operates over TCP in an outboard device, the device would have to implement TCP, including buffering out-of-order packets. 16.2 SESSION KEY ESTABLISHMENT In Chapter 11 Security Handshake Pitfalls we discuss various protocols and pitfalls for doing mutual authentication. Cryptographic mutual authentication is certainly an improvement over passwords in the clear, but it is also important to cryptographically protect the conversation after the initial handshake. Obviously it is desirable to protect the data from disclosure or modification. But another vulnerability if cryptographic protection ends after the initial handshake is session hijacking, in which someone takes over Alice's session to Bob by forging Alice's IP address as the source address on packets sent to Bob, and using TCP sequence numbers larger than what Alice would be using. Without cryptographic protection, Bob can't distinguish these packets from authentic packets from Alice. Once Bob accepts the attacker's larger TCP sequence numbers, Alice's data just gets ignored as duplicate data. It looks to Alice like the connection breaks, but the attacker is now logged in as Alice, and can do anything Alice is allowed to do. So, a session key is used after the initial mutual authentication to cryptographically protect the conversation from disclosure, modification, or hijacking. A sequence number is typically used to prevent replay or reordering. A new session key should be established for each new session, and during a session, if the sequence number might be reused (i.e., wrap around). The session key must be unpredictable to an attacker. (For instance, Goldberg and Wagner [GOLD96] reverse engineered an implementation of the SSL protocol and discovered that the session key was based on three quantities: the time of day in units of microseconds, the process id, and the parent process id. This meant that an attacker who knew the approximate time the session was 14.3 PERFECT FORWARD SECRECY 369 created had a feasible search space to search for keys. Even though the key was 128 bits long, given a bit of cleverness to narrow down the potential process ids, the attacker would only have to search from an approximately 30-bit space to find the key. Both communicating parties should contribute to the session key, for instance by having each side send a value to the other, encrypted with the other side's public key, and using a hash of the two values as the session key. This rule makes it less likely that the protocol will have flaws in which someone will be able to impersonate one side or the other in a replay attack. For instance, in a protocol in which Alice sends the session key to Bob, encrypted with his public key, someone impersonating Alice can simply replay all Alice's messages. Another reason for having both sides contribute to the session key is that if either side has a good random number generator, then the session key will be sufficiently random. 14.3 PERFECT FORWARD SECRECY A protocol is said to have perfect forward secrecy (PFS) if it is impossible for an eavesdropper (Sam) to decrypt a conversation between Alice and Bob even if Sam records the entire encrypted session, and then subsequently breaks into both Alice and Bob and steals their long-term secrets. The trick to achieving perfect forward secrecy is to generate a temporary session key, not derivable from information stored at the node after the session concludes, and then forget it after the session concludes. If the session will last for a long time, it is common to generate and forget keys periodically so that even if Sam seizes Alice's and Bob's computers while the conversation is still going on, he will not be able to decrypt messages received before the last key rollover. Protocol 14-2 is an example of a protocol with perfect forward secrecy. It uses Diffie-Hellman to agree on a session key, which achieves perfect forward secrecy assuming both sides generate an unpredictable DiffieHellman private number and forget both the private number and the agreed-upon session key after the session ends. Also, each side signs the Diffie-Hellman quantity to foil a man-in-the-middle attack (see §6.4.2 Defenses Against Man-in-the-Middle Attack). In the first two messages, each side identifies itself, and supplies a Diffie-Hellman value signed by its private key. In the next two messages, each side proves knowledge of the agreed-upon Diffie-Hellman value gab mod p by sending a hash of it, with each side sending a different hash. If each side forgets gab mod p and its private Diffie-Hellman number (a or b) after the session, there is no way for anyone to reconstruct gab mod p from knowledge of both long-term private keys and the entire recorded conversation. What kind of protocol would not have perfect forward secrecy? Examples include 370 REAL-TIME COMMUNICATION SECURITY 14.3 • Alice sends all messages for Bob encrypted with Bob's public key, and Bob sends all messages for Alice encrypted with Alice's public key. • Kerberos (since the session key is inside the ticket to Bob, and is encrypted with Bob's longterm secret). • Alice chooses the session key, and sends it to Bob, encrypted with Bob's public key. Perfect forward secrecy might seem like it only protects against a fairly obscure threat. However, protocols designed with perfect forward secrecy usually have another property, which is particularly popular with the IETF crowd, which we'll call escrow-foilage. This means that even if the forces of darkness (and we make no value judgments here), have required Alice and Bob to give their long-term private keys to some benign, completely trustworthy organization, the conversation between Alice and Bob will still be secret between only Alice and Bob. In other words, even with prior knowledge of Alice and Bob's long-term keys, a passive eavesdropper cannot decrypt the conversation. Of course if Sam has prior knowledge of all of Alice's and Bob's secrets, then he can impersonate Alice or Bob, and perhaps trick them into divulging what they would have divulged in the conversation. Maybe you'd think Alice and Bob could start off the conversation by asking each other a few personal questions like, "What café did we meet at in Paris?" but Sam could be acting as an active man-in-the-middle, decrypting and reencrypting the traffic, relaying the personal questions and answers, and this would be very difficult to detect. Anything with perfect forward secrecy will also have escrow-foilage against a passive attack, since anything you can do by having recorded the conversation and learned the secret later you can also do knowing the secrets in advance and eavesdropping in real-time. But often with escrowed systems a user has two public key pairs, one for encryption and one for signatures. And in those Alice ["Alice", ga mod p]Alice Bob message 1 ["Bob", gb mod p]Bob message 2 hash(gab mod p) message 3 hash(1, gab mod p) message 4 Protocol 14-2. Diffie-Hellman for perfect forward secrecy using signature keys 14.4 PFS-FOILAGE 371 cases, only the encryption key is escrowed, since law enforcement would like to decrypt data, but does not need the ability to forge signatures. It would be counterproductive for them to have a user's private signature key, because then the user can repudiate his signature since anyone else with access to the key might have signed the message. So assuming the signature keys are not escrowed, then Protocol 14-2 will have escrow-foilage even against active attacks. 14.4 PFS-FOILAGE Even if the protocol is designed for perfect forward secrecy and escrow-foilage, an implementation can fail to achieve those properties. I2 am familiar with a design proposed by someone who wanted to ensure, even in the presence of the types of protocols designed by "those anarchist IETFers," that the ability of Big Brother to monitor traffic would not be impeded. This was done by having the client machines generate all random numbers based on a seed provided by (and stored at) a server. The assumption was that the servers were managed by completely trustworthy individuals, all servers were physically protected at all times, and there were no security vulnerabilities in servers to be exploited, so it was okay that the servers would be able to decrypt all encrypted data. The thing that annoyed me2 most about the design was the claim that forcing the client machine to generate all random numbers from a seed escrowed at the server would enhance security, because "the client machine was incapable of generating as high-quality a random seed as the server machine could generate." Some problems with this reasoning: • The client machine is actually more capable of generating random numbers since it has access to a nice source of randomness, a human, providing all sorts of unpredictable inputs, e.g., keystrokes and mouse movements. • If the server really wanted to "help out" by giving the client a really high quality seed, the client machine should merely use that as one more of the inputs to its random number generator, say by hashing the received seed with the random number the client would have generated without the server's help. • The necessity for the server to securely send the seed to the client is another opportunity for vulnerabilities and design complexity. But of course the real reason for this design was to provide PFS-foilage, not to enhance security. 372 REAL-TIME COMMUNICATION SECURITY 14.5 14.5 DENIAL-OF-SERVICE/CLOGGING PROTECTION Even if Trudy can't impersonate Alice without knowing her secret, in many protocols she can lock out legitimate users by forcing Bob to use up all his state or computation resources with authentication attempts. This sort of attack happens on the Internet, and the impostor often launches the attack packets with forged IP addresses. Using forged IP source addresses makes it difficult to catch the attacker, and it makes it difficult to allow firewalls to easily recognize and discard traffic from the impostor. It is easy to put any address into the source address field of an IP packet, but it is more difficult to be able to receive packets sent to a randomly chosen IP address. 14.5.1 Cookies The designers of Photuris (an early key management protocol for IPsec which will be more fully described in §16.1 Photuris) provided for denial-of-service protection by adding a feature which the designers called cookies. Despite the name, this feature has nothing to do with web browser cookies [see §26.5 Cookies]. The Photuris cookie is a number chosen by Bob, and unpredictable to the side initiating communication with Bob. When Bob receives a connection initiation from IP source address S, Bob sends this number, in the clear, to the IP address S. Bob does no significant computation until he receives the same cookie back from that IP address. This assures that the initiator can receive packets sent to the IP address from which it claims to be transmitting. This feature makes it somewhat harder to mount a certain narrow set of attacks. It doesn't hurt, other than making the protocol slightly more complex, and adding a round-trip delay (in the case of Photuris). In theory, cookies could be used only when a node is getting swamped, saving the round-trip delay in the usual case (see Homework Problem 2). A server has a limited amount of memory for keeping track of connections. If an attacker were sending zillions of connect requests, and Bob had to remember what cookie he sent for each request, then even if the attacker couldn't return the correct cookie, he could still swamp Bob's memory so that Bob couldn't accept connections from legitimate users. This is similar to the attack known as the TCP SYN attack ("SYN" is the first packet that Alice sends to Bob to initiate a TCP connection to Bob), where TCP requires Bob to keep state after receiving the first connection request, and an attacker can fill up Bob's table. So we'd like Bob not to have to remember which cookies he sent to which connection initiators. The Photuris protocol provided for stateless cookies. The idea is to have the cookie be a function of the IP address and a secret known to Bob, so that Bob can calculate what cookie he would have sent to a particular IP address. One possibility is hash(IP address, secret). So the cookie protocol might look like Protocol 14-2. 14.5.2 DENIAL-OF-SERVICE/CLOGGING PROTECTION 373 14.5.2 Puzzles Juels and Brainard [JUEL99] came up with a clever alternative scheme for some amount of protection from a denial-of-service attack of this sort, where there are more initiators than Bob has computation or storage to handle. The idea is that if Bob is getting swamped, then Bob can require initiators to do more computation in order to connect. Bob can require each one to solve a puzzle, for instance making them compute a number whose message digest is a particular value. Bob can require arbitrary amounts of computation from an initiator by varying the size of the unknown number ("what 27-bit number has an MD of x?"). He can also make it stateless, in a similar way to stateless cookies (Homework Problem 3). This would not help very much if there are a lot more attackers than legitimate connection initiators, but it will slow down a single attacker making a lot of connection attempts. Verifying a puzzle answer is fast, since it merely involves doing a message digest. However solving the problem is exponential in the number of the bits Bob chooses for the size of the puzzle. The idea would be that non-hostile connection initiators might not mind having to do a few seconds worth of computation in order to connect, but it would slow down attackers, since any single attacker would not be able to get to the point of requiring storage and significant computation from Bob more than once every few seconds. Some issues are: • There are orders of magnitude of difference in the computational power of machines, and the amount of computation necessary to slow down a powerful machine would make it prohibitive to connect from a wimpy, ancient (e.g., two-year-old) machine. Initiator I want to talk Bob c = hash(IP address, secret) c c, start of rest of protocol Does c = hash(IP address, secret)? If so, continue with protocol. Protocol 14-3. Stateless cookie protocol 374 REAL-TIME COMMUNICATION SECURITY 14.6 • Many attackers could cooperate, or an attacker could create a virus that would attack from many sites simultaneously (this is known as a distributed denial-of-service attack). 14.6 ENDPOINT IDENTIFIER HIDING Another feature in some of these protocols is the ability to hide the identities of the two communicating parties from eavesdroppers. A mechanism for accomplishing this is to first do an "anonymous" Diffie-Hellman exchange, which establishes an encrypted tunnel, but to an unknown endpoint. The tunnel might have a man-in-the-middle, since you have not authenticated the other side (indeed you don't even know who the other side is claiming to be). After the anonymous Diffie-Hellman exchange establishes a key, the two parties divulge their identities, encrypted with the anonymous Diffie-Hellman key. Then a passive attacker will not learn their identities, but an active attacker acting as a man-in-the-middle might (depending on the particular protocol) learn one or both of the identities. In addition to divulging their identities, they should also authenticate each other based on the keys associated with their identities. An active attacker doing a man-in-the middle attack would be detected at this point, after having discovered the endpoint identities. Note that by carefully designing the protocol, you can arrange for the man-in-the-middle to only be able to learn one of the two identities before being discovered by the other side as an impostor. Which identity is better to hide from an active attacker? One argument says that it is better to hide the initiator's identity (Alice) than the responder's identity (Bob), because Bob's identity is probably already known. He has to be sitting at a fixed IP address waiting to be contacted, whereas Alice might be dialing in from anywhere, and her identity could not be guessed from her IP address. But a different argument says that it is better to hide Bob's identity. If Bob divulges his identity first, then anyone can initiate a connection to Bob and get him to divulge his identity. Unless there is a strict client/server model in which clients never accept connections and only initiate them, having a protocol in which the responder divulges his identity first makes it trivial to find out who is at a given IP address. In contrast, for an active attacker to trick Alice into revealing her identity, it requires impersonating Bob's address and waiting for Alice to initiate a conversation. An example protocol, assuming the two sides have public signature keys, might be Protocol 14-4. In this protocol an active attacker will be able to discover Alice's identity, but not Bob's. It is easy to arrange instead to hide Alice's identity (see Homework Problem 4). If Alice and Bob know in advance to whom they will be talking (perhaps they are two spies who will be contacting each other at a specific time), then a protocol based on a shared secret key 14.7 LIVE PARTNER REASSURANCE 375 will hide both identities. This is accomplished by authenticating based on the secret key and not sending identities at all (see Homework Problem 6). If Alice already knows Bob's public encryption key, it is possible to hide both identities from active attackers (see Homework Problem 5). 14.7 LIVE PARTNER REASSURANCE If Trudy can replay messages from previous conversation negotiations, she might be able to get Bob to waste space on a connection, or worse yet, she might be able to replay the subsequent data messages and, even if she can't decrypt the conversation, she might be able to cause Bob to repeat actions. For instance, when Bob (an ATM machine) talked to Alice, she might have requested Bob to put $100 into the money tray, as in Protocol 14-5. An hour later, if Trudy replays Alice's messages, it is important that Bob realize that this is not a conversation with the live Alice. If Bob chose a different b in each Diffie-Hellman exchange, then there wouldn't be a problem, but it is computation intensive to compute gb, so it might be nice to be able to reuse b. A way to allow Bob to reuse b and avoid replay attacks is for Bob to choose a nonce for each connection attempt, and have the session key be a function of the nonce as well as the Diffie-Hellman key. So the protocol might be modified to look like Protocol 14-6. Here the session key is a function of the nonce N as well as the Diffie-Hellman value. This seems similar to a cookie, but it is desirable for a cookie to be stateless, so that Bob does not have to Alice I want to talk, ga mod p Bob OK, gb mod p gab mod p {"Alice", [ga mod p]Alice} gab mod p {"Bob", [gb mod p]Bob} Protocol 14-4. Identity hiding 376 REAL-TIME COMMUNICATION SECURITY 14.7 keep state until he's at least sure the other end can listen at the IP address it claims to be sending from. With the most straightforward implementation of a stateless cookie, the cookie will be reused, so wouldn't work as a nonce. It is possible to design a protocol that will allow something to work both as a nonce and as a stateless cookie (see Homework Problem 10). Note that we've only ensured that Bob knows it's the live Alice (and not replayed messages). How would Alice know it's the real Bob? If Alice chooses a different a each time, and if Alice receives proof from the other side that it knows K (for instance, by acting on her request, which was encrypted with K), then she knows it's the real Bob. But suppose Alice, like Bob, would like to reuse a to save herself from computing ga mod p (see Homework Problem 11). Alice "Alice", [ga mod p]Alice Bob [gb mod p]Bob gab mod p {"please give me $100"} Protocol 14-5. A protocol vulnerable to replay if Bob reuses b Alice "Alice", [ga mod p]Alice Bob [gb mod p]Bob, N K = hash(N, gab mod p) K{"please give me $100"} Protocol 14-6. Using a nonce so Bob knows it's not replayed messages from Alice 14.8 ARRANGING FOR PARALLEL COMPUTATION 377 14.8 ARRANGING FOR PARALLEL COMPUTATION A lot of protocols require both Alice and Bob to compute a shared Diffie-Hellman key. This might take a long time (seconds perhaps, on slow devices). It can speed up the total elapsed time for an exchange if Alice and Bob can be computing at the same time, as in Protocol 14-7. This exchange might seem silly. Why not combine message 2 with message 3? The reason is that telling Alice gb mod p gives Alice a head-start on computing gab. She can be computing gab at the same time Bob is computing it. Al Eldridge probably was the first to invent this trick of sending an extra message in order to allow the computation-intensive calculations to be done in parallel. He implemented it in Lotus Notes. In Lotus Notes, Bob sends something encrypted with Alice's public key in msg k, and then later sends his signature on that message in msg k+1. This lets Alice do the expensive private key decryption while Bob is doing the expensive signature. Notice that although this adds a message it doesn't add any round-trip times, so it can be faster even if Alice and Bob are very far apart, talking to each other via, say, carrier pigeons (see RFC 1149 or RFC 2549). Alice [ga mod p]Alice Bob message 1 [gb mod p]Bob message 2 gab mod p{Bob's message} message 3 gab mod p{Alice's message} message 4 Protocol 14-7. Parallel computation 378 REAL-TIME COMMUNICATION SECURITY 14.9 14.9 SESSION RESUMPTION Another trick is the ability to bypass the initial expensive public key authentication if the server has recently authenticated the client and established a shared secret session key. In Lotus Notes, Bob (the server) has a secret S that he shares with nobody, and changes periodically (once a month). After Bob authenticates Alice, he sends her a secret, SAlice-Bob, which is a hash of her name and S, encrypted with her public key. Until Bob changes S, he'll give Alice the same SAlice-Bob each time she is successfully authenticated. The actual session key (used for encryption and integrity protection) is a function of SAlice-Bob and nonces sent by each side. If Alice shows knowledge of the SAlice-Bob that would result from hashing S and her name, then Bob assumes he's authenticated her in the recent past, and they skip the expensive public key operations. If Bob has switched Ss, then Alice's attempt to bypass the expensive authentication step will fail, and Alice and Bob will start from the beginning exchanging certificates, signing things, etc. The Lotus Notes scheme is especially nice because it does not require Bob to keep state. Other protocols, such as SSL, allow session resumption, but require Bob to remember the AliceBob session secret. Digital's DASS scheme (RFC 1507) had a different interesting method of bypassing the public key cryptography. During the handshake, Alice sends Bob the session key S, encrypted with Bob's public key and signed with Alice's private key. Call that quantity (the encrypted, signed S) XAlice-Bob. If Bob remembers XAlice-Bob from a recent Alice-Bob session, then he merely compares the received value from Alice with the stored XAlice-Bob. If they match, he doesn't have to bother decrypting it to extract S. But if he doesn't remember it, or it's different from the stored one (perhaps because Alice doesn't remember the previous session, or wants to change keys), then he verifies Alice's signature and decrypts with his private key to obtain the new S. If Bob doesn't remember the previous session, but Alice does, then Alice still saves time. If they both remember the previous session, then they both save time. What's interesting about this protocol is that the protocol messages look the same whether state has been kept or not. 14.10 PLAUSIBLE DENIABILITY If a protocol involves having Alice sign something containing Bob's name, then it offers proof that Alice intentionally talked to Bob (though it still gives no indication of what they talked about). In some cases Alice would like to assure Bob that it is her, but not provide proof that she talked to Bob. If Alice and Bob are authenticating each other based on a shared secret, then there is no way to prove to a third party that Alice and Bob communicated with each other, because the entire conver- 14.11 DATA STREAM PROTECTION 379 sation could have been constructed by Alice or Bob. If Alice and Bob authenticate each other using public encryption keys, anyone could create an entire conversation that looks like a conversation between Alice and Bob (e.g., consider Protocol 14-7, and change the first two messages from being signed by the sender to being encrypted with the recipient's public key. No knowledge of either side's private key is required to create such messages). If Alice and Bob authenticate each other using public signature keys, then it is possible to create a protocol in which each signs information including the other's identity, in which case there is no plausible deniability. But it is also possible to avoid signing the other side's identity, and therefore preserving plausible deniability (see Protocol 14-7). 14.11 DATA STREAM PROTECTION TCP's job is to accept a stream of data at the source and deliver that same stream to the process running on top of TCP at the destination. The mechanics of TCP doing its job—breaking the stream into packets, retransmitting lost packets, reordering packets—these are irrelevant to the process running on top of TCP. Protocols such as SSL that run on top of TCP break the data into whatever size chunks are convenient. For instance, SSL sends chunks that might be as large as 16K octets. The chunks will arrive in order (TCP does that), but they may have been broken into smaller pieces, which would certainly happen if the chunk was as large as 16K octets. A chunk cannot be integritychecked by SSL until all its pieces arrive. Alice [ga mod p]Alice Bob message 1 [gb mod p]Bob message 2 gab mod p{Alice's message} message 3 gab mod p{Bob's message} message 4 Protocol 14-8. Plausible deniability with signature keys 380 REAL-TIME COMMUNICATION SECURITY 14.11 With IPsec, however, individual packets are self-contained so that they can be encrypted and integrity-protected independently of other packets in a conversation. This is a performance advantage, since if each packet can be processed individually, it is easier to off-load the cryptographic processing onto an outboard device. It requires no buffering since each packet can be processed immediately. This is an argument which is often made (that IPsec can be implemented in an outboard device and SSL can't), but I2'm not completely comfortable with that argument. If an IP packet as transmitted by the source is too large, it might get fragmented on some intermediate link. In that case IPsec has the same problem, and can't integrity check a received fragment. It has to wait until all the fragments of the originally transmitted packet are reassembled, just as in the SSL case. And if SSL is careful to send chunks that are small enough that the chunks arrive intact, an outboard device that processed SSL (and implemented TCP as well) could in theory be built. With each packet being independently processed, you have to be careful to avoid replay attacks. Therefore you need a sequence number assigned by IPsec. This sequence number has nothing to do with the TCP sequence number. For instance, if TCP is running on top of IPsec, and needs to retransmit a packet, the retransmitted packet will be assigned a different sequence number by IPsec than the originally transmitted packet, and be perceived as a different packet by IPsec. You have to make sure that the IPsec-assigned sequence number is not used twice with the same key. So if a conversation is so long that the sequence number would wrap (i.e., reach a maximum value and start again at zero), a new key must be established for that session. And a new key should be established each time a node starts a conversation, since otherwise a packet from a prior conversation could be slipped into the data stream. To ensure that two plaintext blocks that are equal will not result in the same ciphertext when encrypted with the same session key, there are various tricks that can be used: • Explicitly put an IV into each packet, and use that IV for encrypting that packet in, for instance, CBC mode. Note that although the final ciphertext block of packet n−1 is a reasonable choice for the IV for packet n, since you can't guarantee that packet n−1 is available to IPsec when it is decrypting packet n, the IV must appear explicitly in each packet. • Have the IPsec-assigned sequence number be included in the first encryption block, so that the first block of data in each packet to be encrypted is guaranteed to be different for each packet that is encrypted with the same key. • Use a mode such as counter mode (see §4.2.5 Counter Mode (CTR)), using the IPsecassigned sequence number as the block number. 14.12 NEGOTIATING CRYPTO PARAMETERS 381 14.12 NEGOTIATING CRYPTO PARAMETERS It's fashionable today for security protocols to negotiate cryptographic algorithms, rather than simply having the algorithms be part of the specification of the protocol. It certainly makes the protocols more complex. Examples of things to negotiate are an algorithm for encryption, including key size, or an algorithm for compression, or the prime to use in a Diffie-Hellman exchange. One reason for allowing choice of cryptographic algorithms is so that over time systems can migrate to stronger but slower crypto as attackers' and defenders' machines get faster. Also, it allows migration from a cryptographic algorithm that gets broken. The cynic might observe that it also avoids the necessity for the committee to make a decision, and it allows them to pass the responsibility for a decision to a different committee (or worse yet, the user). One potential security flaw, if the negotiation isn't done right, is that an active attacker Trudy might be able to trick Alice and Bob into using weaker cryptography by removing from Alice's suggestions the ones that Trudy can't break. Alice and Bob would like to agree on the strongest possible cryptography that they both support. If Alice is willing, if necessary, to use weak crypto, and a weak algorithm is among the choices, and Bob is willing, if necessary, to use weak crypto, removing the strong choices from Alice's list may cause them to agree on weak crypto, if the protocol is not carefully designed. The reason this is a common vulnerability is that while Alice and Bob are negotiating about cryptographic algorithms, they probably do not yet have a shared secret with which to do integrity protection of the packets. The solution is to wait until they have established a shared secret, and then detect the tampering by having Alice reiterate (in a cryptographically protected way) what proposals she had made. IKE (§14.5.5 Negotiating Cryptographic Parameters) separately negotiates algorithms for encryption, integrity, authentication, etc. One way of doing this would be to say "I can use any of these algorithms for encryption, any of these for integrity," and so forth. But the IKE designers were concerned that not every set of choices would interoperate. For instance, DSA is supposed to only use SHA-1 as the hash. So the choice for IKE was to separately specify each combination Alice was willing to support. This could lead to an exponential explosion of choices. SSL/TLS's negotiation is much simpler. The specification predefines suites, with each suite specifying each of the algorithms. This is less flexible, unless every possible combination of cryptographic algorithms was a predefined suite. But in reality, there is no reason to support more than a few suites. 382 REAL-TIME COMMUNICATION SECURITY 14.13 14.13 EASY HOMEWORK • Why can an active attacker break an SSL connection, but not an IPsec connection? • Why is it a good idea for the session key to be a function of random numbers chosen by both endpoints? • What does a non-stateless cookie protect against? What is the advantage of making the cookie stateless? • Remove the nonce from Protocol 14-6. What is the security vulnerability? • Show a flawed protocol in which Alice and Bob negotiate what cryptographic algorithms to use, and an active attacker can trick them into using weaker cryptography. Show how to fix the protocol. 14.14 HOMEWORK 1. Talk about the properties of each of the following protocols, such as perfect forward secrecy, escrow foilage against passive attacks, escrow foilage against active attacks, identity hiding, perfect forward secrecy for identity hiding. Assume private encryption keys are escrowed and private signature keys are not escrowed. • Protocol 14-2. • A modified form of Protocol 14-2 in which the first two messages are encrypted with the other end's public key rather than signed by the transmitter's private signature key. So in message 1 Alice sends {"Alice", ga mod p} encrypted with Bob's public key, and Bob in message 2 sends {"Bob", gb mod p} encrypted with Alice's public key. • Protocol 14-4. • Protocol 14-9, where Alice and Bob share a secret key S. • Each side sends a nonce encrypted with other's public encryption key, resulting key is $\oplus$ of two nonces • Assume Alice and Bob share a secret S. Design a protocol in which they can do mutual authentication and establish a shared secret with PFS. Can it be done without DiffieHellman or any other form of public key cryptography? 14.14 HOMEWORK 383 • Protocol 14-2, but with each side deterministically generating the Diffie-Hellman private numbers as described in §14.4 PFS-Foilage from a seed given to the client machine and escrowed at the server machine. 2. Design a protocol in which Bob chooses whether to require Alice to send a cookie (see §14.5.1 Cookies). 3. Design a method of utilizing puzzles in a stateless manner similar to stateless cookies (see §14.5.2 Puzzles). 4. Referring to §14.6 Endpoint Identifier Hiding, modify Protocol 14-4 to hide the initiator's identity rather than the target's identity. 5. As mentioned in §14.6 Endpoint Identifier Hiding, it is possible to design a protocol that will hide both identifiers from an active attacker, assuming that Alice (the initiator) already knows Bob's public key. Show such a protocol. 6. Also as mentioned in §14.6 Endpoint Identifier Hiding, it is possible to hide both identities from active attackers if Alice and Bob share a secret key and there is a small set of entities that might initiate a connection to Bob. Show such a protocol. 7. Devise a protocol based on a pre-shared secret key that hides identities and gives PFS for identity hiding. Make two variants, one in which an active attacker can learn only the initiator's identity, and one in which an active attacker can learn only the target's identity. 8. Assuming private key operations are very slow, show a protocol that runs faster with an extra message. Suppose transmission delay is longer than the time it takes to do a private key operation. Would the protocol still complete faster with an extra message? Alice "Alice", ga mod p Bob message 1 "Bob", gb mod p, hash(S, gab mod p) message 2 hash(1,S, gab mod p) message 3 session key is gab mod p message 3 Protocol 14-9. Protocol for Homework Problem 1 384 REAL-TIME COMMUNICATION SECURITY 14.14 9. Design a variant of Kerberos in which the conversation between Alice and Bob can have perfect forward secrecy. 10. Design a protocol that gives Bob the power of a stateless cookie, and also ensures against replay protection. (Hint: have the value of the nonce/cookie be a function of the time, a secret known to Bob, and the IP address of the connection initiator. Also, have Bob remember all successfully used nonces within the acceptable clock skew.) 11. In the Protocol 14-6, explain why Bob knows that Alice is the real Alice, and not someone replaying Alice's messages. How does Alice know that it's the real Bob if she uses a different a each time? Modify the protocol to allow both Alice and Bob to reuse their a and b values, and yet have both sides be able to know they are talking to a live partner. 12. Suppose a stateless cookie mechanism is used by Bob, and suppose he changes his secret periodically. What can he do to assure a connection attempt will succeed even if he changed his secret between the time the initiator asked for the cookie and returned it (assuming the initiator doesn't wait too long before returning the cookie)? 13. Assume a stateless session resumption scheme as described in §14.9 Session Resumption. Suppose Bob changes his S every ten minutes, and yet you'd like to be able to resume sessions that have been idle for longer than that, say two hours. How might that work? 14. Explain how, and under what circumstances, in the DASS session resumption protocol described in §14.9 Session Resumption, only Alice saves computation, only Bob saves computation, and they both save computation. 15. Describe various methods of having Alice and/or Bob remember state from the last time they authenticated each other that allows them to resume a session and bypass the expensive public key cryptography. Describe a method in which Bob can save computation even if he hasn't kept state. Is there a clever way of having Bob remember state and not Alice? 385 15 IPSEC: AH AND ESP IPsec is an IETF standard for real-time communication security. The concepts behind such a protocol were already covered in Chapter 14 Real-time Communication Security. The main pieces of IPsec are AH and ESP, which describe the IP header extensions for carrying cryptographically protected data, and IKE, which is a protocol for authenticating and establishing a session key. This chapter covers AH and ESP, and Chapter 16 IPsec: IKE will cover IKE. 15.1 OVERVIEW OF IPSEC The part of IPsec that we cover in this chapter assumes that two nodes already have a shared session key, which might have been configured manually, or established through IKE. Since Bob might be receiving IPsec-protected packets from many sources, maybe even different processes using the same source IP address, there has to be a way for Bob to know which cryptographic key and which algorithms to use to process the packet. This is done by inserting an IPsec header (AH and/or ESP) into the IP packet which tells Bob to which security association the packet belongs (see §15.1.1 Security Associations). IPsec works with IPv4 or with IPv6. 1.1.1 Security Associations An IPsec security association (SA) is a cryptographically protected connection. Associated with each end of the SA is a cryptographic key and other information such as the identity of the other end, the sequence number currently being used, and the cryptographic services being used (e.g., integrity only, or encryption+integrity, and which cryptographic algorithms should be used). The SA is considered unidirectional, so a conversation between Alice and Bob will consist of two SAs, one in each direction. The IPsec header includes a field known as the SPI (SECURITY PARAMETER INDEX) which identifies the security association, allowing Alice to look up the necessary information (such as the cryptographic key) in her SA database. The SPI value is chosen by the destination (Bob), so it would 386 IPSEC: AH AND ESP 15.1.2 seem as though the SPI alone should allow Bob to know the SA, since Bob can ensure that the SPI is unique with respect to all the sources that Bob has SAs with. But it is possible for Bob to also be receiving multicast data, in which case Bob would not have chosen the SPI, and it might be equal to one that Bob already assigned. Therefore the SA is defined by both the SPI and the destination address. (The destination address of a packet received by Bob will be Bob for unicast, or a group address if it's multicast.) Furthermore, IPsec allows the same SPI values to be assigned to different SAs if one SA is using AH and one is using ESP, so the SA is defined by the triple ⟨SPI, destination address, flag for whether it's AH or ESP⟩. 15.1.2 Security Association Database A system implementing IPsec keeps a security association database. When transmitting to IP destination X, the transmitter looks up X in the security association database, and that entry will tell it how to transmit to X, i.e., it will provide the SPI, the key, the algorithms, the sequence number, etc. When receiving an IP packet, the SPI of the received packet is used to find the entry in the security association database that will tell the receiver which key, sequence number, etc., to use to process the packet. 15.1.3 Security Policy Database Just as firewalls are configured with tables telling them what type of traffic to allow based on information such as the IP header source and destination addresses and TCP ports, IPsec is assumed to have access to a similar database specifying which types of packets should be dropped completely, which should be forwarded or accepted without IPsec protection, and which should be protected by IPsec, and if protected, whether they should be encrypted and/or integrity-protected. Decisions could, in theory, be based on any fields in the packet, e.g., source IP address, destination IP address, protocol type in the IP header, and layer 4 (TCP or UDP) ports. 15.1.4 AH and ESP AH (Authentication Header, defined in RFC 2402) and ESP (Encapsulating Security Payload, defined in RFC 2406) are the two types of IPsec headers. AH provides integrity protection only. ESP provides encryption and/or integrity protection. Given that ESP optionally provides integrity protection (in addition to optional encryption), it's natural to wonder why AH is needed. In fact many people argue (and we concur) that AH is not necessary. The integrity protection provided by ESP and AH are not identical, however. Both pro- 15.1.5 OVERVIEW OF IPSEC 387 vide integrity protection of everything beyond the IP header, but AH provides integrity protection for some of the fields inside the IP header as well. See §15.1.6 Why Protect the IP Header?. AH can't protect all of the fields, because some of them are intended to be modified by routers (see §15.3.1 Mutable, Immutable). There is one feature that AH provides that ESP does not. Firewalls and routers sometimes look at fields such as layer 4 ports in order to do packet filtering, content screening, or differential queuing. If everything beyond the header is encrypted (as it would be with ESP, if ESP is encrypting), then firewalls and routers are not able to look at those fields. One could use ESP for integrity only, but it would be impossible for a firewall or router to know whether an ESP-protected packet was encrypted or not. The source and destination know—it's either manually configured into the SA or negotiated through IKE. But there's nothing in the packet header that tells a router or firewall whether a packet is encrypted. This "feature" of having routers and firewalls look at the TCP ports can only be used with unencrypted IP traffic, and many security advocates argue that IPsec should always be encrypting the traffic. Fields such as TCP ports should perhaps be hidden to avoid divulging information such as which applications are running. Firewalls base decisions on the port fields, but a malicious user can disguise any traffic to fit the firewall's policy database (e.g., if the firewall allows HTTP, then run all protocols on top of HTTP). 15.1.5 Tunnel, Transport Mode The IPsec specification talks about two "modes" of applying IPsec protection to a packet. Transport mode refers to adding the IPsec information between the IP header and the remainder of the packet. Tunnel mode refers to keeping the original IP packet intact and adding a new IP header and IPsec information (ESP or AH) outside. Transport mode is most logical when IPsec is being applied end-to-end. A common use of tunnel mode is firewall to firewall, or endnode to firewall, where the data is only protected along Transport Mode Tunnel Mode IP header rest of packet original packet IP header rest of packet IP header IPsec rest of packet new IP hdr IPsec IP header rest of packet Figure 15-1. Transport Mode and Tunnel Mode 388 IPSEC: AH AND ESP 15.1.5 part of the path between the endpoints. Suppose two firewalls establish an encrypted tunnel to each other across the Internet (see Figure 15-2). They treat the tunnel as if it is an ordinary, trusted link. In order to forward packets across that link, F1 adds an IP header with destination=F2. When A launches an IP packet to destination B, it will have, in the IP header, source=A and destination=B. When F1 forwards the packet to F2 across the encrypted tunnel, it will use IPsec tunnel mode. F1 will not modify the inner header, other than doing what any router would do when forwarding a packet, such as decrementing the hop count. The outer IP header added by F1 will have source=F1 and destination=F2. The inner header will be unmodified by the routers along the path between F1 and F2. Those routers will only look at the outer IP header. Transport mode is not strictly necessary, since tunnel mode could be used instead. Tunnel mode just uses more header space since there will be two IP headers. The same packet might have multiple layers of IPsec (ESP and/or AH) headers, and might be multiply-encrypted (see Figure 15-3). Suppose A and B are talking with an encrypted end-to-end connection. Their packets will contain an ESP header. When F1 forwards it across the tunnel to F2, F1 takes the entire packet (including the IP+ESP header) and adds its own IP+ESP header. F1 encrypts the entire packet it received, including the IP header, with the key that F1 shares with F2. Tunnel mode is essential between firewalls in order to preserve the original source and destination addresses; and as we said earlier, tunnel mode can be used instead of transport mode at the expense of adding a new IP header. Given that IPsec is too complex, many have argued that getting rid of transport mode would be one way of simplifying IPsec. But transport mode is such a small piece of the complexity of IPsec that we don't feel it's worth worrying about. Far more useful would be to get rid of AH and most of IKE. original packet as launched by A, encrypted with the F1–F2 key IP: src = F1, dst = F2 ESP IP: src = A, dst = B ESP Figure 15-3. Multiply encrypted IP packet Internet F1 F2 A B added by firewall F1 original packet IP: src = F1, dst = F2 ESP IP: src = A, dst = B Figure 15-2. IPsec, tunnel mode, between firewalls 15.1.6 IP AND IPV6 389 15.1.6 Why Protect the IP Header? AH advocates claim AH is needed because it protects the IP header. It is unclear why it is necessary to protect the IP header. If it were necessary, this could be provided by ESP in tunnel mode. Intermediate routers cannot enforce AH's integrity protection, because they would not know the session key for the Alice-Bob security association. So AH can at best be used by Bob to check that the IP header was received as launched by Alice. 15.2 IP AND IPV6 IP is what’s known in ISO terminology as a network layer, or layer 3 protocol. It puts a source address and destination address on data, much like addressing an envelope to be sent through snail mail. IP was designed with 32-bit addresses, and many people started to realize in the mid-1980s that 32 bits was too few. There were other layer 3 protocols deployed at the time with adequate address space, and the IETF could have adopted one of those. In fact, in 1992, the IAB (Internet Architecture Board) recommended replacing IP with the CLNP packet format, a format that was very similar to IP, but had larger addresses. CLNP had been standardized by ISO and implemented by most of the vendors. If the IETF had adopted CLNP in 1992, most likely at this point the Internet would be using larger addresses and would certainly be better off than it is now, but certain very vocal IETF members wanted to invent their own header format. The new header format designed by IETF is known as IPv6, and because they’ve been designing it for so long, it is unfortunately not at all clear whether the world will ever migrate to IPv6. It would have been much easier to migrate to a larger address format in the early ’90s, when the Internet was smaller and people hadn’t out of necessity invented all sorts of kludges (like NAT, see §15.2.1 NAT (Network Address Translation)) to live with a smaller address space. The reason all this is relevant for this book is that you need to understand some of the politics in order to understand the design of IPsec. The “v6” in the name “IPv6” comes from the first four bits of an IP header, which is the VERSION NUMBER field, which is set to 4 for the current version of IP, so IP is sometimes known as IPv4. I2 bet you’d guess that 5 would be the next version number after 4, but 5 was already assigned for a little-known and little-used protocol, so the next version of IP needed to be 6. If you really care about all this, we1,3 recommend my2 book Interconnections: Bridges, Routers, Switches, and Internetworking Protocols. (I2 modestly abstain.) In order to understand IPsec, you need to understand both the IPv4 and IPv6 formats, and you need to understand some of the politics associated with the design of IPv6 in order to understand things like why there is an AH header and why it looks different than the ESP header. 390 IPSEC: AH AND ESP 15.2.1 The IPv6 designers were frustrated that the world didn’t immediately deploy IPv6, after they spent 10 years (so far) designing it. Although bigger addresses are good for you, people don’t get excited about learning something new and doing radical changes to all their software if things are working. “What would I get for converting to IPv6?” “Bigger addresses.” “What’s an address?” You see, it just doesn’t motivate people to turn their environment inside out. Especially when you consider the other thing you get by converting to IPv6, which is noninteroperability with the 600 million current Internet nodes. So the IPv6 proponents hoped that IPsec (among other features) would be the motivator for moving to IPv6. Some IPv6 advocates proposed making it illegal to make any improvements (including IPsec) to IPv4, so that if the world wanted any of the stuff IETF designed in the last 10 years it would have to move to IPv6. But the IPsec designers were more interested in security, and didn’t care whether it was deployed with 4-octet or 16-octet addresses, so they designed it to work with either format. The IPv6 specification says that IPsec is a mandatory feature of IPv6, so sometimes people claim that “Security is built into IPv6, whereas it’s an add-on to IPv4.” In reality, the “mandatory-ness” of IPsec with IPv6 is just words on paper. There are implementations of IPv6 without IPsec, and there are implementations of IPv4 with IPsec. And IPsec works just as well with IPv4 as IPv6. 15.2.1 NAT (Network Address Translation) Another thing you need to understand is one of the kludges that has allowed the Internet to survive and thrive with 32-bit addresses during the many years in which IPv6 was being designed. The kludge is known as NAT, network address translation. With a NAT box, the computers on your internal network do not need global IPv4 addresses in order to connect to the Internet. Instead the NAT box translates an internal node’s IP address into a globally unique address when that node is talking to something on the

Internet. NAT complicates everything, especially because many of the higher layer protocols violate layering and carry IP addresses inside their data. An IPsec tunnel cannot go through a NAT box because the NAT box wants to update the IP addresses inside the encrypted data and it doesn't have the key. Even IPsec transport mode has problems because the IP address is included in the computation of the TCP or UDP checksum—the NAT box cannot correct the checksum because it is encrypted. NAT boxes sometimes make multiple computers appear at the same IP address by assigning each a subset of the TCP and UDP port ranges. This technique also fails with NAT boxes because IPsec encrypts the port information. There are carefully crafted configurations where IPsec can be used through NAT boxes by encapsulating IPsec packets inside UDP packets. Absent encryption, NAT boxes are willing to go to extreme lengths to make protocols work. A particularly amusing and nasty example of this is the protocol FTP (File Transfer Protocol, RFC 959). FTP is an application layer protocol, so it is encapsulated within both layer 3 and layer 4. FTP 15.2.2 IP AND IPV6 391 not only carries IP addresses, but carries them in what is called dotted decimal notation, where each octet of the 4-octet IP address is represented in ASCII as decimal numbers and delimited by periods, e.g., 178.201.19.175. Each octet of the IP address might require between one and three ASCII characters (not counting the periods), since leading 0s are not transmitted. So the NAT box not only needs to change the IP address in the IP header, but has to know that the packet is FTP, and know to change the address carried inside the FTP data! Worse than that, it might have to replace a string such as 178.201.19.175 with a string of a different length, such as 22.51.111.9. The problem with changing the size of the data is that TCP numbers octets, not packets, so if the NAT box changes the length of one of the packets in a conversation, it has to compensate by changing the TCP sequence numbers for the entire duration of that conversation to compensate! So everyone hates NAT, but NAT boxes are very popular because really what users want is for things to work and they don't care about architectural purity. NAT particularly infuriates the IPv6 proponents because it makes it possible for the world to delay migrating to IPv6. The longer the world delays, the harder it will be to get everyone to convert. So the IPv6 proponents actually go out of their way to design things that will not work with NAT, hoping that this will cause the world to abandon NAT (and then, they hope, move to IPv6). That is one reason they like AH, because the AH integrity check will fail if a NAT box modifies the IP header. 15.2.2 Firewalls Another important issue is firewalls. Various network administrators like to have firewalls observe packets and discard (filter) packets based on characteristics such as which protocol is being used. Many IETF people hate firewalls, since they tend to make it difficult to deploy new applications, and they break many existing applications. Security is strongest if done end-to-end. IPsec encrypts information on which firewalls like to base decisions, such as the PORT fields in the TCP header that can help them know whether the data is email or telnet. The people designing the protocols believe such information is nobody's business except the endpoints. Some firewall administrators feel they need that information in order to monitor and control traffic to and from their network. Some people hope that IPsec will make firewalls go away, since IPsec encryption will prevent firewalls from being able to do a lot of what people use firewalls for. But more likely, many firewall administrators would rather forgo end-to-end security than give up their ability to monitor and control the traffic between their net and the Internet. If enough firewall administrators decide to throw away any packets that are encrypted because they can't inspect them, people will be highly discouraged from using IPsec. As much as people would like their traffic to be protected in transit, they're even more anxious for their traffic to be delivered at all. Some would argue that firewalls exist to compensate for poorly protected, poorly managed end systems on the inside, and that the very existence of IPsec on a system means that that system is well-protected and well-managed.

Therefore, they'd argue, it is safe for the firewall to forward 392 IPSEC: AH AND ESP 15.2.3 encrypted traffic and assume that an end system smart enough to have IPsec must know what it is doing. That may well be how things initially roll out, but it's unlikely to last. Systems supporting IPsec are not necessarily any better managed than those without it, and sometimes the firewall administrators are trying to prevent anti-social actions like downloading pornography rather than just trying to protect against carelessness. One of the most amusing consequences of firewalls is that people find ways of disguising the traffic that firewalls administrators would like to block so that it looks to the firewall like the kind of traffic the firewall is configured to allow. For instance, protocols are being defined to work on top of HTTP to make something like file transfer or system administration look like web browsing. This is extremely inefficient because there are extra layers of protocol. And as a final irony, the term for disguising traffic to look like something the firewall is configured to pass through, when the firewall administrator intends to keep that traffic out, is firewall-friendly. 15.2.3 IPv4 Header The IPv4 header is defined in RFC 791. Its fields are For the purposes of this chapter, the most important field is the PROTOCOL field, which indicates what follows the IP header. Common values are TCP (6), UDP (17), and IP (4). When PROTOCOL indicates IP, it is a tunneled packet, i.e., following the IP header is another IP header. Note that the PROTOCOL field value of 4 for IP has nothing to do with the version number for IPv4; if the next header is IPv6, the protocol field will still be 4. size 4 bits version 4 bits header length (in 4-octet units) 1 octet type of service 2 octets length of header plus data in this fragment 2 octets packet identification 3 bits flags (don't fragment, and last fragment) 13 bits fragment offset 1 octet hops remaining, known as TTL (time to live) 1 octet protocol 50=ESP, 51=AH 2 octets header checksum 4 octets source address 4 octets destination address variable options 15.2.4 IP AND IPV6 393 IPsec defines two new values for the protocol field in the IP header: ESP=50 and AH=51. For example, if TCP is on top of IP without IPsec, the PROTOCOL field in the IP header will be 6. If TCP is used with IP using AH, for instance, then the PROTOCOL field in the IP header will equal 51, and the PROTOCOL field in the AH header will be 6 to indicate that TCP follows the AH header. If the packet is encrypted using ESP, then the PROTOCOL field in the IP header will be 50, but the actual PROTOCOL field, the one that would have appeared in the IP header if encryption with ESP was not being used, will be encrypted, and therefore not visible until the packet is decrypted. 15.2.4 IPv6 Header The IPv6 header is defined in RFC 2460, and its fields are In IPv6, the equivalent field to IPv4's PROTOCOL is NEXT HEADER. It has the same values defined as the IPv4 PROTOCOL field, so ESP=50 and AH=51. IPv6-style extension headers (roughly equivalent to OPTIONS in the IPv4 header) are encoded as LENGTH OF THIS HEADER is in units of 8-octet chunks, not counting the first 8-octet chunk. AH looks like an IPv6 extension header, but its PAYLOAD LENGTH is in units of 4-octet chunks instead of 8-octet chunks (and like other IPv6 extension headers, doesn't count the first 8-octets). This violates one of the protocol folklore rules described in Chapter 26 Folklore, which is that the LENGTH field should always be defined the same way for all options, so that it is easy to skip over unknown options. DATA FOR THIS HEADER is a sequence of options, each one TLV-encoded, which means a TYPE field, a LENGTH field, and a VALUE field. The TYPE field is one octet long, and one of the bits in the TYPE field for options that appear in some extension headers indicates whether the option is # octets 4 version (4 bits) |type of service|flow label 2 payload length 1 next header 1 hops remaining 16 source address 16 destination address # octets 1 next header 1 length of this header variable data for this header 394 IPSEC: AH AND ESP 15.3 mutable (might change along the path) or immutable (relevant for AH, see section §15.3 AH (Authentication Header)). The mutable flag is only useful for AH, and if AH ever goes away, the flag in IPv6 will be very mysterious. 15.3 AH (AUTHENTICATION HEADER) The AH header provides authentication only (not encryption), and is defined in RFC 2402. It looks like This is the same format as IPv6 extension headers, which all start with NEXT HEADER and PAYLOAD LENGTH (which gives the length of the AH header), except, as we said in the previous section, AH's PAYLOAD LENGTH is in different units than the equivalent field in an IPv6 extension header. AH is intended not only to protect the data, but the IP header as well. In IPv4, the AH header must be a multiple of 32 bits, and in IPv6 it must be a multiple of 64 bits. So the AUTHENTICATION DATA field must be an appropriate size to make the header size be the right length. Some integrity checks require the data to be a multiple of some block size. If the data is not a multiple of the block size, then AH is computed as if the data were padded to the proper length with 0s, but the 0s are not transmitted. The fields in AH are • NEXT HEADER. Same as PROTOCOL field in IPv4. For example, if TCP follows the AH header, then NEXT HEADER is 6. • PAYLOAD LENGTH. The size of the AH header in 32-bit chunks, not counting the first 8 octets. • SPI. Discussed in §15.1.1 Security Associations. • SEQUENCE NUMBER. The sequence number has nothing to do with TCP's sequence number. This sequence number is assigned by AH and used so that AH can recognize replayed packets and discard them. So, for example, if TCP retransmits a packet, AH will just treat it like a # octets 1 next header 1 payload length 2 unused 4 SPI (Security Parameter Index) 4 sequence number variable authentication data 15.3.1 AH (AUTHENTICATION HEADER) 395 new packet and give it the next sequence number. AH will not know (or care) that this is a retransmitted TCP packet. • AUTHENTICATION DATA. This is the cryptographic integrity check on the data. 15.3.1 Mutable, Immutable Some fields in the IP header get modified by routers, so they can't be included in AH's end-to-end integrity check. For example, the TTL field must be decremented by every router. The immutable fields are the ones that the AH designers do not believe should ever legitimately be modified in transit. The IPv4 AH defines the mutable fields as TYPE OF SERVICE, FLAGS, FRAGMENT OFFSET, TIME TO LIVE, and HEADER CHECKSUM. Some of the choices for which fields are considered mutable are surprising. As envisioned by the original IP designers, TYPE OF SERVICE would be an immutable quantity, and indeed worthy of protecting. It contained the desired routing metric and priority chosen by the source. But that use of the TYPE OF SERVICE field as originally defined in the IP header did not prove to be useful, and now network administrators are playing around with various uses of the field, such as categorizing the packet when it enters their domain. So routers today want to be free to modify that field. Why is PAYLOAD LENGTH considered immutable? If a packet requires fragmentation, the PAYLOAD LENGTH must be modified (since PAYLOAD LENGTH is the length of the data in this fragment, not the original size of the packet as launched by the source), so it would seem like it should be considered mutable. However, the reason it can be considered immutable is that IP at the destination must reassemble the packet before AH can verify the integrity check, in which case although PAYLOAD LENGTH was modified en route, it has been restored to its original value before the AH header is processed. Theoretically the same logic would apply to FRAGMENT OFFSET, which should be set to 0 when launched by the source, and might be modified by routers along the path if the packet is fragmented, but will be restored to 0 again after the packet is reassembled at the destination, and before the AH header is processed. But AH has chosen to define FRAGMENT OFFSET as a mutable field, and therefore not covered by the AH integrity check. Not that it matters, since as we said earlier, there's no reason to bother protecting any of the IP header. Note that since FRAGMENT OFFSET is always 0 when the AH header is processed, whether it's immutable or not is irrelevant. In IPv6 the mutable fields are TYPE OF SERVICE (because routers want to be able to modify it), FLOW LABEL (because the IPv6 designers still don't know what to do with the field, so maybe whatever they'll decide to use it for will require it to change en route), and HOP LIMIT (which is decremented by each router along the path). 396 IPSEC: AH AND ESP 15.3.2 15.3.2 Mutable but Predictable The

DESTINATION ADDRESS is not quite considered immutable, but it is included in the AH integrity check, since there is one situation in which it might be modified in a way considered legitimate by the AH designers (as opposed to being modified by a NAT box, which the AH designers do not consider legitimate). This case is when the source specifies source routing as an option. The way source routing works in IP is that the source chooses a path consisting of a sequence of intermediate destinations to be visited in order, say D1, D2, D3, D, where D is the ultimate destination. In IPv4 there is an option called “source routing” that would include these intermediate destinations. In IPv6 it’s an extension header called the “route header”, which is the same idea. When source routing is specified, the DESTINATION ADDRESS in the IP header (in both v4 and v6) specifies the next destination in the source route header, not the ultimate destination. So DESTINATION ADDRESS is clearly mutable, since in this example it would start out, when launched by S, as D1, get overwritten as D2 once it reached D1, get overwritten as D3 once it reached D2, and ultimately wind up at the destination as D. But the source knows what the DESTINATION ADDRESS will look like when it arrives at D (the DESTINATION ADDRESS field will specify D). So even though S launches it with DESTINATION ADDRESS set to D1, S computes the AH integrity check as though DESTINATION ADDRESS were set to D. Then when D evaluates the AH, it will compute properly as received. So fields that are mutable but predictable are included in the AH integrity check, but with the values they will have when received at the other end. The only one listed in the AH spec is the DESTINATION ADDRESS, but in theory the TOTAL LENGTH field in IPv4 should be listed as mutable but predictable rather than immutable, since if the packet were fragmented, that field would be modified en route, but would be restored by IP at the destination before IPsec at the destination saw the packet. But it isn’t a problem since immutable and mutable-but-predictable fields are treated the same way at the destination. IP Source routing S D1 D2 D3 D 15.4 ESP (ENCAPSULATING SECURITY PAYLOAD) 397 15.4 ESP (ENCAPSULATING SECURITY PAYLOAD) ESP allows for encryption and/or integrity protection. If you want encryption, you must use ESP. If you want integrity protection only, you could use ESP or AH. If you want both encryption and integrity protection, you could use both ESP and AH, or you could do both integrity protection and encryption with ESP. The security association database would tell you what to use when transmitting to a particular IP address. It’s a little odd to call ESP a “header” because it puts information both before and after the encrypted data, but everyone seems to call it a header so we will too. Technically, ESP always does encryption, but if you don’t want encryption you use the special “null encryption” algorithm. The working group had a lot of fun writing the RFC specifying the null encryption algorithm, extolling its virtues like how fast it was and how it could take any size keys, and source and destination would even interoperate if they had different keys. For a bit of security-geek humor, read RFC 2410. The presence of the ESP header is indicated by having the PROTOCOL field in IPv4 or the NEXT HEADER field in IPv6 equal to 50. The ESP envelope itself consists of The fields are • SPI. Same as for AH, discussed in §15.1.1 Security Associations. • SEQUENCE NUMBER. Same as for AH, explained in §15.3 AH (Authentication Header). • INITIALIZATION VECTOR. An IV is required by some cryptographic algorithms, such as encryption with CBC mode. Although the IV is variable length (it may even be zero length), it’s fixed length for a particular cryptographic algorithm. Once the SA is established, the cryptographic algorithm is known, and therefore the length of the IV field is fixed for the duration of the SA. This is also true of the field AUTHENTICATION DATA. # octets 4 SPI (Security Parameters Index) 4 sequence number variable IV (initialization vector) variable data variable padding 1 padding length (in units of octets) 1 next header/protocol type variable authentication data 398 IPSEC: AH AND ESP 15.5 • DATA. This is the protected data, probably encrypted. If it is a tunnel mode packet, then the beginning of the data would be an IP header. If it would have been TCP, and ESP is being used in Transport mode, then the beginning of the data would be the TCP header. • PADDING. Padding is used for several reasons: to make the data be a multiple of a block size for cryptographic algorithms that require it; to make the encrypted data be a different size than the plaintext, so as to somewhat disguise the size of the data (limited because PADDING LENGTH is only one octet); and to ensure that the combination of the fields DATA, PADDING, PADDING LENGTH, and NEXT HEADER are a multiple of four octets. • PADDING LENGTH. Number of octets of padding. • NEXT HEADER. Same as PROTOCOL field in IPv4 or NEXT HEADER in IPv6, or NEXT HEADER in AH. • AUTHENTICATION DATA. The cryptographic integrity check. Its length is determined by the authentication function selected for the SA; it is zero length if ESP is providing encryption only. If encryption is used, the fields DATA, PADDING, PADDING LENGTH, and NEXT HEADER are encrypted. The AUTHENTICATION DATA appears only if the security association requests integrity protection with ESP. If ESP integrity protection is used, all fields in the ESP (starting with SPI and ending with NEXT HEADER) are included in the ESP integrity check. 15.5 SO, DO WE NEED AH? We clearly need ESP, since it’s the only one that provides encryption. But since you can do integrity-only with ESP, as well as encryption+integrity with ESP, it might seem like AH is unnecessary. What reasons do people give for keeping AH? • AH protects the IP header itself, whereas ESP only protects everything beyond the ESP header. We don’t think protecting the IP header matters for security. • An implementation of IPsec that only implemented AH might be more exportable. Indeed, at least one vendor reported that they were able to get export approval for their implementation of IPsec specifically because it only supported AH. But other vendors were able to get export approval for ESP. It’s not unusual for different companies to get different answers for export, and the rules change with time. We think that even if an implementation of IPsec that only did AH were more exportable, it’s not very important because who would buy it? 15.6 COMPARISON OF ENCODINGS 399 • As we discussed in §15.1.4 AH and ESP, with ESP, even when not using encryption, firewalls and routers cannot look beyond the layer 3 header at information such as TCP ports. This sounds like a reasonable argument, and indeed a proposal was made for TF-ESP (transportfriendly ESP), which allowed for copying all fields of interest into an unencrypted portion of the header. The arguments against TF-ESP which convinced people not to bother with it were ♦ Routers and firewalls have no right to look at anything above layer 3. ♦ Anything copied over in cleartext exposes some information that might be better hidden from eavesdroppers. ♦ Since the IPsec key is end-to-end, it is impossible for intermediate devices to verify that the cleartext fields are accurate. It would be possible to sneak past a firewall, for instance, if you set the ports in cleartext to something the firewall would allow through. The destination could, in theory, check that the cleartext fields are the same as the fields inside the ESP-protected payload when they arrive. But someone colluding with the source could change them back after the packet gets through the firewall, or if the destination were colluding with the source, it could simply not do the check. The arguments against TF-ESP are credible, but it is unfair to simultaneously claim that intermediate devices have no right or need to look at upper layer information, and to criticize ESP for not exposing layer 4 information. It seems very unlikely that anyone would bother with IPsec for integrity protection only, and so “fixing” ESP to give people (e.g., firewall administrators, and router vendors who want to claim that their routers do some fancy thing) what they want is probably the only way to provide those features. In other words, rather than seeing the feature of exposed layer 4 information as a reason to keep AH, we believe that IPsec should be considered as essentially always providing encryption, and therefore if the layer 4 information needs to be exposed, there will have to be a way of exposing it with ESP. Or (our preference) decide the world can and should do without this information, especially since there's no way, even with AH, to ensure that the information is accurate, except at the endpoints. 15.6 COMPARISON OF ENCODINGS AH was designed by IPv6 fans, and looks similar to IPv6 extension headers. (The only difference is that its LENGTH field is expressed in different units.) The ESP designers didn't really care about IPv6 and designed ESP to be as technically good as it could be, unconstrained with having to look like something else. And there's no reason why, even if used with IPv6, that ESP has to look like the other IPv6 extension headers. 400 IPSEC: AH AND ESP 15.7 There are two wasted octets in AH (the "unused" octets) in order for all the fields to be on 4- octet boundaries. But AH can cleverly avoid padding the data, because the integrity check is calculated as if the data is padded to a multiple of a block size, but the padding is not transmitted. Therefore, if the data needed more than two octets of padding, AH winds up being smaller. Of course, this is assuming that IPsec is being used for integrity-only, which as we said, we think will be rare. If the data is encrypted, then it would be much more overhead to use AH and ESP simultaneously than just using ESP for both integrity protection and encryption. Having the MAC appear before the data (as it is in AH) means that the data needs to be buffered and the integrity check computed before the packet can be transmitted. In contrast, in ESP, the MAC appears after the data. IPv6 proponents hate NAT because it makes it possible for the world to continue to live with IPv4. This is another reason why they like AH, because having a NAT modify the IP addresses in the header will cause AH to reject the packet, and so they hope that AH will cause NAT to go away. But as we said, AH isn't intrinsically useful enough to kill NAT. More likely NAT would kill AH. Attempting to protect the IP header, and needing to classify every field and every option according to whether it's mutable or immutable, makes AH very complicated. At one of the final IETF meetings before AH and ESP were finalized, someone from Microsoft got up and gave an impassioned speech about how AH was useless given the existence of ESP, cluttered up the spec, and couldn't be implemented efficiently (because of the MAC in front of the data). Our1,2 impression of what happened next was that everyone in the room looked around at each other and said, "Hmm. He's right, and we hate AH also, but if it annoys Microsoft let's leave it in, since we hate Microsoft more than we hate AH." 15.7 EASY HOMEWORK 1. Why isn't the SPI value sufficient for the receiver to know which SA the packet belongs to? 2. Why is the integrity check processing more convenient with ESP than with AH? 15.8 HOMEWORK 1. Suppose Alice is sending packets to Bob using IPsec. Suppose Bob's TCP acknowledgment gets lost, and Alice's TCP, assuming the packet was lost, retransmits the packet. Will Bob's IPsec implementation notice that the packet is a duplicate and discard it? 15.8 HOMEWORK 401 2. Suppose a company's network is attached to the Internet via two NAT boxes, and packets might exit via either one. How might a protocol such as FTP complicate implementation of multiple NAT boxes? Suggest methods of making this work. 3. Suppose you wanted the transmitter to assign the SPI rather than the receiver. What problems might this cause? Can it be made to work? 4. Would it be possible for the SA to be defined only by the destination address and the SPI (i.e., leave out whether it's ESP or AH)? Would this require any changes to the IPsec protocol? Would an implementation of a receiver that defined the SA based solely on destination address and SPI interwork with one that did what the IPsec specification says? 5. When sending encrypted traffic from firewall to firewall, why does there need to be an extra IP header? Why can't the firewall simply encrypt the packet, leaving the source and destination as the original source and destination? 6. Referring to Figure 15-2, suppose A and B already have an IPsec SA between them and are using ESP. What would be the advantages/disadvantages of having F1, in the case where there's already an ESP header, merely forwarding the packet to F2 without doing a second encryption? 7. Referring to Figure 15-2, assume that A and B are using IPsec in transport mode, and F1 and F2 have established an encrypted tunnel using IPsec. Assume A sends a TCP