key technology is being used for keys) or by using the key she shares with Bob (if secret key technology is being used). However, this is not how mail encryption is generally done, for several reasons: • If Alice has a long message to send to multiple recipients, the long message would have to be encrypted once for each recipient, producing a different version to be sent to each recipient. • Public key encryption is far less efficient than secret key encryption. So it is desirable to encrypt the message with secret key encryption even if Bob's key is a public key. • As described in §7.8 Session Key Establishment, it's not desirable to use a long-term key more than necessary. So to preserve the useful lifetime of the long-term key Alice and Bob share, it is preferable to encrypt as little as possible with a key that is expensive to replace. 17.5.2 PRIVACY 449 So the way it is done is that Alice chooses a random secret key S to be used only for encrypting that one message. She encrypts the message with S, encrypts S with Bob's key, and transmits both quantities to Bob. Even if the message is being sent to multiple recipients, she only encrypts the message once, with key S. But she encrypts S once for each recipient, with the appropriate key, and includes each encrypted S with the encrypted message. Let's assume Bob's key is KBob, Carol's key is KCarol, and Ted's key is KTed. If public key technology is being used, then KBob is Bob's public key. If secret key technology is being used, then KBob is the key that Alice shares with Bob. The message Alice would like to send to Bob, Carol, and Ted is m. S is the secret key Alice chose specifically for encrypting m. (We'll write K{x} to indicate x encrypted with key K.) The mail message Alice will send includes: • Bob's name; KBob{S} • Carol's name; KCarol{S} • Ted's name; KTed{S} • S{m} So, for example, the message Alice sends might look like this: To: Bob, Carol, Ted From: Alice Key-info: Bob-4280234208034 Key-info: Carol-48235235344848488 Key-info: Ted-99349248 Msg-info: RJekcr283hkjsdf8ghn327&3489724&#$*92 In that way, each recipient can decrypt the appropriate encrypted version of the secret key, and each will use the same secret key to decrypt the message. 17.5.2 Privacy with Distribution List Exploders Suppose Alice is sending a message to a distribution list which will be remotely exploded, and Bob is only one of the recipients. Assume the distribution list is stored at some node remote from Alice, and that Alice does not even know the individuals on the distribution list. We can't assume she has keys for all the members of the distribution list. Instead, she has a key only for the distribution list exploder. Alice does not need to treat the distribution list exploder any differently than any other recipient of a message. It is merely someone she shares a key with and sends messages to. The distribution list exploder will need keys for all the individuals on the list. As described in the end-to-end case, Alice will choose a random per-message secret key S, and encrypt the message with S. The distribution list exploder will decrypt S (but it does not need to 450 ELECTRONIC MAIL SECURITY 17.6 decrypt the message!), and re-encrypt S with the key for each recipient to whom it is forwarding the message. Local exploding requires different mechanisms. Alice still has to trust the maintainer of the mailing list, since a bad guy could insert extra names into the distribution list. The distribution list is most likely just a list of names. Alice will not be able to send a secure message to a name without establishing a key for that individual. 17.6 AUTHENTICATION OF THE SOURCE With traditional (unsecured) mail systems, it is possible for Carol to send a message to Bob where the FROM: field says Alice. Obviously this can cause great harm if Bob takes the message seriously. For instance, Alice might be Bob's manager, and the message Carol sends might say Urgent meeting with a customer in Juneau, Alaska; I've left; meet me there tomorrow. It would be desirable for the message system to assure Bob that the message did indeed come from Alice. 17.6.1 Source Authentication Based on Public Key Technology Assuming Bob knows Alice's public key, Alice can digitally sign the message, using her private key, which will assure Bob that Alice wrote the message. The method usually chosen to sign a message is for Alice to first compute a hash of the message (for instance using MD5), and then to sign the message digest, since computing a message digest is faster than public key operations, and the message digest is usually a smaller quantity to sign than the message. This method extends easily to multiple recipients and distribution lists. If Alice is sending a message to multiple recipients, the same signature will work for all recipients. If Alice does not know that Bob knows her public key, she could send the message anyway and let Bob fetch the public key (together with finding a chain of certificates to certify Alice's key) if he needs to. Or she could include the public key in the header of the message, together with a chain of certificates. Or she might guess that he already has a lot of the certificates in the chain, and she might just furnish what she thinks is likely to be the remainder. Note that Alice doesn't even have to know whether Bob is crypto-capable to send a signed message. Sometimes users send signed messages just to advertise the fact that they can. If the signature is formatted innocuously, it just looks like cybercrud in the mail header to non-crypto-capa- 17.6.2 AUTHENTICATION OF THE SOURCE 451 ble recipients. Of course, added cybercrud sometimes annoys humans, even if it doesn't prevent them from deciphering the message. For example, the message would look something like this when received by Bob: 17.6.2 Source Authentication Based on Secret Keys If Alice has a shared key with Bob, then she can reassure Bob that she is Alice by proving she knows the shared secret key. She does this by performing some cryptographic computation on the message using the shared secret key. This quantity also serves as an integrity check, and is called a MIC (message integrity code) by some and a MAC (message authentication code) by others (i.e., MIC and MAC are synonyms). The PEM standard calls it a MIC, but the term MAC is more commonly used today. Various possibilities for computing a MAC include: 1. The MAC is the CBC residue of the message computed with the shared secret key. 2. The MAC is the message digest of the shared secret appended to the message. 3. The MAC is the encrypted message digest, where the 128-bit message digest is encrypted with the shared secret key, for instance in ECB or CBC mode. From: Alice To: Bob Subject: I got PEM working ... and I'm sending this message with PEM. To: Bob Subject: I got PEM working Date: Fri, 01 Apr 94 10:12:37 -0400 From: Alice -----BEGIN PRIVACY-ENHANCED MESSAGE----- Proc-Type: 4,MIC-CLEAR Content-Domain: RFC822 DEK-Info: DES-CBC,31747476B4831B1D Originator-ID-Asymmetric: MEMxCzAJBgNVBAYTAlVTMSYwJAYDVQQKEx1EaWd pdGFsIEVxdWlwbWVudCBDb3Jwb3JhdGlvbjEMMAoGA1UECxMDTEtH,02 MIC-Info: RSA-MD5,RSA,u1OHP1RwLqePAoaN5nPk9W7Q2EfjaP+yDBSaLyMcSgc MK2YicGSAqLz47Ol+TUR4YmMD/JnHMtsCJerV2QFtFQ== ... and I'm sending this message with PEM. -----END PRIVACY-ENHANCED MESSAGE----- 452 ELECTRONIC MAIL SECURITY 17.6.3 What if there are multiple recipients? Alice's shared secret with Bob will be different from Alice's shared secret with Ted. If the MAC method is 1 or 2 above, then Alice would have to do a computation-intensive cryptographic pass over the long message for each recipient. The third method is therefore preferable. Alice computes the message digest once, and computes the MAC on the message digest independently for each recipient. 17.6.3 Source Authentication with Distribution Lists With public keys, source authentication is easy, even with distribution lists. Basically, the distribution list will merely forward the message without changing the authentication information. Alice's public key will authenticate the message. The distribution list exploder plays no part in the source authentication. With secret keys it is more complicated. We can't assume Alice will share a secret key with every recipient, or even that Alice will know who the recipients are. The only choice is for Alice to share a key with the distribution list exploder. The distribution list exploder will be able to verify that the message came from Alice, but whatever authentication information Alice puts into the message will become meaningless after the distribution list exploder forwards the message. It must remove Alice's authentication information, and replace it with its own authentication information for each recipient. The recipients will know the message came from the distribution list exploder, but they will have to take the exploder’s word that the message really did originate from Alice. When using mail exploders and secret key technology, it is vital that the mail exploder verify the source before forwarding the message, and include the name of the sender in the body of the message that the mail exploder cryptographically protects. PEM does not define a field to accomplish this, which would make it problematic to use secret key PEM with mail exploders. However, this was never a problem because: • Nobody deployed PEM-aware mail exploders. • Nobody used secret key PEM. 17.7 MESSAGE INTEGRITY When Bob receives a message from Alice, how does he know that Carol did not intercept the message and modify it? For instance, she might have changed the message Fire Carol immediately to Promote Carol immediately. 17.7.1 MESSAGE INTEGRITY 453 It makes little sense to provide for authentication of the source without also guaranteeing integrity of the contents of the message. Who would care that the message originated with Alice, if the contents might have been modified en route? Indeed, all of the mail standards provide message integrity and source authentication together. They either provide both or neither. The mechanisms discussed in the previous section for source authentication provide message integrity as well. 17.7.1 Message Integrity without Source Authentication Might someone want to provide message integrity protection without source authentication? The only vaguely plausible example we could come up with for when this might be useful was a ransom note. The kidnappers would want to prove that they were the kidnappers, perhaps by extracting some secret information from the victim. The note would give the proof that they were the kidnappers, and include directions for dropping off the ransom money. They’d want to keep the message private, since they wouldn’t want someone else to get that secret, convince the family they were the kidnappers, and collect the ransom. Nor would they want some third party to see the directions for dropping off the ransom money, since someone else might try to pick up the money. (Kidnappers are such suspicious people.) The message would be anonymous, so there would be no source verification. Integrity protection would be somewhat important, to prevent someone from guessing that the first part of the message was the proof that they were the kidnappers, and substitute a different second part of the message, with different directions for dropping off ransom money. In some sense in this example there is source authentication—it’s just not done with cryptography by the mail system. Instead the contents of the message provide source authentication by humans. We know of no systems that provide this odd combination (message integrity without source authentication), and kidnappers represent such a small market that none is likely to be developed. It’s meaningless to provide integrity protection without source authentication if the message is unencrypted, because anyone can remove the message from the wire and construct a different message. It isn’t possible to do message integrity without source authentication with secret key technology, since the sender must share a secret with the recipient, so the recipient will know who the message came from based on which secret key is used. If the message is encrypted (where the sender knows the public key of the recipient), it is possible to do something that could be considered integrity protection without source authentication. It is still possible for someone to remove the entire message from the wire and substitute a different message. But what can be ensured with an encrypted message is that the entire message came from one source, although the source is anonymous. 454 ELECTRONIC MAIL SECURITY 17.8 17.8 NON-REPUDIATION Repudiation is the act of denying that you sent a message. If the message system provides for non-repudiation, it means that if Alice sends a message to Bob, Alice cannot later deny that she sent the message. Bob can prove to a third party that Alice did indeed send the message. For instance, if Alice sends a message to the bank that she wishes to transfer a million dollars to

Bob’s account, the bank might not honor the transaction unless it is sent in such a way that not only can the bank know the message was from Alice, but they can prove it to a court if necessary. It is not always desirable to have non-repudiation. For instance, Alice might be the head of some large organization and want to give the go-ahead to her underlings for some scheme like, say, selling arms to Iran to raise money to fund the Contras. The underlings must be absolutely certain the orders came from Alice, so it is necessary to have source authentication of the message, but Alice wants plausible deniability (what a great phrase!), so that if any of her underlings are caught or killed, she can disavow any knowledge of their actions. With public keys, it is very natural to provide non-repudiable source authentication, and more difficult to provide repudiable source authentication. With secret keys, it is very natural to provide repudiable source authentication and very difficult to provide non-repudiable messages. 17.8.1 Non-Repudiation Based on Public Key Technology The usual method Alice uses to prove the authenticity of her message to Bob is to include her public key signature on the message digest of the message, using her private key. This method of source authentication not only reassures Bob that Alice sent the message, but Bob can prove to anyone else that Alice sent the message. Only someone with knowledge of Alice’s private key could have signed the message digest with it. But anyone knowing her public key can verify her signature. 17.8.2 Plausible Deniability Based on Public Key Technology How can Alice send a message to Bob in such a way that Bob knows it came from Alice, but Bob can’t prove to anyone else that it came from Alice? Alice can send a public key based repudiable message m to Bob by doing the following. First we’ll review our notation. We use curly braces {} for encrypting something with a public key, with a subscript specifying the name of the individual whose public key we are using. We use square brackets [] for signing something with a private key, with a subscript specifying the name of the individual whose private key is being used. 17.8.3 NON-REPUDIATION 455 1. Alice picks a secret key S, which she will use just for m. 2. She encrypts S with Bob’s public key, getting {S}Bob. 3. She signs {S}Bob with her private key, getting [{S}Bob]Alice. 4. She uses S to compute a MAC for m (for example, by using DES to compute the CBC residue of the message). 5. She sends the MAC, [{S}Bob]Alice, and m to Bob. Bob will know that the message came from Alice, because Alice signed the encrypted S. But Bob can’t prove to anyone else that Alice sent him m. He can prove that at some point Alice sent some message using key S, but it might not have been m. Once Bob has the quantity [{S}Bob]Alice he can construct any message he likes and construct an integrity code using S. 17.8.3 Non-Repudiation with Secret Keys It is surprising that it is possible to provide non-repudiation with secret keys. Let’s say Alice is sending a message to Bob, and it has to be possible for Bob to prove later, to “the judge”, that Alice sent the message. We’ll need a notary N who must be trusted by Bob and the judge. (A notary is a trusted service that vouches for certain information.) Alice sends the message to N, and does source authentication with N so that N knows the message came from Alice. N does some computation on the message and Alice’s name using a secret quantity SN that N shares with nobody. We’ll call the result of this computation N’s seal on the message. (For instance N’s seal might be the message digest of the concatenation of Alice, the message, and SN.) Then the message together with the seal is sent to Bob. Bob can’t tell whether N’s seal is genuine, because that would require knowing SN (which N shares with nobody). If Alice were dishonest and wanted to send a message to Bob that Bob would only accept if it were non-repudiable, then Alice could put in any cybercrud and pretend that it is authentication information from N. There are two things we can do to assure Bob that the message was vouched for by N, and that therefore he can later prove to the judge that N vouched for the message. • Bob and N could establish a shared key, and N could use it to generate a MAC for the concatenation of Alice, the message, and the seal. Bob can verify this

MAC (since he knows the shared key) and thus verify the validity of the seal. • Bob could, after receiving the message, send a request to N asking N if N's seal is genuine. This request would of course have to include Alice, the message, and the alleged seal. 456 ELECTRONIC MAIL SECURITY 17.9 Now let's assume at some later time Bob needs to prove to the judge that Alice sent the message. Bob sends Alice, the message, and the seal to the judge, who sends it to N in an integrity-protected conversation. N verifies the validity of the seal and reports that to the judge. Note that N does not actually have to see the message, in case Alice and Bob want to keep the message private. Alice could instead just send the message digest to N, and N could compute the seal of the message digest. 17.9 PROOF OF SUBMISSION When we send paper mail, the postal service allows us to send it certified, which means we pay extra money and get a piece of paper that verifies we submitted something on a particular date to a particular address. Some electronic mail systems offer a similar service, but by using cryptography on the contents of the message, it turns out to be more powerful than the postal service certified mail concept. The mail system can implement this by computing a message digest of the message concatenated with any other information that might be useful, like the time of submission, and then sign the message digest. The user could later use the receipt to prove that the message was sent (though not that it was delivered). 17.10 PROOF OF DELIVERY This feature corresponds to the paper mail postal service return receipt requested. It can be implemented in electronic mail by having either the destination or the mail delivery service sign a message digest of the message concatenated with any other information that might be useful (like the time of receipt). If the mail service gives the proof of delivery, it would be done after transmitting the message to the destination. If the destination gives the proof of delivery, it would be done after the destination receives the message. In either case this requires the cooperation of the recipient. It obviously requires the cooperation of the recipient if the recipient returns the proof of delivery. But it also requires the cooperation of the recipient if the mail system returns the proof of delivery, since the recipient might refuse to acknowledge the last packet of the message (once the recipient realizes the message is an eviction notice, for instance). Note that it is impossible to provide a receipt if and only if the recipient got the message. The problem is, if the recipient signs before the message is delivered, then the message can get lost, but 17.11 MESSAGE FLOW CONFIDENTIALITY 457 the mail system has the signature. If the recipient signs after receiving the message, the recipient might not furnish a signature at that point, but yet have the message. 17.11 MESSAGE FLOW CONFIDENTIALITY This feature allows Alice to send Bob a message in such a way that no eavesdropper can find out that Alice sent Bob a message. This would be useful when the mere fact that Alice sent Bob a message is useful information, even if the contents were encrypted so that only Bob could read the contents. (For instance, Bob might be a headhunter. Or Bob might be a reporter and Alice might be a member of a secret congressional committee who is leaking information to the press.) The X.402 standard hints that such a service might be provided and hints at mechanisms for providing it. If Alice knows that intruder Carol is monitoring her outgoing mail to look to see if she is sending messages to Bob, she can utilize a friend as an intermediary. She can send an encrypted message to Fred, with the message she wants to send to Bob embedded in the contents of the message to Fred. So the message Fred would read is This is Alice. Please forward the following message to Bob. Thanks a lot. followed by Alice's message to Bob. If Alice were sufficiently paranoid, she might prefer a service that is constantly sending encrypted dummy messages to random recipients. If Fred forwards Alice's message after a random delay, then even if Carol knows Fred provides this service, she cannot know to which recipient Alice was asking Fred to forward a message. To be even more paranoid, a path through several intermediaries could be used. Alice does this by taking the (encrypted) message she'd like to send to Bob, enclosing that in an encrypted message to the last intermediary, enclosing that in an encrypted message to the penultimate intermediary, encloses that—you get the idea. Using the multiple intermediary technique, even if someone bribed one of the intermediaries to remember the directions from which it received and sent mail, Alice could not be paired with Bob. To discover that Alice had sent a message to Bob would require the cooperation of all the intermediaries. 17.12 ANONYMITY There are times when Alice might want to send Bob a message, but not have Bob be able to tell who sent the message. One might think that would be easy. Alice could merely not sign the message. However, most mail systems automatically include the sender's name in the mail message. One 458 ELECTRONIC MAIL SECURITY 17.12 could imagine modifying one's mailer to leave out that information, but a clever recipient can get clues. For instance, the network layer address of the node from which it was sent might be sent with the message. Often the node from which the recipient receives the message is not the actual source, but instead is an intermediate node that stores and forwards mail. However, most mail transports include a record of the path the message took, and this information might be available to the recipient. If Alice really wants to ensure anonymity, she can use the same technique as for message flow confidentiality. She can give the message to a third party, Fred, and have Fred send the unsigned message on to Bob. If Fred does not have enough clients, he can't really provide anonymity. For instance, if Alice is the only one who has sent a message to Fred, then Bob can guess the message came from her, even if Fred is sending out lots of messages to random people. Fred can have lots of clients by providing other, innocuous services, such as weather reports, bookmaking, or pornographic screen savers (where you are required to transmit a fixed-length, encrypted message in order to get a response), or by having the user community cooperate by sending dummy messages to Fred at random intervals to provide cover for someone who might really require the service. (We specified fixed-length messages to Fred so that people could not correlate the lengths of messages into and out of Fred—someone wanting to send a longer message would have to break it into multiple messages, and pad short messages.) There's an amusing anecdote about anonymous mail. Someone does provide an anonymous mail service. The way it works is you send an ordinary mail message to the service, telling it to whom you'd like to send an anonymous message. It strips off your real address and substitutes a pseudonym for you. The first time you send a message it assigns you a pseudonym, but on subsequent messages it fills in the same pseudonym. So people won't know from whom the message came, but they'll know that different messages came from the same person. Mail sent to any of the pseudonyms assigned by the anonymity service will be directed to that service, which will substitute the real address instead of the destination's pseudonym (and incidentally, assign a pseudonym to the source and substitute that—this particular service didn't allow for non-anonymous mail to be sent to a pseudonym). Anyway, one December day in 1993, someone decided to add him/herself to the IETF mailing list as the pseudonym assigned by the service. Then someone sent a message to the IETF mailing list (it happens). When the message was distributed by the mailing list, it was sent to all the members, including the pseudonym. When this pseudonym-addressed copy was received by the anonymous mail service, it assigned a pseudonym to the source (the mailing list) and sent an assignment message back to the mailing list, which of course got distributed to all the mailing list members: To: ietf@nat-flnt{MHS:ietf@CNRI.Reston.VA.US} From: 00000000 {MHS:""@sjf-domain-hub.Novell} Subject: Anonymous code name allocated. Date: 12/28/93 Time: 9:08a You have sent a message using the anonymous contact service. You have been allocated the code name an60254. You can be reached anonymously using the address an60254@anon.penet.fi. If you want to use a nickname, please send a message to nick@anon.penet.fi, with a Subject: field containing your nickname. For instructions, send a message to help@anon.penet.fi. This strange behavior was caused by the anonymity service's assumption that both parties using its service would want anonymity, which is not a reasonable assumption. It should be possible for two anonymous parties to communicate, but it is also reasonable for a non-anonymous party to communicate with an anonymous party. A pseudonym should not be assigned unless requested. 17.13 CONTAINMENT This feature is modeled after nondiscretionary access controls (see §1.13.1 Mandatory (Nondiscretionary) Access Controls). The idea is that the network would be partitioned into pieces that are capable of handling certain security classes. Each message would be marked with its security classification, and the message routers would refuse to forward a message to a portion of the network that didn't handle its security class. 17.14 ANNOYING TEXT FORMAT ISSUES There is unfortunately no single standard for text representation. Some systems have (line feed, ASCII 10) as the line delimiter. Others have (carriage return, ASCII 13). Others have (followed by). Some systems have line-length limitations. Others happily wrap lines. Some systems expect parity on the high-order bit of each octet. Others get upset if the high-order bit is on. Still others don't get upset, but calmly clear the bit. Some systems don't even use ASCII, but use some other encoding for characters, such as EBCDIC. 460 ELECTRONIC MAIL SECURITY 17.14 When sending a text message to someone on another system, we'd like the message to appear to the human on the other end about the way it looked to the human who sent it. This is a messy, boring problem. Your system can't really know what the conventions are at the remote site, and the destination might have difficulties translating appropriately because it would not know what the conventions were at the source. The usual method of solving problems like these is to define a canonical format. The data is translated at the source into canonical format, and at the destination out of canonical format into local format. This way, each system only needs to know how to convert its local format to and from canonical format (as opposed to needing to know how to convert its local format to and from each other possible format). Unfortunately, there is no single standard canonical format. SMTP (Simple Mail Transport Protocol, RFC 821) is a standard for Internet mail. The companion RFC 822 defines a canonical format that uses as the line delimiter. But there are non-SMTP mail systems, and messages can get modified in non-reversible ways as messages pass through various sorts of gateways. For instance, some mail gateways remove white space at the ends of lines (spaces immediately preceding the end of the line), add line breaks to lines that seem too long to fit on a screen, or turn the tab character into five spaces. These sorts of transformations are mostly harmless when trying to send simple text messages from one user to another across diverse systems, but it causes problems when trying to send arbitrary data between systems. Certainly if you're trying to send arbitrary binary data, having all your high-order bits cleared or having inserted periodically will hopelessly mangle the data. But even if you are sending text, security features can get broken by mail gateways diddling with the data. For instance, a digital signature will no longer verify if any of the message is modified. It's difficult to predict exactly what kinds of modifications an arbitrary mail gateway will make to your file in its attempt to be helpful. Mail is not the only application which has this problem. File transfer also has the problem. It would be easy just to send the bits in a file and have them appear at the destination machine exactly as they were at the source machine, but that is often not what is wanted. Because of the different representations for text, a text file might not display intelligibly on a different system. (And of course there are other problems besides displaying text. A floating point number is represented differently on different machines. Maybe you'd even like your executable code to run on a different machine!) Most file transfer protocols give you the choice of whether you are sending binary data (in which case it won't muck with your data) or text (in which case it will attempt to do transformations on the data so that it will display properly). SMTP made the assumption that the only thing anyone would ever want to

mail was text, so there is no capability of sending arbitrary data. The mail infrastructure is always allowed to play around with your data in an attempt to be helpful. 17.14.1 ANNOYING TEXT FORMAT ISSUES 461 17.14.1 Disguising Data as Text People have long wanted to mail types of data other than simple text (for instance, diagrams, voice, pictures, or even text in languages with large character sets), and they have done so by coming up with an encoding that will pass through all known maulers ;-). The classic way UNIX systems did this was to run a program called uuencode on the data, mail the result, and have the recipient uudecode it. The encoding process accepts arbitrary octet strings and produces strings containing only "safe" characters with an occasional end-of-line delimiter to keep the mailers happy. There are obviously fewer than 256 safe values for an octet (otherwise, arbitrary binary data would work just fine). It turns out there are at least 64 safe characters, but not much more than that, so the encoding packs 6 bits of information into each character, by translating each of the possible values of the 6 bits into one of the safe characters, and adding sufficiently often. This encoding expands the information by about 33%. S/MIME, PEM, and PGP make the assumption that they have to work with a mail system such as SMTP that will assume their data is text, so they both have a convention similar to uuencode/uudecode for packing and unpacking 6 bits into a character. We will use the term encode to mean any similar mechanism, even if it isn't exactly the transformation that the UNIX utility provides. X.400 has a method of transmitting data unmodified, so it does not need to do this. So there are two transformations of data. One is canonicalization, which is minor diddling such as changing the end-of-line character into . The other is encoding, which is packing 6 bits per character, thereby making the data completely unintelligible to humans and expanding it by about 33%. There is a troublesome consequence of encoding, in that you can't always assume that the recipient of the mail message has S/MIME, PEM or PGP. The recipient might have to read the message with an ordinary mail program. Obviously they would not be able to read an encrypted message with an ordinary mail program, but you'd want them to be able to read a message that is unencrypted but integrity-protected. Suppose Alice sends the message Bob, your terminal is about to explode. You'd better quickly run out of your office. She decides to use PEM so that it will add an integrity check to the message, since she really cares that Bob gets the message without it getting garbled. PEM helpfully reformats the message so that it now looks like: -----BEGIN PRIVACY-ENHANCED MESSAGE----- Proc-Type: 4,MIC-ONLY Originator-ID-Asymmetric:
MEMxCzAJBgNVBAYTAlVTMSYwJAYDVQQKEx1EaWd
pdGFsIEVxdWlwbWVudCBDb3Jwb3JhdGlvbjEMMAoGA1UECxMDTEtH,05 MIC-Info: RSA-MD5,RSA,ZHiLe5DJdd4fE1/w++csvPbpFG9K9PWkPwU3fSLh9I+
ZNyFdZ7pBk1zPvXrya5+14XQcwIcb3Dcqk2RbnyzWzQ==
Qm9iLCB5b3VyIHRlcm1pbmFsIGlzIGFib3V0IHRvDQpleHBsb2RlLiAgWW91J2Qg 462 ELECTRONIC MAIL SECURITY 17.14.1
YmV0dGVyIHF1aWNrbHkgcnVuIG91dCBvZiB5b3VyIG9mZmljZS4NCg== -----END PRIVACY-ENHANCED MESSAGE----- The first five lines of cybercrud include information like the integrity check. The actual message is the bottom two lines:
Qm9iLCB5b3VyIHRlcm1pbmFsIGlzIGFib3V0IHRvDQpleHBsb2RlLiAgWW91J2Qg
YmV0dGVyIHF1aWNrbHkgcnVuIG91dCBvZiB5b3VyIG9mZmljZS4NCg== Suppose Bob's mail program has not yet implemented PEM. It is reasonable to expect him to have problems understanding an encrypted message, but Alice did not encrypt the message. She merely requested an integrity check be added to the message. But PEM had to reformat the data, because the integrity check would fail if the text was modified in any way. The human Bob will have trouble processing
Qm9iLCB5b3VyIHRlcm1pbmFsIGlzIGFib3V0IHRvDQpleHBsb2RlLiAgWW91J2Qg

YmV0dGVyIHF1aWNrbHkgcnVuIG91dCBvZiB5b3VyIG9mZmljZS4NCg== It's not encrypted—it is easy to turn it back into human-readable text, if there is a utility at Bob's node to do this. Without such a utility, Bob will probably take longer than is prudent (given the contents of Alice's message) to read the message. Because of this problem, S/MIME, PEM, and PGP have the capability of sending something without modification. When you use that mode, they will not modify the message. It's your problem if mail gateways modify the message and the integrity check no longer matches. The programs add an integrity check without encoding the message. The message would then look like this: -----BEGIN PRIVACY-ENHANCED MESSAGE----- Proc-Type: 4,MIC-CLEAR Originator-ID-Asymmetric: MEMxCzAJBgNVBAYTAlVTMSYwJAYDVQQKEx1EaWd pdGFsIEVxdWlwbWVudCBDb3Jwb3JhdGlvbjEMMAoGA1UECxMDTEtH,05 MIC-Info: RSA-MD5,RSA,ZHiLe5DJdd4fE1/w++csvPbpFG9K9PWkPwU3fSLh9I+ ZNyFdZ7pBk1zPvXrya5+14XQcwIcb3Dcqk2RbnyzWzQ== Bob, your terminal is about to explode. You'd better quickly run out of your office -----END PRIVACY-ENHANCED MESSAGE----- If Alice sends her message to Bob in this mode using something like PEM, then it is possible that she'll be lucky and one of the following will be true: • The message happens to be encoded in a way which no mail utility en route will feel obliged to modify. Bob's mailer has PEM and the integrity check will succeed. 17.15 NAMES AND ADDRESSES 463 • Bob's mailer does not have PEM, and the message does not get maliciously corrupted en route by some third party Carol. What Bob sees is the actual text of the message, plus some cybercrud for the signature and so on, which he ignores. It is possible that she'll be unlucky (or rather, Bob will be unlucky in this case) and one of the following will happen: • Bob's mail has PEM, and the message gets modified by a helpful mail utility in a "harmless" sort of way. The integrity check fails and Bob ignores the message. • Bob's mailer does not have PEM, and bad-guy Carol modifies the message en route into something like, Are you still free for tennis today after work? In either case, Bob gets blown to bits. 17.15 NAMES AND ADDRESSES The designers of PEM (and S/MIME) chose X.509 for the certificate format. This was an unfortunate choice, since the email applications did not use X.500 names, which is what appeared in X.509 certificates (at the time when these standards were developed). What good is a certificate that certifies the public key of an X.500 name such as /C=US/O=CIA/OU=drugs/PN='Manuel Noriega' when the human specified the recipient with an Internet email address such as Alice@Wonderland.Edu? The designers were assuming that the user would specify the X.500 name of the recipient, and some service would translate that into an email address by looking it up in some sort of directory. There was no need to have a certificate stating the recipient's email address because at worst the encrypted message to the name would get delivered to the wrong mailbox, but encrypted with a key that the reader of the wrong mailbox would not have. But a user interface of specifying an X.500 name of a recipient rather than an email address of the recipient did not catch on (and the directories for mapping X.500 name to email address didn't get deployed). So to solve the problem that the user identified the recipient by a name that was not in the X.509 certificate, the S/MIME standard specifies that the email address should appear as an alternate name in the certificate. Hopefully the issuer of the certificate would be at least as careful about checking the mapping of the key with the alternate name containing the email address, as they would be with the subject name (the X.500 name) in the certificate. Especially since, if used with email, the subject name would be ignored by the application, according to the standard. But since support of certificates with the alternate name field (X.509 version 3) was slower than the rollout of S/MIME, what implementations did was to invent a new component of the X.500 464 ELECTRONIC MAIL SECURITY 17.16 name which would contain the email address, e.g., /C=US/O=CIA/OU=drugs/PN='Manuel Noriega'/E=Alice@Wonderland.Edu. The other components of the X.500 name are ignored (except that the user could display them through some optional UI). 17.16 VERIFYING WHEN A MESSAGE WAS REALLY SENT You can't depend on the date in a signed message being accurate. Someone who steals your private key can put an old date in a signed message. Someone with use of a key can sign a message with a date in the future as well. Keys get revoked. Keys get changed. How can one look at a five-year-old message and know whether the mail message was valid at the time it was sent? We'll discuss several scenarios and ways for coping with these circumstances, based on use of a notary (see §17.8.3 Non-Repudiation with Secret Keys). People have suggested all sorts of digital notary services, but we claim that all services can be accomplished with a notary that simply, given a pile of bits, will date and sign the pile of bits. 17.16.1 Preventing Backdating An example problem that this protects against is a dishonest buyer. Suppose there is a company that allows electronic purchase orders. The deal is that you are responsible for any purchase orders signed with your key, unless you had reported your key stolen. From the time you declare the key stolen, all purchase orders signed with the key are invalid, but all purchase orders signed before that time are valid. Suppose Alice signed a purchase order with her private key a week ago. Then she decides she wants to renege on the deal, so she reports her key stolen. The company knows they received the purchase order a week ago, but how can they prove it? We can assume the purchase order has a date in it, which is integrity-protected with the signature, but anyone who has the private key can backdate a message. One way the company can protect itself is to send each purchase order, when received, to a notary. Our version of the notary simply dates and signs the message, so the company can prove they received the message before the timestamp added by the notary. It will also be convenient for the company to store, with the purchase order, the certificates and certificate revocation lists (CRLs) it used originally to verify Alice's signature. Now if Alice denies having sent the message, the company can give a judge Alice's notarized message along with the certificates and CRLs, and the judge can determine that Alice's message was legally binding at the time it was received. 17.16.2 HOMEWORK 465 If the company is worried about Alice controlling a certification authority (CA), and therefore being able to fake dates on certificates or CRLs, then the company can have the notary also sign and date the certificates and CRLs. See Homework Problem 10. 17.16.2 Preventing Postdating The notary, as described above, can only prove that a message was generated before the date on which the notary signed it. Someone could hold a message for years, and then have the notary sign it. Suppose you want to prove that a message was generated after a certain date. This can be accomplished by including in the message something that you could not have known until that date, for instance the winning lottery number in some daily lottery drawing with sufficiently large numbers, or a signed timestamp from a notary for that day. Kidnappers often use something like this to prove their victim is still alive (a photo of the victim holding the latest newspaper). In the cryptography world, this technique might be useful to ensure that someone that has temporary use of the CRL signing key does not postdate a CRL (without his certificate on it), to substitute for the real CRL generated in the future that would have his certificate on it. 17.17 HOMEWORK 1. Outline a scheme for sending a message to a distribution list, where distribution lists can be nested. Attempt to avoid duplicate copies to recipients that are members of multiple lists. Discuss both the local exploder and remote exploder methods of distribution list expansion. 2. If using secret keys for user keys, and sending a multi-recipient message, why is encrypting the MD of the message with the secret key associated with the recipient a more efficient MAC than doing a keyed MD of the message (again, using the secret key associated with the recipient)? 3. Using tickets as in §17.4.2 Establishing Secret Keys, suppose Alice's KDC is different from Bob's KDC. Alice will have to use a chain of KDCs to get a ticket for Bob (see §7.7.4.1 Multiple KDC Domains). Does Alice need to include the entire chain of tickets or merely the final ticket to Bob? How is this different from use of certificates? 4. Suppose Alice sends an encrypted, signed message to Bob via the mechanism suggested in §17.8.2 Plausible Deniability Based on Public Key Technology. Why can't Bob prove to third 466 ELECTRONIC MAIL SECURITY 17.17 party Charlie that Alice sent the message? Why are both cryptographic operations on S necessary? (Alice both encrypts it with Bob's public key and signs it with her private key.) 5. Assume we are using secret key technology. What is wrong with the following source authentication scheme? Alice chooses a per-message secret key K, and puts an encrypted version of K in the header for each recipient, say Bob and Ted. Then she uses K to compute a MAC on the message, say a DES-CBC residue, or by computing a message digest of K appended to the message. (Hint: it works fine for a single recipient, but there is a security problem if Alice sends a multiple-recipient message.) 6. Using the authentication without non-repudiation described in §17.8.2 Plausible Deniability Based on Public Key Technology, Bob can forge Alice's signature on a message to himself. Why can't he forge Alice's signature on a message to someone else using the same technique? 7. Suppose you changed the protocol in §17.8.2 Plausible Deniability Based on Public Key Technology so that Alice first signs S, and then encrypts with Bob's public key. So instead of sending [{S}Bob]Alice to Bob, she sends {[S]Alice}Bob. Does this work? (i.e., can Bob be sure that the message came from Alice, but not be able to prove it to a third party) 8. Which security features can be provided without changing the mail delivery infrastructure, i.e., by only running special software at the source and destination? 9. Which security features (privacy, integrity protection, repudiability, non-repudiability, source authentication) would be desired, and which ones would definitely not be used, in the following cases: • submitting an expense report • inviting a friend to lunch • selling illegal pornography over the network • running an (illegal) gambling operation on the network • sending a mission description to the Mission Impossible team • sending a purchase order • sending a tip to the IRS to audit a neighbor you don't like but are afraid of • sending a love letter • sending an (unwanted) obscene note • sending a blackmail letter • sending a ransom note 17.17 HOMEWORK 467 • sending a resume to a headhunter 10. Suppose you are a judge trying to decide a dispute between a buyer and a supplier. The buyer claims not to have produced a particular email purchase order, while the supplier shows you the purchase order, and certificates and CRLs demonstrating that the purchase order was signed by the buyer, all signed by a notary. How would the dates on the various pieces of evidence influence your decision? What if only the purchase order was signed by the notary? Note that the security community continues to debate issues such as whether the CRL should be required to have a later date than the notary's signature. 11. Consider the following format for Alice sending signed, encrypted email to Bob. Alice invents a secret key S and sends {S}Bob, the message encrypted with S, and the message digest of the message signed by Alice. Is this secure? (hint: the signed message digest is not encrypted) This page is intentionally left blank 469 18 PEM & S/MIME 18.1 INTRODUCTION PEM (Privacy Enhanced Mail) was developed by the Internet community in the late '80s and early '90s as a means of adding encryption, source authentication, and integrity protection to ordinary text messages—the dominant form of email at the time. It is documented in four pieces. RFC 1421 describes the message formats. RFC 1422 describes the CA hierarchy. RFC 1423 describes a base set of cryptographic algorithms that can be used with PEM. RFC 1424 describes mail message formats for requesting certificates and for requesting or posting certificate revocation lists. At about the same time, email was evolving to be more than just text. MIME (Multipurpose Internet Mail Extensions, RFC 2045) specified a standard way of encoding arbitrary data in email: pictures, rich text, video clips, binary files, etc. And it specified a way to indicate what the format of the data was so that a recipient could figure out how to process the data. S/MIME (RFC 2633) took many of the design principles of PEM, and incorporated them into the MIME structure, adding signed data and encrypted data as new types of data. Even though PEM has pretty much died out, we'll discuss it since its text based encoding makes it possible to show examples. S/MIME has a format that is similar if you are a computer, but which is much harder to read if you're a human. Once you understand the concepts in PEM, it's simple to translate the ideas into S/MIME. PEM and S/MIME were designed to work by having smart software only at the source and destination (as opposed to in the mail infrastructure, for instance in the mail gateways). They assume that the message they want to send will be carried by a mail infrastructure which might be based on some simple-minded mail transport protocol that might only be able to send ordinary text. A lot of PEM's, MIME's, and S/MIME's complexity involves encoding information in such a way that it will pass unmodified through all the mailers known at the time (there's no guarantee that someone won't invent a mail gateway which will mangle data in new and innovative ways, though). The design of PEM lets you base user keys on secret key or public key technology, but basing user email keys on secret keys never caught on, for reasons that are pointed out in Chapter 17 Electronic Mail Security. (Message encryption is always done with secret key technology, even when user keys are based on public key technology—see §17.5.1 End-to-End Privacy.) Even though 470 PEM & S/MIME 18.2 secret key PEM never caught on, it's interesting to see how the PEM designers envisioned their use. S/MIME only defines how to use public key technology for user keys. PEM and S/MIME were both designed to be used with a variety of cryptographic algorithms, and over time the set of specified algorithms has grown to include RSA, DSS, DES, RC2, 3DES, AES, and others. First we'll describe PEM, and then we'll describe how to translate the ideas into S/MIME. 18.2 STRUCTURE OF A PEM MESSAGE A mail message can contain pieces that have been processed in different ways by PEM. For instance, part of a message might be encrypted, and another part might be integrity-protected. PEM puts markers before and after such blocks so that the PEM at the destination will know which pieces need what processing. PEM marks a piece that it has processed in some way (for instance encrypted) with a text string before and after the piece. PEM will insert -----BEGIN PRIVACY-ENHANCED MESSAGE----- before and insert -----END PRIVACY-ENHANCED MESSAGE----- at the end of the piece of the message that PEM has processed. Extra information needs to be sent along with the PEM-processed message, for instance the (encrypted) key used to encrypt the data, or the message integrity code (MIC)1 . S/MIME encodes this information as pure cybercrud, but PEM inserts a label for each piece of information attached to the message it has processed (apparently to make books like this one more readable). There are labels for each piece of PEM information, for instance Proc-Type, Originator-ID-Asymmetric, and MIC-Info. The intention is that most humans will ignore that stuff. The delimiting strings are indeed helpful to alert a human that might not have a PEM-capable mail reader as to why the message seems like junk. In §18.15 Message Formats we will walk through the PEM message syntax in its full glory, explaining exactly what each field means and which variations are allowed. The different types of pieces PEM can combine into a message are • ordinary, unsecured data • integrity-protected unmodified data—An integrity check is added to the message, but the original message is included unmodified as part of the PEM message. The PEM terminology for this kind of data is MIC-CLEAR. The assumption is that the text will not be garbled in annoying ways along the route by meddling mail utilities. Otherwise the integrity check will 1. PEM refers to the cryptographic message integrity code as a MIC, which is a synonym for MAC, the term that has become more widely used. 18.2 STRUCTURE OF A PEM MESSAGE 471 not work. See §18.7 Reformatting Data to Get Through Mailers. (Note that, as with the types below, the integrity check also provides source authentication.) • integrity-protected encoded data—PEM first encodes the message so that it will pass through all mailers unmodified. Then PEM adds an integrity check. PEM calls this MIC-ONLY. Since the encoded message is not readable by normal humans, the mail program at the destination must be able to convert the text back into human readable format. • encoded encrypted integrity-protected data—PEM computes an integrity check on the message. PEM then encrypts the message and the integrity check with a randomly selected per-message secret key. The encrypted message, the encrypted integrity check, and the per-message key (encrypted by the interchange key—see §18.3 Establishing Keys) are each then encoded to pass through mailers as ordinary text. PEM calls this ENCRYPTED. Clearly in this case the mail program at the destination must be PEM-aware so that it can decrypt the message. For example, let's assume the simple message The MIC-CLEAR version would look like this: From: Alice To: Bob Subject: Keep this proposition private! Date: Fri, 01 Apr 94 10:12:37 -0400 Care to meet me at my apartment tonight? From: Alice To: Bob Subject: Keep this proposition private! Date: Fri, 01 Apr 94 10:12:37 -0400 -----BEGIN PRIVACY-ENHANCED MESSAGE----- Proc-Type: 4,MIC-CLEAR Content-Domain: RFC822 Originator-ID-Asymmetric: MEMxCzAJBgNVBAYTAlVTMSYwJAYDVQQKEx1EaWd pdGFsIEVxdWlwbWVudCBDb3Jwb3JhdGlvbjEMMAoGA1UECxMDTEtH,02 MIC-Info: RSA-MD5,RSA,u1OHP1RwLqePAoaN5nPk9W7Q2EfjaP+yDBSaLyMcSgc MK2YicGSAqLz47Ol+TUR4YmMD/JnHMtsCJerV2QFtFQ== Care to meet me at my apartment tonight? -----END PRIVACY-ENHANCED MESSAGE----- 472 PEM & S/MIME 18.2 The MIC-ONLY version would look like this: The ENCRYPTED version would look like this: From: Alice To: Bob Subject: Keep this proposition private! Date: Fri, 01 Apr 94 10:15:02 -0400 -----BEGIN PRIVACY-ENHANCED MESSAGE----- Proc-Type: 4,MIC-ONLY Content-Domain: RFC822 Originator-ID-Asymmetric: MEMxCzAJBgNVBAYTAlVTMSYwJAYDVQQKEx1EaWd pdGFsIEVxdWlwbWVudCBDb3Jwb3JhdGlvbjEMMAoGA1UECxMDTEtH,02 MIC-Info: RSA-MD5,RSA,u1OHP1RwLqePAoaN5nPk9W7Q2EfjaP+yDBSaLyMcSgc MK2YicGSAqLz47Ol+TUR4YmMD/JnHMtsCJerV2QFtFQ== G9yYXRpb24xDDAKBgNVBAsTA0xLRzESMBAGA1UEAxMJSm9obiBMaW5uMFcwCgYE -----END PRIVACY-ENHANCED MESSAGE----- From: Alice To: Bob Subject: Keep this proposition private! Date: Fri, 01 Apr 94 10:10:31 -0400 -----BEGIN PRIVACY-ENHANCED MESSAGE----- Proc-Type: 4,ENCRYPTED Content-Domain: RFC822 DEK-Info: DES-CBC,31747476B4831B1D Originator-ID-Asymmetric: MEMxCzAJBgNVBAYTAlVTMSYwJAYDVQQKEx1EaWd pdGFsIEVxdWlwbWVudCBDb3Jwb3JhdGlvbjEMMAoGA1UECxMDTEtH,02 Key-Info: RSA,Pv3W7Ds86/fQBnvB5DsvUXgpK7+6h5aSVcNeYf9uWtly9m2VHzC v2t7A6qgbXtIcf4kaMj1FL2yl9/N9mWpm4w== MIC-Info: RSA-MD5,RSA,FUiVRM3x5Ku0aZveGIJ1hv/hi3Iowpm1iypd9VP7MGw PPQra+42TkbR/2jhnqXyVEXLaJ7BSyNhBh/9znIUj5uk0N7IXeBxX Recipient-ID-Asymmetric: MEMxCzAJBgNVBAYTAlVTMSYwJAYDVQQKEx1EaWdp dGFsIEVxdWlwbWVudCBDb3Jwb3JhdGlvbjEMMAoGA1UECxMDTEtH,05 Key-Info: RSA,dpUp7/QoY9YOZzZCVcIwxIDMN0WbGCFAGN3T+xlV/0pBu2n5+x8 PmBvMUN0NcBi5vtqBS4cfmgShiK0I4zu05Q== 21OHDuHTP5BABnlsqENz1WVerZxxWo2AsPHhm2SIz9qpLMvxT/x0+8UEf8dDsTDr wbQo9/x+ -----END PRIVACY-ENHANCED MESSAGE----- 18.3 ESTABLISHING KEYS 473 Not only can these types of data be combined in a message, but they can be nested inside one another. For instance, assuming public key based user keys, Alice might enclose an integrity-protected message from Fred in an encrypted message to Bob. (Bob, I received the following message from Fred. Should I alert the authorities?) 18.3 ESTABLISHING KEYS Suppose Alice is sending a message to Bob. For an integrity check, or for encryption, PEM has to find appropriate cryptographic keys. The per-message key used to encrypt a message is just a randomly selected number. But there is also a long-term key which PEM refers to as an interchange key.

When public keys are used, the interchange key is Bob's public key. When secret keys are used, the interchange key is a key Alice and Bob share. The interchange key is used to encrypt the per-message key: PEM gives no hint as to how to establish secret key based interchange keys. The assumption is that there is some out-of-band mechanism for doing it, for instance calling someone on the phone and agreeing on a number. The PEM designers could have allowed fields for Kerberos-style tickets to be carried in the message, but interest in secret key based interchange keys for PEM is low, so there hasn't been a lot of thought on how to deploy secret key based PEM. In the case of PEM based on public key technology, PEM defines a certification hierarchy based on the X.500 naming hierarchy. A hierarchical naming scheme means that names contain multiple components, like A/B/C/D, reflecting a tree structure where each successive component gives the next branch downwards. Having the certification hierarchy follow the naming hierarchy means that there is a CA named A/B/C that issues certificates for all the names of the form A/B/C/*, and there's a CA named A/B that certifies all the names of the form A/B/* (including CA A/B/C). While it is hoped that eventually there will exist a global directory service in which all certificates can be stored to make them universally available, the PEM designers considered it important to be able to operate before such a directory service was deployed. For that reason, they specified how the header of a PEM message could contain certificates relevant to authenticating the sender of the message. This not only permits the receiver to verify the signature of a message without access to a directory service, it provides a convenient method for Bob to deliver certificates to someone who needs to know Bob's public key in order to send him an encrypted message. The key is 2582 encrypted with interchange key data... encrypted with message key (2582) 474 PEM & S/MIME 18.4 Thus a convenient way for Bob to send Alice the necessary certificates is for Bob to use PEM to send Alice a message including all necessary certificates in the header. Note that even if Alice and Bob intend that all of their conversations be encrypted, the first message one of them sends the other must be unencrypted (since neither yet knows the other's public key). This will not be necessary if Alice can look up Bob's certificates in some directory service. Note that certificates take up a lot of room, and they may not be necessary. Indeed, Alice may have cached Bob's certificates. In that case, there is no reason for Bob to send the certificates. Some PEM implementations allow the user to tell PEM to leave out the certificates for a particular message. If Bob sends a message without the certificates, but Alice really doesn't have them, then Alice can send Bob a message (or call him on the phone) and ask him to send a message with the certificates. Or perhaps she can retrieve them from the directory service. 18.4 SOME PEM HISTORY The PEM Certificate Hierarchy is somewhat baroque, and to understand why it is the way it is, it helps to know some history. It got to where it is by a series of compromises, each one of which made sense in context. Although it never got deployed and at this point never will be, it is instructive to understand it as an example PKI design. The original vision for the PEM Certificate Hierarchy was to have a large number of independently operated CAs, each having responsibility for a subtree of the global namespace. Most X.500 names would list individuals as members of an organization, and it seemed natural for the organization to operate the CA for its own members. For instance, the organization CIA might have a CA called /C=US/O=CIA, and all the names of people in CIA start out with the string /C=US/O=CIA, for example /C=US/O=CIA/CN=Noriega. If an organization was large, it might create subsidiary CAs and delegate to them control of sub-parts of the organization's name space. For instance, there might be /C=US/O=CIA/OU=plumbers. Someone who worked in that organization would then have a name like /C=US/O=CIA/OU=plumbers/CN=Liddy. A single global root CA would certify the names and public keys of all the independent organizations. To be considered valid by the recipient, a certificate would have to follow the rule that the issuer name (the name of the CA that signs the certificate) be a prefix of the subject name (the name of the owner of the public key being certified). This would guarantee that a CA in one organization could not issue a valid certificate for a user in another. So for instance, with the certificate for Liddy, the issuer name would be /C=US/O=CIA/OU=plumbers and the subject name would be /C=US/O=CIA/OU=plumbers/CN=Liddy. (Note that PEM allows a CA to certify anything in its subtree, so the CA for /C=US/O=CIA would be allowed to issue a certificate for /C=US/O=CIA/OU=plumbers/CN=Liddy.) 18.4 SOME PEM HISTORY 475 PEM was intended to be deployed in the free-wheeling internet community where it would replace a system with virtually no security at all. The vision was for PEM to open up new applications by offering a single uniformly high standard of authentication. It might, for example, be possible to sign purchase orders as PEM messages and consider them legally binding. And it should be possible to send an encrypted message to someone around the world with complete confidence that only the intended recipient would be able to read it. Such confidence requires faith in both the cryptographic mechanisms used and the operational procedures of the people who create certificates. If there are “sloppy” CAs around who will issue certificates to anyone with an honest face, how can you know who is really sending and reading your messages? The original vision for PEM, therefore, set a uniformly high standard for the operational procedures of CAs. Since the compromise of a CA’s key would be particularly disastrous, it would be required that CAs be implemented in specially packaged hardware that made it nearly impossible to extract the CA’s private key from the hardware. Strict administrative rules would be followed in qualifying the people who could operate the CAs and in the procedures they would follow in ensuring that they issued certificates only to people who had adequately proven their identities. The root CA would be particularly sensitive, since its compromise would render the entire PEM infrastructure useless. The PEM designers assumed the root CA would be operated by an organization that would use draconian procedures to protect the integrity of its private key and its signing procedures. Since there would be no way to change the root key, whichever organization was chosen to be the root would have an eternal monopoly. RSADSI (the organization that licensed the RSA patent) was positioning itself to be that organization. This structure had a side effect that it made possible a licensing structure and mechanism for RSADSI to collect fees for issuing certificates, and once this mechanism was sufficiently entrenched, it would allow them to continue to collect fees for certificates even after the RSA patent expired. The same CA hardware that protects the CA’s private key could also act as a “postage meter” which keeps track of the number of certificates it issues and can require payment of a per-certificate fee to RSADSI. Collection of royalties is therefore distributed. RSADSI could enforce the use of metering hardware (or some alternative royalty arrangement) by refusing to certify an organizational CA until some accommodation was reached. Without such an organizational certificate, the certificates issued by a CA would not be accepted by “standard” PEM implementations. This vision of the PEM certificate hierarchy was ultimately rejected by the IETF standards community for a number of reasons. • Universities were not amenable to enforcing the same administrative procedures for authenticating their students as defense contractors were in authenticating their employees. No “one size fits all” compromise could be reached. It may have been discussions of mandatory drug testing for CA operators that pushed this over the edge. 476 PEM & S/MIME 18.5 • There were concerns about the availability and cost of the CA hardware. For several years, organizational certificates and the hardware remained “a few months” away. • There was reluctance to trust RSADSI—or any single party—with the security of the entire system. • There was a reluctance to expand the RSADSI monopoly. RSADSI was the sole licensing authority for the RSA algorithm, so it might seem that it was necessary to make peace with them in order to use PEM, but this was simplistic. The U.S. Government had rights to use the algorithm without payment of royalties. Several companies had already reached agreements with RSADSI and were afraid of having to “pay again”. RSA was only patented within the U.S., and foreign concerns were afraid of having to pay for what they currently used for free. And since the RSA patent would expire on September 20, 2000, people were not happy about the possibility of having to continue to pay RSADSI after that date in order to obtain certificates. In any case, it was decided that this relatively straightforward solution to the certificate hierarchy was unacceptable, and the designers embellished it in the minimal fashion that worked—they added another layer to the hierarchy and explicitly allowed people to go into competition with RSADSI. The result is the hierarchy we’re about to describe. 18.5 PEM CERTIFICATE HIERARCHY RFC 1422 recommends an organization of CAs. It recommends a single root CA, called the IPRA, which stands for Internet Policy Registration Authority and is managed by the Internet Society. Certified by the IPRA are CAs called PCAs (for Policy Certification Authorities), so named because each one has a written policy it will enforce in its issuance of certificates. The design envisions at least three kinds of policies that will be enforced. The names have not been standardized, so we’ll arbitrarily name them for ease of explanation. • High Assurance (HA) A HA CA is meant to be super-secure. The rules aren’t standardized, but some of the ideas are as follows. A HA CA is supposed to be implemented on special hardware designed so that it is tamper-resistant (impossible to extract its private key even if you were to steal the whole box), and managed by people who have passed security checks and perhaps periodic drug tests or other procedures. Also, the people who run a HA CA have to have wonderfully paranoid criteria for authen- 18.5 PEM CERTIFICATE HIERARCHY 477 ticating you before issuing you a certificate. Furthermore, a HA CA will refuse to grant a certificate to any organization that doesn't have the same strict rules about how its CA is managed and how its CA grants certificates. • Discretionary Assurance (DA) A DA CA is intended to be well managed at the top level, but it doesn’t impose any rules on the organizations to which it grants CA certificates (other than that they actually “own” the name listed in their certificate). The managers of a DA CA will make sure that when an organization, say the Chaos Computer Club, asks for a certificate for their CA, that they really are the Chaos Computer Club. However, the DA CA managers will not make any constraints on how the Chaos Computer Club manages its CA. Note that technically the IPRA is of the DA type. It can't be of the HA type, because it will issue certificates to a DA CA and a NA CA (see below) • No Assurance (NA) A NA CA has no constraints except that it is not allowed to issue two certificates with the same name. It is expected that users certified by a NA CA will not in general use their real names and may in fact operate under personas (pseudonyms) like Rabid Dog. Even the issuing NA CA might not know the true identity of the users it certifies. The rule in RFC 1422 is that there can only be a single path through the CA hierarchy to any individual. What that means is that if an organization, say MIT, decides it wants to be certified with a DA CA, then it cannot also be certified with a HA CA. It also means that cross certificates (see §12.9 Hierarchy of Realms) are not allowed. The CA hierarchy is a tree. This rule makes it easy to know the proper chain of certificates to give to someone. The maximal set of certificates they can need is the chain of certificates beginning with the IPRA. There are disadvantages to the restrictions, however. Cross certificates are often useful, so that the chain of certificates can be shorter between two organizations with a lot of mail traffic. And sometimes the two organizations would trust each other’s CAs more than they’d trust the entire chain of CAs to and from the IPRA. Also the rule prevents an organization that would like to get certificates from a HA CA (or made that decision at some point in the past) from granting a certificate to some suborganization's CA that will be less strict than the HA rules demand. An organization can accomplish this by operating two CAs, one which will follow the HA rules, and another which is not constrained by those rules. 478 PEM & S/MIME 18.6 18.6 CERTIFICATE REVOCATION LISTS (CRLS) It is possible for PEM users to obtain both certificates and CRLs without the aid of PEM, assuming some sufficiently sophisticated (and as yet undeployed) directory service. PEM wants to be usable even if such a directory service is for some reason unavailable, so it allows Bob to send Alice the relevant certificates for him by including them in the header of a PEM message he sends to her. PEM doesn’t include a field in the header for CRLs, though. Instead PEM defines a CRL service that sends the latest CRLs as a mail message in response to receipt of a request in a mail message. PEM defines two message types for the CRL service—CRL-RETRIEVAL-REQUEST and CRL. You send a CRL-RETRIEVAL-REQUEST message to the CRL service. The request is unencrypted and unsigned, and consists of a sequence of encoded X.500 names of CAs whose CRLs you want. The service responds with a CRL message containing (as text) the requested CRLs encoded, unencrypted, and unsigned, though each included CRL is signed by the CA that issued it. For each CA, the CRL service sends only the CA’s most recently issued CRL that it knows about. Note that you don’t always want the latest CRLs. You really want the ones that were valid at the time a message was received. You might be checking the signatures (on every relevant certificate) for a message that had been forwarded to you months after it was originally received. PEM doesn’t provide such a service. The original recipient would have had to obtain the relevant CRLs immediately upon receiving the message, and then kept them with the message. Alternatively, some sort of library service could be set up to archive all CRLs. Figure 18-1. RFC 1422 Certification Authority Hierarchy IPRA HACA DACA NACA HACA HACA individual HACA individual CA CA CA CA various organizations’ CAs PCAs individuals or CAs CA individual individual CA etc. 18.7 REFORMATTING DATA TO GET THROUGH MAILERS 479 18.7 REFORMATTING DATA TO GET THROUGH MAILERS As discussed in §17.14 Annoying Text Format Issues, the mail infrastructure on which PEM depends for delivering messages might modify messages. PEM needs a reversible encoding of a message that will transit through all known kinds of mail gateways (and even those the PEM designers don’t know about, hopefully) without having the gateways feel compelled to mess with the data. The theory is, if no “funny” characters are used, lines are a reasonable length, you don't care about the value of the high-order bit (the parity bit), and all lines are terminated with , your data will be safe from helpful mail forwarders. PEM packs 6 bits of information into each 8-bit character. It takes 24 bits of input and converts that into 32 bits (4 characters, each of which will be transmitted as 8 bits). If the data does not consist of an integral number of 24-bit chunks, it is padded with 0 bits to the next multiple of 24 bits, and each octet of pad bits is encoded by =. Actually, PEM packs 65 values into the 8 bits, since the padding gets converted to the = character, which is not one of the 64 characters used for encoding information. The encoding characters consist of the 26 upper-case (A–Z) and 26 lower-case (a– z) letters, the ten digits (0–9), +, and /. PEM uses the character = for the one or two possible padding characters necessary at the end of the data. For those who care, the table of encodings is as follows: value character ASCII representation 0 A 101 octal … … … 25 Z 132 octal 26 a 141 octal … … … 51 z 172 octal 52 0 60 octal … … … 61 9 71 octal 62 + 53 octal 63 / 57 octal padding = 75 octal Figure 18-2. PEM 6-Bit Encoding 480 PEM & S/MIME 18.8 18.8 GENERAL STRUCTURE OF A PEM MESSAGE A PEM message is sent like an ordinary mail message. As a matter of fact, the PEM portion might only be a small piece inside an ordinary mail message. If public key interchange keys are being used and Alice is sending a message to Bob, the PEM portion has the following structure: It is optional how many certificates in the chain to include. If Alice is pretty sure Bob has the remaining certificates, she can stop at some point before the IPRA. It is possible that no certificates appear explicitly, in which case what appears in their place is the serial number and issuer name of Alice’s certificate. marker indicating PEM-processed portion of the message follows: -----BEGIN PRIVACY-ENHANCED MESSAGE----- header, including information such as which of the three modes is being used (MIC-CLEAR, MIC-ONLY, ENCRYPTED) Initialization Vector (IV) for DES-CBC [field present only for ENCRYPTED message] certificate for Alice, signed by Alice's CA optional (see below) certificate for Alice's CA, signed by CA2 ... certificate signed by IPRA Message Integrity Code (MIC), which is encrypted if the message is encrypted secret key used to encrypt the message, encrypted with Bob's public key [field present only for ENCRYPTED message] blank line message, either encrypted [if ENCRYPTED], merely mangled to fit into 6-bit canonical get-through-any-mailer format [if MIC-ONLY], or actually human-intelligible [if MIC-CLEAR] marker indicating PEM-processed portion of the message is finished: ------END PRIVACY-ENHANCED MESSAGE------ 18.9 ENCRYPTION 481 18.9 ENCRYPTION As discussed in §17.14 Annoying Text Format Issues, the mail infrastructure on which PEM depends for delivering messages might modify messages. PEM needs a reversible encoding of a message that will transit through all known kinds of mail gateways (and even those the PEM designers don't know about, hopefully) without having the gateways feel compelled to mess with the data. The theory is, if no "funny" characters are used, lines are a reasonable length, you don't care about the value of the high-order bit (the parity bit), and all lines are terminated with , your data will be safe from helpful mail forwarders. PEM packs 6 bits of information into each 8-bit character. It takes 24 bits of input and converts that into 32 bits (4 characters, each of which will be transmitted as 8 bits). If the data does not consist of an integral number of 24-bit chunks, it is padded with 0 bits to the next multiple of 24 bits, and each octet of pad bits is encoded by =. Actually, PEM packs 65 values into the 8 bits, since the padding gets converted to the = character, which is not one of the 64 characters used for encoding information. The encoding characters consist of the 26 upper-case (A–Z) and 26 lower-case (a– z) letters, the ten digits (0–9), +, and /. PEM uses the character = for the one or two possible padding characters necessary at the end of the data. For those who care, the table of encodings is as follows: value character ASCII representation 0 A 101 octal ... ... 25 Z 132 octal 26 a 141 octal ... ... 51 z 172 octal 52 0 60 octal ... ... 61 9 71 octal 62 + 53 octal 63 / 57 octal padding = 75 octal Figure 18-2. PEM 6-Bit Encoding 482 EM & S/MIME 18.10 case where several messages are sent and the attacker happens to know they all begin with the same text and knows that text (e.g., "Dear sir:"). Assume 56-bit keys. In that case, an attacker with a fast crypto engine could start encrypting that text with all 256 keys and check whether the resulting block matched the first block of any of the captured messages. In 256 encryptions, the attacker can crack all of the messages. With PEM's IV, however, the attacker has to try 256 keys for each message, since the first blocks of text will be different after the IVs are ⊕ed. Each message to be CBC-encrypted is padded with 1 to 8 bytes to make it a multiple of 8 bytes (even if it started out as a multiple of 8 bytes). The content of each of the pad bytes is the number of pad bytes. So a message that was already a multiple of 8 bytes would be padded with eight 8s. A message that was 2 more than a multiple of 8 would have 6 bytes of 6s. 18.10 SOURCE AUTHENTICATION AND INTEGRITY PROTECTION Integrity protection is provided by calculating a MIC (Message Integrity Code) for the message. The defined types of MIC are MD2 and MD5, though originally the DES CBC residue was also defined. That mode of MIC was removed after we1,2 showed that it was completely insecure (see §18.16 DES-CBC as MIC Doesn't Work). But a message digest by itself is not a MIC since anyone can modify the message and compute the message digest of the modified message. So the message digest has to be cryptographically protected. In the public key case, this is done by signing the message digest; in the secret key case, it is done by encrypting the message digest with the interchange key. In the public key case, why must the MIC be encrypted when the message is encrypted? The reason is that the message digest can be computed from the MIC by using

Alice's public key. Someone can guess the contents of the message, and then be able to verify it by checking the message digest of the guess against that of the message. For instance, if they guess the message will say either attack or retreat, they can compute the message digest on both guessed messages, and see whether either guess is correct. It's nice that the MIC doesn't need to be encrypted when the message is not encrypted. In that case a per-message key can be dispensed with entirely and the sender need not learn the public key of the recipient in order to send a signed message. 18.11 MULTIPLE RECIPIENTS 483 18.11 MULTIPLE RECIPIENTS A signed but unencrypted message can be sent to any number of recipients without modification. PEM merely signs the message with the sender's private key: To transmit an encrypted message to multiple recipients, PEM encrypts the message once (with, as usual, a randomly chosen key), and then separately encrypts the per-message key for each recipient: Note that the message is encrypted only once, but the per-message key must be encrypted once for each recipient. It is possible to mix usage of public and secret keys. For instance, suppose Alice is sending a message to Bob and Carol, where Bob has a public key, but Carol has a secret key shared with Alice, and Carol's node cannot execute public key cryptographic algorithms (perhaps because Carol hasn't made the proper licensing arrangement). Alice picks a per-message secret key, encrypts the message with that secret key, and then encrypts the per-message key with Bob's public key (for Bob), and encrypts it with the key she shares with Carol (for Carol). The integrity check need only appear once to take care of all recipients capable of dealing with public keys. Alice merely signs the message digest with her private key. However, just because Alice has a public key does not mean that all the recipients can make use of it. For instance, those U.S. recipients who -----BEGIN PRIVACY-ENHANCED MESSAGE----- header, saying MIC-CLEAR or MIC-ONLY MIC (message digest signed with Alice's private key) unencrypted message -----END PRIVACY-ENHANCED MESSAGE----- -----BEGIN PRIVACY-ENHANCED MESSAGE----- header, saying ENCRYPTED MIC (message digest signed with Alice's private key and encrypted with message key) Message key encrypted with Bob's public key Message key encrypted with Ted's public key encrypted message -----END PRIVACY-ENHANCED MESSAGE----- 484 PEM & S/MIME 18.12 haven't licensed RSA cannot legally verify the signature. Alice needs to include a separate MIC for each recipient who cannot deal with her public key. For instance, for Carol, Alice will include the message digest encrypted with the secret key that Alice and Carol share. 18.12 BRACKETING PEM MESSAGES Bob has just received a message. How does his PEM processor know whether the message is a PEM message, or just an ordinary message? Even if PEM knows (somehow) that the message is a PEM message, it's not always obvious where PEM should start processing, and where to end processing. For instance, Alice might have sent a signed message to Bob, so that the entire message as transmitted by Alice should be processed by PEM, but mail gateways en route might add cybercrud to the beginning and/or end of the message. (Luckily most mail gateways are sufficiently well-behaved so that they don't add things to the middle of a message—if they did, PEM would not be able to cope.) By the time Bob receives the message his PEM processor might need to skip the first few lines, start processing at some point, and end at some other point in the message. As a mat- -----BEGIN PRIVACY-ENHANCED MESSAGE----- header, saying ENCRYPTED MIC (message digest signed with Alice's private key and encrypted with message key) Message key encrypted with Bob's public key MIC (message digest encrypted with the key Alice and Carol share) Message key encrypted with the key Alice and Carol share encrypted message -----END PRIVACY-ENHANCED MESSAGE----- 18.12 BRACKETING PEM MESSAGES 485 ter of fact, it isn't strictly necessary for an entire message from Alice to be PEM-processed. She might just want to encrypt a small portion of the message. For example: That is why PEM marks the beginning of the PEM-processed portion of the message with the string -----BEGIN PRIVACY-ENHANCED MESSAGE----- and the end of the message with the string -----END PRIVACY-ENHANCED MESSAGE----- This might sound straightforward, but there are problems. A human being is perfectly capable of typing the above strings (we did, for instance). Suppose -----END PRIVACY-ENHANCED MESSAGE----- appears in the middle of a message. For instance, Alice might be transmitting a portion of this book (after getting written permission from the publisher, of course!). Note that if the message is encoded (as a MIC-ONLY or ENCRYPTED message will be), then the string -----END PRIVACY-ENHANCED MESSAGE----- can't possibly appear inside the (encoded) body of the message (even if it was in the original message) because "-" is not one of the encode characters. But PEM does have to worry about disguising the PEM end marker in MIC-CLEAR messages. Although the end marker is the only string that could confuse PEM, instead of only searching for -----END PRIVACY-ENHANCED MESSAGE----- PEM always modifies every line of the text of a MIC-CLEAR message that begins with a dash, just because RFC 934 did it that way for forwarded mail markers. When PEM sees a line that begins with a dash, it adds the two characters "- " (dash space) to the beginning of the line. So the string -----END PRIVACY-ENHANCED MESSAGE----- would appear as - -----END PRIVACY-ENHANCED MESSAGE----- if it appeared in the middle of a MIC-CLEAR message. Any other line that started with a dash Hello Bob This is to inform you that your raise has gone through and your new salary is -----BEGIN PRIVACY-ENHANCED MESSAGE----- header saying encrypted afdsjklasdfjklas;f -----END PRIVACY-ENHANCED MESSAGE----- Once again congratulations on the good work. Sincerely, Alice 486 PEM & S/MIME 18.12 would also appear that way. For example, --Sincerely, Alice-- would appear as: - --Sincerely, Alice-- Bob's PEM processor will most likely remove the extra characters before displaying the text of the message to Bob. However, if Bob does not have PEM, Bob will see the PEM cybercrud, and see the mysterious "- " added to the beginning of some of the lines of the text of the message. PEM really can't defend itself from being confused in all cases. For instance, Alice might have typed the following message and sent it with a non-PEM mailer. If Bob's machine has PEM, it will see those markers and attempt to parse the message as a PEM message. It will probably notice that the message is ill-formed, though Alice could have typed something that happened to have all the text looking like correct fields. At that point it would probably notice that signatures would not verify or that it was lacking the proper certificates. If Alice did indeed manage to type in a completely correct PEM message, then PEM at Bob's node will not notice (Alice will have passed a warped form of the Turing test). Another issue is nested messages. It is permissible to have, for instance, an encrypted message inside a signed message, or Alice might be forwarding a signed message from Fred as an enclosure in a signed message to Bob (see §18.13.1 Forwarding a Message). The PEM processor at Bob's node presumably has to process the outer message, which might involve decrypting and/or decoding. Once the outer message is processed, the string -----BEGIN PRIVACY-ENHANCED MESSAGE----- might be uncovered, and PEM has the same problem as we mentioned before, which is to somehow determine whether that is really the beginning of a PEM message, or whether Alice just happened to have included that particular string in her message. The PEM protocol was designed so that it is conceptually possible to nest PEM processing, but it would require PEM processors far more sophisticated than anyone is likely to ever implement to deal with all possible cases, and furthermore the user interface would be horrible to contemplate. (See Homework Problem 8.) To: Bob Have you heard about this new PEM standard? I don't completely understand it, but it seems to imply that I can make my message to you secret by adding the string -----BEGIN PRIVACY-ENHANCED MESSAGE----- and some other stuff like MIC-INFO and RFC-1822 at the beginning of the message and putting the string -----END PRIVACY-ENHANCED MESSAGE----- at the end. I don't understand how this could prevent anyone from reading the message, although maybe enough cybercrud on the beginning of the message would convince them the message must be