hopelessly garbled and they might not even attempt to read it. --Alice 18.13 FORWARDING AND ENCLOSURES 487 18.13 FORWARDING AND ENCLOSURES Alice might want to forward to Bob a message she received from someone else, inside a message of her own: 18.13.1 Forwarding a Message What should you do to a message from Fred before forwarding it to Bob so that Bob can verify Fred's signature? With non-PEM mailers, one often extracts text from one message and includes it in another. PEM certainly doesn't stop you from doing that, but if you don't send a message with the PEM information intact, there's no way for Bob to cryptographically verify that the original text was really from Fred. With public keys, it is possible to usefully forward PEM signatures. But secret key based source authentication information cannot be usefully forwarded because such information cannot be verified by a third party. Therefore, in this section, we'll assume user keys are based on public key technology. Suppose Alice is just forwarding a message from Fred to Bob without adding any annotations of her own, and in the same format as she received it (ENCRYPTED, MIC-CLEAR, or MIC-ONLY). In that case she processes the message like a distribution list exploder: -----BEGIN PRIVACY-ENHANCED MESSAGE----- header, saying MIC-CLEAR MIC (message digest signed with Bob's private key) Bob— Get a load of the outrageous stuff Fred is sending me! - --Alice - ----BEGIN PRIVACY-ENHANCED MESSAGE----- header, saying MIC-CLEAR MIC (message digest signed with Fred's private key) Alice— The rhythm of your typing at the terminal is driving me insane. Either type completely rhythmically or completely unrhythmically. I will complain to upper level management if you don't adjust your typing. - - --Fred - ----END PRIVACY-ENHANCED MESSAGE----- -----END PRIVACY-ENHANCED MESSAGE----- 488 PEM & S/MIME 18.13.1 • In the case of a MIC-CLEAR or MIC-ONLY message, she just forwards it, without having to do any processing on it. • In the case of an ENCRYPTED message, she decrypts the per-message key, reencrypts it with Bob's key, and adds the encrypted key and Bob's name to the header. It could be that Alice wants to change the format of the message. The term upgrading means that she turns an unencrypted message into an encrypted message. The term downgrading means she turns an encrypted message into an unencrypted message. The PEM designers were careful to have the signature on a message apply to the unencrypted message, so that a message can be downgraded or upgraded without changing the signature. For example, let's assume Alice has a MIC-ONLY message from Fred. It contains • header, saying MIC-ONLY • MIC (message digest signed with Fred's private key) • message (encoded but unencrypted) In order to encrypt it to send it to Bob, Alice picks a per-message key K in order to encrypt the message. Now, she can't simply encrypt the encoded message, since PEM mandates that encryption is based on an unencoded message. So she has to decode the message, encrypt the result, and encode the result of the encryption. So what she sends is • header, saying ENCRYPTED • K encrypted with Bob's public key • encrypted MIC, consisting of the message digest signed with Fred's private key and encrypted with K. Note that Alice can't sign the MIC with Fred's private key, so she has to use the MIC she received. Since the MIC she received from Fred is encoded (to pass through mailers), she has to decode the MIC to obtain the actual message digest signed with Fred's public key. That is the quantity she encrypts and then encodes. • message encrypted with K Downgrading a message is just reversing the process. If Alice wants to annotate Fred's message before forwarding it, then she can enclose the complete message inside the text of her message, and then sign or encrypt her own message as she desires. 18.14 UNPROTECTED INFORMATION 489 18.14 UNPROTECTED INFORMATION PEM only provides integrity protection or privacy on the contents of a message. There is other information in a message, however, and if people use PEM carelessly there are interesting security flaws that can be exploited. PEM does not protect the SUBJECT, TO, FROM, or TIMESTAMP field. How can a bad guy exploit the fact that PEM does not protect these fields? Why would it be nice to cryptographically protect

the subject line? A naive user might put private information into the subject line, and expose the information to an eavesdropper. Or suppose Alice sees a message from Fred, in which Fred suggests some outrageous idea. Alice forwards Fred's message, integrity-protected, with the subject line Fire this Bozo immediately, but someone modifies the unprotected subject line to Great idea! Implement this suggestion immediately. Since the subject line is not protected, the message will arrive signed by Alice, but having the subject line modified completely changes what she intended to say. The TO and FROM fields cannot be encrypted because the mail infrastructure needs to see them. Theoretically they could be included in the integrity protection even though they couldn't be privacy-protected. PEM didn't do this because mailers tend to mangle these fields. PEM could have gotten around this by making a copy of that information and moving it inside the message. It's unfortunate that PEM doesn't integrity-protect the TO field. For instance, Alice might send the signed message I agree to donate $1000 to your organization to organization Z, and the message might be copied and used by organization Y to fool Alice's underlings into sending them money also. An example of where it is unfortunate that PEM didn't do something with the FROM field is where secret key based interchange keys are used and Alice sends a message to a distribution list located on a remote exploder. The remote exploder has to check Alice's MIC and then add its own per-recipient MIC. PEM does not provide any cryptographically protected method for the exploder to assert that it did indeed check and the message had come from Alice. The other piece of header information that is not cryptographically protected is the timestamp. This doesn't seem too important. Even naive users would realize that if the time of sending a message were important, they should date the message in the text. Naive users would probably not even know that the mail infrastructure adds a time of posting. Note that even if PEM protected all the header fields, it would be possible to misuse electronic mail. For instance, if Alice sends Bob the signed message I approve, Bob can't use the message later to prove that Alice okayed his action, since the message isn't explicit about what Alice was approving. If Alice is careful, most of these PEM problems can be worked around. To protect the header information, Alice should include the header information in the text. The only problem we've described that has no PEM workaround is that a remote mail exploder cannot cryptographically assert that a message using secret interchange keys did originate with Alice. 490 PEM & S/MIME 18.15 18.15 MESSAGE FORMATS The PEM message formats have fields that consist of text. Sometimes the fields are encrypted, and sometimes they are encoded (which looks encrypted to humans but isn't). The types of messages are MIC-CLEAR, MIC-ONLY, and ENCRYPTED, and the format for each of these varies slightly depending on whether public or secret key cryptography is being used. We'll describe the public key and secret key variants separately. If some recipients use public key and some use secret key, then fields from both variants must be present. Each field in the message has a label indicating what it contains. There are a few reasons for this label: • To provide a visual clue to a human reader wondering what all the cybercrud is (but a civilized PEM mailer ought not show the cybercrud to the human) • To identify the field—the position of the field in the message does not uniquely identify it because some fields are optional and/or repeated (e.g., CA CERTIFICATE) 18.15.1 MESSAGE FORMATS 491 18.15.1 ENCRYPTED, Public Key Variant -----BEGIN PRIVACY-ENHANCED MESSAGE----- PRE-ENCAPSULATION BOUNDARY. This is just to show that a PEM-processed portion of the mail message is about to begin. Proc-Type: 4,ENCRYPTED MESSAGE TYPE. The first subfield (4) specifies that this is version four of PEM. There are currently no other legal values for this field. The second subfield (ENCRYPTED) specifies that this is an encrypted message. The other legal values for this field are MIC-ONLY and MIC-CLEAR. ----BEGIN PRIVACY-ENHANCED MESSAGE----- pre-encapsulation boundary Proc-Type: 4,ENCRYPTED type of PEM message (version,type) Content-Domain: RFC822

message form DEK-Info: DES-CBC,16 hex digits message encryption algorithm, IV Originator-Certificate: cybercrud sender's encoded certificate (optional) Originator-ID-Asymmetric: cybercrud,number sender ID (present only if sender's certificate not present) Key-Info: RSA,cybercrud key-info for CC'd sender (if needed) Issuer-Certificate: cybercrud sequence of zero or more CA certificates (possibly whole chain from the sender's certificate to the IPRA's) ... MIC-Info: RSA-MDx,RSA,cybercrud message digest algorithm, message digest encryption algorithm, encoded encrypted MIC Recipient-ID-Asymmetric: cybercrud,number Key-Info: RSA,cybercrud for each recipient: recipient ID (encoded X.500 name of CA that signed certificate, certificate serial number); key-info for recipient ... blank line cybercrud encoded encrypted message ----END PRIVACY-ENHANCED MESSAGE----- post-encapsulation boundary 492 PEM & S/MIME 18.15.1 Content-Domain: RFC822 MESSAGE FORM. The intention is that there might eventually be lots of types of mail messages, such as PostScript, spreadsheet, bitmap, executable image for an XYZ processor, and so on. However, the only defined type is RFC822, which is an Internet standard for ordinary text messages. DEK-Info: DES-CBC,0123456789ABCDEF MESSAGE ENCRYPTION ALGORITHM, IV. The first subfield identifies the cryptographic algorithm used to encrypt the message. Currently the only "registered" algorithm is DES-CBC. The second subfield specifies the IV as 16 hex digits. So in the example above, the IV is 0123456789ABCDEF16. Originator-Certificate: MIIBfzCCASkCAQIwDQYJKoZIhvcNAQECBQAwQzELM
AkGA1UEBhMCVVMxJjAkBgNVBAoTHURpZ2l0YWwgRXF1aXBtZW50IENvcnBvcmF0a
W9uMQwwCgYDVQQLEwNMS0cwHhcNOTIwOTA4MjAxODMzWhcNOTQwOTA4MjAxODMzW
jBXMQswCQYDVQQGEwJVUzEmMCQGA1UEChMdRGlnaXRhbCBFcXVpcG1lbnQgQ29yc
G9yYXRpb24xDDAKBgNVBAsTA0xLRzESMBAGA1UEAxMJSm9obiBMaW5uMFcwCgYEV
QgBAQICAgADSQAwRgJBAMEom520pxxpN7Y+0e8nWsI2yVK4YPu30+meJEwH9w0Xu
O3hmHTHowzrNwgdgFeic4SHlkFOwf1K7RrOgnTurQ0CAQMwDQYJKoZIhvcNAQECB
QADQQBq/+L5hKVZvN/jtgsjeUE5eQBrMpSf3ND4kqyP65xj29OhFk5Mbh4DpfJ+7
wOwJATwUjOERhzwVJm5WmI0oRs4 SENDER'S CERTIFICATE. BIG pile of cybercrud as specified in X.509. Since a certificate is ASN.1-coded information (as opposed to nice printable text strings), the certificate is encoded as per §18.7 Reformatting Data to Get Through Mailers. Certificates are described in Chapter 15 PKI (Public Key Infrastructure). Originator-ID-Asymmetric: MEMxCzAJBgNVBAYTAlVTMSYwJAYDVQQKEx1EaWd
pdGFsIEVxdWlwbWVudCBDb3Jwb3JhdGlvbjEMMAoGA1UECxMDTEtH,216 SENDER ID. This field can replace the SENDER'S CERTIFICATE field to save space. It consists of the X.500 name of the CA that issued the sender her certificate, coded in ASN.1 and encoded 6 bits per character, followed by a comma, followed by the serial number of the sender's certificate in decimal using decimal digits. In the example above, the serial number is 216. Key-Info: RSA,Pv3W7Ds86/fQBnvB5DsvUXgpK7+6h5aSVcNeYf9uWtly9m2VHzC
v2t7A6qgbXtIcf4kaMj1FL2yl9/N9mWpm4w== KEY-INFO. This is strictly necessary only if the sender CC's herself on the message. It is also useful when the mail system returns the message to the sender due to some error condition. The purpose of this field is to enable the sender to decrypt the message. The first subfield specifies the cryptographic algorithm used for encrypting the per-message key. The only defined public key algorithm is RSA. The second subfield contains the per-message key encrypted with the sender's public key and then encoded. 18.15.1 MESSAGE FORMATS 493 Issuer-Certificate: MIIBXzCCAQkCAQowDQYJKoZIhvcNAQECBQAwNzELMAkGA
1UEBhMCVVMxKDAmBgNVBAoTH1RydXN0ZWQgSW5mb3JtYXRpb24gU3lzdGVtcyBQQ
0EwHhcNOTIwOTA4MTk0NTQzWhcNOTQwOTA4MTk0NTQzWjBDMQswCQYDVQQGEwJVU
zEmMCQGA1UEChMdRGlnaXRhbCBFcXVpcG1lbnQgQ29yYXRpb24xDDAKBgNVB

AsTA0xLRzBXMAoGBFUIAQECAgIAA0kAMEYCQQDkwhgRGX6ScwOHTVk48NK07tiih t9bBExsbxphdp7brUrypA4kZDYqZNVv9Ee5kHv5qn428Pc3lbGH+zlS7bq/AgEDM A0GCSqGSIb3DQEBAgUAA0EARfI+2huFXK9jsROeK+CaohcB23cBFjzqhslhI50UU Y4rBx8QhcApY/g+mhXzBz1JPe/HANQMG567DHB3AJKaTQ== CA CERTIFICATE. The certificate of one of the CAs in the chain from the sender to the IPRA. The first one certifies the sender's CA, the next one certifies the sender's CA's parent, ..., the last one certifies the IPRA. However, it is permissible to include in the message only an initial subsequence of this certificate chain (even an empty subsequence!), as long as no certificates are skipped. MIC-Info: RSA-MD5,RSA,FUiVRM3x5Ku0aZveGIJ1hv/hi3Iowpm1iypd9VP7MGw PPQra+42TkbR/2jhnqXyVEXLaJ7BSyNhBh/9znIUj5uk0N7IXeBxX MESSAGE DIGEST ALGORITHM, MESSAGE DIGEST ENCRYPTION ALGORITHM, MIC. The choices for message digest algorithm are RSA-MD2 and RSA-MD5. RSA-MD2 is merely MD2. RSA-MD5 is MD5. In both cases, RSA is included in the name to acknowledge that MD2 and MD5 were donated for use in PEM by RSADSI. The choices for message digest encryption algorithm are RSA. (Luckily RSA is a good choice). The MIC (the message digest encrypted with the sender's private key, then encrypted with the per-message key) is an encoded 512-bit number. Recipient-ID-Asymmetric: MEMxCzAJBgNVBAYTAlVTMSYwJAYDVQQKEx1EaWdp dGFsIEVxdWlwbWVudCBDb3Jwb3JhdGlvbjEMMAoGA1UECxMDTEtH,729 RECIPIENT ID. This is an identifier for a recipient's certificate. It consists of the X.500 name of the CA that issued this recipient's certificate, coded in ASN.1 and encoded 6 bits per character, followed by a comma, followed by the serial number of the certificate in decimal using decimal digits. In the above example, the serial number is 729. Note that identifying the recipient in this arcane manner, rather than simply by name, allows a recipient with multiple public keys to know which key to use to decrypt his KEY-INFO field: Key-Info: RSA,dpUp7/QoY9YOZzZCVcIwxIDMN0WbGCFAGN3T+xlV/0pBu2n5+x8 PmBvMUN0NcBi5vtqBS4cfmgShiK0I4zu05Q== KEY-INFO. This field provides the recipient (identified by the immediately preceding RECIPIENT ID field) with the per-message key. The second subfield contains the per-message key encrypted with the recipient's public key and then encoded. The first subfield specifies the cryptographic algorithm used for encrypting the per-message key. The only defined public key algorithm is RSA. Note that even if there were more choices, there would be no reason to specify the algorithm since the algorithm is whatever algorithm is specified for the key in the 494 PEM & S/MIME 18.15.2 recipient's certificate. However, bit-bumming the encoding of mail messages did not seem to be a priority with the PEM designers. 21OHDuHTP5BABnlsqENz1WVerZxxWo2AsPHhm2SIz9qpLMvxT/x0+8UEf8dDsTDr wbQo9/x+ ENCRYPTED MESSAGE. The message DES-CBC encrypted with the per-message key and then encoded. -----END PRIVACY-ENHANCED MESSAGE----- POST-ENCAPSULATION BOUNDARY. This is just to show that a PEM-processed portion of the mail message has ended. 18.15.2 ENCRYPTED, Secret Key Variant -----BEGIN PRIVACY-ENHANCED MESSAGE----- PRE-ENCAPSULATION BOUNDARY. This is just to show that a PEM-processed portion of the mail message is about to begin. Proc-Type: 4,ENCRYPTED MESSAGE TYPE. The first subfield (4) specifies that this is version four of PEM. There are currently no other legal values for this field. The second subfield (ENCRYPTED) specifies that ----BEGIN PRIVACY-ENHANCED MESSAGE---- - pre-encapsulation boundary Proc-Type: 4,ENCRYPTED type of PEM message (version,type) Content-Domain: RFC822 message form DEK-Info: DES-CBC,16 hex digits message encryption algorithm, IV Originator-ID-Symmetric: entity identifier,issuing authority,version/expiration sender ID Recipient-ID-Symmetric: entity identifier,issuing authority,version/expiration Key-Info: DES-ECB,RSA-MDx,16 hex digits,32 hex digits for each recipient: ... recipient ID; key-info

for recipient blank line cybercrud encoded encrypted message ----END PRIVACY-ENHANCED MESSAGE----- post-encapsulation boundary 18.15.2 MESSAGE FORMATS 495 this is an encrypted message. The other legal values for this field are MIC-ONLY and MIC-CLEAR. Content-Domain: RFC822 MESSAGE FORM. The intention is that there might eventually be lots of types of mail messages, such as PostScript, spreadsheet, bitmap, executable image for an XYZ processor, and so on. However, the only defined type is RFC822, which is an Internet standard for ordinary text messages. DEK-Info: DES-CBC,0123456789ABCDEF MESSAGE ENCRYPTION ALGORITHM, IV. The first subfield identifies the cryptographic algorithm used to encrypt the message. Currently the only "registered" algorithm is DES-CBC. The second subfield specifies the IV as 16 hex digits. So in the example above, the IV is 0123456789ABCDEF16. Originator-ID-Symmetric: Alice@Wonderland.edu,, SENDER ID. This field uniquely identifies the sender to the recipients. There are three subfields. The first subfield identifies the sender. The other two subfields are useless, but are there to make the sender ID have the same format as the recipient IDs. In practice, the two useless subfields are omitted (except for the delimiting commas). There is no standard for the format of any of the subfields, but RFC 1421 strongly recommends using an internet mail address for the first subfield. Recipient-ID-Symmetric: WhiteRabbit@Wonderland.edu, MadHatter@Wonderland.edu,1 RECIPIENT ID. This field uniquely identifies the interchange key to be used by the recipient. There are three subfields. The first subfield identifies the recipient (so that the recipient knows this field and the immediately following KEY-INFO field are for him). The second subfield identifies the issuer of the shared key (in the event that multiple issuing authorities have assigned shared keys between the sender and recipient). The third subfield uniquely identifies the key (in the event that multiple keys have been issued between the sender and recipient by the same issuing authority). There is no standard for the format of any of the subfields, but RFC 1421 strongly recommends using an internet mail address for the first subfield. Key-Info: DES-ECB,RSA-MD5,0123456789ABCDEF, FEDCBA98765432100123456789ABCDEF KEY-INFO. This field provides the recipient (identified by the immediately preceding RECIPIENT ID field) with the per-message key and the MIC. The first subfield specifies the cryptographic algorithm used to encrypt the per-message key. The choices are DES-ECB (see §4.2.1 Electronic Code Book (ECB)) and DES-EDE (see §4.4 Multiple Encryption DES). DES-EDE is used when the shared interchange key is double length. The second subfield 496 PEM & S/MIME 18.15.3 specifies the algorithm used to compute the message digest. The choices are RSA-MD2 (for MD2) and RSA-MD5 (for MD5). The third subfield is the per-message key encrypted with the interchange key, expressed as a 16-digit hexadecimal number. The fourth subfield is the MIC, which is the message digest encrypted with the per-message key in ECB mode, expressed as a 32-digit hexadecimal number.
21OHDuHTP5BABnlsqENz1WVerZxxWo2AsPHhm2SIz9qpLMvxT/x0+8UEf8dDsTDr wbQo9/x+ ENCRYPTED MESSAGE. The message DES-CBC encrypted with the per-message key and then encoded. -----END PRIVACY-ENHANCED MESSAGE----- POST-ENCAPSULATION BOUNDARY. This is just to show that a PEM-processed portion of the mail message has ended. 18.15.3 MIC-ONLY or MIC-CLEAR, Public Key Variant The only differences between an ENCRYPTED public key PEM message and a MIC-ONLY or MIC-CLEAR public key PEM message are the following: • The TYPE OF MESSAGE field specifies MIC-ONLY or MIC-CLEAR. ----BEGIN PRIVACY-ENHANCED MESSAGE----- pre-encapsulation boundary Proc-Type: 4,MIC-ONLY or MIC-CLEAR type of PEM message (version,type) Content-Domain: RFC822 message form Originator-Certificate: cybercrud sender's encoded certificate (optional) Originator-ID-Asymmetric: cybercrud,number sender ID (present only if sender's certificate not present) Issuer-Certificate: cybercrud sequence of zero or more CA certificates (possibly whole chain from sender to the

root) ... MIC-Info: RSA-MDx,RSA,cybercrud message digest algorithm, message digest encryption algorithm, encoded MIC blank line message message (encoded if MIC-ONLY) ----END PRIVACY-ENHANCED MESSAGE----- post-encapsulation boundary 18.15.4 MESSAGE FORMATS 497 • The MIC is not encrypted, though it is encoded 6 bits per character. • The message is not encrypted; for MIC-ONLY it is encoded 6 bits per character, while for MIC-CLEAR it is included unmodified (except for the insertion of a "- " before each beginning-of-line "-"). • The MESSAGE ENCRYPTION ALGORITHM, IV field is omitted. • All the fields giving per-recipient information are omitted. • The KEY-INFO field for giving information about the sender's key is omitted. 18.15.4 MIC-ONLY and MIC-CLEAR, Secret Key Variant The only differences between an ENCRYPTED secret key PEM message and a MIC-ONLY or MIC-CLEAR secret key PEM message are the following: • The TYPE OF MESSAGE field specifies MIC-ONLY or MIC-CLEAR. • The message is not encrypted; for MIC-ONLY it is encoded 6 bits per character, while with MIC-CLEAR it is included unmodified (except for the insertion of a "- " before each beginning-of-line "-"). • The MESSAGE ENCRYPTION ALGORITHM, IV field is omitted. ----BEGIN PRIVACY-ENHANCED MESSAGE----- pre-encapsulation boundary Proc-Type: 4,MIC-ONLY or MIC-CLEAR type of PEM message (version,type) Content-Domain: RFC822 message form Originator-ID-Symmetric: entity identifier,issuing authority,version/expiration sender ID Recipient-ID-Symmetric: entity identifier,issuing authority,version/expiration Key-Info: DES-ECB,RSA-MDx,16 hex digits,32 hex digits for each recipient: recipient ID; key-info for recipient ... blank line message message (encoded if MIC-ONLY) ----END PRIVACY-ENHANCED MESSAGE----- post-encapsulation boundary 498 PEM & S/MIME 18.15.5 18.15.5 CRL-RETRIEVAL-REQUEST 18.15.6 CRL 18.16 DES-CBC AS MIC DOESN'T WORK Originally, PEM had an additional method of obtaining a MIC on a message, which was the DES-CBC residue. The PEM documentation referred to this form of integrity check as DES-MAC. Ironically, they included this because the MD functions were new and therefore cryptographically suspect, whereas DES CBC residue was used in banking and therefore known to be secure. And it was in the banking context. But not for email. In banking it is used as an integrity check, and has the property that if someone doesn't know the secret key that Alice and Bob share, they cannot modify or forge a message between Alice and Bob. It is instructive to see how something as seemingly straightforward as an integrity check, especially one known to be secure in a highly sensitive application like banking, can have problems. The DES-MAC version of integrity checking has several interesting security problems: • If Alice uses secret key interchange keys and ever sends a message to multiple recipients, say Bob and Ted, then Bob can thereafter send DES-MAC integrity check messages ----BEGIN PRIVACY-ENHANCED MESSAGE----- pre-encapsulation boundary Proc-Type: 4,CRL-RETRIEVAL-REQUEST type of PEM message (version,type) Issuer: cybercrud for each CRL requested: the encoded X.500 name of the issuing CA ... ----END PRIVACY-ENHANCED MESSAGE----- post-encapsulation boundary ----BEGIN PRIVACY-ENHANCED MESSAGE----- pre-encapsulation boundary Proc-Type: 4,CRL type of PEM message (version,type) CRL: cybercrud Originator-Certificate: cybercrud for each CRL retrieved: encoded X.509 format CRL; encoded X.509 certificate of the CA that issued the CRL ... ----END PRIVACY-ENHANCED MESSAGE----- post-encapsulation boundary 18.16 DES-CBC AS MIC DOESN'T WORK 499 (MIC-CLEAR, MIC-ONLY, or ENCRYPTED) to Ted, claiming to be Alice (and Ted can likewise send such messages to Bob claiming to be Alice). This is true regardless of the choice of integrity check (DES-MAC, MD2, or MD5) Alice chose when she sent the multi-recipient message. • If Alice is using public key technology for interchange keys and uses DES-MAC as the integrity check when sending an ENCRYPTED message to Bob, Bob can thereafter send messages to anyone he wants, claiming to be Alice. • With public key interchange keys, if Alice sends a MIC-CLEAR or MIC-ONLY (as opposed to ENCRYPTED) message to Bob, anyone

who eavesdrops on the message can thereafter send messages to anyone she wants, claiming to be Alice. • Suppose Alice, knowing that DES-MAC is not a secure integrity check, never uses it herself. But Ted is happy to accept DES-MAC messages. If Alice sends a message using either MD2 or MD5, someone can use her signature on that message to create a forged message that uses DES-MAC and would appear to come from Alice; Ted would accept such a message as genuine. (Vulnerabilities depend on whether interchange keys are secret or public, and whether the message was encrypted or not. See Homework Problem 12). We'll have to describe how DES-MAC is specified in the earlier PEM specification. Suppose Alice is sending a message to Bob and Ted. The algorithm for using DES-MAC as the integrity check is as follows: 1. Choose a per-message key (this is the same key PEM would use if PEM was going to encrypt the message). 2. Modify the per-message key by ⊕ing it with F0F0F0F0F0F0F0F016 (to avoid the classic cryptographic flaw described in §4.3.1 Ensuring Privacy and Integrity Together). 3. Compute the 64-bit CBC residue with the modified per-message key. 4. Encrypt the residue. If secret key based interchange keys are used, encrypt the CBC residue with the interchange key associated with Ted, plus encrypt the CBC residue with the interchange key associated with Bob, and send both quantities along with the message. If public key based interchange keys are used, Alice signs the residue with her private RSA key. The problem is that it is possible, given a 64-bit quantity x and a DES key k, to generate a message m such that the CBC residue of m with key k is x. (We'll demonstrate that later in this section.) Let's look at several cases: • Alice sends a MIC-ONLY message to Bob, using public key interchange keys, using DES-MAC as the integrity check. Carol eavesdrops on the message. She can see that it came from Alice. She finds Alice's public key, extracts the signed DES-MAC from the message 500 PEM & S/MIME 18.16 header, and reverses the signature on the DES-MAC, thereby retrieving the DES-MAC of the message. Carol now knows the quantity x (the DES-MAC) and the quantity y, which is Alice's signature on x. As we said, she can invent any key k and construct a message m with DES-MAC x (using key k). She can then send m to anyone she wants, say Ted, and claim that the message was from Alice, because she'll include y as the signed DES-MAC for m. • Suppose instead that Alice sends an encrypted message to Bob. In this case only Bob knows the quantities x (the DES-MAC of the message) and y (Alice's signature on x) because PEM specifies that if the message is encrypted, the integrity check is encrypted with the key used to encrypt the per-message key. But that means that Bob can now send messages claiming to be Alice, just like any eavesdropper could have done in the previous example. • Alice sends a message with DES-MAC integrity check to both Bob and Ted, using secret key interchange keys. So she shares a key KAB with Bob, and shares a key KAT with Ted. The DES-MAC is included twice, once encrypted with KAB and once encrypted with KAT. Furthermore the per-message key k is sent twice, once encrypted with KAB and once encrypted with KAT. An eavesdropper cannot discover the DES-MAC of the message even if the message isn't encrypted—she can't compute it because she doesn't know k, and she can't decrypt the encrypted DES-MAC because she doesn't know either KAB or KAT. However, Bob does know the DES-MAC x, and can see KAT{x}. He also knows k and KAT{k}. So after receiving this message from Alice, Bob can construct a message m with CBC residue x, send it to Ted using per-message key k, claim that Alice is sending it, and send KAT{k} as the encrypted per-message key and KAT{x} as the encrypted DES-MAC. • Alice sends a message with MD5 integrity check to Bob and Ted, using secret key interchange keys. PEM specifies that each of the two 64-bit halves of the MD5 is encrypted with the interchange key of each recipient. Let's say that the halves are x1 and x2. So Alice will send KAT{x1} and KAB{x1} as well as KAT{x2} and KAB{x2}. Now even though Alice wasn't using DES-MAC, Bob knows a 64-bit quantity (say x2) and what that quantity looks like when encrypted with KAT. He also knows the per-message key k and KAT{k}. So Bob can construct a message m

with CBC-residue x2 and send it to Ted using per-message key k, claiming that it was from Alice and that DES-MAC was the integrity check used. Note that any of the quantities x1, x2, or k could have been chosen to be the per-message key and/or the DES-MAC of the forged message. Now we'll show how, given a 64-bit value r and a key k, it is possible for Carol to construct a message m with r as the CBC residue (using key k) (see Figure 18-3). Carol can actually construct any message she wants. The only constraint on the message is that somewhere inside the message there has to be a 64-bit block (8 characters) that must be filled with a value determined only by the rest of the message and r and k. 18.17 DIFFERENCES IN S/MIME 501 Let's assume that there's one 8-byte block of the message, say m4, that Carol can fill with "garbage" without suspicion. She constructs the message she wants in the remaining plaintext blocks. She will be sending a message to Ted, signed by Alice. Ted might think it strange to have 8 bytes of crud in the middle of the message, but maybe Carol can cleverly write the message so that somewhere inside the message she says something like, Hey, did I tell you about how I fixed my porch this weekend? It went well until I hit my thumb with the hammer. Then I said, "@f#!Ruoe"! Now, back to business ... Alternatively, mailers are so wonderfully user-friendly that cybercrud appears in them all the time. PEM headers certainly are ugly and would be ignored by a human. Even non-PEM messages often contain obscure-looking headers and postmarks. If Carol were to embed arbitrary cybercrud there it probably would not create suspicion. Anyway, Carol works backwards from r. DES is reversible, so she can decrypt r and ⊕ that value with m6 to discover what c5 should be. She then decrypts c5 and ⊕s it with m5, to compute c4. Also, she works forward. She can encrypt m1 to get c1. Then she ⊕s c1 with m2 and encrypts the result to get c2 and then ⊕s c2 with m3 and encrypts the result to get c3. Now she has gotten to where the two ends of the computation meet. She ⊕s c3 with the decryption of c4 to get m4. Why don't the other types of MIC have the problems we've described in this section? The reason is that with message digests it is impossible for Carol to come up with another message with a particular message digest. So the PEM designers simply removed DES-MAC as a permissible integrity check. 18.17 DIFFERENCES IN S/MIME MIME specifies how to encode non-text data and type labels into what will look like a text message to intermediate mailers. Like PEM, it had to deal with overly helpful mail transfer agents and defined its own 6-bits-per-byte encoding, so S/MIME doesn't have to. S/MIME says that its m1 m2 m3 m4 m5 m6 ⊕⊕⊕⊕⊕ EEEDDD secret key k c1 c2 c3 c4 c5 CBC Residue r Figure 18-3. Generating a Message with a Given CBC Residue 502 PEM & S/MIME 18.17 encrypted blobs are binary data and MIME takes care of encoding them. But S/MIME still has some encoding issues to deal with. As with PEM, signed messages can be in one of two forms: clear-signed and encoded. Clear signed messages can be read by mail readers that don't understand S/MIME (they notice the signature as a part of the message they don't know how to process and they ignore it). MIME itself was designed so that mailers that don't understand MIME will display text messages so that users will be able to find the message amid a substantial amount of cybercrud. That makes it possible for a user with an old mailer to make sense of an S/MIME signed message if they are sufficiently determined. MIME includes text canonicalization rules that increase the chances that the message will arrive unmodified, but it is still somewhat safer to send messages in an encoded format. The encoded format has an additional advantage. If the recipient has a mailer that does not understand From: Alice To: Bob Subject: Keep this proposition private! MIME-Version: 1.0 Date: Sun, 20 Jan 2002 15:38:21 -0500 Content-Type: multipart/signed; protocol="application/pkcs7-signature"; micalg=sha1; boundary=-------boundarymarker ---------boundarymarker Content-Type: text/plain; charset="US-ASCII" Care to meet me at my apartment tonight? ---------boundarymarker Content-Type: application/pkcs7-signature; name="smime.p7s" Content-Transfer-Encoding: base64 Content-Disposition: attachment;

filename="smime.p7s" Content-Description: S/MIME Cryptographic Signature
MIAGCSqGSIb3DQEHAqCAMIIIlgIBATELMAkGBSsOAwIaBQAwCwYJKoZIhvcNAQcBoIIG7jCCAX
4w
ggEooAMCAQICBDY+EqcwDQYJKoZIhvcNAQEEBQAwRjELMAkGA1UEBhMCVVMxDTALBgNVBA
gTBE1B
U1MxDTALBgNVBAoTBElyaXMxGTAXBgNVBAMTEElyaXMgSW50ZXJuZXQgQ0EwHhcNOTgxMTA
yMDgx
NDAwWhcNMDgxMDMwMDgxNDAwWjBGMQswCQYDVQQGEwJVUzENMAsGA1UECBMETUFT
UzENMAsGA1UE
ChMESXJpczEZMBcGA1UEAxMQSXJpcyBJbnRlcm5ldCBDQTBcMA0GCSqGSIb3DQEBAQUAA0s
AMEgC -----30 more lines like this deleted from this example-----
AQkFMQ8XDTAyMDEyMDIwMzgyMVowIwYJKoZIhvcNAQkEMRYEFMfSZKBY4a55CM7L1rg+sk3T
RlM2
MEMGCSqGSIb3DQEJDzE2MDQwBwYFKw4DAh0wDgYIKoZIhvcNAwICAgCAMAoGCCqGSIb3D
QMHMA0G
CCqGSIb3DQMCAgEoMA0GCSqGSIb3DQEBAQUABE8DVCKj7dYs0Uho2cY/o4VFDe6wUBdBC
3R1bhK5
FnUKIrJLUC3QnBQgm70tWQvL5vK9IGyqvZfovPnDUI/2zd0nS9JHnyjqo2pEVvE0uzb0AAAAAA==
---------boundarymarker 18.17 DIFFERENCES IN S/MIME 503 S/MIME, the encoded format will appear to it as an empty message with a file attachment with the name smime.p7m. If the mailer is configured to process file attachments with external utilities based on the file name suffix, it can be set up to hand off the smime.p7m file to an S/MIME program without any special configuration in the mailer. S/MIME defines two new data formats for MIME to encode: application/pkcs7-signature, and application/pkcs7-mime. Type application/pkcs7-signature holds “detached” signatures within a multipart/signed structure. The format application/pkcs7-mime is used for both encrypted messages and encoded signed messages. Because MIME includes recursive encapsulation of messages as a natural thing to do, S/MIME does not have a concept of a message that is signed and encrypted as a single operation. Instead, S/MIME will sign a message yielding a signed message, and then encrypt the result. A minor security advantage gained with this approach is that the identity of the signer, and even the fact that the message is signed, is invisible to anyone who does not have a key to decrypt the message. This does make life difficult for mailers that would like to display message status (encrypted?, signed?, already read?, sender, ...) in a summary screen before opening individual messages. The S/MIME specification explicitly allows a sender to encrypt a message and then sign the result, or even sign the message, encrypt the result, and then sign the encrypted version again. Each variation has subtly different security properties, but none are actually used. S/MIME defines a large set of header fields for both encrypted and signed messages. Since more years have passed since the time of the PEM design, ever larger numbers of options have been specified. Examples include being able to indicate in the header of a signed message what cryptographic algorithms the sender of the message supports, and which public key (possibly of many) she would like the recipient to use when responding. This evolved from the fact that such information is not encoded in certificates, and the common certificate distribution technique of sending out signed messages in order that the recipient can reply with an encrypted one. Unlike PEM, which encoded its header fields individually with human readable text labels, S/MIME encodes its header information and data using ASN.1. Thus in the box on page 502, the signature field contains sender certificates and other such information, but it’s not readily apparent. The encoding is specified in RFC2630: Cryptographic Message Syntax. Issues around mailing list exploders for S/MIME messages are the same as those for PEM

messages, except that examples are harder to look at. A remailer can simply forward signed messages. To forward encrypted messages, it would have to decrypt the message header and then reencrypt it for each recipient. The layering of encryption over signatures makes it even more straightforward to do this without invalidating the signature. 504 PEM & S/MIME 18.18 18.18 S/MIME CERTIFICATE HIERARCHY An important reason why S/MIME succeeded while PEM failed is that S/MIME did not try to prescribe a particular public key infrastructure. An important side effect was that the security gained from using it is limited (though that could improve if a workable PKI gets deployed). There are several hierarchies people use with S/MIME. 18.18.1 S/MIME with a Public Certifier Several companies, most notably Verisign and Thawte, have tried to make a business of issuing certificates to S/MIME users related to their businesses creating SSL certificates for public web servers. They have different levels of assurance with which users can be authenticated, where that assurance level is indicated in the certificates issued. Presumably, the certificates with a greater degree of assurance cost more, since the higher levels of assurance involve more administrative overhead. Both companies have been fairly successful with their free certificates with low assurance that they issue as a loss leader to get people into the system. 18.18.2 S/MIME with an Organizational Certifier An alternative to getting certificates from a public certifier is to get them from an organization with which you have some affiliation—typically an employer. Several commercial mail products include CAs that allow certificate generation for their users. The challenge with non-public certifiers is getting the recipients' software to trust the issuer of the certificate. For mail sent within organizations, this can be done as a matter of configuration. Also within organizations, users' certificates can be listed in an organizational LDAP directory so that senders can look up recipients' public keys in order to send them encrypted mail. 18.18.3 S/MIME with Certificates from Any Old CA The way S/MIME is most frequently used is without a directory for storing certificates and without a need for trusted certifiers. If I want to send you encrypted mail, I first get you to send me signed mail. Your signed mail contains your certificate, which I accept into my personal address book and tell my system to trust. (If you have a certificate from a certifier I already trust, this is a little easier, but if not I just say "OK" to one more security warning that I don't understand.) 18.19 HOMEWORK 505 This mechanism for securing mail has many theoretical flaws. An attacker could impersonate us to one another or read or forge mail. But in practice, attackers do not yet have the technical sophistication to do these things. S/MIME used in this manner is very effective against passive eavesdropping, which is the attack most people worry about. Even the most sophisticated PKI can't protect you from someone breaking into your machine, stealing your keys, and reading your email that way, which is probably the most common attack in practice. 18.19 HOMEWORK 1. Why doesn't the message portion of a PEM message need a label (such as KEY-INFO:?). In other words, how does a parser know which portion of the PEM message is the actual body of the message? 2. How does Bob, when receiving a signed PEM message from Alice using a public key interchange key, verify the signature on the message? 3. In the public key case of ENCRYPTED messages, there is a KEY-INFO field for each recipient (where the per-message encryption key is encrypted with that recipient's public key), plus a KEY-INFO field for the sender of the message, say Alice, where the per-message encryption key is encrypted with Alice's public key. The purpose of the KEY-INFO field for Alice is so that if she were later to see a copy of the encrypted message, she'd be able to decrypt it.) In the secret key case of ENCRYPTED messages, however, there is no such field for Alice (though there is still one for each recipient). Why was it considered necessary to include a KEY-INFO field for Alice in the public key case and not in the secret key case? 4. Design a mechanism to allow Alice and Bob to establish a secret key cryptographic interchange key, using something like Kerberos KDCs. Assume for simplicity that Alice and Bob are in the

same realm. If Alice does not keep track of KDC-assigned interchange keys for people to whom she has previously sent messages, how can your design ensure Alice would be able to later decrypt an encrypted message she had sent to Bob? 5. Why are the KEY-INFO fields not necessary in public key MIC-CLEAR and MIC-ONLY messages, whereas they are necessary in public key ENCRYPTED messages and in all secret key messages? 6. If Alice is sending an ENCRYPTED message, she first signs the message digest with her private key (as she would have done for a MIC-CLEAR or MIC-ONLY message), and then encrypts the message digest with the per-message secret key. Why was this last encryption considered necessary for encrypted messages and not for MIC-CLEAR or MIC-ONLY? 506 PEM & S/MIME 18.19 Now assume Alice sends Bob a MIC-ONLY or MIC-CLEAR message with secret key based interchange keys. PEM requires her to encrypt the MIC with Bob's interchange key. Why is this necessary? 7. In §18.16 DES-CBC as MIC Doesn't Work we described how Carol, given a 64-bit value x and a key k, could construct a message m with DES-CBC x. Suppose Carol can't really send an arbitrary 8 bytes of garbage embedded in the message. For instance, suppose she wants to send the forged message MIC-CLEAR, so each byte is constrained to have one of only 64 valid values. Assume Carol is reasonably flexible about the message she'd like to send (for instance there might be two places in which crud could be embedded). Find a computationally feasible method for Carol to generate a message which will satisfy the constraint. 8. Describe a plausible algorithm and user interface for a PEM processor at a receiver that deals with nested PEM messages, embedded strings that look like begin and end PEM markers, and messages in which some of the PEM enclosures fail PEM processing due to signatures not verifying or certificates missing. 9. Given that each message will be encrypted with its own per-message key, why does PEM also provide an IV? 10. Suppose PEM signed an encrypted message (rather than signing the unencrypted message and encrypting the result)? Design a mechanism for allowing Bob, the recipient of such a message from Alice, to forward it to Carol, so that Carol can both verify Alice's signature and read the contents of the message. 11. Assume an encryption mechanism such as $\oplus$ with a one-time pad, where the one-time pad is the per-message key. Why would such an encryption scheme not work with the design in Homework Problem 10? (Hint: Carol should not believe Alice signed the message.) 12. As described in §18.16 DES-CBC as MIC Doesn't Work, there are many scenarios in which messages can be forged. Suppose Alice sends an ENCRYPTED message to Bob, using MD5 as the MIC, but Ted accepts DES-CBC. Can an eavesdropper now forge messages from Alice to Ted? How about Bob? If so, how? If not, why not? Suppose Alice's message was MIC-CLEAR or MIC-ONLY? 13. Why is DES CBC secure for banking, but not for email? (Hint: What cryptographic properties does banking depend on? What cryptographic properties does email depend on?) 14. Suppose DES-CBC was only used with secret interchange keys, and the CBC was always computed using the interchange keys (rather than the per-message key). Would this be secure? 507 19 PGP (PRETTY GOOD PRIVACY) 19.1 INTRODUCTION PGP is another secure mail protocol. It started out as a single public-domain implementation that was, in the words of Phil Zimmermann, the original author, guerrilla freeware. It was the author's intention that it be distributed widely (anyone with a copy is invited to send copies to friends; copies can also be obtained from a bunch of Internet FTP sites). Unfortunately, both RSADSI, by enforcing the RSA patent, and government authorities, by enforcing export control of dangerous technologies like nuclear weapons and the ability to encrypt mail, caused PGP to start its life as contraband. Its history is colorful, serving as a test case in a number of legal challenges to U.S. export laws. We recommend GARF95 as an entertaining and informative telling of its early history and use. Ironically, PGP has been, from the start, legal and freely available in many other countries because the RSA patent is U.S.-only and other governments have different policies about the

export, import, and use of privacy protection technology. For a while, the most popular site distributing PGP over the Internet was in Finland. Finland is very liberal about cryptography (but is still snitty about selling nuclear weapons, we presume). The PGP documentation is delightfully conspiratorial and fun to read. Phil starts out by asserting that although we honest people don't really think we need to encrypt our email—we're not hiding anything—we should all start encrypting our mail so that in case someone needs privacy, the poor soul won't arouse suspicion by being the only one encrypting mail. Phil uses the analogy with paper mail. If the "standard" scheme were to write everything on postcards, and then only in unusual cases where you felt you didn't want the government or the next door neighbor reading your mail would you enclose it in an envelope, then a letter in an envelope would arouse suspicion and save the snooper time—every letter in an envelope would probably contain something interesting to the snooper. The questionable legality of PGP inspired a cult following, with people using it as a statement in support of personal freedoms and geek pride, independent of any need for secrecy. 508 PGP (PRETTY GOOD PRIVACY) 19.2 If privacy is outlawed, only outlaws will have privacy. —Phil Zimmermann In trying to accommodate patents, export laws, a desire for commercialization, and committees of people trying to help, PGP has unfortunately evolved into a Tower of Babel. PGP Classic (Version 2.6.*) gained a large user community using the RSA and IDEA encryption algorithms. Through arduous negotiations, one version (2.6.2) became legal for non-commercial use within the U.S. and was freely downloadable from an MIT web site. And it was possible to buy a license making it legally usable for any purpose. Not satisfied with that, an incompatible PGP using patent-free algorithms (DSS, Diffie-Hellman, and Triple DES) was developed. It also avoided U.S. export law by a novel mechanism. U.S. export law covers software, but not books (books existed when the first amendment was drafted, but the Internet did not). The source code for PGP was published as a book, the book was legally exported from the U.S., and then someone outside the U.S. optically scanned the book and turned it back into software outside the U.S. In large part to publicly embarrass the export authorities, the book was published in an OCR font (a font designed for Optical Character Recognition before scanners became good enough to accurately scan almost any font) and digital checksums were included on each page. Surprisingly, the book also sold well to the public, who bought it either out of ignorance or to be able to display a great hack. This had the potential for getting PGP out from under patent and export restrictions, but it fragmented the user community (since the two versions could not interoperate). In trying to harness the market buzz around "Open Software", the internal data representations of PGP were redesigned by an IETF committee and dubbed "Open PGP". Several independent (and not always interoperating) versions of Open PGP have been deployed including one called GPG (Gnu Privacy Guard). While all of this was going on, the RSA patent expired in September 2000 potentially making PGP Classic finally legal for all uses everywhere. Unfortunately, a patent on the IDEA algorithm that expires in 2007 has postponed that possibility. The formats described in the remainder of this chapter are for PGP Classic. The newer ones are more complex, but similar in spirit and goals. 19.2 OVERVIEW PGP is not just for mail. It performs encryption and integrity protection on files. Mail is not treated any differently from ordinary files. Someone wishing to send a secure mail message could first transform the file to be mailed using PGP, and then mail the transformed file using a traditional mailer. Similarly, if one were to receive a PGP-encrypted mail message, one could treat the received message as a file and feed it to PGP to process. This is a bit inconvenient. So PGP source code 19.3 KEY DISTRIBUTION 509 comes with modifications for a number of common mail systems, enabling people to integrate PGP into their mail systems. 19.3 KEY DISTRIBUTION PGP uses public key cryptography for personal keys. The most important differences between PGP, PEM, and S/MIME lie in how public keys are certified and

how certificate chains are verified. PEM assumes a rigid hierarchy of CAs. PGP assumes anarchy. S/MIME is agnostic, but as deployed assumes a number of parallel independent hierarchies, a subset of which are trusted by each user. With PGP, each user decides which keys to trust. If Bob has talked in person to Alice, and Alice has written down her key for him, Bob can store Alice's public key and trust that it is hers. If he trusts Alice sufficiently, and Bob receives a certificate signed by Alice's key that says Carol's public key is P, then Bob can trust that P really does belong to Carol. While with PEM the infrastructure decides whom you should trust to certify people, PGP leaves it up to you. This means PEM is potentially more convenient, while PGP is potentially more secure (if you're careful). It also means that PEM is unusable until you "hook into" an infrastructure. Deployment of PEM was held up for years trying to settle the details of how that infrastructure would work. PGP is easy to use "out of the box", but it's an open question whether it will be able to scale to large environments. In the PEM model, finding a path of certificates is very simple. The certificate chain follows the naming hierarchy (approximately), and the path is simple—one simply goes down from the root (the Internet PCA Registration Authority). Knowing whether to trust a path is simple in PEM. There is only one path and you are expected to trust it. S/MIME assumes there are multiple certificate hierarchies, and software will be configured with which ones to trust. You trust all certificates in a trusted hierarchy. You can trust me. —used car salesman, Westboro, MA, April 4, 1993 PGP doesn't require certificates at all, though they can make life simpler. To send someone encrypted mail or to verify their signature, you need to know their public key. People publish their PGP fingerprints (cryptographic hashes of public keys) on their web sites, on their business cards, or even in their books. For example, my1 PGP fingerprint is 29 6F 4B E2 56 FF 36 2F AB 49 DF DF B9 4C BE E1. They are also short enough to be read over the phone. Certificates are an optional in PGP. Anyone can issue a certificate to anyone else. And users decide whose certificates they are willing to trust in authenticating someone. As a result, with PGP 510 PGP (PRETTY GOOD PRIVACY) 19.3 there might be several certificate paths to a particular person. For instance, you might have the following certificates: It might even be the case that the last certificate instead said There are several issues: • You might have a disorganized mass of certificates. How can you find a chain that leads from a key you know to Carol's key? • There might be multiple chains, and some might lead to different keys for Carol. • If you do find a chain, how much do you trust that chain? In order to trust a chain of certificates you have to believe that the first public key in the chain really belongs to whom you think it does, and that you trust every individual in the chain. Should you trust a certificate signed by what you believe is Ted's key? Even if you're sure that Ted's public key is P5, can you trust Ted not to certify anything in sight for a $100 bribe? Even if Ted is honest, can you trust him not to be sloppy about checking that the key really belongs to the named person? In other words, are Ted's pre-certification policies and procedures for authenticating users adequate? Suppose Ted is honest and usually very careful about checking a key before signing a certificate. Suppose Carol is Ted's ex-spouse, from whom he had a bitter divorce. Should you then trust Ted's signature to certify Carol's public key? PGP cannot answer these questions, of course. Not even people can really answer these questions. But PGP asks you for advice, and stores with each key a quantity indicating how much the key should be trusted as being legitimate and how much the owner of the key should be trusted in certifying other keys. With a particular signature (as in the case where you'd trust Ted to certify just about any key except Carol's), PGP also asks its user for advice and stores information about how much that particular signature should be trusted. It's worth noting that the commercial version of PGP tried to address these issues with a less anarchistic structure. That's because it was marketed to organizations where the users were not considered sufficiently motivated and competent to make the right decisions, but where the

system administrators were. Carol's key is P1 signed with P2 Alice's key is P2 signed with P4 Carol's key is P1 signed with P5 Carol's key is P3 signed with P5 19.4 EFFICIENT ENCODING 511 19.4 EFFICIENT ENCODING To appreciate how clever PGP is, let's review inefficiencies of encoding in PEM (S/MIME inherits most of its encodings from MIME, making it non-comparable). Both PEM and PGP are designed to work in the environment where the contents of a message might be restricted by two problems. The first problem is that intermediate mail relays insist that the message contain only innocuous ASCII characters and limited-length lines. The second problem is that the representation of text at the destination machine may be different from the representation at the source machine. The solution to this problem is to turn the message into canonical format before encrypting or signing it. This includes converting the end-of-line indication, which might be linefeed only () at the source, into which is the SMTP standard. This may make the message a little bit bigger. PEM expects to be handed ordinary text. If the message being sent with PEM is not in fact text consisting of innocuous ASCII characters, then in order to send it through PEM it must be encoded with a utility such as uuencode, which expands it by about 33%. Then PEM does whatever processing it needs to do. For instance it might encrypt the (newly expanded) message. Once encrypted, the message has to get encoded by PEM so that it will consist of only innocuous ASCII characters with s occurring sufficiently frequently. If the message started out as ordinary text, then only PEM will do the encoding and the message will expand by about 33%, but if the message needed to be encoded for submission to PEM, then the data will have been encoded twice, resulting in an expansion of about 78%. A message sent MIC-CLEAR will not need to be expanded even once. To summarize, there are four cases: 1. Text sent with PEM as MIC-CLEAR—the message does not get expanded at all. 2. Text sent with PEM as MIC-ONLY or ENCRYPTED—the message gets expanded once (33%). 3. Non-text sent with PEM as MIC-CLEAR—the message gets encoded before submittal to PEM, thus expanded once (33%). 4. Non-text sent with PEM as MIC-ONLY or ENCRYPTED—the message gets expanded twice (78%). Since PEM is going to encode the message anyway, why do you need to encode the message before handing it to PEM? The reason is the canonicalization step that PEM insists on performing. If you have a file that would be damaged by the canonicalization, you have to encode it so that PEM won't damage it. In contrast, PGP allows you to specify, when handing a file to PGP to process, whether the file is text or binary. (Though PGP checks up on you when you tell it you're giving it text—if it 512 PGP (PRETTY GOOD PRIVACY) 19.5 doesn't look like text, PGP will change the state of the flag associated with the file). If you tell PGP the file is binary, then PGP will not canonicalize it, and PGP will mark the PGP-processed message as binary so that the PGP at the receiver will know not to perform reverse canonicalization. By merely marking the file this way, PGP manages to avoid ever needing two encodings on a single message. Furthermore, PGP does not always assume that the encrypted file needs to be encoded in order to get through mail relays. For instance, the file might be sent through a mail system (like X.400) that supports transparent transport of data. Or it might not be mailed at all, but merely transported via floppy disk. In some cases, PGP might know, based on the mail system into which it is integrated, whether the file needs to be encoded. In other cases, the user might explicitly tell PGP whether it should encode the output to prepare it for simple-minded mailers. An additional trick that PGP does to conserve bits is to compress a file before sending it, whether it is binary or text. PGP uses the utility ZIP for compression. Typically, 50% compression is achieved, so even though PGP will need to do one expansion encoding after encrypting a file, what PGP actually has to transmit will often be smaller than the original file. 19.5 CERTIFICATE AND KEY REVOCATION A certificate can be revoked by whoever signed the certificate. It does not necessarily mean that the key in the certificate is bad; it just means that whoever signed the certificate no longer

wants to vouch for its authenticity. Given that anyone can issue a certificate for anyone, and then issue a revocation of the certificate, having a revoked certificate does not necessarily mean anything bad about the individual certified. Otherwise, an enemy of yours could issue a certificate for you and then immediately revoke it, making everyone suspicious of you. (The reasoning might be: Why was the certificate revoked? Must be because the person had lied or something...) If a certificate is revoked, then whoever was relying on that certificate to authenticate you will no longer be able to authenticate you. Certificates can optionally have a validity period, in which case they expire after the specified time period. They can be renewed by being reissued. The current custom in PGP is to issue non-expiring certificates (by omitting the VALIDITY PERIOD field). Keys can be revoked too. My key can only be revoked by me (or someone who stole my private key—but gee, if the bad guy is so obliging as to tell the world he stole my key, I'm not worried about him forging a key revocation for my key). The idea is that I will issue a key revocation only when I think someone has discovered my private key. I'll presumably generate a new key for myself, and distribute the key revocation for the old key as widely as possible (perhaps by publishing it in The New York Times as a legal notice). 19.6 SIGNATURE TYPES 513 Key and certificate revocations are distributed informally, just as are public keys and certificates. There are public servers where you can post and search for certificates. A ritual has developed for populating those servers. Frequently, at gatherings of nerds, there is a PGP key signing "party". People stand and state their name and public key fingerprint. People who know that person are encouraged to issue certificates for them. If you go to enough of these events, you're likely to get certificates from enough people that members of the community will be able to trust your key. 19.6 SIGNATURE TYPES For each signed quantity, PGP indicates whether what is being signed is a message or a certificate. The reason for this is to guard against the remote possibility that you'd sign a message saying something like Let's do lunch, and the encoding of Let's do lunch could be parsed as a valid certificate. It's good practice whenever a key can be used for multiple things to explicitly say what kind of thing is being processed. This avoids any possibility of aliasing (in this case confusing messages and certificates). One could argue that it's really ridiculously unlikely that a message could be parsed as a certificate, or vice versa, but PGP's SIGNATURE TYPE field is only one octet long, so why not? 19.7 YOUR PRIVATE KEY If all you want to do is verify signatures on signed messages, you don't need a private key. However, if you want to sign your own messages, or if you want to receive encrypted PGP mail, you need a private key. PGP will generate a private key for you. You can specify the size of the key. Then it asks you for a password, and it converts the string you type into an IDEA key by doing an MD5 message digest on it. Then it uses that IDEA key to encrypt the private key. Encryption is done with 64-bit CFB using a random IV which is stored with the encrypted private key. 514 PGP (PRETTY GOOD PRIVACY) 19.8 19.8 KEY RINGS A key ring is a data structure you keep that contains some public keys, some information about people, and some certificates. Usually this information is just stored locally, but some PGP users might share their key ring information, which would provide a quick method of building up your database of public keys. PGP allows you to assert how much trust you place on different people. There are three levels of trust: none, partial, or complete. You might not trust Fred at all, so you would not trust a certificate signed by Fred, but you might want to keep Fred's public key around so that you can verify messages you receive from him, or be able to send encrypted messages to him. PGP computes the trust that should be placed on certificates and public keys in your key ring based on the trust information you asserted on the people. A certificate signed by someone you indicated you don't trust at all will be ignored. Whom or what you trust is a very private decision. You could have your own version of PGP that had arbitrarily complex trust rules. It does not have to be compatible with anyone else's. It could even always ask you to

make the decision for each key and certificate, but give you the raw data (Carol and Ted both said this was Alice. You indicated before you have level 4 trust in Carol, level 6 trust in Carol's public key, level 5 trust in Ted, and level 2 trust in Ted's public key. How much trust would you like indicated for Alice's key?) The public domain version of PGP gives you only three levels of trust (none, partial, or complete, as we said above). One completely trusted individual signing a certificate will yield a completely trusted certificate. Several partially trusted individuals signing certificates attesting to the same public key for Alice will yield a completely trusted public key for Alice. If you have a key for Alice that is not completely trusted, you can still verify messages from Alice, and send encrypted messages to Alice, but PGP will give you a warning. 19.9 ANOMALIES 19.9.1 File Name PGP includes a FILE NAME field with any message (see §19.10 Object Formats). This field specifies the name of the file that was read to produce the PGP object so that the recipient can ask PGP to store the information using the same file name. We don't think this is very useful in the context of mail, although it can be helpful when PGP is used as a file encryption tool. But for mail, it means you have to be careful what you call the file on your machine (a name like TO-THE-DWEEB 19.9.2 OBJECT FORMATS 515 might offend the recipient, and you might have forgotten that PGP tells the recipient what you called the file on your machine). Aside from being mostly useless, this feature is slightly dangerous. A careless user might be tricked into storing a received message that is actually a virus-containing executable with a file name (such as .login) that the system might understand and wind up executing. Admittedly, PGP warns you when it is about to overwrite an existing file, but the user might not have such a file, or might ignore the warning. PGP also includes the time the file was last modified. As with FILE NAME, this can be useful for a file encryption tool, but its existence is probably a bad idea for mail, because again it's telling information that you may not realize it's divulging. 19.9.2 People Names PGP allows you to choose any name you want. It does not enforce uniqueness or reasonableness. This causes an interesting problem. You can name yourself Richard Nixon (that might even be your name but you might not be the Richard Nixon). You might have lots of people willing to sign certificates as to your public key. An unsuspecting person, say Julie, has a public key associated with the name Richard Nixon. She would completely trust the Richard Nixon she knows, but doesn't realize that the key she holds is yours, not his. If I have two public keys, both indicating that the associated human is John Smith, I won't know which key to use when I send a message to one of the John Smiths. For this reason, it is helpful if the user identification string contains a little more information than just the name of the user, for instance, the user's email address. PGP recommends (but does not require) including Internet email addresses in the name. 19.10 OBJECT FORMATS PGP reads and writes a variety of objects. Objects may contain a variety of fields, and some objects contain other objects. Each object can be represented in two formats—compact and SMTP mailable. The SMTP mailable format is computed from the compact format by performing the same encoding that PEM uses (see §18.7 Reformatting Data to Get Through Mailers) and adding a human-readable header and trailer so that a human (or a smart mail receiver) will know the cybercrud between the header and trailer needs to be processed by PGP. The human-readable header is 516 PGP (PRETTY GOOD PRIVACY) 19.10.1 -----BEGIN PGP MESSAGE----- Version 2.2 and the human readable trailer is -----END PGP MESSAGE----- 19.10.1 Message Formats PGP messages are composed of a sequence of primitive objects (see §19.10.2 Primitive Object Formats). We'll describe the message formats in this section. Encrypted Message. PGP encrypts messages with the secret key algorithm IDEA using 64-bit CFB mode with a randomly chosen IV that is included with the message. The sending PGP chooses a random IDEA key for encrypting the message, and then encrypts that IDEA key with the public key of the recipient, placing the encrypted IDEA key in the header. If the message is being sent to multiple

recipients, the IDEA key is individually encrypted with the key of each recipient, and all the encrypted keys are placed in the header. When the final object is decrypted with the IDEA key, the result is either PLAINTEXT DATA or COMPRESSED DATA, depending on whether PGP decided the data was compressible. If PGP decided the data was compressible, then when the COMPRESSED DATA is uncompressed, the result is PLAINTEXT DATA. Signed Message. This is likely to be COMPRESSED DATA. When uncompressed, the result is Encrypted Signed Message. This is just like an ENCRYPTED MESSAGE, but with the message replaced by a SIGNED MESSAGE. IDEA key encrypted with recipient1's public key IDEA key encrypted with recipient2's public key ... IDEA key encrypted with recipientk's public key message encrypted with IDEA key signature plaintext data IDEA key encrypted with recipient1's public key IDEA key encrypted with recipient2's public key ... IDEA key encrypted with recipientk's public key signed message encrypted with IDEA key 19.10.2 OBJECT FORMATS 517 When the final object is decrypted with the IDEA key, the result will probably be COMPRESSED DATA (unless PGP noticed that it was not compressible), which when uncompressed is a SIGNED MESSAGE. Signed Human-Readable Message. Usually PGP modifies even an unencrypted message before mailing it, for two reasons. It compresses it for ecological reasons (saves bits), and it encodes it so it will have some hope of making it through mailers unmodified. This is nice but it means that if the recipient does not have PGP, the human at the other end will not be able to read the message (see §18.7 Reformatting Data to Get Through Mailers). So PGP allows you to tell it not to muck with the message. You will hold PGP harmless if the message gets modified by mailers and the signature then does not verify. This form of message is Just like with PEM, we have to ensure that the string -----BEGIN PGP SIGNATURE----- does not appear in the text of the message. And just like PEM, PGP protects against such a mishap by prepending '- ' to any line that begins with '-'. The PGP at the receiver is smart enough to strip that off before attempting to verify the signature. 19.10.2 Primitive Object Formats Primitive object formats are where CTB stands for cipher type byte (aren't you glad you asked?) and specifies the PGP object type and the length of length. We'll describe the interpretation of stuff for the various object types in this section. Certain object types contain one or more integers in their stuff field. Integers are encoded as a length in bits (two octets, most significant first) followed by the binary integer with leading 0 bits to pad to a multiple of 8 bits then packed into octets most significant first. The CTB is interpreted bit by bit (most significant to least significant) as follows: bit 7: 1 bit 6: reserved, i.e. set to 0 and ignored bit 5–2: a 4-bit field indicating which type of object follows. The defined values are -----BEGIN PGP SIGNED MESSAGE----- cleartext message -----BEGIN PGP SIGNATURE----- Version: 2.2 signature -----END PGP SIGNATURE----- CTB length stuff 518 PGP (PRETTY GOOD PRIVACY) 19.10.2 0001—something encrypted using someone's public key (the only thing ever actually encrypted this way is an IDEA key) 0010—signature 0101—private key encrypted with a password 0110—public key 1000—compressed data 1001—something encrypted with a secret key 1011—plaintext data plus a file name and mode (binary or text) 1100—key ring trust information 1101—user identification 1110—comment bits 1–0: a 2-bit field indicating the length of the length field. The defined values are: 00—1-octet length field follows CTB 01—2-octet length field follows CTB 10—4-octet length field follows CTB 11—there is no length field (this can only occur for the last object in a message—the length is to the externally determined end of data) Key Encrypted under a Public Key (0001 in CTB). IDEA key before RSA-encryption. PGP follows the PKCS #1 standard, which defines how to encrypt something with RSA, usually a key (see §6.3.6 Public-Key Cryptography Standard (PKCS)). PGP encrypts a little more than just the IDEA key. PGP encrypts a 19-octet quantity consisting of an octet = 1, the 16-octet IDEA key, and a 2-octet CRC-16 of the IDEA key. The purpose of the extra data is so that in case you accidentally decrypt with the wrong RSA key the contents will not

look right. So PGP encrypts, using RSA, field description 1 octet version number (2) 8 octet key ID (low order 64 bits of RSA modulus) 1 octet algorithm ID (1 for RSA, no others defined) integer IDEA key encrypted with RSA 0 2 n random non-zero octets 0 1 IDEA key CRC 19.10.2 OBJECT FORMATS 519 Signature (0010 in CTB). The defined signature types (written in hex) are 00—signature on a binary message or document 01—signature on a text message with indicating EOL 10—certificate with unspecified assurance 11—key certification with no assurance 12—key certification with casual assurance 13—key certification with heavy-duty assurance 20—key revocation (attempt to revoke key that signed this) 30—certificate revocation (revokes a certificate signed with the key that signed this) 40—signature of a signature (notary service) The message digest is computed on The quantity signed using RSA is The 18-octet constant is ASN.1 cybercrud meaning MD5. field description 1 octet version number (2) 1 octet # of octets before key ID field (5 if validity period missing, else 7) 1 octet signature type (see below) 4 octet time when signed 0 or 2 octet validity period in days (0 means infinite) 8 octet key ID (low order 64 bits of RSA key) 1 octet public key algorithm (1 for RSA, no others defined) 1 octet message digest algorithm (1 for MD5, no others defined) 2 octet first two octets of message digest, to reassure us that we decrypted with the proper key integer message digest signed with RSA (see below) message to be signed signature type timestamp validity period 0 1 n octets of FF16 0 3020300c06082a864886f70d02050500041016 message digest 520 PGP (PRETTY GOOD PRIVACY) 19.10.2 Private key encrypted with password (0101 in CTB). The encrypted quantity, before encryption, looks like Encryption is done using IDEA in 64-bit CFB mode (see §3.4 International Data Encryption Algorithm (IDEA) and §4.2 Encrypting a Large Message) Public key (0110 in CTB). Compressed Data (1000 in CTB). Any PGP object or sequence of objects can be compressed. field description 1 octet version number (2) 4 octet time when key created 2 octet validity period in days (0 means infinite) 1 octet public key algorithm (1 for RSA, no others defined) integer public modulus integer public exponent 1 octet algorithm encrypting private key (1 for IDEA, no others defined) 8 octet random if key encrypted, else 0 integer encrypted quantity field description integer private exponent integer p (the first prime, in format of integer) integer q (the second prime, in format of integer) integer information for speeding up the RSA operations (see §6.3.4.4 Optimizing RSA Private Key Operations) 2 octet CRC-16 of preceding fields field description 1 octet version number (2) 4 octet time when key created 2 octet validity period in days (0 means infinite) 1 octet public key algorithm (1 for RSA, no others defined) integer public modulus integer public exponent field description 1 octet compression algorithm (1 for ZIP, no others defined) – compressed data 19.10.2 OBJECT FORMATS 521 Something Encrypted with a Secret Key (1001 in CTB). Encryption is done using IDEA in 64-bit CFB mode with an IV of 0. The data being encrypted is The purpose of the random data is so that successive encryptions of the same unencrypted data will result in different ciphertext. The purpose of repeating the last two octets of random data is to recognize that decryption worked properly. Plaintext Data (1011 in CTB) . Key Ring. The key ring stores public keys and information about each key. The information about each key consists of the name of the human associated with the key (the user ID), the set of certificate signatures you've received vouching for the fact that that user has that public key, and trust information about each of the pieces of information. Key revocations and certificate revocations (see SIGNATURE object format) are also stored on the key ring. So the key ring looks like The trust information following the public key indicates how much that public key is trusted to sign things, and is manually set. If the key has been revoked, the key revocation is inserted immediately before the user ID. The trust information following the user ID indicates how much confidence is placed on the public key belonging to the user, and is computed from the trust information following the signatures. The trust information following each signature is copied from the trust information of the public key that

did the signing. Certificate revocations are included with the normal signatures. field description – encrypted data field description 8 octet random data 2 octet last two octets of random data – unencrypted data field description 1 octet mode (ASCII b for binary, t for text) 1 octet length of following field in octets file name (ASCII string) 4 octet time file last modified, or 0 – message public key trust user ID trust signature trust ... signature trust public key trust user ID trust signature trust ... signature trust ... public key trust user ID trust signature trust ... signature trust 522 PGP (PRETTY GOOD PRIVACY) 19.10.2 PGP encourages people to share the information in their key rings. However, users should make their own trust decisions. So when key ring information is transmitted, the trust information is removed. Key Ring Trust Information (1100 in CTB). Since this format is only used locally, we won't bother documenting its gory details, but we will say that it has two different interpretations depending on whether the previous item is a user ID or not. User Identification (1101 in CTB). Comment (1110 in CTB). field description 1 octet how much the preceding item should be trusted field description – ASCII string giving user's name, perhaps email address also field description – ASCII string PART 5 LEFTOVERS This page is intentionally left blank 525 20 FIREWALLS A firewall (see Figure 20-1) is a computer that sits between your internal network and the rest of the network and attempts to prevent bad things from happening (such as internal users sending company secrets outside, or outside people breaking into systems inside) without preventing good things from happening (such as employees accessing information available externally). It is sometimes called other things, like a security gateway, or various more colorful names thought up by frustrated network users when it prevents them from doing what they want to do. Why is a firewall needed? It isn't if the systems on your internal network are properly secured, meaning that every system in the network has sophisticated authentication, the ability to do integrity-protected and encrypted communication whenever talking across the network, and is well-managed (when users rely on passwords, they choose good passwords; system accounts are not left with no password or a default password; all the latest security patches are installed; ...). Unfortunately many applications exist that were designed (and we use the term loosely) for an environment without bad guys, and most users view security (such as being forced to use long passwords) as an annoyance. Most corporate networks are not designed for security, and have survived because they have not been attacked. But now users want connectivity to the Internet. You want to be able to exchange mail with anyone. You want to occasionally share files. You want to communicate with publicly available services, and make some of your own services available to customers located outside your corporate network. But the Internet is a scary place. There are spies from unfriendly countries, users from competing companies, playful undergraduates, press people eager for a juicy scoop, criminals anxious to steal information for profit, disgruntled ex-employees, and vandals who attempt to compensate for their lack of a social life by annoying others. your net firewall Internet Figure 20-1. Firewall 526 FIREWALLS One of the reasons systems are hard to secure is that they provide so many services, many by default. You might think that a user workstation would only initiate requests to servers on the user's behalf, but most user workstations provide services like remote access to their file systems and configuration databases. This makes workstations easier to centrally manage and makes it easier to share files, but every service is a point of attack. Even if the services are properly configured to only serve authorized users, they could be missing bounds checks in their input parsers so that a clever attacker can send an ill-formed request and sometimes get access to things the service was not designed to provide. It's common for a user who wants to share a file with another user to make his entire file system remotely mountable (because that's easier than limiting access to a single user and a single file). Often, he intends that things only be opened for a short time, but he forgets and leaves his system exposed indefinitely. Firewalls

centrally manage access to services in ways that individual systems should but don't. They can enforce policies such as systems outside the firewall can't access file services on any systems inside the firewall. With such a restriction, even if the internal systems are more open than they should be or have bugs, they can't be attacked directly from systems outside the firewall. Also, there are many ways to communicate with a system, since people keep coming up with mechanisms intended for debugging or convenience, and worse yet, many different ways of accomplishing the same purpose. In the university environment, in which a lot of these utilities were developed, it might have been reasonable to trust any message received. But each of these mechanisms is a potential new door into the system. Firewalls attempt to protect systems inside from attack from outside. But in order to do this, the firewall administrator needs to understand all the mechanisms and come up with some way of disallowing the dangerous ones without disabling the necessary ones. Some example mechanisms include: • finger (port 79), which gives information such as who is logged in, or when a specific user last logged in. • telnet (port 23), which allows someone to log into the system from across a network. • rlogin (port 513), similar to telnet, but UNIX-specific. Also, whereas telnet requires explicit user authentication, rlogin (along with a few other UNIX utilities known as r utilities, such as rsh) avoids the inconvenience of explicit user authentication by trusting that the source address in the IP header is correct, and if that address is known to be the address of a host trusted by this system, then anything that the message asserts about the identity of the user is believed. • ftp (port 21), which allows sharing of files. This protocol is tricky for firewalls because there are two connections made—one for control information and one for data. Although a host inside might initiate the transfer (in which case the firewall administrator would like to allow the transfer), the data connection is created by the other end. There is a modified version of FIREWALLS 527 the ftp protocol (called passive ftp) in which both connections originate at the requester in order to make firewall logic easier; but until it is deployed everywhere, firewalls will need to deal with the original ftp. • X Windows (port 177), which (like telnet and rlogin) allows you to interact with a remote machine, but the display assumes a real monitor that can handle colors and fonts, rather than a "dumb terminal". • ICMP (Internet Control Message Protocol, RFC 792), is a protocol for sending messages to a node informing it of various conditions. There is no authentication of an ICMP message. ICMP packets can be recognized because the PROTOCOL field in the IP header is equal to 1. The other mechanisms run on top of TCP or UDP, and therefore can be distinguished based on the PORT field in the layer 4 header. Examples of ICMP messages a system Alice might receive include: ♦ notification that some packet Alice had previously sent can't be delivered because the destination address, or an entire range of addresses including the destination's, is unreachable. ♦ a redirect, telling Alice to use a particular router for forwarding to a particular address, presumably because the router Alice chose on a previous packet was not the best path to the destination. ♦ a ping, which is supposed to be echoed back by the system that receives it. Ping is useful for seeing if a system is alive and reachable. Notice that there are many ways of doing the same thing. To be logged into a remote machine, one might use telnet, rlogin, or the X Windows utility. None of these utilities were designed to be secure. rlogin is a particularly egregious example. Its primary "advantage" over telnet is that you often won't have to type your username or password. There is a file on a UNIX machine (called /etc/hosts.equiv) that lists IP addresses or host names (from which the IP address will be looked up in DNS). The hosts listed in that file are trusted to assert any user identity, and no password is required. The only authentication of the machine is that the IP header contains the trusted machine's IP address. Also, each user can have in his home directory a file named .rhosts which lists IP