UNIT V COMMUNICTION SECURITY **10 hrs.**

Communication Security: Authentications Protocols, E-mail Security, Web security, Social Issues, SDL based protocol verification and validation, Internet protocol, SDL based interoperability testing of CSMA/CD and CSMA/CA protocol using Bridge, Scalability testing-Applications

5.1. Authentication Protocols

Authentication is the technique by which a process verifies that its communication partner is who it is supposed to be and not an imposter. Verifying the identity of a remote process in the face of a malicious, active intruder is surprisingly difficult and requires complex protocols based on cryptography. In this section, we will study some of the many authentication protocols that are used on insecure computer networks.

As an aside, some people confuse authorization with authentication. Authentication deals with the question of whether you are actually communicating with a specific process. Authorization is concerned with what that process is permitted to do. For example, a client process contacts a file server and says: I am Scott's process and I want to delete the file *cookbook.old*. From the file server's point of view, two questions must be answered:

1. Is this actually Scott's process (authentication)?
2. Is Scott allowed to delete *cookbook. Old* (authorization)?

Only after both of these questions have been unambiguously answered in the affirmative can the requested action take place. The former question is really the key one. Once the file server knows to whom it is talking, checking authorization is just a matter of looking up entries in local tables or databases. For this reason, we will concentrate on authentication in this section.

The general model that all authentication protocols use is this. Alice starts out by sending a message either to Bob or to a trusted **KDC** (**Key Distribution Center**), which is expected to be honest. Several other message exchanges follow in various directions. As these messages are being sent Trudy may intercept, modify, or replay them in order to trick Alice and Bob or just to gum up the works.

Nevertheless, when the protocol has been completed, Alice is sure she is talking to Bob and Bob is sure he is talking to Alice. Furthermore, in most of the protocols, the two of them will also have established a secret **session key** for use in the upcoming conversation. In practice, for performance reasons, all data traffic is encrypted using symmetric-key cryptography (typically AES or triple DES), although public-key cryptography is widely used for the authentication protocols themselves and for establishing the session key.

The point of using a new, randomly-chosen session key for each new connection is to minimize the amount of traffic that gets sent with the users' secret keys or public keys, to reduce the amount of cipher text an intruder can obtain, and to minimize the damage done if a process crashes and its core dump falls into the wrong hands. Hopefully, the only key present then will be the session key. All the permanent keys should have been carefully zeroed out after the session was established.

5.1.1. Authentication Based on a Shared Secret Key

For our first authentication protocol, we will assume that Alice and Bob already share a secret key, K_{AB}. This shared key might have been agreed upon on the telephone or in person, but, in any event, not on the (insecure) network.

This protocol is based on a principle found in many authentication protocols: one party sends a random number to the other, who then transforms it in a special way and then returns the result. Such protocols are called **challenge-response** protocols. In this and subsequent authentication protocols, the following notation will be used:

A, *B* are the identities of Alice and Bob.

R_i's are the challenges, where the subscript identifies the challenger.

K_i are keys, where *i* indicate the owner.

K_S is the session key.

The message sequence for our first shared-key authentication protocol is illustrated in Fig.. In message 1, Alice sends her identity, *A*, to Bob in a way that Bob understands. Bob, of course, has no way of knowing whether this message came from Alice or from Trudy, so he chooses a challenge, a large random number, R_B, and sends it back to "Alice" as message 2, in plaintext. Random numbers used just once in challenge-response protocols like this one are called **nonces**. Alice then encrypts the message with the key she shares with Bob and sends the ciphertext, $K_{AB}(R_B)$, back in message 3. When Bob sees this message, he immediately knows that it came from Alice because Trudy does not know K_{AB} and thus could not have generated it. Furthermore, since R_B was chosen randomly from a large space (say, 128-bit random numbers), it is very unlikely that Trudy would have seen R_B and its response from an earlier session. It is equally unlikely that she could guess the correct response to any challenge.

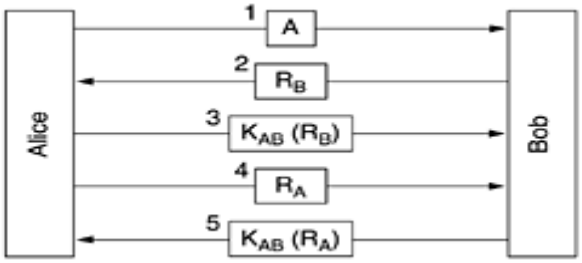


Fig. 5.1. Two-way authentication using a challenge-response protocol.

At this point, Bob is sure he is talking to Alice, but Alice is not sure of anything. For all Alice knows, Trudy might have intercepted message 1 and sent back R_B in response. Maybe Bob died last night. To find out to whom she is talking, Alice picks a random number, R_A and sends it to Bob as plaintext, in message 4. When Bob responds with $K_{AB}(R_A)$, Alice knows she is talking to Bob. If they wish to establish a session key now, Alice can pick one, K_S, and send it to Bob encrypted with K_{AB}.

The protocol of Fig. 5.1contains five messages. Let us see if we can be clever and eliminate some of them. One approach is illustrated in Fig. 5.2. Here Alice initiates the challenge-response protocol instead of waiting for Bob to do it. Similarly, while he is responding to Alice's

challenge, Bob sends his own. The entire protocol can be reduced to three messages instead of five.

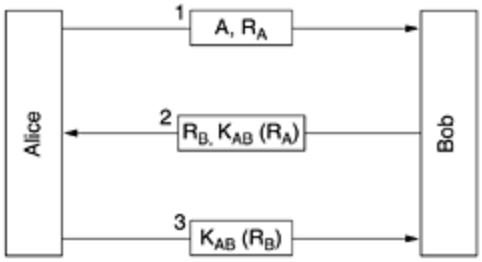


Fig. 5.2. A shortened two-way authentication protocol.

Is this new protocol an improvement over the original one? In one sense it is: it is shorter. Unfortunately, it is also wrong. Under certain circumstances, Trudy can defeat this protocol by using what is known as a **reflection attack**. In particular, Trudy can break it if it is possible to open multiple sessions with Bob at once. This situation would be true, for example, if Bob is a bank and is prepared to accept many simultaneous connections from teller machines at once.

Trudy's reflection attack is shown in Fig. 5.3. It starts out with Trudy claiming she is Alice and sending R_T. Bob responds, as usual, with his own challenge, R_B. Now Trudy is stuck. What can she do? She does not know $K_{AB}(R_B)$.

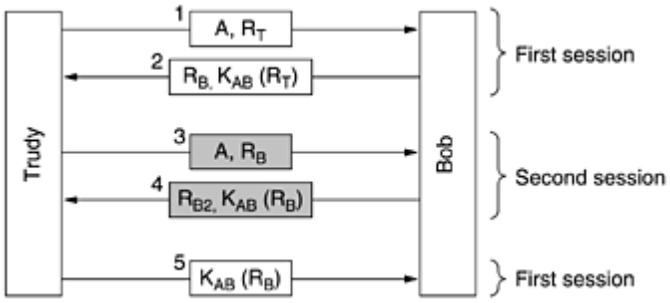


Fig. 5.3. The reflection attack.

She can open a second session with message 3, supplying the R_B taken from message 2 as her challenge. Bob calmly encrypts it and sends back $K_{AB}(R_B)$in message 4. We have shaded the messages on the second session to make them stand out. Now Trudy has the missing information, so she can complete the first session and abort the second one. Bob is now convinced that Trudy is Alice, so when she asks for her bank account balance, he gives it to her without question. Then when she asks him to transfer it all to a secret bank account in Switzerland, he does so without a moment's hesitation.

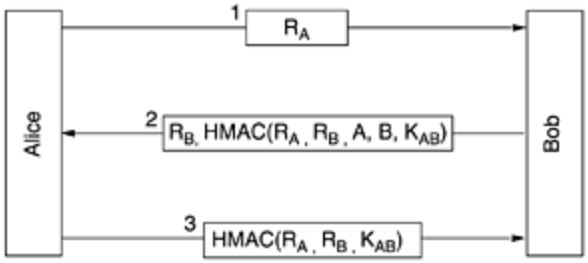


Fig. 5.4. Authentication using HMACs.

Can Trudy somehow subvert this protocol? No, because she cannot force either party to encrypt or hash a value of her choice, as happened and Fo. Both HMACs include values chosen by the sending party, something which Trudy cannot control.

Using HMACs is not the only way to use this idea. An alternative scheme that is often used instead of computing the HMAC over a series of items is to encrypt the items sequentially using cipher block chaining.

5.1.2 Establishing a Shared Key: The Diffie-Hellman Key Exchange

So far we have assumed that Alice and Bob share a secret key. Suppose that they do not (because so far there is no universally accepted PKI for signing and distributing certificates). How can they establish one? One way would be for Alice to call Bob and give him her key on the phone, but he would probably start out by saying: How do I know you are Alice and not Trudy? They could try to arrange a meeting, with each one bringing a passport, a drivers' license, and three major credit cards, but being busy people, they might not be able to find a mutually acceptable date for months. Fortunately, incredible as it may sound, there is a way for total strangers to establish a shared secret key in broad daylight, even with Trudy carefully recording every message.

The protocol that allows strangers to establish a shared secret key is called the **Diffie-Hellman key exchange** (Diffie and Hellman, 1976) and works as follows. Alice and Bob have to agree on two large numbers, n and g, where n is a prime, $(n - 1)/2$ is also a prime and certain conditions apply to g. These numbers may be public, so either one of them can just pick n and g or tell the other openly. Now Alice picks a large (say, 512-bit) number, x, and keeps it secret. Similarly, Bob picks a large secret number, y.

Alice initiates the key exchange protocol by sending Bob a message containing (n, g, g^x mod n), as shown in Fig. 5-1.4. Bob responds by sending Alice a message containing g^y mod n. Now Alice raises the number Bob sent her to the xth power modulo n to get $(g^y \bmod n)^x \bmod n$. Bob performs a similar operation to get $(g^x \bmod n)^y \bmod n$. By the laws of modular arithmetic, both calculations yield $g^{xy} \bmod n$. Lo and behold, Alice and Bob suddenly share a secret key, $g^{xy} \bmod n$.

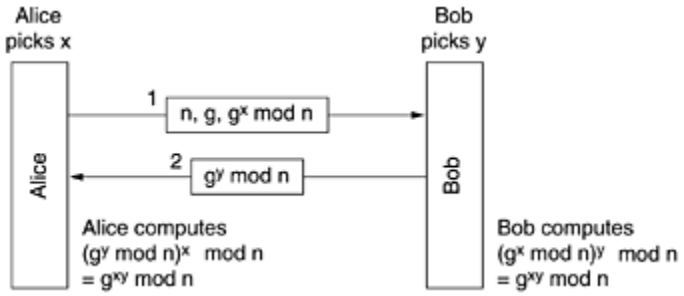


Fig. 5.5. The Diffie-Hellman key exchange.

Trudy, of course, has seen both messages. She knows *g* and *n* from message 1. If she could compute *x* and *y*, she could Fig. out the secret key. The trouble is, given only g^x mod *n*, she cannot find *x*. No practical algorithm for computing discrete logarithms modulo a very large prime number is known.

To make the above example more concrete, we will use the (completely unrealistic) values of *n* = 47 and *g* = 3. Alice picks *x* = 8 and Bob picks *y* = 10. Both of these are kept secret. Alice's message to Bob is (47, 3, 28) because 3^8 mod 47 is 28. Bob's message to Alice is (17). Alice computes 17^8 mod 47, which is 4. Bob computes 28^{10} mod 47, which is 4. Alice and Bob have independently determined that the secret key is now 4. Trudy has to solve the equation 3^x mod 47 = 28, which can be done by exhaustive search for small numbers like this, but not when all the numbers are hundreds of bits long. All currently-known algorithms simply take too long, even on massively parallel supercomputers.

Despite the elegance of the Diffie-Hellman algorithm, there is a problem: when Bob gets the triple (47, 3, 28), how does he know it is from Alice and not from Trudy? There is no way he can know. Unfortunately, Trudy can exploit this fact to deceive both Alice and Bob, as illustrated in Fig. 5.1.5. Here, while Alice and Bob are choosing *x* and *y*, respectively, Trudy picks her own random number, *z*. Alice sends message 1 intended for Bob. Trudy intercepts it and sends message 2 to Bob, using the correct *g* and *n* (which are public anyway) but with her own *z* instead of *x*. She also sends message 3 back to Alice. Later Bob sends message 4 to Alice, which Trudy again intercepts and keeps.

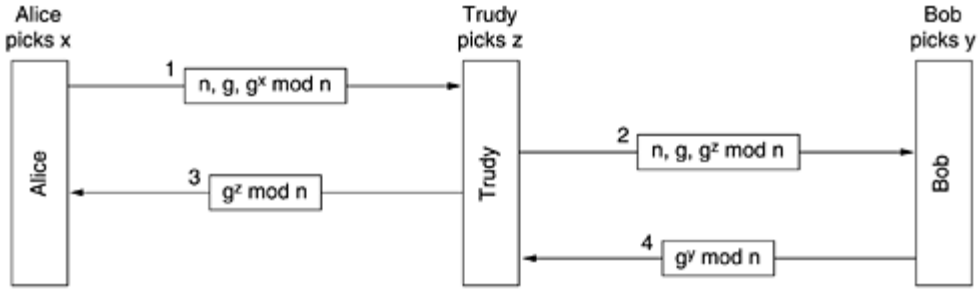


Fig. 5.6. The bucket brigade or man-in-the-middle attack.

Now everybody does the modular arithmetic. Alice computes the secret key as g^{xz} mod *n*, and so does Trudy (for messages to Alice). Bob computes g^{yz} mod *n* and so does Trudy (for messages to Bob). Alice thinks she is talking to Bob so she establishes a session key (with Trudy). So does Bob. Every message that Alice sends on the encrypted session is captured by Trudy, stored, modified if desired, and then (optionally) passed on to Bob. Similarly, in the other direction. Trudy sees everything and can modify all messages at will, while both Alice and Bob

are under the illusion that they have a secure channel to one another. This attack is known as the **bucket brigade attack**, because it vaguely resembles an old-time volunteer fire department passing buckets along the line from the fire truck to the fire. It is also called the **man-in-the-middle attack**.

5.1.3 Authentication Using a Key Distribution Center

Setting up a shared secret with a stranger almost worked, but not quite. On the other hand, it probably was not worth doing in the first place (sour grapes attack). To talk to *n* people this way, you would need *n* keys. For popular people, key management would become a real burden, especially if each key had to be stored on a separate plastic chip card.

A different approach is to introduce a trusted key distribution center (KDC). In this model, each user has a single key shared with the KDC. Authentication and session key management now goes through the KDC. The simplest known KDC authentication protocol involving two parties and a trusted KDC is depicted in Fig. 5.7.

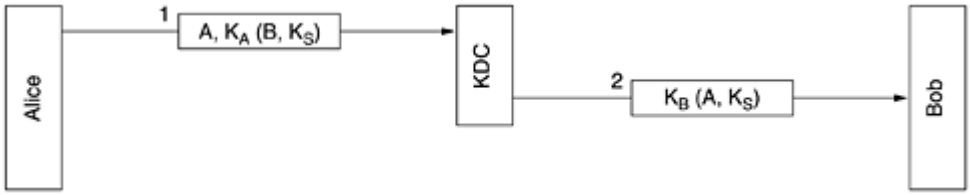


Fig. 5.7. A first attempt at an authentication protocol using a KDC.

The idea behind this protocol is simple: Alice picks a session key, K_S, and tells the KDC that she wants to talk to Bob using K_S. This message is encrypted with the secret key Alice shares (only) with the KDC, K_A. The KDC decrypts this message, extracting Bob's identity and the session key. It then constructs a new message containing Alice's identity and the session key and sends this message to Bob. This encryption is done with K_B, the secret key Bob shares with the KDC. When Bob decrypts the message, he learns that Alice wants to talk to him and which key she wants to use.

The authentication here happens for free. The KDC knows that message 1 must have come from Alice, since no one else would have been able to encrypt it with Alice's secret key. Similarly, Bob knows that message 2 must have come from the KDC, whom he trusts, since no one else knows his secret key.

Unfortunately, this protocol has a serious flaw. Trudy needs some money, so she Fig.s out some legitimate service she can perform for Alice, makes an attractive offer, and gets the job. After doing the work, Trudy then politely requests Alice to pay by bank transfer. Alice then establishes a session key with her banker, Bob. Then she sends Bob a message requesting money to be transferred to Trudy's account.

Meanwhile, Trudy is back to her old ways, snooping on the network. She copies both message 2 in fig_and the money-transfer request that follows it. Later, she replays both of them to Bob. Bob gets them and thinks: Alice must have hired Trudy again. She clearly does good work. Bob then transfers an equal amount of money from Alice's account to Trudy's. Sometime after the 50th message pair, Bob runs out of the office to find Trudy to offer her a big loan so she can expand her obviously successful business. This problem is called the **replay attack**.

Several solutions to the replay attack are possible. The first one is to include a timestamp in each message. Then if anyone receives an obsolete message, it can be discarded. The trouble with this approach is that clocks are never exactly synchronized over a network, so there has to be some interval during which a timestamp is valid. Trudy can replay the message during this interval and get away with it.

The second solution is to put a nonce in each message. Each party then has to remember all previous nonces and reject any message containing a previously-used nonce. But nonces have to be remembered forever, lest Trudy try replaying a 5-year-old message. Also, if some machine crashes and it loses its nonce list, it is again vulnerable to a replay attack. Timestamps and nonces can be combined to limit how long nonces have to be remembered, but clearly the protocol is going to get a lot more complicated.

A more sophisticated approach to mutual authentication is to use a multiway challenge-response protocol. A well-known example of such a protocol is the **Needham-Schroeder authentication** protocol (Needham and Schroeder, 1978), one variant of which is shown in Fig. 5.8.

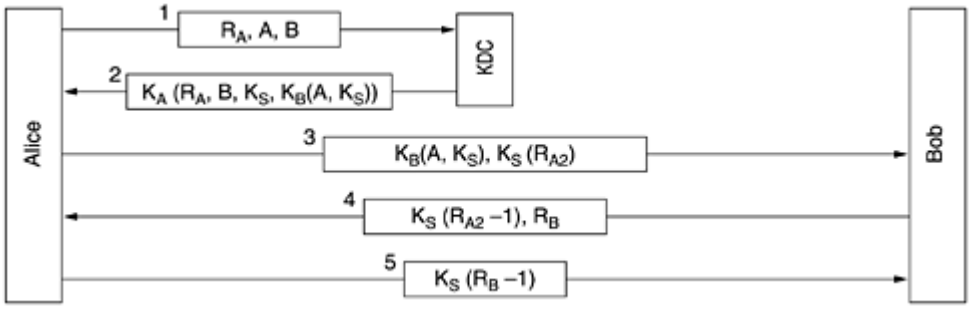


Fig. 5.8. The Needham-Schroeder authentication protocol.

The protocol begins with Alice telling the KDC that she wants to talk to Bob. This message contains a large random number, R_A, as a nonce. The KDC sends back message 2 containing Alice's random number, a session key, and a ticket that she can send to Bob. The point of the random number, R_A, is to assure Alice that message 2 is fresh, and not a replay. Bob's identity is also enclosed in case Trudy gets any funny ideas about replacing *B* in message 1 with her own identity so the KDC will encrypt the ticket at the end of message 2 with K_T instead of K_B. The ticket encrypted with K_B is included inside the encrypted message to prevent Trudy from replacing it with something else on the way back to Alice.

Alice now sends the ticket to Bob, along with a new random number, R_{A2}, encrypted with the session key, K_S. In message 4, Bob sends back $K_S(R_{A2} - 1)$ to prove to Alice that she is talking to the real Bob. Sending back $K_S(R_{A2})$ would not have worked, since Trudy could just have stolen it from message 3.

After receiving message 4, Alice is now convinced that she is talking to Bob and that no replays could have been used so far. After all, she just generated R_{A2} a few milliseconds ago. The purpose of message 5 is to convince Bob that it is indeed Alice he is talking to, and no

replays are being used here either. By having each party both generate a challenge and respond to one, the possibility of any kind of replay attack is eliminated.

Although this protocol seems pretty solid, it does have a slight weakness. If Trudy ever manages to obtain an old session key in plaintext, she can initiate a new session with Bob by replaying the message 3 corresponding to the compromised key and convince him that she is Alice (Denning and Sacco, 1981). This time she can plunder Alice's bank account without having to perform the legitimate service even once.

Needham and Schroeder later published a protocol that corrects this problem (Needham and Schroeder, 1987). In the same issue of the same journal, Otway and Rees (1987) also published a protocol that solves the problem in a shorter way. Fig. 5.9.shows a slightly modified Otway-Rees protocol.

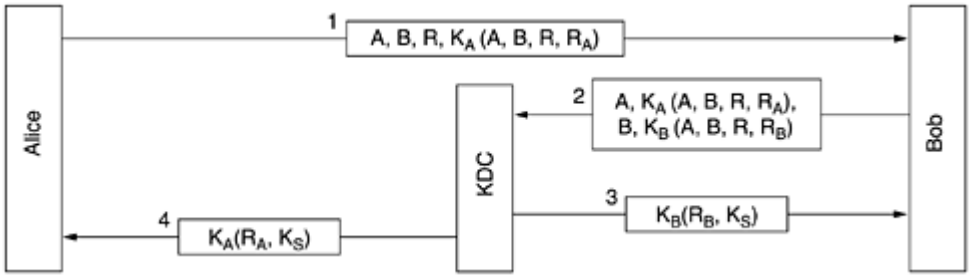


Fig. 5.9. The Otway-Rees authentication protocol (slightly simplified).

In the Otway-Rees protocol, Alice starts out by generating a pair of random numbers, *R*, which will be used as a common identifier, and R_A, which Alice will use to challenge Bob. When Bob gets this message, he constructs a new message from the encrypted part of Alice's message and an analogous one of his own. Both the parts encrypted with K_A and K_B identify Alice and Bob, contain the common identifier, and contain a challenge.

The KDC checks to see if the *R* in both parts is the same. It might not be because Trudy tampered with *R* in message 1 or replaced part of message 2. If the two *R*s match, the KDC believes that the request message from Bob is valid. It then generates a session key and encrypts it twice, once for Alice and once for Bob. Each message contains the receiver's random number, as proof that the KDC, and not Trudy, generated the message. At this point both Alice and Bob are in possession of the same session key and can start communicating. The first time they exchange data messages, each one can see that the other one has an identical copy of K_S, so the authentication is then complete.

5.1.4 Authentication Using Kerberos

An authentication protocol used in many real systems (including Windows 2000) is **Kerberos**, which is based on a variant of Needham-Schroeder. It is named for a multiheaded dog in Greek mythology that used to guard the entrance to Hades (presumably to keep undesirables out). Kerberos was designed at M.I.T. to allow workstation users to access network resources in a secure way. Its biggest difference from Needham-Schroeder is its assumption that all clocks are fairly well synchronized. The protocol has gone through several iterations. V4 is the version most widely used in industry, so we will describe it. Afterward, we will say a few words about its successor, V5. For more information, see (Steiner et al., 1988).

Kerberos involves three servers in addition to Alice (a client workstation):

- Authentication Server (AS): verifies users during login

- Ticket-Granting Server (TGS): issues "proof of identity tickets"
- Bob the server: actually does the work Alice wants performed

AS is similar to a KDC in that it shares a secret password with every user. The TGS's job is to issue tickets that can convince the real servers that the bearer of a TGS ticket really is who he or she claims to be.

To start a session, Alice sits down at an arbitrary public workstation and types her name. The workstation sends her name to the AS in plaintext. What comes back is a session key and a ticket, $K_{TGS}(A, K_S)$, intended for the TGS. These items are packaged together and encrypted using Alice's secret key, so that only Alice can decrypt them. Only when message 2 arrives does the workstation ask for Alice's password. The password is then used to generate K_A in order to decrypt message 2 and obtain the session key and TGS ticket inside it. At this point, the workstation overwrites Alice's password to make sure that it is only inside the workstation for a few milliseconds at most. If Trudy tries logging in as Alice, the password she types will be wrong and the workstation will detect this because the standard part of message 2 will be incorrect.

Authentication Using Public-Key Cryptography

Mutual authentication can also be done using public-key cryptography. To start with, Alice needs to get Bob's public key. If a PKI exists with a directory server that hands out certificates

for public keys, Alice can ask for Bob's, as shown in Fig. 5.10._as message 1. The reply, in message 2, is an X.509 certificate containing Bob's public key. When Alice verifies that the signature is correct, she sends Bob a message containing her identity and a nonce.

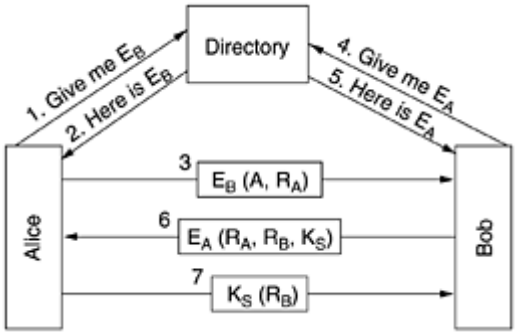


Fig. 5.10. Mutual authentication using public-key cryptography.

When Bob receives this message, he has no idea whether it came from Alice or from Trudy, but he plays along and asks the directory server for Alice's public key (message 4) which he soon gets (message 5). He then sends Alice a message containing Alice's R_A, his own nonce, R_B, and a proposed session key, K_S, as message 6.

When Alice gets message 6, she decrypts it using her private key. She sees R_A in it, which gives her a warm feeling inside. The message must have come from Bob, since Trudy has no way of determining R_A. Furthermore, it must be fresh and not a replay, since she just sent Bob R_A. Alice agrees to the session by sending back message 7. When Bob sees R_B encrypted with the session key he just generated, he knows Alice got message 6 and verified R_A.

What can Trudy do to try to subvert this protocol? She can fabricate message 3 and trick Bob into probing Alice, but Alice will see an R_A that she did not send and will not proceed further. Trudy cannot forge message 7 back to Bob because she does not know R_B or K_S and cannot determine them without Alice's private key. She is out of luck.

5.2. E-Mail Security

When an e-mail message is sent between two distant sites, it will generally transit dozens of machines on the way. Any of these can read and record the message for future use. In practice, privacy is nonexistent, despite what many people think. Nevertheless, many people would like to be able to send e-mail that can be read by the intended recipient and no one else: not their boss and not even their government. This desire has stimulated several people and groups to apply the cryptographic principles we studied earlier to e-mail to produce secure e-mail. In the following sections we will study a widely-used secure e-mail system, PGP, and then briefly mention two others, PEM and S/MIME. For additional information about secure e-mail, see (Kaufman et al., 2002; and Schneier, 1995).

5.2.1. PGP—Pretty Good Privacy

Our first example, **PGP** (**Pretty Good Privacy**) is essentially the brainchild of one person, Phil Zimmermann (Zimmermann, 1995a, 1995b). Zimmermann is a privacy advocate whose motto is: If privacy is outlawed, only outlaws will have privacy. Released in 1991, PGP is a complete e-mail security package that provides privacy, authentication, digital signatures, and compression, all in an easy-to-use form. Furthermore, the complete package, including all the source code, is distributed free of charge via the Internet. Due to its quality, price (zero), and easy availability on UNIX, Linux, Windows, and Mac OS platforms, it is widely used today.

PGP encrypts data by using a block cipher called **IDEA** (**International Data Encryption Algorithm**), which uses 128-bit keys. It was devised in Switzerland at a time when DES was seen as tainted and AES had not yet been invented. Conceptually, IDEA is similar to DES and AES: it mixes up the bits in a series of rounds, but the details of the mixing functions are different from DES and AES. Key management uses RSA and data integrity uses MD5, topics that we have already discussed.

PGP has also been embroiled in controversy since day 1 (Levy, 1993). Because Zimmermann did nothing to stop other people from placing PGP on the Internet, where people all over the world could get it, the U.S. Government claimed that Zimmermann had violated U.S. laws prohibiting the export of munitions. The U.S. Government's investigation of Zimmermann went on for 5 years, but was eventually dropped, probably for two reasons. First, Zimmermann did not place PGP on the Internet himself, so his lawyer claimed that *he* never exported anything (and then there is the little matter of whether creating a Web site constitutes export at all). Second, the government eventually came to realize that winning a trial meant convincing a jury that a Web site containing a downloadable privacy program was covered by the arms-trafficking law prohibiting the export of war materiel such as tanks, submarines, military aircraft, and nuclear weapons. Years of negative publicity probably did not help much, either.

As an aside, the export rules are bizarre, to put it mildly. The government considered putting code on a Web site to be an illegal export and harassed Zimmermann for 5 years about it. On the other hand, when someone published the complete PGP source code, in C, as a book (in a large font with a checksum on each page to make scanning it in easy) and then exported the book, that was fine with the government because books are not classified as munitions. The sword is mightier than the pen, at least for Uncle Sam.

Another problem PGP ran into involved patent infringement. The company holding the RSA patent, RSA Security, Inc., alleged that PGP's use of the RSA algorithm infringed on its patent, but that problem was settled with releases starting at 2.6. Furthermore, PGP uses another patented encryption algorithm, IDEA, whose use caused some problems at first.

Since PGP is open source, various people and groups have modified it and produced a number of versions. Some of these were designed to get around the munitions laws, others were focused on avoiding the use of patented algorithms, and still others wanted to turn it into a closed-source commercial product. Although the munitions laws have now been slightly liberalized (otherwise products using AES would not have been exportable from the U.S.), and the RSA patent expired in September 2000, the legacy of all these problems is that several incompatible versions of PGP are in circulation, under various names. The discussion below focuses on classic PGP, which is the oldest and simplest version. Another popular version, Open PGP, is described in RFC 2440. Yet another is the GNU Privacy Guard.

PGP intentionally uses existing cryptographic algorithms rather than inventing new ones. It is largely based on algorithms that have withstood extensive peer review and were not designed or influenced by any government agency trying to weaken them. For people who tend to distrust government, this property is a big plus.

PGP supports text compression, secrecy, and digital signatures and also provides extensive key management facilities, but oddly enough, not e-mail facilities. It is more of a preprocessor that takes plaintext as input and produces signed ciphertext in base64 as output. This output can then be e-mailed, of course. Some PGP implementations call a user agent as the final step to actually send the message.

To see how PGP works, let us consider the example of Fig. 5.11. Here, Alice wants to send a signed plaintext message, *P*, to Bob in a secure way. Both Alice and Bob have private (D_X) and public (E_X) RSA keys. Let us assume that each one knows the other's public key; we will cover PGP key management shortly.

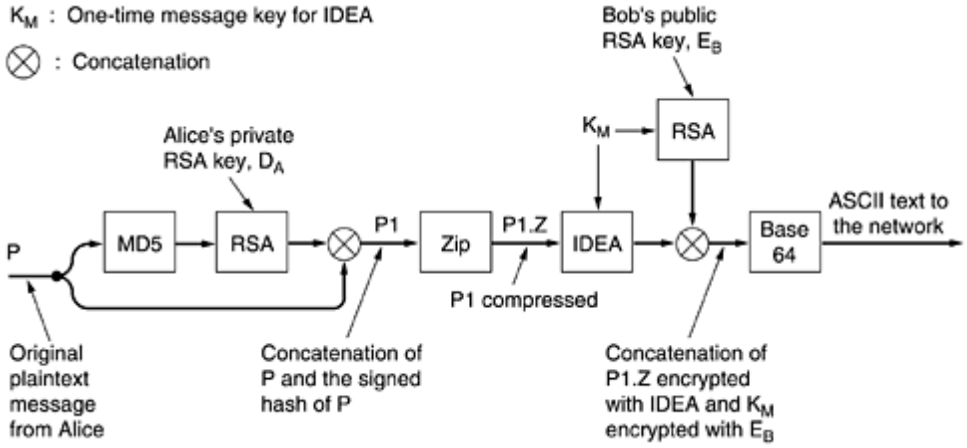


Fig. 5.11. PGP in operation for sending a message.

Alice starts out by invoking the PGP program on her computer. PGP first hashes her message, *P*, using MD5, and then encrypts the resulting hash using her private RSA key, D_A. When Bob eventually gets the message, he can decrypt the hash with Alice's public key and verify that the hash is correct. Even if someone else (e.g., Trudy) could acquire the hash at this stage and decrypt it with Alice's known public key, the strength of MD5 guarantees that it would be computationally infeasible to produce another message with the same MD5 hash.

The encrypted hash and the original message are now concatenated into a single message, *P1*, and compressed using the ZIP program, which uses the Ziv-Lempel algorithm (Ziv and Lempel, 1977). Call the output of this step *P1.Z*.

Next, PGP prompts Alice for some random input. Both the content and the typing speed are used to generate a 128-bit IDEA message key, K_M (called a session key in the PGP literature, but this is really a misnomer since there is no session). K_M is now used to encrypt *P1.Z* with IDEA in cipher feedback mode. In addition, K_M is encrypted with Bob's public key, E_B. These two components are then concatenated and converted to base64, as we discussed in the section on MIME . The resulting message then contains only letters, digits, and the symbols +, /, and =, which means it can be put into an RFC 822 body and be expected to arrive unmodified.

When Bob gets the message, he reverses the base64 encoding and decrypts the IDEA key using his private RSA key. Using this key, he decrypts the message to get *P1.Z*. After decompressing it, Bob separates the plaintext from the encrypted hash and decrypts the hash using Alice's public key. If the plaintext hash agrees with his own MD5 computation, he knows that *P* is the correct message and that it came from Alice.

It is worth noting that RSA is only used in two places here: to encrypt the 128-bit MD5 hash and to encrypt the 128-bit IDEA key. Although RSA is slow, it has to encrypt only 256 bits, not a large volume of data. Furthermore, all 256 plaintext bits are exceedingly random, so a considerable amount of work will be required on Trudy's part just to determine if a guessed key is correct. The heavy duty encryption is done by IDEA, which are orders of magnitude faster than RSA. Thus, PGP provides security, compression, and a digital signature and does so in a much more efficient way than the previous scheme.

PGP supports four RSA key lengths. It is up to the user to select the one that is most appropriate. The lengths are

1. Casual (384 bits): can be broken easily today.
2. Commercial (512 bits): breakable by three-letter organizations.
3. Military (1024 bits): Not breakable by anyone on earth.
4. Alien (2048 bits): Not breakable by anyone on other planets, either.

Since RSA is only used for two small computations, everyone should use alien strength keys all the time.

The format of a classic PGP message is shown in Fig.5.12. Numerous other formats are also in use. The message has three parts, containing the IDEA key, the signature, and the message, respectively. The key part contains not only the key, but also a key identifier, since users are permitted to have multiple public keys.

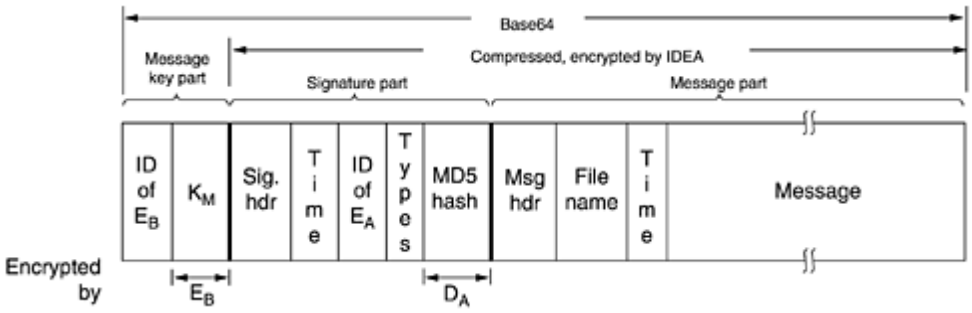


Fig. 5.12. A PGP message.

The signature part contains a header, which will not concern us here. The header is followed by a timestamp, the identifier for the sender's public key that can be used to decrypt the signature hash, some type information that identifies the algorithms used (to allow MD6 and RSA2 to be used when they are invented), and the encrypted hash itself.

The message part also contains a header, the default name of the file to be used if the receiver writes the file to the disk, a message creation timestamp, and, finally, the message itself.

Key management has received a large amount of attention in PGP as it is the Achilles heel of all security systems. Key management works as follows. Each user maintains two data structures locally: a private key ring and a public key ring. The **private key ring** contains one or more personal private-public key pairs. The reason for supporting multiple pairs per user is to permit users to change their public keys periodically or when one is thought to have been compromised, without invalidating messages currently in preparation or in transit. Each pair has an identifier associated with it so that a message sender can tell the recipient which public key was used to encrypt it. Message identifiers consist of the low-order 64 bits of the public key. Users are responsible for avoiding conflicts in their public key identifiers. The private keys on disk are encrypted using a special (arbitrarily long) password to protect them against sneak attacks.

The **public key ring** contains public keys of the user's correspondents. These are needed to encrypt the message keys associated with each message. Each entry on the public key ring contains not only the public key, but also its 64-bit identifier and an indication of how strongly the user trusts the key.

The problem being tackled here is the following. Suppose that public keys are maintained on bulletin boards. One way for Trudy to read Bob's secret e-mail is to attack the bulletin board and replace Bob's public key with one of her choice. When Alice later fetches the key allegedly belonging to Bob, Trudy can mount a bucket brigade attack on Bob.

To prevent such attacks, or at least minimize the consequences of them, Alice needs to know how much to trust the item called "Bob's key" on her public key ring. If she knows that Bob personally handed her a floppy disk containing the key, she can set the trust value to the highest value. It is this decentralized, user-controlled approach to public-key management that sets PGP apart from centralized PKI schemes.

Nevertheless, people do sometimes obtain public keys by querying a trusted key server. For this reason, after X.509 was standardized, PGP supported these certificates as well as the traditional PGP public key ring mechanism. All current versions of PGP have X.509 support.

5.2.2 PEM—Privacy Enhanced Mail

In contrast to PGP, which was initially a one-man show, our second example, **PEM** (**Privacy Enhanced Mail**), developed in the late 1980s, is an official Internet standard and described in four RFCs: RFC 1421 through RFC 1424. Very roughly, PEM covers the same territory as PGP: privacy and authentication for RFC 822-based e-mail systems. Nevertheless, it also has some differences from PGP in approach and technology.

Messages sent using PEM are first converted to a canonical form so they all have the same conventions about white space (e.g., tabs, trailing spaces). Next, a message hash is computed using MD2 or MD5. Then the concatenation of the hash and the message is encrypted using DES. In light of the known weakness of a 56-bit key, this choice is certainly suspect. The encrypted message can then be encoded with base64 coding and transmitted to the recipient.

As in PGP, each message is encrypted with a one-time key that is enclosed along with the message. The key can be protected either with RSA or with triple DES using EDE.

Key management is more structured than in PGP. Keys are certified by X.509 certificates issued by CAs, which are arranged in a rigid hierarchy starting at a single root. The advantage of this scheme is that certificate revocation is possible by having the root issue CRLs periodically.

The only problem with PEM is that nobody ever used it and it has long-since gone to that big bit bin in the sky. The problem was largely political: who would operate the root and under what conditions? There was no shortage of candidates, but many people were afraid to trust anyone company with the security of the whole system. The most serious candidate, RSA Security, Inc., wanted to charge per certificate issued. However, some organizations balked at this idea. In particular, the U.S. Government is allowed to use all U.S. patents for free, and companies outside the U.S. had become accustomed to using the RSA algorithm for free (the company forgot to patent it outside the U.S.). Neither was enthusiastic about suddenly having to pay RSA Security, Inc., for doing something that they had always done for free. In the end, no root could be found and PEM collapsed.

5.2.3. S/MIME

IETF's next venture into e-mail security, called **S/MIME** (**Secure/MIME**), is described in RFCs 2632 through 2643. Like PEM, it provides authentication, data integrity, secrecy, and nonrepudiation. It also is quite flexible, supporting a variety of cryptographic algorithms. Not surprisingly, given the name, S/MIME integrates well with MIME, allowing all kinds of messages to be protected. A variety of new MIME headers are defined, for example, for holding digital signatures.

IETF definitely learned something from the PEM experience. S/MIME does not have a rigid certificate hierarchy beginning at a single root. Instead, users can have multiple trust anchors. As long as a certificate can be traced back to some trust anchor the user believes in, it is considered valid. S/MIME uses the standard algorithms and protocols we have been examining so far, so we will not discuss it any further here. For the details, please consult the RFCs.

5.3. Web Security

We have just studied two important areas where security is needed: communications and e-mail. You can think of these as the soup and appetizer. Now it is time for the main course: Web security. The Web is where most of the Trudies hang out nowadays and do their dirty work. In the following sections we will look at some of the problems and issues relating to Web security.

Web security can be roughly divided into three parts. First, how are objects and resources named securely? Second, how can secure, authenticated connections be established? Third, what happens when a Web site sends a client a piece of executable code? After looking at some threats, we will examine all these issues.

5.3.1 Threats

One reads about Web site security problems in the newspaper almost weekly. The situation is really pretty grim. Let us look at a few examples of what has already happened. First, the home page of numerous organizations has been attacked and replaced by a new home page of the crackers' choosing. (The popular press calls people who break into computers "hackers," but many programmers reserve that term for great programmers. We prefer to call these people

"crackers.") Sites that have been cracked include Yahoo, the U.S. Army, the CIA, NASA, and the New York Times. In most cases, the crackers just put up some funny text and the sites were repaired within a few hours.

Now let us look at some much more serious cases. Numerous sites have been brought down by denial-of-service attacks, in which the cracker floods the site with traffic, rendering it unable to respond to legitimate queries. Often the attack is mounted from a large number of machines that the cracker has already broken into (DDoS atacks). These attacks are so common that they do not even make the news any more, but they can cost the attacked site thousands of dollars in lost business.

In 1999, a Swedish cracker broke into Microsoft's Hotmail Web site and created a mirror site that allowed anyone to type in the name of a Hotmail user and then read all of the person's current and archived e-mail.

In another case, a 19-year-old Russian cracker named Maxim broke into an e-commerce Web site and stole 300,000 credit card numbers. Then he approached the site owners and told them that if they did not pay him $100,000, he would post all the credit card numbers to the Internet. They did not give in to his blackmail, and he indeed posted the credit card numbers, inflicting great damage to many innocent victims.

In a different vein, a 23-year-old California student e-mailed a press release to a news agency falsely stating that the Emulex Corporation was going to post a large quarterly loss and that the C.E.O. was resigning immediately. Within hours, the company's stock dropped by 60%, causing stockholders to lose over $2 billion. The perpetrator made a quarter of a million dollars by selling the stock short just before sending the announcement. While this event was not a Web site break-in, it is clear that putting such an announcement on the home page of any big corporation would have a similar effect.

We could (unfortunately) go on like this for many pages. But it is now time to examine some of the technical issues related to Web security. For more information about security problems of all kinds, see (Anderson, 2001; Garfinkel with Spafford, 2002; and Schneier, 2000). Searching the Internet will also turn up vast numbers of specific cases.

5.4. Social Issues

The Internet and its security technology is an area where social issues, public policy, and technology meet head on, often with huge consequences. Below we will just briefly examine three areas: privacy, freedom of speech, and copyright. Needless to say, we can only scratch the surface here. For additional reading, see (Anderson, 2001; Garfinkel with Spafford, 2002; and Schneier, 2000). The Internet is also full of material. Just type words such as "privacy," "censorship," and "copyright" into any search engine. Also, see this book's Web site for some links.

5.4.1 Privacy

Do people have a right to privacy? Good question. The Fourth Amendment to the U.S. Constitution prohibits the government from searching people's houses, papers, and effects without good reason, and goes on to restrict the circumstances under which search warrants shall be issued. Thus, privacy has been on the public agenda for over 200 years, at least in the U.S.

What have changed in the past decade is both the ease with which governments can spy on their citizens and the ease with which the citizens can prevent such spying. In the 18th century, for the government to search a citizen's papers, it had to send out a policeman on a horse to go to the citizen's farm demanding to see certain documents. It was a cumbersome procedure.

Nowadays, telephone companies and Internet providers readily provide wiretaps when presented with search warrants. It makes life much easier for the policeman and there is no danger of falling off the horse.

Cryptography changes all that. Anybody who goes to the trouble of downloading and installing PGP and who uses a well-guarded alien-strength key can be fairly sure that nobody in the known universe can read his e-mail, search warrant or no search warrant. Governments well understand this and do not like it. Real privacy means it is much harder for them to spy on criminals of all stripes, but it is also much harder to spy on journalists and political opponents. Consequently, some governments restrict or forbid the use or export of cryptography. In France, for example, prior to 1999, all cryptography was banned unless the government was given the keys. France was not alone. In April 1993, the U.S. Government announced its intention to make a hardware crypto processor, the **clipper chip**, the standard for all networked communication. In this way, it was said; citizens' privacy would be guaranteed. It also mentioned that the chip provided the government with the ability to decrypt all traffic via a scheme called **key escrow**, which allowed the government access to all the keys. However, it promised only to snoop when it had a valid search warrant. Needless to say, a huge furor ensued, with privacy advocates denouncing the whole plan and law enforcement officials praising it. Eventually, the government backed down and dropped the idea.

Anonymous Remailers

PGP, SSL, and other technologies make it possible for two parties to establish secure, authenticated communication, free from third-party surveillance and interference. However, sometimes privacy is best served by *not* having authentication, in fact by making communication anonymous. The anonymity may be desired for point-to-point messages, newsgroups, or both.

Let us consider some examples. First, political dissidents living under authoritarian regimes often wish to communicate anonymously to escape being jailed or killed. Second, wrongdoing in many corporate, educational, governmental, and other organizations has often been exposed by whistleblowers, who frequently prefer to remain anonymously to avoid retribution. Third, people with unpopular social, political, or religious views may wish to communicate with each other via e-mail or newsgroups without exposing themselves. Fourth, people may wish to discuss alcoholism, mental illness, sexual harassment, child abuse, or being a member of a persecuted minority in a newsgroup without having to go public. Numerous other examples exist, of course.

Let us consider a specific example. In the 1990s, some critics of a nontraditional religious group posted their views to a USENET newsgroup via an **anonymous remailer**. This server allowed users to create pseudonyms and send e-mail to the server, which then re-mailed or re-posted them using the pseudonym, so no one could tell where the message really came from. Some postings revealed what the religious group claimed were trade secrets and copyrighted documents. The religious group responded by telling local authorities that its trade secrets had been disclosed and its copyright infringed, both of which were crimes where the server was located. A court case followed and the server operator was compelled to turn over the mapping information which revealed the true identities of the persons who had made the postings. (Incidentally, this was not the first time that a religion was unhappy when someone leaked its secrets: William Tyndale was burned at the stake in 1536 for translating the Bible into English).

A substantial segment of the Internet community was outraged by this breach of confidentiality. The conclusion that everyone drew is that an anonymous remailer that stores a mapping between real e-mail addresses and pseudonyms (called a type 1 remailer) is not worth much. This case stimulated various people into designing anonymous remailers that could withstand subpoena attacks.

These new remailers often called **cypherpunk remailers**, work as follows. The user produces an e-mail message, complete with RFC 822 headers (except *From:*, of course), encrypts it with the remailer's public key, and sends it to the remailer. There the outer RFC 822 headers are stripped off, the content is decrypted and the message is remailed. The remailer has no accounts and maintains no logs, so even if the server is later confiscated, it retains no trace of messages that have passed through it.

Many users who wish anonymity chain their requests through multiple anonymous remailers, as shown in Fig.5.13. Here, Alice wants to send Bob a really, really, really anonymous Valentine's Day card, so she uses three remailers. She composes the message, *M*, and puts a header on it containing Bob's e-mail address. Then she encrypts the whole thing with remailer 3's public key, E_3. (Indicated by horizontal hatching). To this she prepends a header with remailer 3's e-mail address in plaintext. This is the message shown between remailers 2 and 3 in the Fig..

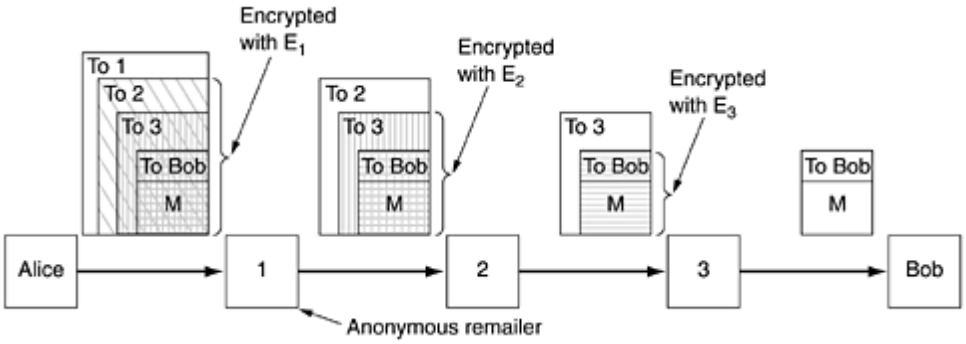


Fig.5.13. How Alice uses 3 remailers to send Bob a message.

Then she encrypts this message with remailer 2's public key, E_2 (indicated by vertical hatching) and prepends a plaintext header containing remailer 2's e-mail address. This message is shown between 1 and 2 in Fig.5.4.1. Finally, she encrypts the entire message with remailer 1's public key, E_1, and prepends a plaintext header with remailer 1's e-mail address. This is the message shown to the right of Alice in the Fig. and this is the message she actually transmits.

When the message hits remailer 1, the outer header is stripped off. The body is decrypted and then e-mailed to remailer 2. Similar steps occur at the other two remailers.

Although it is extremely difficult for anyone to trace the final message back to Alice, many remailers take additional safety precautions. For example, they may hold messages for a random time, add or remove junk at the end of a message, and reorder messages, all to make it harder for anyone to tell which message output by a remailer corresponds to which input, in

order to thwart traffic analysis. For a description of a system that represents the state of the art in anonymous e-mail, see (Mazières and Kaashoek, 1998).

Anonymity is not restricted to e-mail. Services also exist that allow anonymous Web surfing. The user conFig.s his browser to use the anonymizer as a proxy. Henceforth, all HTTP requests go to the anonymizer, which requests the page and sends it back. The Web site sees the anonymizer as the source of the request, not the user. As long as the anonymizer refrains from keeping a log, after the fact no one can determine who requested which page.

Steganography

In countries where censorship abounds, dissidents often try to use technology to evade it. Cryptography allows secret messages to be sent (although possibly not lawfully), but if the government thinks that Alice is a Bad Person, the mere fact that she is communicating with Bob may get him put in this category, too, as repressive governments understand the concept of transitive closure, even if they are short on mathematicians. Anonymous remailers can help, but if they are banned domestically and messages to foreign ones require a government export license, they cannot help much. But the Web can.

People who want to communicate secretly often try to hide the fact that any communication at all is taking place. The science of hiding messages is called **steganography**, from the Greek words for "covered writing." In fact, the ancient Greeks used it themselves. Herodotus wrote of a general who shaved the head of a messenger, tattooed a message to his scalp, and let the hair grow back before sending him off. Modern techniques are conceptually the same, only they have a higher bandwidth and lower latency.

5.5. Specification & Description Language (SDL)

- SDL is object-oriented, formal language developed and standardized by ITU-T.
- Intended for specification of complex, event-driven, real time interactive application involving many concurrent activities that communicate using discrete signals.
- Specification language specifies the communication protocols either by using formal or graphical notation or both. Specification language is like any programming language which follows syntax and semantics.
- SDL has been applied to system analysis and design in many application domains.
- SDL uses FSMs and its extensions for specification
- Graphical representation to specify behavior of protocols.

Salient Features of SDL

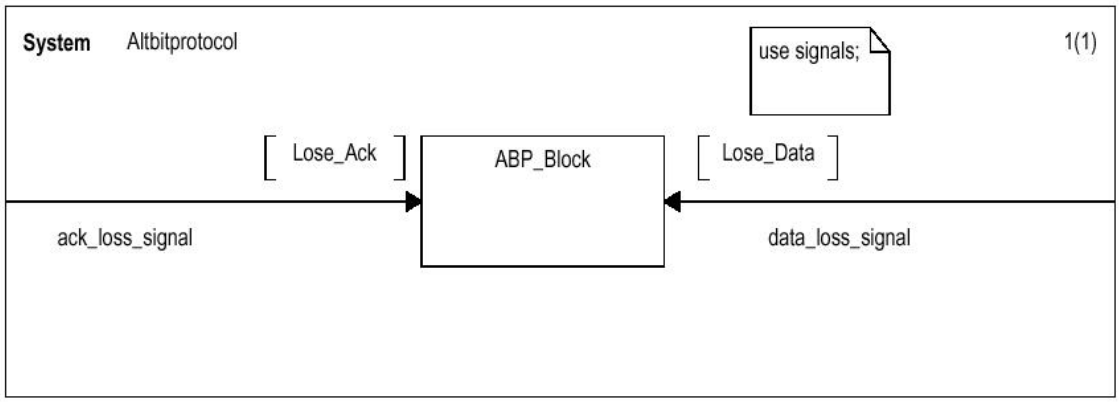
- Well defined set of concepts.
- Unambiguous, clear, precise, and concise specifications
- Thorough basis for analyzing specifications and conformance testing.
- Basis for determining the consistency of specifications.
- Good computer support interface for generating applications without the need for the traditional coding phase.

- High degree of testability as a result of its formalism for parallelism, interfaces, communication and time.
- portability, scalability and open specification
- High degree of reuse because of visual clarity, object oriented concepts, clear interfaces, and abstraction mechanisms.
- facility for applying optimization techniques improving protocol

SDL Based Protocol Verification

• Create the FSMs of all the entities of the protocol.
● Translate the FSMs into the SDL based specification (like system, block and process diagrams).
● Run the simulation and check that all the processes are running in the given specifications.
● Input the required signals to check for safety and liveness properties of the specified protocol.
● Generate the MSC diagrams using the SDL tool.
● Check the MSC diagrams of the test-cases to verify the safety and liveness properties.

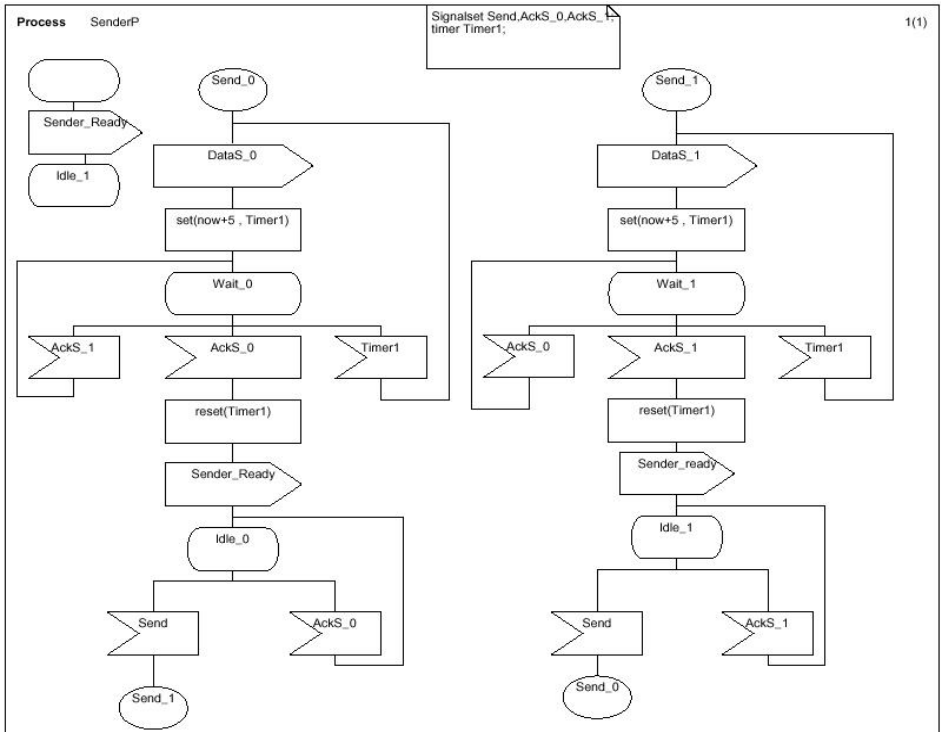
Alternating Bit Protocol



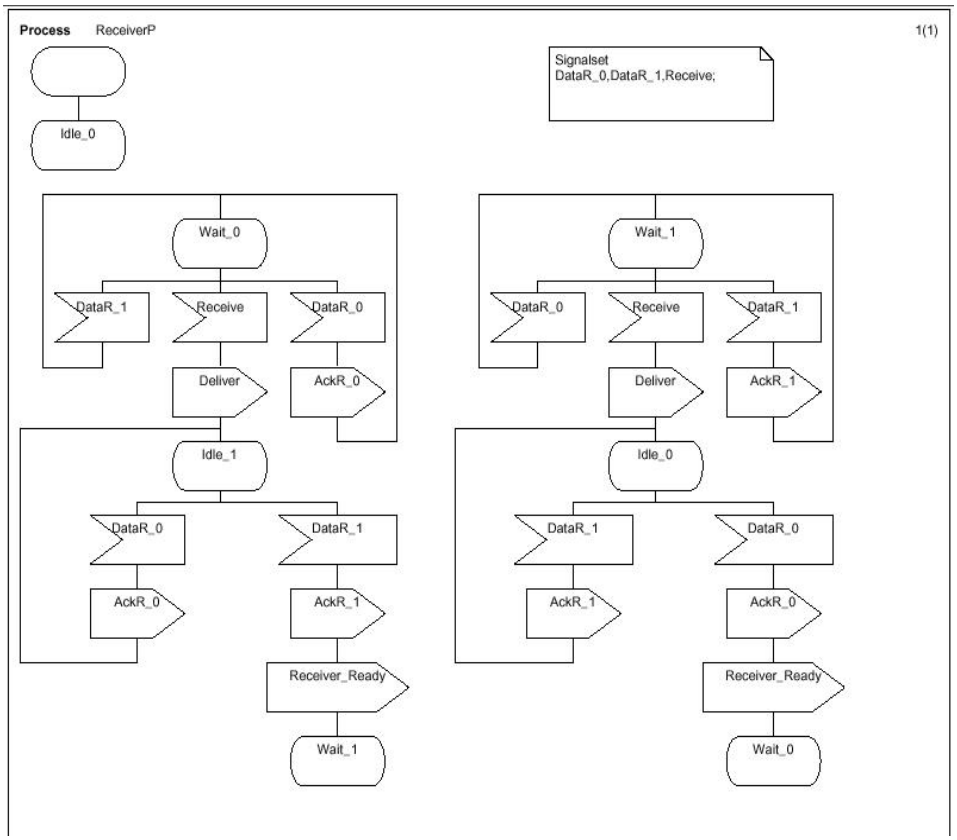
a)

Alternating Bit Protocol

• The block consists of the following processes
• UL sender P: upper layer sender process
• UL Receiver: upper layer receiver process
• Data Medium: provides service to transmit data from sender to receiver
• Sender P: sender process to transmit the frames
• Receiver P: receiver process to receive the data and deliver to upper layer process as well as acknowledge the received data.
• Ack Medium: provides service to transmit the acknowledgments
Received from the receiver to sender.



Reciever Process of ABP



Verification of the ABP

Safety properties:

- To verify the ABP sends the packet with correct sequence number to the receiver, even if the medium loses the data frames.
- We Send lose_data signal to the data medium. When sender process sends data_0 packet, the data medium does not forwards it to the receiver but sender times out ultimately and retransmits the packet and reaches the destination.
- To verify that ABP sends the ack signal with correct sequence number to sender, even if the medium loses the ack, we send lose_ack signal to ack_medium. sender sends data_0, receiver send ack_0 to ack medium which loses ack making sender to timeout and resend data_0, receiver retxm. ack_0 which reaches destination.

Protocol Validation

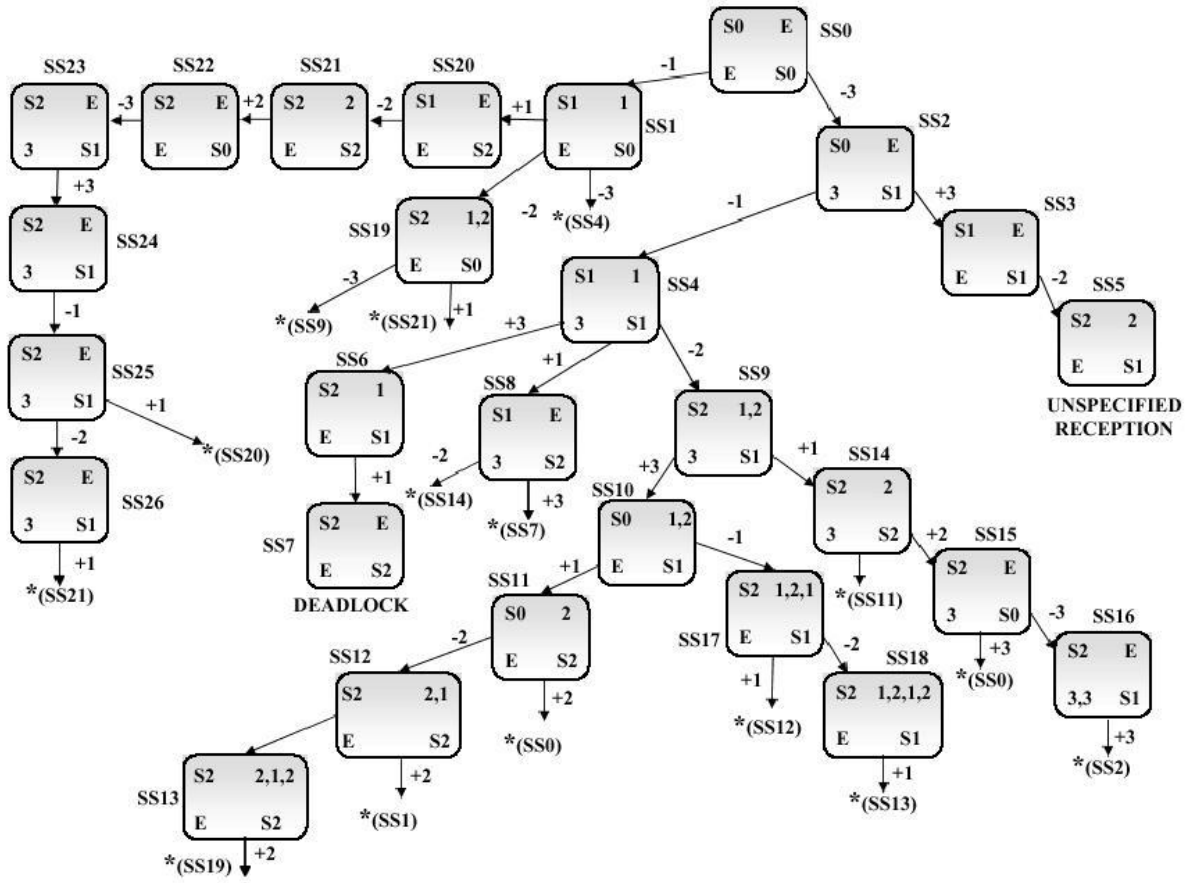
● Definition:– **Protocol validation** is a method of checking whether the interactions of protocol entities are according to the protocol specification,

– do indeed satisfy certain properties or conditions which may be either general or specific to the particular protocol system directly derived from the specifications.

● Validation sometimes refers to check the protocol specification such that it will not get into protocol design errors like deadlock, unspecified receptions, and live lock errors.

Protocol Validation Approaches

● Perturbation Technique: Reachability Analysis



Reachability Analysis

– consider global stateSS4inwhich:
● P1is inS1state
● P2is inS1state
● Channel P12 is containingmessage1, and
● Channel P21 is containing message3. From this the following transitions are possible:
● P1canreceivethemessage3andtransit to global stateS6,
● P2canreceivethemessage1andtransit toSS8,
● P1 can transmit a message2 and reachSS9.

Pros &Cons of Reachability Analysis

– The advantages of using the reachability tree (sometimes called as graphs) for protocol validation are:
● The tree generation can be easily automated
● Many logical errors can be detected by only examining individual global states in the reachability graph.
– The disadvantages of using reachability graphs are:
● State space explosion problem;
● Does not work on unbounded protocols; and
● Many relationships among the protocol state variables, expressing the desirable logical correctness properties of the protocol, are not apparent from simply traversing the reachability tree.

Fair Reachability Graphs

Sequence Number	Transitions					
1	S12	R21	S12	R21	S12	R21
2	S12	R21	R21	R21	S12	R21
3	S12	R21	S12	R21	R21	R21
4	S12	S12	R21	S12	R21	R21
5	S12	S12	S12	R21	R21	R21

Fair Reachability Graphs

● Consider simple message exchange protocol where Process P1(sender) sendes3 messages toprocessP2(receiver) via a queue that has a maximum capacity of 3 messages.
● The possible sequences are shown in the Fig., have the common property that both process execute the same sequence of interaction steps, that isP1sends3 messages and P2receives 3 messages.
● To Illustrate sequence, consider sequence5and1.Sequence5canonlybe execute dif at least 3messagesareallowedinthequeue. similarly sequence1 requires only single queued message(transitionS12) , ie reception is followed by every transmission(transitionsS12,R21,S12,R21,S12,R21).
● the sequence is treated as a series of executing steps, where each sequence consists of a protocol process sending or receiving message.
● assumed that any transition starts and ends in a stable global system states.
● This restriction requires that protocol execution periodically results in a global state in which all transmitted messages have been received.

5.6. Internet Protocol

•In OSI reference model terminology - the IP protocol covers the network layer. •IP can be used on many data-link layers (can support many network hardware implementations).

Internet Protocol

The IP in UDP/IP and TCP/IP

•IP is the network layer • packet delivery service (host-to-host). •translation between different data-link protocols.

IP Datagrams

•IP provides connectionless, unreliable delivery of IP datagram. • Connectionless: each datagram is independent of all others. • Unreliable: there is no guarantee that datagrams are delivered correctly or at all

IP Addresses

•IP addresses are not the same as the underlying data-link (MAC) addresses.
IP is a network layer - it must be capable of providing communication between hosts on different kinds of networks (different data-link implementations). • The address must include information about what network the receiving host is on. This makes routing feasible.

IP addresses are logical addresses (not physical) • 32 bits. • Includes a network ID and a host ID. • Every host must have a unique IP address. • IP addresses are assigned by a central authority (Internet Corporation for Assigned Names and Numbers -- ICANN)

The four formats of IP Addresses

Class A

128 possible network IDs

over 4 million host IDs per network ID

128 possible network IDs

over 4 million host IDs per network ID

Class B

16K possible network IDs 64K host IDs per network ID

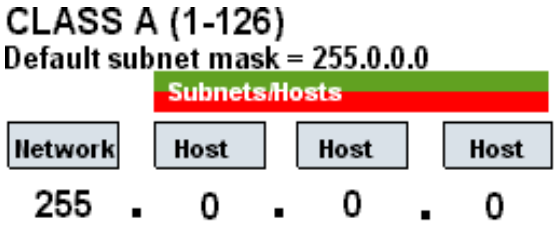
16K possible network IDs

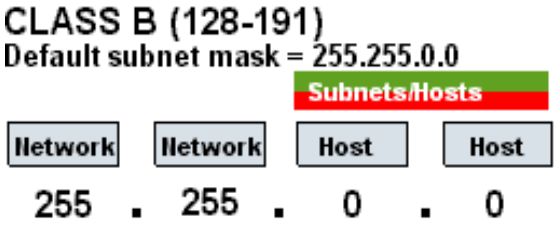
64K host IDs per network ID

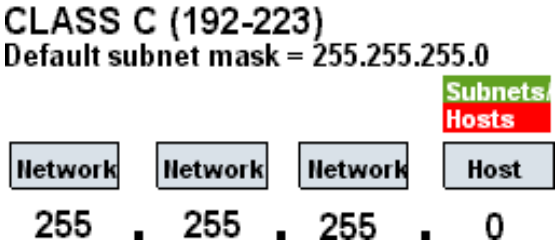
Class C

over 2 million possible network IDs

about 256 host IDs per network ID







Network and Host IDs

• A Network ID is assigned to an organization by a global authority. • Host IDs are assigned locally by a system administrator. • Both the Network ID and the Host ID are used for routing.

IP Addresses

•IP Addresses are usually shown in dotted decimal notation:

1.2.3.4 00000001 00000010 00000011 00000100

• cs.rpi.edu is 128.213.1.1

10000000 11010101 00000001 00000001

CS has a class B network

Host and Network Addresses

• A single network interface is assigned a single IP address called the host address. • A host may have multiple interfaces, and therefore multiple host addresses. • Hosts that share a network all have the same IP network address (the network ID).

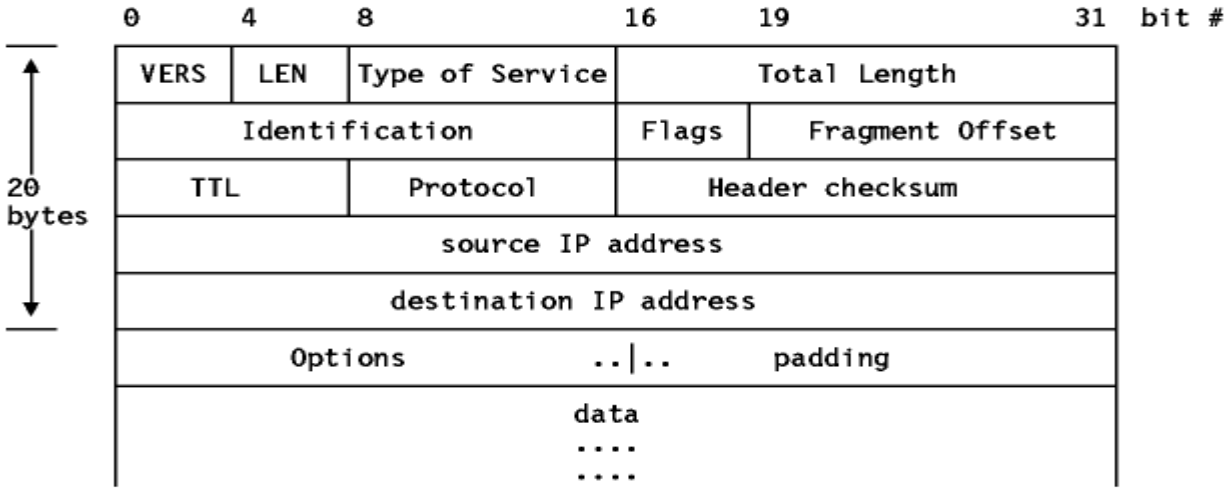
IP Broadcast and Network Addresses

• An IP broadcast addresses has a host ID of all 1s. •IP broadcasting is not necessarily a true broadcast; it relies on the underlying hardware technology. • An IP address that has a host ID of all 0s is called a network address and refers to an entire network.

Subnet Addresses

• An organization can subdivide its host address space into groups called subnets. • The subnet ID is generally used to group hosts based on the physical network topology.

IP Datagram



IP Datagram Fragmentation

• Each fragment (packet) has the same structure as the IP datagram.

• IP specifies that datagram reassembly is done only at the destination (not on a hop-by-hop basis).

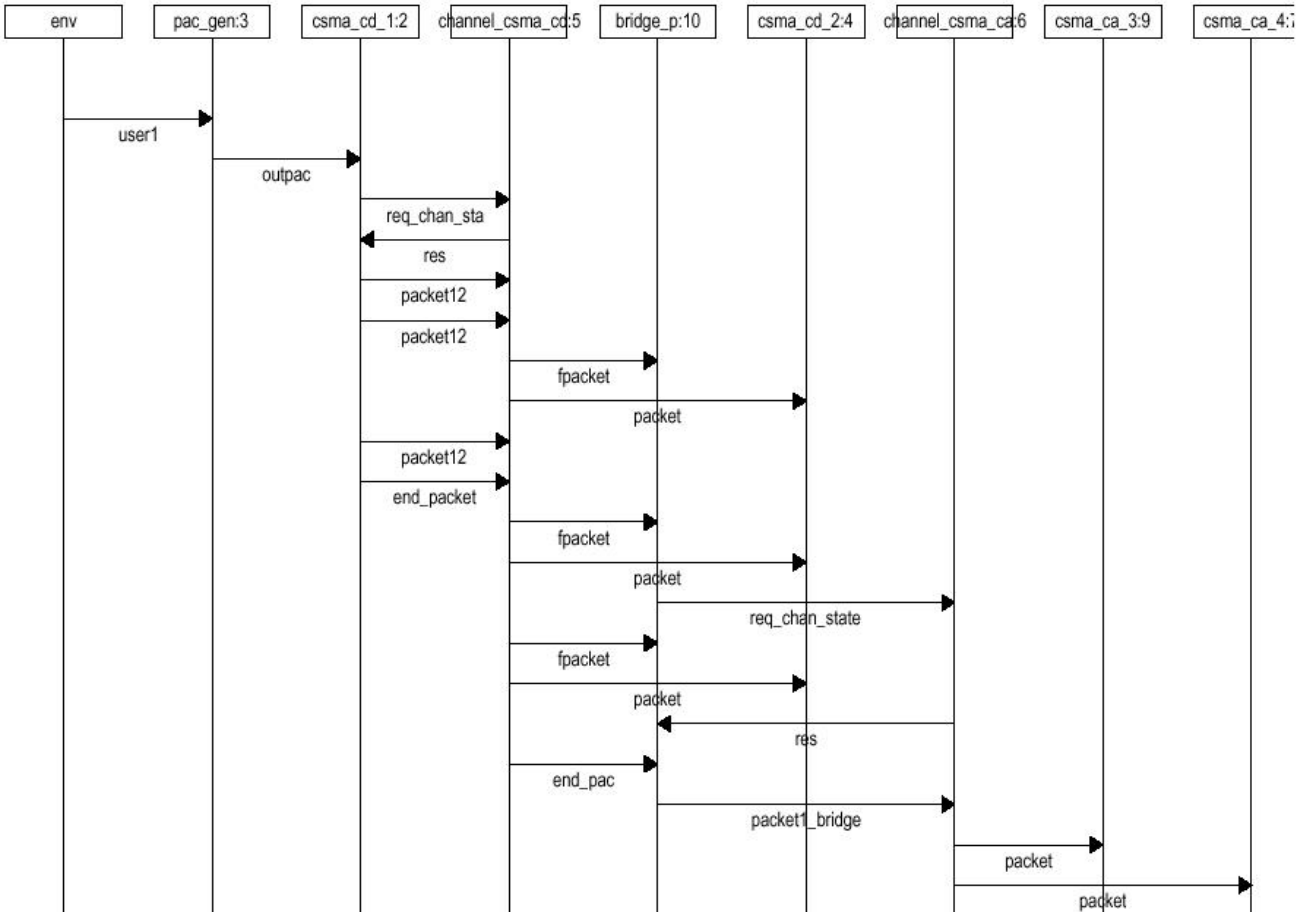
• If any of the fragments are lost – the entire datagram is discarded (and an ICMP message is sent to the sender).

IP Flow Control & Error
Detection
• If packets arrive too fast – the receiver discards excessive packets and sends an ICMP message to the sender (SOURCE QUENCH).
• If an error is found (header checksum problem) the packet is discarded and an ICMP message is sent to the sender

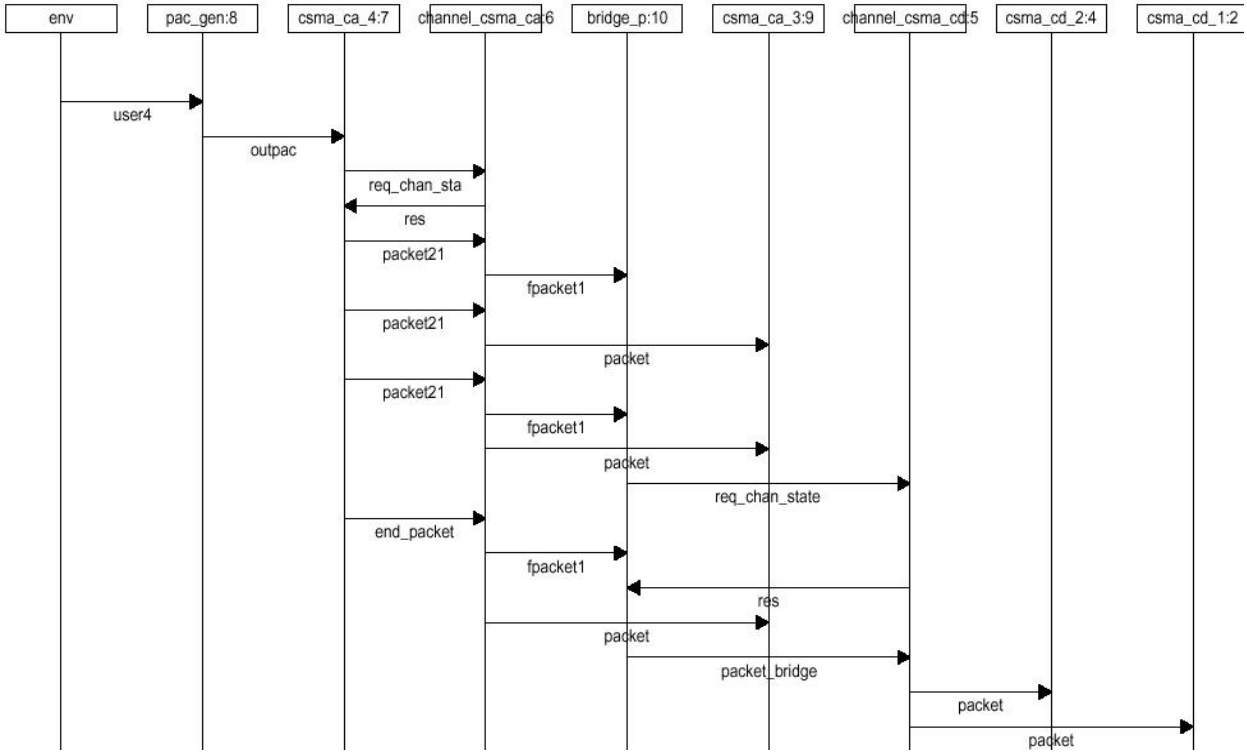
5.7. SDL based interoperability testing of CSMA/CD and CSMA/CA protocol using bridge

● The procedure for interoperability testing by using SDL is as follows. – Create FSMs for the protocols and their interoperations using a bridge. – Create SDL diagrams of the protocols and the bridge operations. – Run the system by giving the inputs (test sequence) from the environment such as source, destination and number of packets. – Create the MSCs for the different kinds of inputs. – Observe from MSCs interworking of CSMA/CA and CSMA/CD using a bridge.

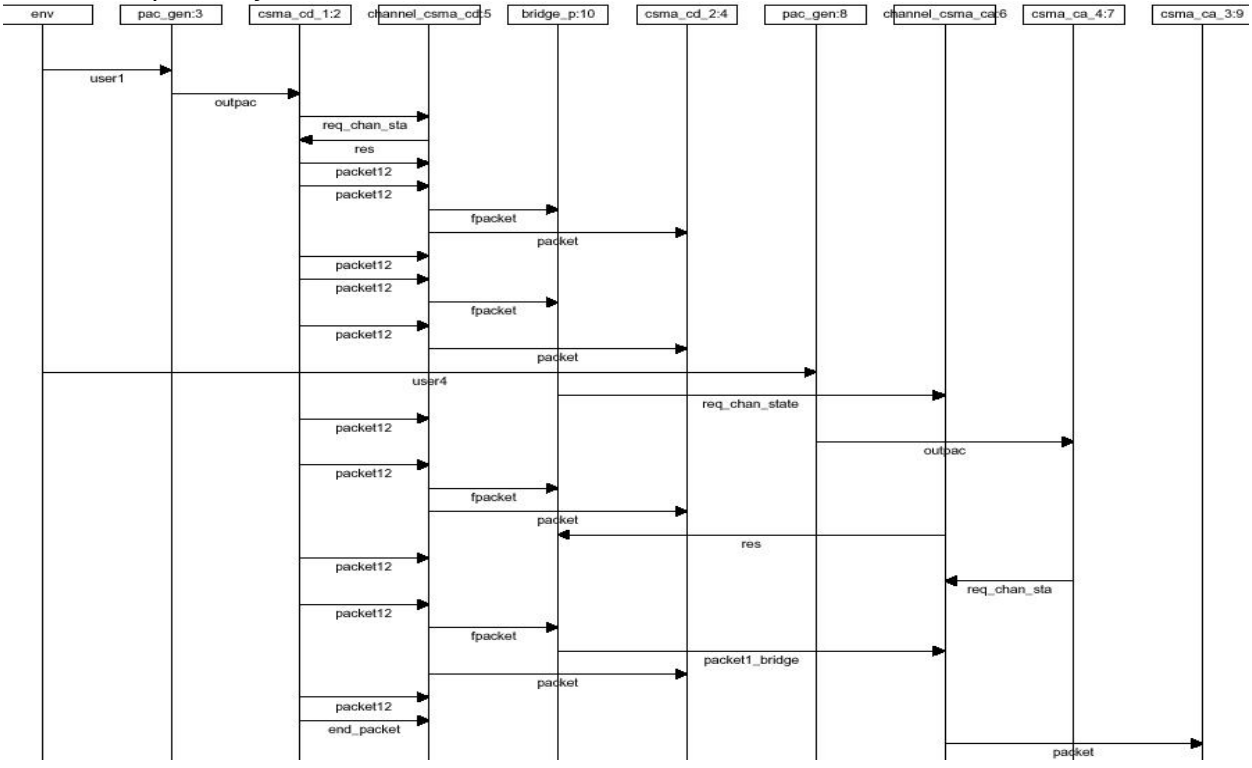
MSC Interoperability test 1



MSC Interoperability test 2



MSC Interoperability test 3



Results

1) MSC for the movement of 3 packets from node 1 (on CSMA/CD) to node 3 (on CSMA/CA). The bridge checks the destination of the packet, if it is on the same link, then it will not forward the packet.
• In case destination is on different link, the bridge buffers the packet until CSMA/CA channel is free.
• Once the channel is free, packets are sent one by one to the receiver.
2) MSC for the movement of 3 packets from node 4 (CSMA/ CA) to node 2 (on CSMA/CD).
• The bridge is forwarding packets reliably to the node 2.
3) MSC for the movement of packets in both directions simultaneously (from node 1 to 3, and node 4 to 2).

• bridge is able to handle the data transfer reliably for both sides.

5.8. Scalability Testing

Scalability refers to the ability of communication protocol to support the network even when the network size grows without consuming much of the resources such as bandwidth and buffers. −Large networks like internet has some scalability limitation when it comes to managing individual traffic flows. Internet protocol RSVP are believed to be non scablable..

Scalability Testing of BGP: −

BGP is designed to support scalability. The first item to test is the routing table capacity as the internet currently has 120,000 routes and still continues to grow. A BGP tester should generate Ipv4 routers and then validate the capacity of the local router to correctly process and store these routes. −The upper limits should be discovered and communicated to the appropriate network architects. −The current Internet routing table should also be loaded into the router so that real world scalability can be verified.

Scalability testing of BGP

The following are the test procedures used in scalability testing of BGP.

Test case 1: Initially a baseline set of tests was performed which determined the number of routes which could be held in a VRF (Virtual routing and forwarding) table relative to the number of VRFs that were conFig.d on the router.

Test case 2: Once the baseline set of tests has been performed the configuration on the router was altered so that each VRF had a firewall filter and a counter on its outgoing interface. This subset of tests was performed with 5, 10 and 15 filter statements and counters per VRF.

Test case 3: The third set of tests involved altering the configuration again by adding a policer to each VRF in addition to the filter and counter added in test case 2. Once this was established a similar subset of tests to test case 2 was performed.

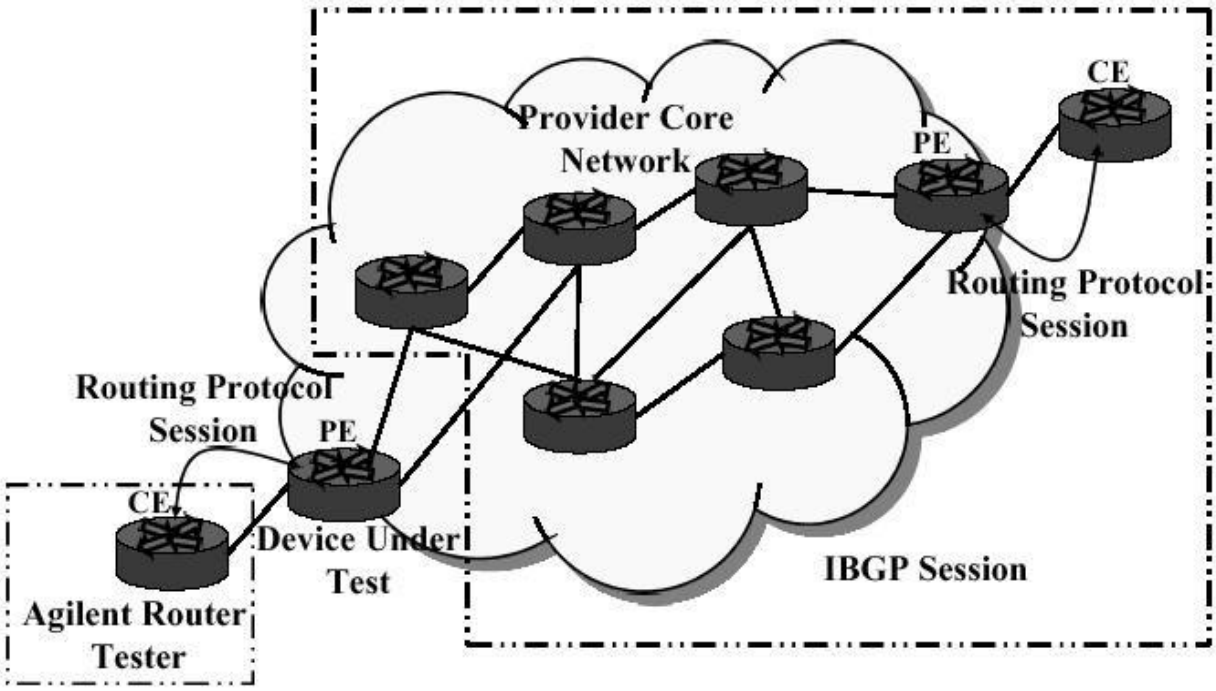


Fig. 5.14.BGP layer 3 MPLS VPN network

Important Questions:

1. Discuss in detail about Authentication protocols.
2. Explain the various aspects of Internet Protocol (IP).
3. Describe the social issues in communication security.
4. With neat diagram brief the SDL based interoperability testing of CSMA/CD and CSMA/CA protocol using bridge