CHAPTER 10

Case Studies on Cryptography and Security

10.1 Introduction

In this appendix, we discuss a few interesting case studies, based on our earlier technical details. As we shall note, implementing cryptography and security is not so straightforward. One needs to consider a lot of practical issues and technical concerns when implementing security.

The organization of the chapter is done in a manner that should help classroom discussions. Each case study begins with a few points for classroom discussion, which should augment the detailed discussion of the case study itself.

The coverage of case studies is very broad in nature. We have discussed diverse areas, such as cryptographic toolkits, Single Sign On (SSO), network-layer attacks, financial transaction security and a few cryptographic problems, which are very interesting to solve.

10.2 Cryptographic Solutions—A Case Study

Points for classroom discussions

1. *What are the key security aspects of a transaction?*
2. *How can the client side on the Internet have cryptographic capabilities?*
3. *What are cryptographic toolkits?*
4. *What are digital certificates, what is their use and how can their validity be checked?*

Functional and Technical Requirements Our case study deals with a banking application that is expected to provide cryptographic functionalities. For simplicity, we have kept the actual business logic quite straightforward, which may not be the case in real life. However, because the focus here is on the cryptographic functionalities, we shall treat the business requirements as ancillary.

Let us consider the requirements statement first.

Our cryptographic application would run on top of an existing banking application, wherein the end customers can perform a single task: funds transfer. A customer can transfer funds from her bank account to any other person's account with the same bank. We shall consider the Internet (i.e. HTTP protocol) as the mechanism of communication between the client and the server. The following cryptographic services are required, depending on the amount of funds being transferred, as shown in Table 10.1.

Table 10.1 *Application Requirements Depending on the Transaction Amount*

Funds Transfer Amount	*Cryptographic Functionality Required*
1-2000	***Message digest*** – To verify the *finger print* of the transaction
2001-5000	***Digital signature*** – To ensure message integrity and non-repudiation
5001 and above	***Digital signature*** – To ensure message integrity and non-repudiation ***Encryption*** – To ensure confidentiality

Let us understand these requirements.

- As we can see, if the funds transfer transaction amount is up to 2000, we simply require a message digest to obtain and verify the *finger print* or integrity of the message. We assume the use of SSL, so that an attacker cannot alter both the message and the message digest. This is an example of *cryptography services*.
- If the transaction amount is between 2000 and 5000, we require a digital signature to ensure not only message integrity, but also non-repudiation. This is an example of *authorization services*.
- Finally, if the transaction amount exceeds 5000, we must not only sign a message but also encrypt it. This is a combination of *authorization services* and *cryptography services*.

Moreover, the authentication mechanism used by our application would be certificate-based authentication. That is, the user must sign a random challenge coming from the server with her private key and send it back to the server for verification, as a part of the authentication process.

Finally, in order to validate the user's certificate, we must provide for Certificate Revocation Lists (CRL) and Online Certificate Validation Protocol (OCSP) services.

We can broadly classify these requirements into appropriate categories as shown in Fig. 10.1.

Hardware and Software Requirements We have two clear portions of the application: the client-side and the server-side.

- ***Client-side requirements*** The client-side requirements would be a browser-based workstation that has Internet Explorer browser installed. The requirement for a specific browser is because we shall use the services of Microsoft's MS-CAPI cryptographic toolkit on the client-side. Since MS-CAPI is installed automatically as a part of Internet Explorer, we shall require that the user must have it installed. Of course, the user is free to use another browser (such as Netscape Communicator) for actual surfing/making transactions. We are simply saying that the user's workstation should still have Internet Explorer installed.
- ***Server-side requirements*** The server-side requirements will mainly consist of the presence of a cryptographic toolkit. Although JCA/JCE, MS-CAPI etc. can suffice, we would like to use a specialist toolkit such as the one from RSA, Entrust or Baltimore. There are no other special requirements on the server-side.

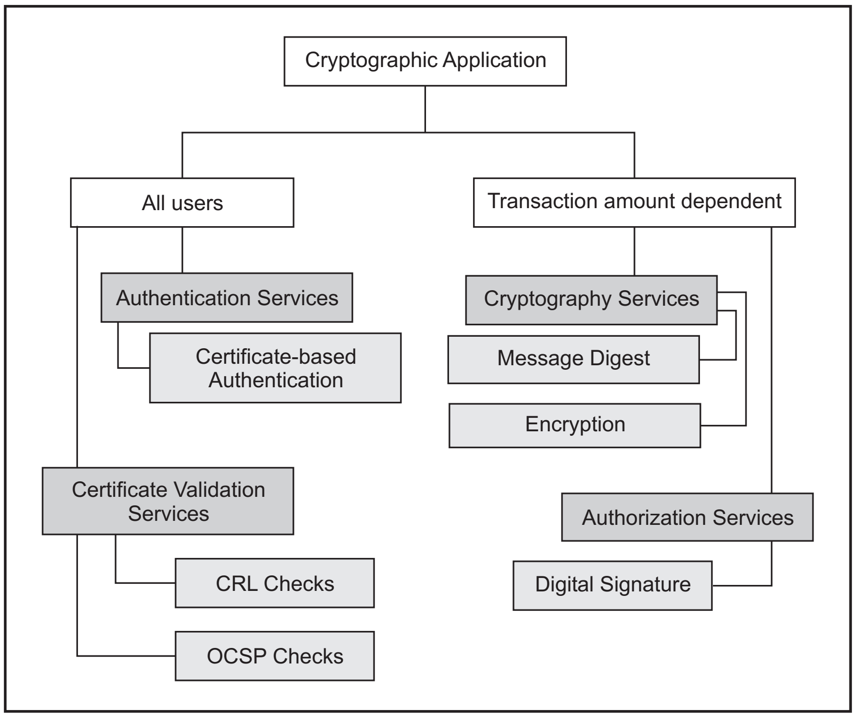

Fig. 10.1 *Requirements classified into various cryptographic services*

Data Definitions Let us now think what data storage requirements our application demands. We certainly need to modify the *User* table (used in chapter 8) to now add the user's digital certificate details in the place of the password field. These details would be necessary for our cryptographic functionalities. Portion of a sample *User* table is shown in Table 10.2.

Table 10.2 *Portion of the User Table*

Column Name	*Data type and Size*	*Comments*
User_Id	Char (10)	Not null, Primary key
First_Name	Char (30)	
Last_Name	Char (30)	
…	…	
Certificate	Binary	

Similarly, we need to create a new table for storing the random challenge during certificate-based authentication. Do not worry if you do not understand this. We shall revisit it soon. This *Session* table would look as shown in Table 10.3.

Application Architecture Let us now think about the architecture of our application. This involves thinking about the portions that are required on the client-side and those required on the server-side.

Table 10.3 *New Session Table*

Column Name	*Data type and Size*	*Comments*
User_Id	Char (10)	Not null, Primary key
Random_Challenge	VarChar (30)	

Clearly, the user would perform the cryptographic operations such as message digests, digital signatures and encryption on the client-side (i.e. on the browser machine). We must verify the results of these operations on the server-side, such as checking if a digital signature was correct or to try and decrypt an encrypted message.

Client-side Cryptographic Processing On the client-side (i.e. the browser side), we have decided to use MS-CAPI for cryptographic functionalities. In order to make use of the MS-CAPI services, pure HTML pages are not sufficient. HTML pages (i.e. a Web page written in the HTML language) can be used for displaying text on the browser in a desired format. However, it cannot perform any client-side processing. Therefore, we must have some other mechanisms to utilize the services of MS-CAPI on the client-side.

As we know, the most widely used approaches for client-side programming are: client-side scripting (e.g. JavaScript or VBScript), Java applets or browser plug-ins (ActiveX controls). Unfortunately, JavaScript cannot directly access MS-CAPI. Therefore, that option is ruled out. We must now decide between applets and plug-ins. Applets are safer to use. But the drawback with applets is that they are quite slow, both because they are written in Java and also because they must be downloaded every time from the Web server to the browser when a cryptographic operation is to be performed. Consequently, we shall go for the approach of browser plug-ins.

A plug-in is downloaded only once – for the very first time it is referenced in a Web page. After that, it resides inside the browser and can be reused directly without requiring a fresh download. Plug-ins do create some psychological doubts in the mind of the users, such as the possibility of performing malicious operations on the user's computer (e.g. introducing a virus or overwriting the disk contents). However, to guard against such doubts, we shall digitally sign the plug-ins, to create confidence in the mind of the end users. When a signed plug-in is downloaded from the Web server to the browser, the user is shown a message that the plug-in is signed and whether the user wants to trust the organization that has signed the plug-in.

Server-side Cryptographic Processing Server-side cryptographic functionalities are pretty much straightforward. We shall provide a dedicated cryptographic API, which can perform the cryptographic operations. The business application of funds transfer would need to call the functions provided by our cryptographic API. The cryptographic API, in turn, will call the appropriate toolkit API.

Thus, the overall cryptographic operations can be summarized as shown in Fig. 10.2.

In order to understand how this works, we shall consider all the desired cryptographic functionalities one-by-one, in turn.

Message Digest As we know, the creation of message digest involves the usage of an algorithm such as MD5 or SHA-1. We will provide a method *create_digest ( )* in the plug-in, which, in turn, will call the message digest function provided by MS-CAPI. Let us understand how this works, step-by-step.

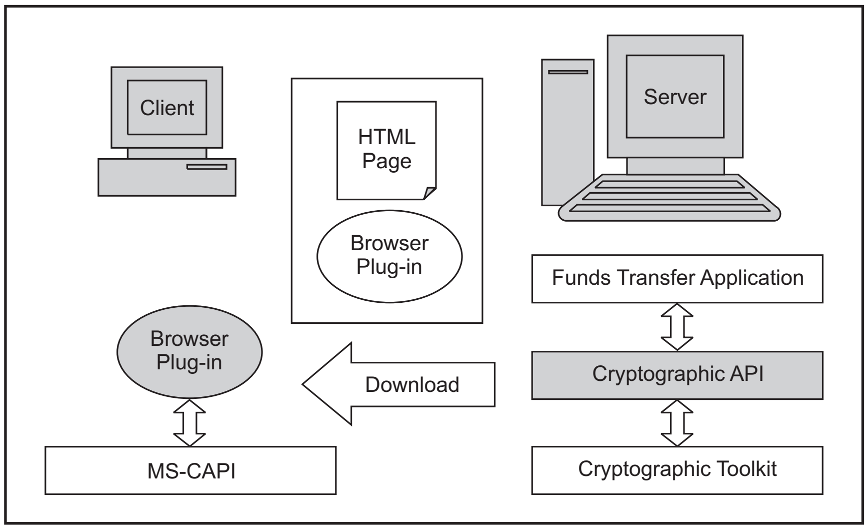

Fig. 10.2 *Conceptual view of the cryptographic operations on the server and the client*

1. The user receives a screen for funds transfer, as shown in Fig. 10.3. The user has to enter the *From*, *To* account numbers and the amount to be transferred. Of course, in real life, the account numbers could be displayed in the form of a list from which the user can choose the account numbers. However, for simplicity, we shall ignore this possibility.

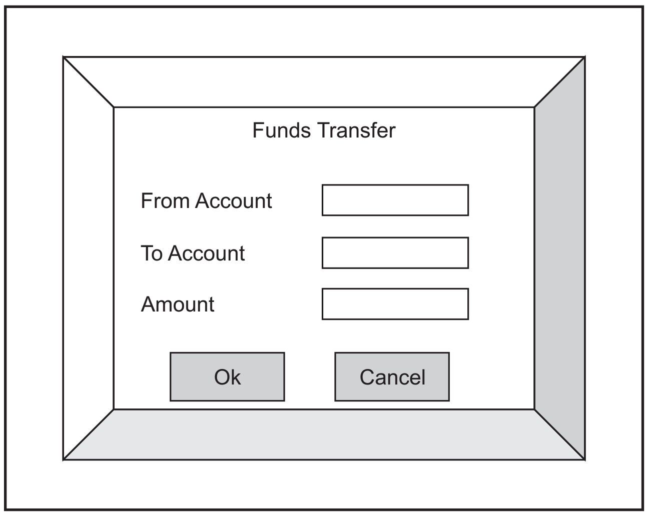

Fig. 10.3 *Funds transfer screen*

2. Now, the actual message digest creation process will work as follows.
   (a) When the user enters these details and presses the *Ok* button, a JavaScript function, which would be a part of the HTML page just displayed (i.e. *Funds Transfer*), would check the amount field. If the amount is less than or equal to 2000 (which is what we will

assume here), it will call the *create_digest ( )* method of the plug-in, passing the form data (i.e. the two account numbers and the amount) to it.

(b) The plug-in, in turn, will call the appropriate message digest function provided by MS-CAPI, passing on the same parameters.

(c) The MS-CAPI message digest function will return a message digest of the data to the plug-in.

(d) The plug-in will, in turn, return this message digest to JavaScript.

(e) JavaScript will submit the form to the server.

3. On the server, we will have a corresponding method, such as *verify_digest*. The server-side program (ASP, JSP, Servlet, etc), which receives the client's form data, will call this method. This method will accept the original message and the message digest (say MD1) from the client, compute its own message digest (say MD2) and compare the two message digests (i.e. MD1 and MD2). Depending on the result of the message digest comparison, it will return an appropriate message back to the funds transfer application on the server side.

Digital Signature The digital signature operation can be described as follows:

1. The input screen shown to the user will be the same as it was shown earlier. The user would enter the two account numbers and the amount and press the *Ok* button.
2. The JavaScript will sense that the amount is between 2001 and 5000, therefore, will call a plug-in method, such as *digital_signature( )*.
3. As we know, to perform digital signature, we need the use's private key. The private key of a user is stored on the disk of the user's computer in a password-protected file. Therefore, at this stage, the plug-in will prompt the user to enter the password, which can open up her private key file.
4. Once the above operation is successful, the plug-in will take the user's private key and pass it along with the original message to the appropriate MS-CAPI method for performing a digital signature.
5. MS-CAPI will first internally create a message digest of the message and encrypt the message digest with the user's private key to form the user's digital signature. MS-CAPI will return the signature to the plug-in.
6. The plug-in will give the original data and the signature to JavaScript, which will submit the form to the server.

To verify the signature, the server will perform the following operations.

1. On the server side, a method such as *verify_signature ( )* will receive the original data and the signature. It has to now de-sign the signature, i.e. decrypt the message digest, which the client had encrypted with her private key. For this operation, the method needs to obtain the user's public key. Therefore, the method will consult the *User* table and based on the user id, retrieve the user's certificate details.
2. It will then perform the certificate validation checks using the CRL/OCSP services and if the certificate is validated successfully, the *verify_signature ( )* method will pass the signature and the user's public key to the appropriate toolkit method.
3. The toolkit method will de-sign (i.e. decrypt) the encrypted message digest, using the signature and the public key received from the *verify_signature ( )* method.

4. Based on the result of the toolkit operation, the *verify_signature ( )* method will send an appropriate message back to the calling method.

Encryption The encryption operation can be described as follows:

1. The input screen shown to the user will be the same as it was shown earlier. The user would enter the two account numbers and the amount and press the *Ok* button.
2. The JavaScript will sense that the amount is above 5000 and therefore, will call a plug-in method, such as *encrypt ( )*.
3. Our specifications suggest that we should first perform a digital signature and then encrypt the original data and the signature. Therefore, the *encrypt ( )* method of the plug-in will first perform a digital signature, as described earlier, by calling the *digital_signature ( )* method.
4. After the signature is performed, we now wish to encrypt the original data and the signature together. Actually, this is enveloping and not encryption, as we shall see. For this purpose, the *encrypt ( )* method of the plug-in will pass the original data, the signature and the server's public key to the appropriate MS-CAPI method.
5. The MS-CAPI method will automatically generate a one time symmetric key. It will encrypt the original data and the signature together with this symmetric key. It will then encrypt the symmetric key with the server's public key. This forms the digital envelope, which it returns back to the *encrypt ( )* method of the plug-in.
6. The plug-in passes the digital envelope to the JavaScript, which submits the form to the server.

On the server-side, the following operations take place.

1. The server-side application will call the appropriate method supplied by our cryptographic API, such as *decrypt ( )*. It will also pass the digital envelope to the *decrypt ( )* method.
2. The *decrypt ( )* method will extract the server's private key from a pre-defined location and pass that along with the digital envelope to the underlying toolkit's appropriate method, such as *open_envelope ( )*.
3. The toolkit method will open the envelope using the server's private key and obtain the one-time symmetric key. Using this symmetric key, it will decrypt the inside data block, to obtain the original data and the digital signature. It will return these values to the *decrypt ( )* method.
4. The *decrypt ( )* method has to now verify the signature. For this, it will call the *verify_signature ( )* method, which we have discussed earlier.
5. Based on the result of the de-signing operation, the *verify_signature ( )* method will send an appropriate message back to the *decrypt ( )* method.
6. Depending on the return value of the *verify_signature ( )* method, the *decrypt ( )* method will send an appropriate message back to the calling method.

Certificate-based Authentication We have discussed the concept of certificate-based authentication in great detail earlier. Therefore, we shall not repeat the discussion here. However, one point must be noted. We have created an additional *Session* table here. This table will be used to store the random challenge created by the server to verify the user's proof of possession of the digital certificate. For the sake of completeness, let us quickly revisit the steps involved in this process.

1. The user submits a login request to the server, only containing the user id.

2. The server validates the user id and if it is ok, it calls a method such as *create_random_challenge ( )*, supplied by our cryptographic API, passing it the user id.
3. This method creates a random challenge and adds a row to the *Session* table for this user. This row contains the user id and the random challenge. The method returns the random challenge back to the server.
4. The server sends the random challenge to the client.
5. The plug-in on the client's computer prompts the user for the password to open her private key file. After this is done, it simply calls the *digital_signature ( )*, passing the random challenge.
6. The digital signature using the user's private key and the random challenge happens as described earlier.
7. The encrypted random challenge is sent back to the server for verification.

On the server side, the same process as was done in *verify_signature ( )* is performed. The only change is that the server-side authenticator now matches the decrypted random challenge (i.e. the de-signed data) with the random challenge stored in the *Session* table. If the two match, we know that the user really holds the private key corresponding to the user's public key certificate and therefore, we consider the user as valid.

CRL-OCSP Checks The CRL-OCSP checks are used for validating digital certificates. CRL and OCSP checks will be performed only on the server side (i.e. during verification of the cryptographic operations done on the client). Only if a certificate is valid, should we proceed with validations (e.g. verifying message digests, verifying signatures or opening envelopes). For this, our cryptographic API layer will use the CRL/OCSP APIs offered by the underlying toolkit. As we know, CRL is an offline check, whereas OCSP is an online check.

- CRL checking requires the downloading of the latest CRL file periodically from the CA's specified location and then checking if the certificate being validated is in the CRL file (i.e. in the list of revoked certificates). If not, the certificate is treated as valid. For this, we would retrieve the certificate details (such as serial number, distinguished name, public key, etc) from the *User* table and pass them on to the toolkit API. The toolkit API will do a lookup in the CRL file and return an appropriate result to our API.
- OCSP also works on a similar model. However, because this is an online check, we must connect to an *OCSP responder* online (using the OCSP protocol, as studied earlier). We would pass the certificate details via the OCSP protocol to the responder, which would return us a valid/invalid certificate message. Based on the outcome, we would decide if we should trust the certificate or reject it.

10.3 Single Sign On (SSO)

Points for classroom discussions

1. *What is Single Sign On (SSO)?*
2. *Why is SSO required?*
3. *What are the main ways of achieving SSO?*
4. *Discuss the working of Kerberos as a SSO protocol.*

Functional and Technical Requirements The National Bank of India (NBI) is a very successful bank in India for many years. To keep itself tuned to the modern world, the bank had started its computerization many years ago. Now, the bank has moved into the arena of Internet banking. The bank was into retail, corporate and investment banking. All these services were moved to the Internet. Therefore, the bank's customers could access all the necessary banking services via the Internet. Within each category of services, the bank offered many individual applications. For instance, within the retail-banking segment, the bank offered solutions in the areas of Internet account access, electronic bill payment, direct debits, etc.

This case study deals with the retail-banking segment only. Each of the applications within this segment for the bank works fine. Customers are very happy with the Internet account access facilities and with the provisions of online bill payment and direct debits. They use these services quite frequently and with ease. This has caused more customers to opt for Internet banking services of the bank. However, a major concern has surfaced of late, which is described as follows.

The applications within each segment work very fine. However, since the applications were developed with an isolated design in mind, each application has its own user authentication model. That is, Internet account access, electronic bill payment and direct debit, all maintain their own user databases and the user has to log on to the particular application as and when she wants to access it. For instance, suppose that the user logs on to the electronic bill payment application and pays her electricity bill. To check the effect of this payment on her bank account, the user is required to now log on to the Internet account access module separately! This is quite annoying for the users, since the bank hosts all the applications and the end users feel that they should not have to worry about the internal design issues of the applications hosted by the bank. They should get a single authentication module. In other words, once they log on to any one application (say Internet account access) using the id and password, they should automatically be logged on to the other applications (i.e. electronic bill payment and direct debit). It is quite tiring for them to have to remember three separate user ids and passwords and use them during the application logons.

Thus, the requirement is to group all the user logins into a single login and offer a single user id and password for the bank users. A user should be able to log on to the bank's site using this id and password and once she is logged on, she must not have to log on separately to access each of the applications. The applications should automatically detect that the user has already authenticated her to one of the applications and simply reuse the credentials of that authentication.

Clearly, this requirement calls for the solution of Single Sign On (SSO). SSO provides a single authentication interface to end-users. Once a user logs on to one of the applications within a group of applications successfully, she does not have to log on to other applications separately. The authentication credentials of the user are simply picked up from the first log on and are reused by the other applications.

Proposed Solution SSO solutions are based on one of the two broad level approaches: the script approach and the agent approach. We can choose either one. However, since the agent approach is considered more suitable for Web-based applications, we shall use it here. As we know, an agent is a small program that runs on each of the Web servers that host an application within the application framework. This agent helps coordinate the SSO workflow in terms of user authentication and session handling.

The bank's application runs on Intel-based servers, on Windows NT 4.0 operating system. These applications are developed on the Microsoft technology, using ASP 2.0 and SQL Server 6.0. The Web server is Microsoft's Internet Information Server (IIS) 4.0. There is an involvement of Microsoft Transaction Server (MTS) for transaction handling. However, the SSO requirement need not be concerned with it.

In order to develop the agents, no special hardware/software requirements are visualized. The agents are simple programs sitting on the IIS Web server and they can be written in the form of ISAPI applications (i.e. the filters on the IIS Web server).

The broad level solution architecture is depicted in Fig. 10.4.

As we can see, the SSO architecture contains two main pieces: the agents sitting on the Web server and a dedicated SSO server. The purpose of these two pieces is as follows:

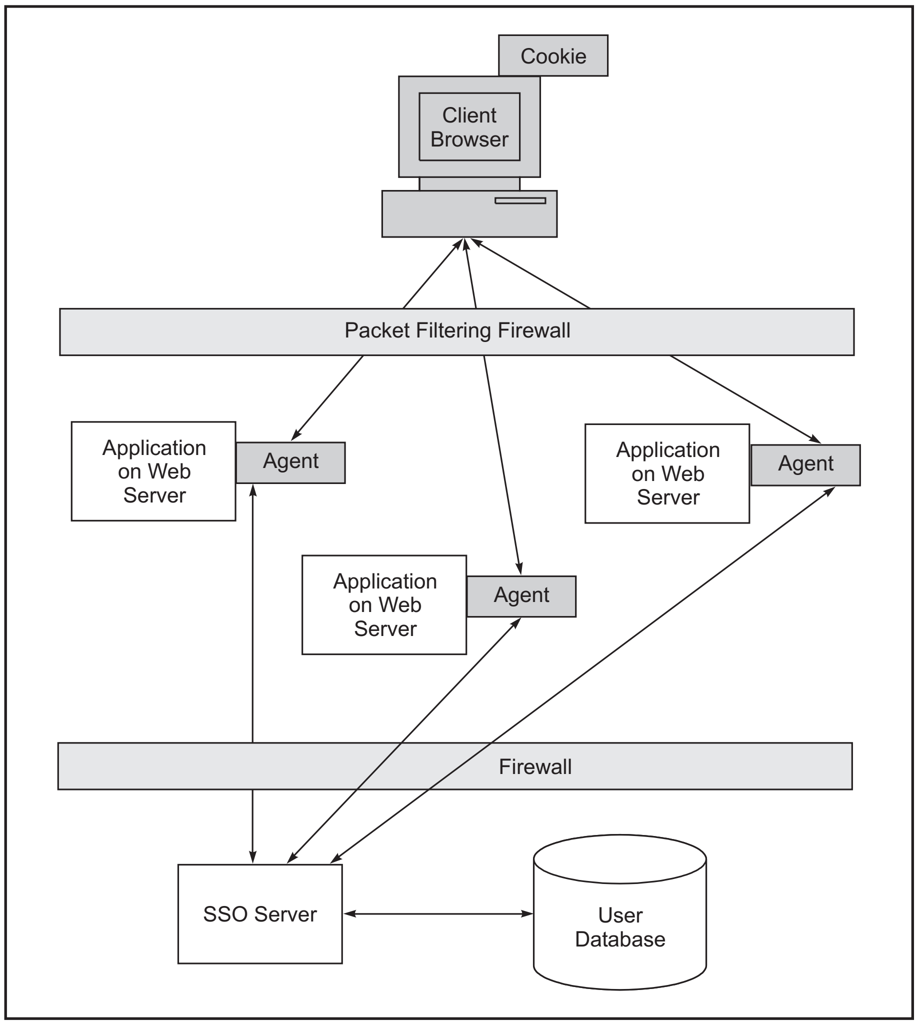
Cookie
Client Browser
Packet Filtering Firewall
Application on Web Server
Agent
Application on Web Server
Agent
Application on Web Server
Agent
Firewall
SSO Server
User Database

Fig. 10.4 *SSO Architecture*

- **Agents**: An agent would intercept every HTTP request arriving at the Web server. There is one agent per Web server, which hosts an application. It interacts with the client browser on the user side and with the SSO server on the application side.
- **SSO server**: The SSO server uses transient cookies to provide session management functionalities. A cookie contains information such as the user id, session id, session creation time, session expiration time, etc.

Application Flow The application flow would be as follows:

1. For every HTTP request that is intercepted, the agent will look for the existence of a valid cookie. There are two possibilities:

   (a) If the cookie is not found, it will initiate a challenge screen to allow the user to enter her credentials. The credentials may be a simple user id/password or user id and digital certificate, depending on the mechanism chosen for user authentication. The agent would receive these details entered by the user and forward them to the SSO server, which would validate them against the user database.

   If the user is authenticated successfully, the SSO server will respond back with a credential token. The agent may forward part of the token to the client browser as a cookie. The cookie may contain basic information like session identifier, session expiry time, etc.

   (b) If the agent finds an existing cookie along with an intercepted HTTP request, it will request the SSO server to decrypt the same and determine whether:

   - The user is already authenticated
   - The authentication is still valid
   - The user can access the application associated with this agent

   If the authentication has expired, it will ask the user to provide authentication details once again.
2. The SSO server will receive authentication requests from the agents. It will then initiate a call to an authentication ASP. This ASP will authenticate the user against the user database and return success or failure.

   On successful authentication, the SSO server will build a credential token with some information and return the whole or part of this token to the agent.

   If the user is already authenticated and the agent requests for verification, the SSO server will determine whether the user is allowed an access to the system. Accordingly, it will initiate the authentication process or will inform the agent to allow user to access the application, if the session is still valid.

10.4 Secure Inter-branch Payment Transactions

Points for classroom discussions

1. *What is the technology to achieve non-repudiation? How is this guaranteed?*
2. *How is the problem of key distribution resolved in PKI?*
3. *Why are cryptographic toolkits required?*
4. *How can smart cards be used in cryptography?*

General Bank Of India (GBI) has implemented an Electronic Payment System called as *EPS* in about 1200 branches across the country. This system transfers payment instructions between two computerized branches of GBI. A central server is maintained at the EPS office located in Mumbai. The branch offices connect to the Local VSAT of a private network by using dial-up connection. The local VSAT has a connectivity established with the EPS office. GBI utilizes its proprietary messaging service called as *GBI-Transfer* to exchange payment instructions.

Currently, EPS has minimal data security. As the system operates in a closed network, the current security infrastructure may suffice the need. The data moving across the network is in encrypted format.

Current EPS Architecture EPS is used to transmit payment details from the payer branch to the payee branch via the central server in Mumbai. Fig. 10.5 depicts the flow, which is also described step-by-step.

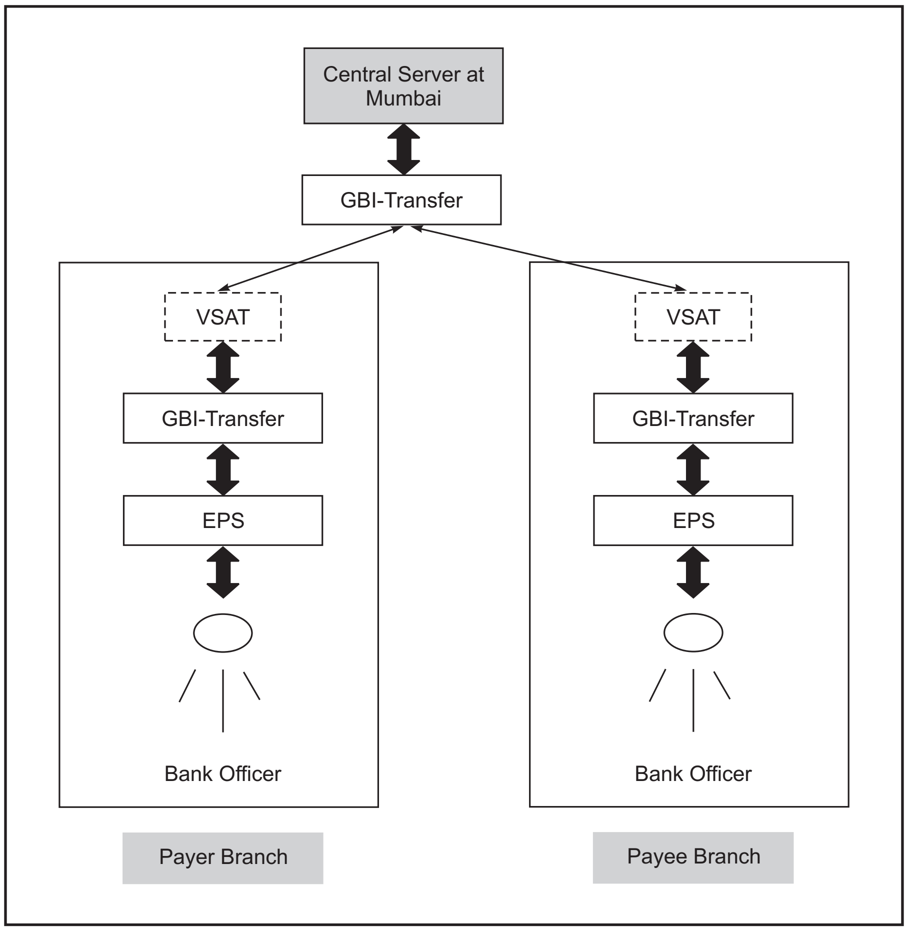

Fig. 10.5 *EPS transaction flow*

A typical payment transfer takes the following steps:

1. A data-entry person in the *Payer Branch* enters transaction details through the EPS interface.
2. A Bank Officer checks the validity of the transaction through the EPS interface.
3. After validating the transaction, the Bank Officer authorizes the transaction. Authorized transaction is stored in a local Payment Master (PM) database.
4. Once the transaction is stored in PM, a copy of the same is encrypted and stored in a file. This transaction file is stored in OUT directory.
5. The *GBI-Transfer* application looks for any pending transactions (i.e. for the presence of any files in the OUT directory) by a polling mechanism and if it finds such transactions, it sends all these files one-by-one to the EPS central office located in Mumbai by dialing the local VSAT.
6. The local VSAT gets connectivity to the EPS central office and the transaction is transferred and stored in the IN directory at the EPS central office.
7. The interface program at the EPS central office collects the file pending in the IN directory and sends it to the PM application at that office.
8. In order to send the Credit Request to PM, the transaction headers are changed. The transaction with changed headers in encrypted format is then placed in OUT directory of the EPS central office.
9. The GBI-Transfer application at the EPS central office collects the transactions pending in the OUT directory and sends them to the *Payee Bank* through the VSAT.
10. The transaction is transferred and stored in the IN directory of the *Payee Branch*.
11. The interface program at the *Payee Branch* collects the transaction and posts it in PM.
12. PM marks the credit entry and returns back an acknowledgement of the same. The acknowledgement is placed in OUT directory of the *Payee Branch*.
13. The acknowledgement is picked by GBI-Transfer at the *Payee Branch* and sent to the EPS central office through the VSAT.
14. The EPS central office receives the credit acknowledgement and forwards it to *Payer Branch*.
15. The *Payer Branch* receives the credit acknowledgement receipt. This completes the transaction.

Requirements to Enhance EPS As GBI is in the process of complete automation and setting up connectivity over the Internet or a private network, they need to ensure stringent security measures, which demand the usage of a Public Key Infrastructure (PKI) framework.

As a part of implementing security, GBI wants the following aspects to be ensured:

- Non-repudiation (Digital Signatures)
- Encryption – 128-bit (Upgrade to the current 56-bit encryption)
- Smart card support for storing sensitive data & on-card digital signing
- Closed loop Public Key Infrastructure

Proposed Solution Since providing cryptographic functionalities require the usage of a cryptographic toolkit, it is assumed that GBI will implement an appropriate Certification Authority (CA) infrastructure and a PKI infrastructure offering.

The transaction will be digitally signed and encrypted/decrypted at the Payer and Payee branches, as well as at the EPS central office. The signing operation can be performed on the system or on external hardware like a smart card. On the server side, a provision of automated signing without any manual intervention will be provided.

The transaction flow described earlier would now be split into two legs:

- The Payer Leg (*Payer Branch* to the EPS central office)
- The Payee Leg (EPS central office to the *Payee Branch*)

The architecture for the Payer Leg is shown in Fig. 10.6. As shown, after verifying the transaction, the EPS Officer authorizes the transaction at the *Payer Branch*. Internally, the application digitally signs the transaction. This signature, along with the transaction data is stored in the local PM Database and then encrypted and placed in the IN directory. For signature and encryption, a cryptographic toolkit is required at the *Payer Branch*. The signed-and-encrypted transaction is sent to the EPS central office in the same way as before.

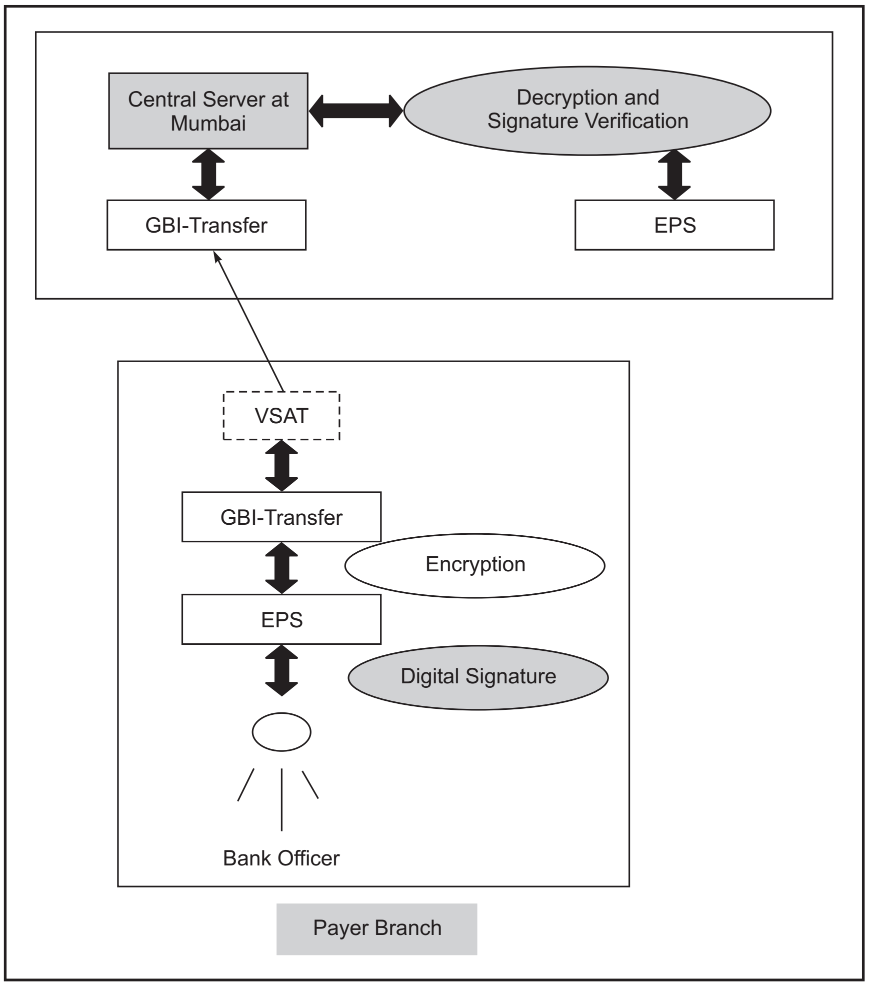

Fig. 10.6 *New EPS transaction flow at the Payer Branch*

The encrypted file is decrypted at EPS central office. Before storing the transaction in the database, the digital signature is verified using an appropriate cryptographic toolkit. The verification process may also check the status of the user's digital certificate by either CRL or OCSP check. If the status of the certificate is invalid, the transaction will be rejected, otherwise it will be stored in the local PM database.

On the Payee Leg, the EPS central office will create a Credit Request as before, sign and encrypt it with the bank officer's digital certificate. This signed-and-encrypted request will be forwarded to the *Payee Branch*. The flow is shown in Fig. 10.7.

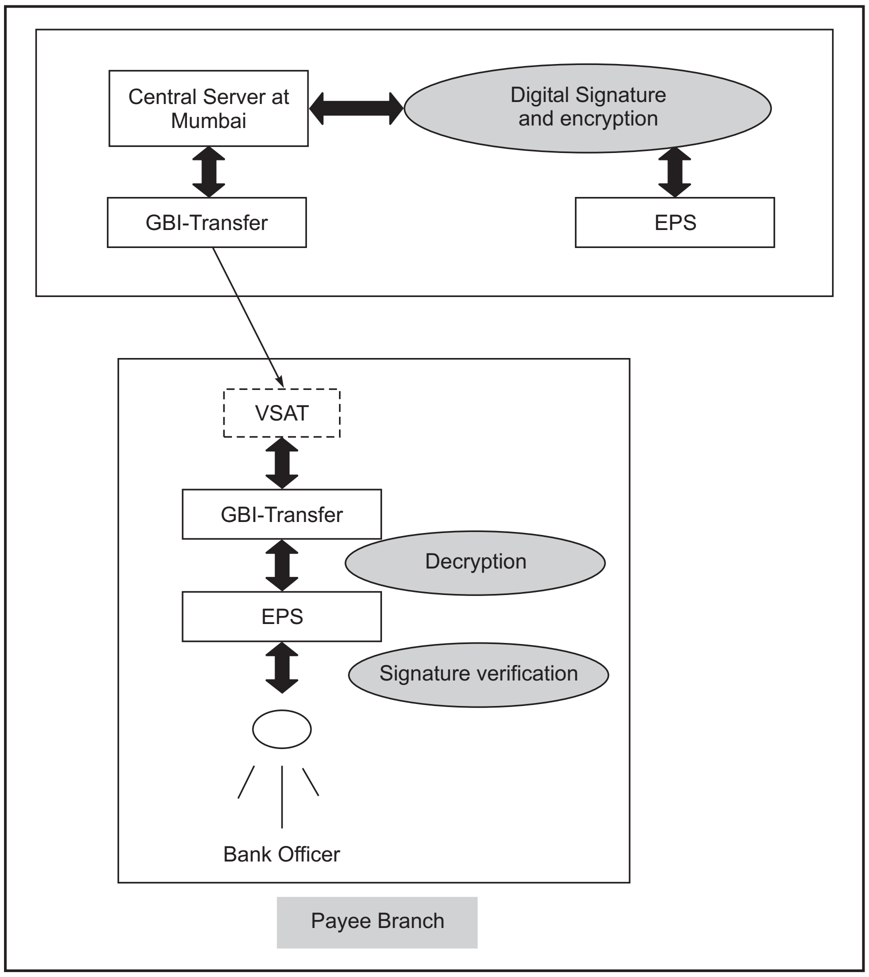

Fig. 10.7 *New EPS transaction flow at the Payee Branch*

In the Payee Leg, the PM software at the EPS central office will generate a Credit Request for the *Payee Bank*. This request will be digitally signed. The signature along with the Credit Request will be encrypted and sent to the *Payee Branch*.

The *Payee Branch* will decrypt the Credit Request and verify the digital signature. If the signature is verified successfully, the transaction is entered into database. Otherwise, it gets rejected and the status of the same is sent to EPS central Office. The Credit Response to the EPS central office can also be digitally signed and encrypted in a similar fashion.

10.5 Denial of Service (DOS) Attacks

Points for classroom discussions

1. *What is the Denial Of Service (DOS) attack?*
2. *Which of the DOS attacks – the one launched using TCP protocol or the one launched using the UDP protocol, more severe? Why?*
3. *What is the meaning of the term 'service' in DOS?*
4. *What can possibly prevent DOS attacks?*

The **Denial Of Service (DOS)** attack has gained a lot of attention in the last few years. Most notable DOS attacks were launched against famous Web sites such as Yahoo, Amazon, etc. In April 2000, a 15-yead old Canadian called as Mafiaboy launched DOS attacks, which caused a lot of concern regarding the security mechanisms of Web sites.

The basic purpose of a DOS attack is simply to flood/overhaul a network so as to deny the authentic users services of the network. A DOS attack can be launched in many ways. The end result is the flooding of a network or change in the configurations of routers on the network.

The reason it is not easy to detect a DOS attack is because there is nothing apparent to suggest that a user is launching a DOS attack and is actually not a legitimate user of the system. This is because in a DOS attack, the attacker simply goes on sending a flood of packets to the server/network being attacked. It is up to the server to detect that certain packets are from an attacker and not from a legitimate user and take an appropriate action. This is not an easy task. Failing this, the server would fall short of resources (memory, network connections, etc. and come to a grinding halt after a while.

A typical mechanism to launch a DOS attack is with the help of the SYN requests. On the Internet, a client and a server communicate using the TCP/IP protocol. This involves the creation of a TCP connection between the client and the server, before they can exchange any data. The sequence of these interactions is as follows:

1. The client sends a SYN request to the server. A SYN (abbreviation of *synchronize*) request indicates to the server that the client is requesting for a TCP connection with it.
2. The server responds back to the client with an acknowledgement, which is technically called as SYN ACK.
3. The client is then expected to acknowledge the server's SYN ACK.

This is shown in Fig. 10.8.

Only after all the three steps above are completed that a TCP connection between a client and a server is considered as established. At this juncture, they can start exchanging the actual application data.

An attacker interested in launching a DOS attack on a server performs Step 1. The server performs Step 2. However, the attacker does not perform Step 3. This means that the TCP connection is not complete. As a result, the server needs to keep the entry for the connection request from the client as *incomplete* and must wait for a response (i.e. Step 3) from the client. The client (i.e. the attacker) is not at all interested in executing step 3. Instead, she simply keeps quiet. Now, imagine that the client sends

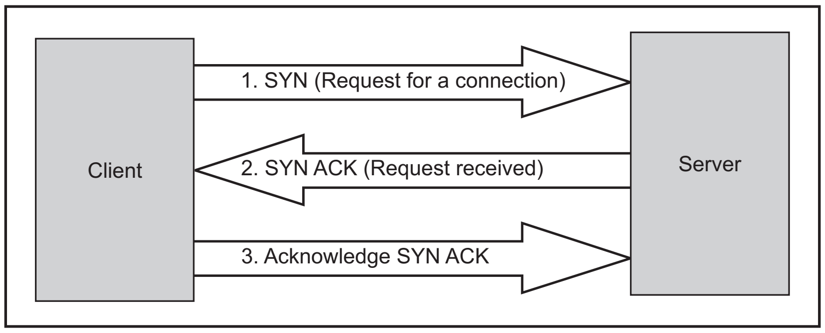
Client
Server
1. SYN (Request for a connection)
2. SYN ACK (Request received)
3. Acknowledge SYN ACK

Fig. 10.8 *TCP connection establishment process*

many such SYN requests to the same server and does not perform Step 3 in any of the requests. Clearly, a lot of incomplete SYN requests would be pending in the server's memory and if these are too many, the server could come to a halt!

Another mechanism to launch the SYN attack is, in Step 1, the attacker forges the source address. That is, the attacker puts the source address as the address of a non-existing client. Therefore, when the server executes Step 2, the SYN ACK never reaches any client at all, fooling the server!

In a more severe case, the attacker launches a Distributed DOS (DDOS) attack. Here, the attacker sends many SYN requests to the server from many physically different client computers. Thus, even if the server detects a DOS attack, it cannot do much by blocking SYN requests coming from a particular IP address – there are many such requests from a variety of (forged) clients!

Mafiaboy performed an attack, which was quite similar to what we have discussed here. He flooded the Web servers with packets sent one after the other in quick succession. This brought a few sites to halt for more than three hours. To stop these attacks, the site administrators started blocking packets coming from suspected source IP addresses. After a few hours, the attacks seemed to be nullified. What had actually happened was quite different. Mafiaboy had simply stopped his attacks! Perhaps he was too tired or bored!

How does one prevent DOS attacks? There are no clear answers here. The following mechanisms can be tried out:

1. Investigate the incoming packets and look for a particular pattern. If such a pattern emerges, then try blocking incoming packets from the concerned IP addresses. However, this is not easy as we have mentioned and this is even tougher to be performed in real time.
2. Another mechanism is to configure the services offered by a particular application so that it never accepts more than a particular number of requests in a specified time interval. Therefore, even if the attacker keeps sending more and more requests, the server would process them at its own pace and use that time to analyze/detect the attacks and take a corrective action.
3. Blocking a particular IP address, port number or a combination of such factors can also prevent a DOS attack. Of course, doing this in real time is not easy.
4. As a precaution, it is always good to have a backup of the firewall and the servers ready. If the main machine is compromised, it should be quickly brought down and the backup can take its place until a proper cleanup is performed.

10.6 IP Spoofing Attacks

Points for classroom discussions

1. *What is the IP spoofing attack?*
2. *Why is it not easy to detect IP spoofing attacks?*

IP Spoofing attacks are more challenging than the DOS attacks, if the attacker intends to use the consequence of the attack for his own benefit. Kevin Mitnick launched such an attack successfully on Tsutomu Shimomura's home computer and the computer at the University of Southern California. Before describing the attack, the technical mechanism to achieve it is first described as follows:

1. The attacker creates an IP datagram (packet) to be sent to the server, just like any other normal packet. This is a SYN request.
2. The main difference between the packet created by the attacker and any other packet is that the attacker puts the *source address* as another computer's IP address. That is, the attacker *forges* or *spoofs* the source IP address.
3. As usual, the server responds back with a SYN ACK response, which travels to the computer with the forged IP address (i.e. not to the attacker).
4. The attacker has to somehow get hold of this SYN ACK response sent by the server and acknowledge it, so as to complete a connection with the server.
5. Once this is done, the attacker can try various commands on the server computer.

Kevin Metnick was a person who had a lot of *background* in hacking/attacking computer systems. Following are some of the incidents he was involved in.

(a) In his early days, Kevin joined a group of hackers. Among their many attacks, the most notable was the one in which they changed a home phone to a pay phone! Thus, when the phone subscriber wanted to make a call, a message greeted her with a request to first pay 20 cents!
(b) At the age of 17, Kevin and his friend actually entered a phone company's office and stole passwords from there! This resulted into a jail term for them.
(c) In 1983, Kevin tried to break into a Pentagon computer over the ARPAnet, an act that was detected, causing another imprisonment for Kevin.

There were many such examples.

In the IP spoofing attack, Kevin performed the following tasks in a very intelligent fashion. Before we explain it, it should be noted that Tsutomu Shimomura had a trusted relationship between his home computer (X) and the computers at the University of Southern California (Y).

1. Kevin first flooded Tsutomu's home computer (X) with a series of SYN requests, causing it to virtually come to a halt.
2. Kevin then sent a SYN request to the main server (Y) at the University. In this packet, Kevin put the source address as X. That is, the source address was spoofed.
3. The server (Y) detected this as an attempt to establish a request for a TCP connection and responded back with a SYN ACK response. As expected, the SYN ACK response went back to Tsutomu's home computer (X), because that was the source address in the original SYN request of Step 2.

4. Kevin had flooded X in Step 1. So, X is not able to see Y's response.
5. Kevin now guessed the sequence number used by Y in the SYN ACK response (after a few failures) and used that number in the acknowledgement of the SYN ACK message, which it sent to Y. That is, Kevin sent many acknowledgements (with different sequence numbers) of the SYN ACK message from Y, to Y.
6. In each case, Kevin immediately sent a command to add a wildcard entry to the trust relationship file maintained by Y, which made Y to trust anybody and everybody. Obviously, this command failed for all the unsuccessful attempts of Kevin to send an acknowledgement of the SYN ACK message. But it succeeded for the one acknowledgement, which succeeded. This opened up Y for all outside users!

10.7 Cross Site Scripting Vulnerability (CSSV)

Points for classroom discussions

1. *What is the purpose of scripting technologies on the Internet?*
2. *What can prevent CSSV attacks?*
3. *What sort of testing can the creators of a Web site perform in order to guard against possible CSSV attacks?*

Cross Site Scripting Vulnerability (CSSV) is a relatively new form of attacks that exploits inadequate validations on the server-side. The term *Cross Server Scripting Vulnerability (CSSV)* is actually not completely correct. However, this term was coined when the problem was not completely understood and has stuck ever since. Cross-site scripting happens when malicious tags and/or scripts attack a Web browser via another site's dynamically generated Web pages. The attacker's target is not a Website, but rather its users (i.e. clients or browsers).

The idea of CSSV is quite simple to understand and is based on exploiting the scripting technologies, such as JavaScript, VBScript or JScript. Let us understand how this works. Consider the following Web page containing a form as shown in Fig. 10.9, in which the user is expected to enter her postal address. Suppose that the URL of the site sending this page is www.test.com and when the user submits this form, it would be processed by a server-side program called as address.asp.

We would typically expect the user to enter the house number, street name, city, postal code and country, etc. However, imagine that the user enters the following weird string, instead:

```
<SCRIPT>Hello World</SCRIPT>
```

As a result, the URL submitted would be something like www.test.com/address.asp?address=<SCRIPT>Hello World </SCRIPT>.

Now suppose that the server-side program address.asp does not validate the input sent by the user and simply sends the value of the field address to the next Web page. What would this translate to? It would mean that the next Web page would receive the value of address as <SCRIPT>Hello World</SCRIPT>.

As we know, this would most likely treat the value of the address field as a script, which would be executed as if it is written in a scripting language, such as JavaScript etc on the Web browser. Therefore, the user would get to see *Hello World*.

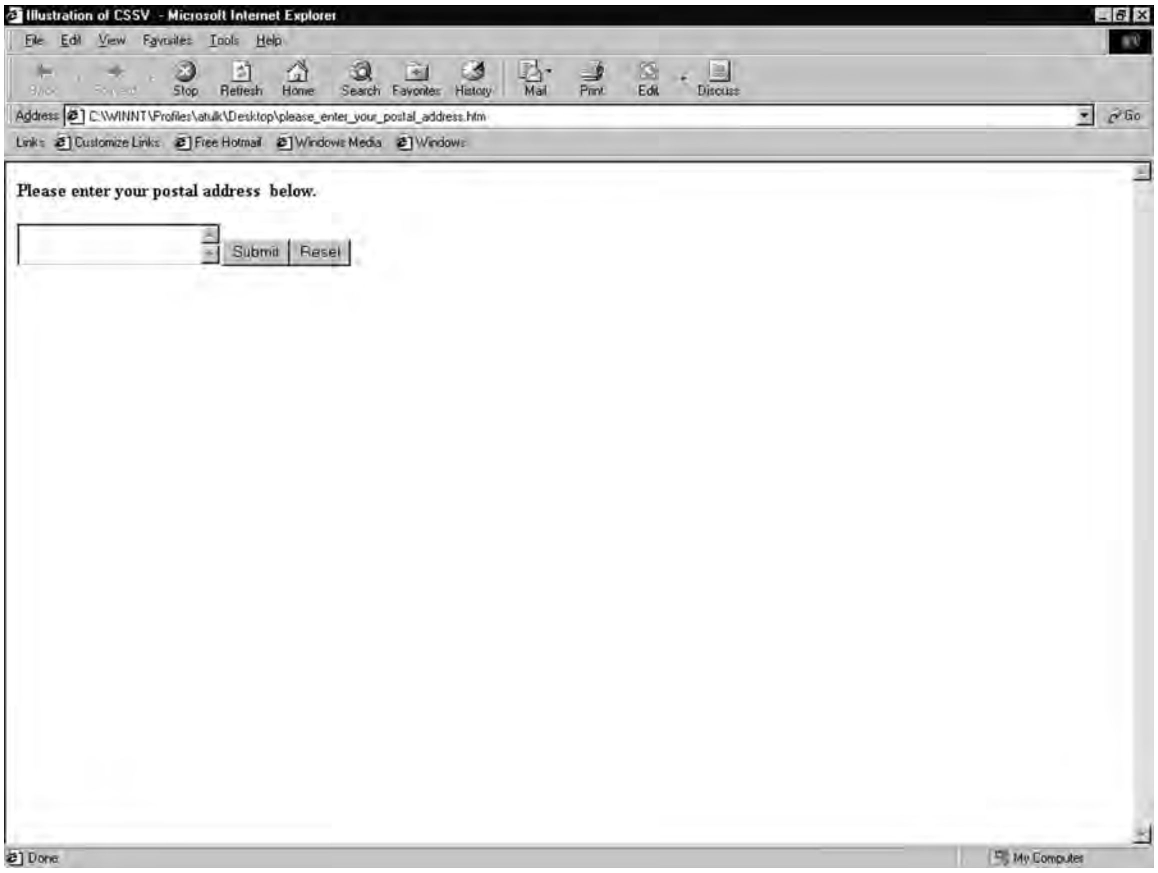
Illustration of CSSV - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search Favorites History Mail Print Edit Discuss
Address C:\WINNT\Profiles\atulk\Desktop\please_enter_your_postal_address.htm
Go
Links Customize Links Free Hotmail Windows Media Windows
Submit Reset
Done
My Computer

Please enter your postal address below.

Fig. 10.9 *Sample HTML form*

Obviously, no serious damage is done. However, extrapolate this possibility to other situations where a user can actually send damaging scripts to the server. This can cause the same or another client to receive a Web page whose contents/look-and-feel are changed. In a more damaging case, the confidential information entered by a user could also be captured and sent to another user and so on. How can this be done?

When a JavaScript program gets downloaded on a browser through a CSSV attack, the JavaScript, in turn, can call up the services of an ActiveX control. An ActiveX control is a small program that gets downloaded from the server to the client and executes on the client. The ActiveX control can write to the disk or read from it and perform many such tasks. Once downloaded to the client via the malicious JavaScript call, the ActiveX control, therefore, can do real damage in this case.

The chief measure to protect against these attacks is to validate all the input fields for tags that look suspicious (e.g.. <, >, SCRIPT, APPLET, OBJECT, etc). The server-side program should not trust a browser-based user to enter data with only good intention. It can very well be malicious, causing damage to other clients.

One can test for CSSV on any Web site, by simply trying to enter scripts or script-like tags in the input areas, such as text boxes.

10.8 Contract Signing

Points for classroom discussions

1. *When can a contract in real-life considered to be complete?*
2. *Is it sufficient if only one of the parties signs a contract?*
3. *What mechanism can be used in cryptography to sign electronic contracts?*
4. *How can disputes be settled if a party later refuses having signed a contract?*

There is a builder, Bob, who has constructed a building for residential purposes. This building contains 20 flats (apartments). Bob wants to sell all of them to individual customers. Alice is one such customer, who is interested in buying a flat. She approaches Bob over telephone and expresses her desire to buy the flat. After a couple of such discussions, Alice decides to go ahead with her purchase. However, Bob has only one requirement. This requirement deals with the way the contract or the agreement between Bob and Alice is signed. Since Bob is Internet-savvy, he wants that the contract between him and Alice be digitally signed over the Internet and that Alice deposit the money into his account by using Internet banking.

In this discussion, we shall restrict our scope to only the first aspect, i.e. digitally signing contracts. How would Bob ensure that the contract signing process is complete in all respects? How would Alice be sure that she is not being cheated? How would a dispute be settled, if it arises?

In order to ensure that the contract signing happens smoothly, Bob contacts a respected third party, Trent. Alice also speaks with Trent over phone and is assured that Trent is a responsible third party, in which she could trust. With these ideas in place, let us write down the steps that would ensure that the digital contract signing is complete and comprehensive in all respects.

1. Alice digitally signs a copy of the contract and sends it to Trent over the Internet.
2. Bob digitally signs a copy of the contract and sends it to Trent over the Internet.
3. Trent sends a message to both Alice and Bob, informing them that he has received the digitally signed contracts from both.
4. At this stage, Alice digitally signs two more copies of the contract and sends both of them to Bob.
5. Bob now digitally signs both the contract copies received from Alice. He keeps one of the copies for his records and sends the other one to Alice.
6. Alice and Bob both inform Trent that they have a copy of the contract, which is digitally signed by both of them.
7. Trent now destroys the original contract copies received from Alice and Bob in Steps 1 and 2.

Is this solution foolproof? Let us examine it.

Can Alice deny at a later date that she never signed the contract? She cannot, because Bob has a copy of the contract, which was signed by him as well as by Alice. It is also the case the other way round. Bob cannot refute having signed the contract, because Alice has a copy of the contract signed by both.

Thus, the solution is indeed comprehensive. This solution also involves the use of a third party (also called as an *arbitrator*), Trent.

10.9 Secret Splitting

Points for classroom discussions

1. *What is the need of secret splitting?*
2. *What would happen if we split a secret and one of the persons knowing the secret leaves the organization? Can the original secret still be recovered? Can the person leaving the organization break the secret before leaving?*
3. *Is the concept of secret splitting the same as XML signatures (signing only a portion of a document)?*

Many times, it is important to *split* a secret in such a fashion that the individual pieces of the secret make no sense. For example, suppose that in a bank, the account details are maintained in a database, which is protected by two *different* passwords. The two passwords are distinct and are known to two different officers. None of them knows both the passwords. Each one knows only her password. This ensures that to access the whole set of accounts, both the officers have to enter their passwords independently and only then the database can be accessed. This sort of scheme is actually used in many banks and other institutions as well.

Let us now think how a variation of this scheme can be implemented using cryptography. Suppose that Alice and Bob are the two officers and Trent is an arbitrator. The scheme should be devised in such a way that neither Alice nor Bob has an access to the complete secret, but they can combine their secrets so as to come up with the required combined secret. The scheme works as follows:

1. Suppose that the plain text secret (e.g.. a combined password, which needs to be split into two parts) is P.
2. Trent generates a random number R, whose length in bits is exactly the same as that of P.
3. Trent performs an XOR operation on P and R to generate the combined secret, S, as follows:

$$S = P \oplus R$$

4. Trent now sends S to Alice and R to Bob.

This process is shown in Fig. 10.10.

The only way for Alice and Bob to regenerate the original secret P, is to perform the following operation:

$$P = S \oplus R$$

Alice cannot do this alone; because she does not have R. Bob cannot do it either, as it does not know S. The only way to regenerate P is for Alice and Bob to come together, enter their respective pieces of the secret (i.e. S and R, respectively), perform an XOR operation and generate P.

10.10 Virtual Elections

Points for classroom discussions

1. *Is it technically possible to have elections on the Internet? How? What sort of infrastructure would be needed for this?*
2. *What would be the main concerns in such a virtual election?*
3. *What would be the use of digital signatures and encryption in virtual elections?*

Another situation where cryptography is useful is virtual elections. Computerized voting would become quite common in the next few decades. As such, it is important that the protocol for virtual

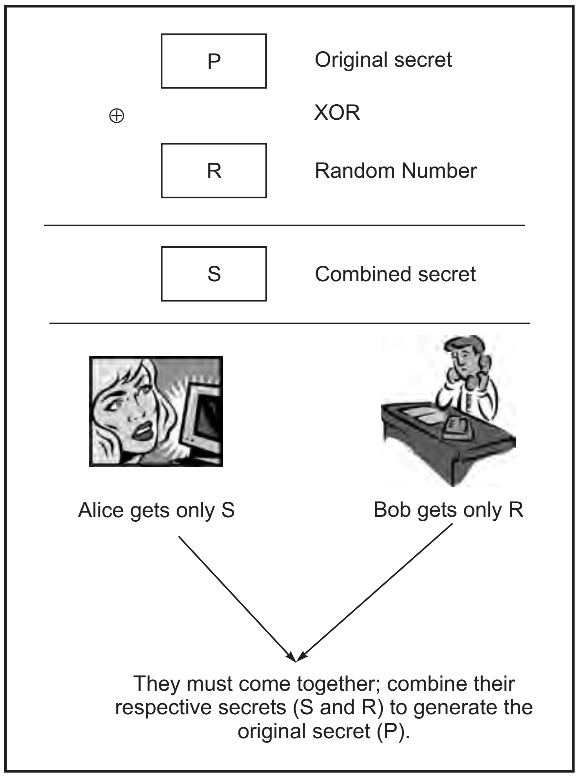
P
Original secret
⊕
XOR
R
Random Number
S
Combined secret
Alice gets only S
Bob gets only R
They must come together; combine their respective secrets (S and R) to generate the original secret (P).

Fig. 10.10 *Secret splitting*

elections should protect individual privacy and should also disallow cheating. Consider the following protocol in order that voters can send their votes electronically to the Election Authority (EA).

1. Each voter casts the vote and encrypts it with the public key of the EA.
2. Each voter sends the encrypted vote to the EA.
3. The EA decrypts all the votes to retrieve the original vote, tabulates all the votes and announces the result of the election.

Is this protocol secure and does it provide comfort both to the voters as well as to the EA? Not at all! There are following problems in this scheme:

1. The EA does not know whether the authorized voters have voted or it has received fake (bogus) votes.
2. Secondly, there is no mechanism to prevent duplicate voting.

What is the advantage of this protocol? Clearly, no one would be able to change another voter's vote, because it is first encrypted with the EA's public key and is then sent to the EA. However, if we observe this scheme carefully, an attacker need not change someone's vote at all. The attacker can simply send duplicate votes!

How can we improve upon this protocol to make it more robust? Let us rewrite it, as follows:

1. Each voter casts the vote and signs it with her private key.

2. Each voter then encrypts the signed vote with the public key of the EA.
3. Each voter sends the vote to the EA.
4. The EA decrypts the voter with its private key and verifies the signature of the voter with the help of the voter's public key.
5. The EA then tabulates all the votes and announces the result of the election.

This protocol would now ensure that duplicate voting is disallowed. Because the voter has signed the vote (with her private key) in Step 1, this can be checked. Similarly, no one can change another voter's vote. This is because a vote is digitally signed and any changes to it will be detected and exposed in the signature verification process.

Although this protocol is a lot better, the trouble with this scheme is that the EA would come to know who voted for whom, leading to privacy concerns. We shall leave it to the reader to figure out how this problem can be solved.

10.11 Secure Multiparty Calculation

Points for classroom discussions

1. *Can you think of any practical situations where secure multiparty calculations would be required?*
2. *Can symmetric key encryption alone suffice the needs of secure multiparty calculations? If yes, what are the possible issues/constraints?*
3. *Is an arbitrator mandatory in such a scheme?*

Suppose that we have the following problem:

Alice, Bob, Carol and Dave are four people working in an organization. One fine day, they are interested in knowing their average salary. However, they want to (obviously) ensure that no one comes to know about the salary of anyone else. Unfortunately, there is no arbitrator, who can take this task upon itself.

How can this be achieved? An interesting protocol can be used to fulfill these requirements, as follows:

1. Alice generates a random number, adds that number to her salary, encrypts the resulting value with the public key of Bob and sends it to Bob.
2. Bob decrypts the information received from Alice with his private key. He adds his salary to the decrypted number (which is Alice's salary + random number). He then encrypts it with the public key of Carol and sends the result to Carol.
3. Carol decrypts the value received from Bob with her private key, adds her salary to it, encrypts the result with Dave's public key and sends the result to Dave.
4. Dave decrypts the value received from Carol with his private key, adds his salary to it, encrypts the result with Alice's public key and sends the result to Dave.
5. Alice decrypts the value received from Dave with her private key and subtracts the original random number from it. This gives her the total salary.
6. Alice divides the total salary by the number of people (4). This produces the value of the average salary, which she announces to Bob, Carol and Dave.

10.12 Creating a VPN

Points for classroom discussions

1. *Review the concepts of a VPN.*
2. *Discuss the pre-requisites of a VPN.*
3. *Create a VPN and test it, as explained below.*

There are many companies/Web sites that offer free VPN software. An example is http://sunsite.dk/vpnd. This VPN software runs on Linux operating system. For the purpose of setting up the VPN, proceed as follows:

1. Use five layer-2 hubs, two routers (R1 and R2), two PCs running Linux (P1 and P2), which function as routers and five general purpose computers to form the intranet shown in Fig. 10.11.

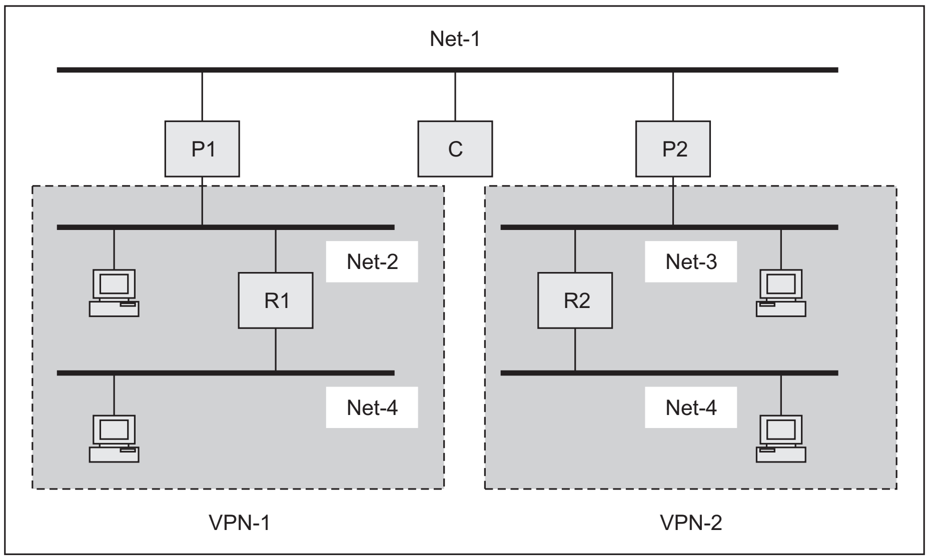

Fig. 10.11 *VPN set up*

2. Download and compile a VPN software for Linux and install it on the two Linux computers (P1 and P2).
3. Configure the VPN software so that it encrypts all the communication between the two computers.
4. Run software on the *checker computer* (C), which will capture and display all packets traveling on network 1.
5. Do a *TELNET* from a computer in location 1 on to another computer in location 2. Verify that all data is encrypted.

10.13 Cookies and Privacy

Points for classroom discussions

1. *Discuss the concept of a cookie*
2. *What are the practical usages of cookies?*
3. *Which technologies can create and read cookies?*
4. *How can cookies damage privacy?*

Cookies are simple text files stored on client computers. Whenever a user browses the Internet, the server-side code can create a small text file, called as cookie, on the user's computer. This is because the next time the user visits the same site again (during the same or a different session), the server can identify the client, based on the cookie. This overcomes the drawback of the HTTP protocol, which is stateless. This means that every request from a client, even to the same server in the same session, is treated as a completely new request.

Cookies can be transient (live for the lifetime of a session) or persistent (remain on the user's computer beyond the lifetime of a session). Cookies themselves cannot cause any damage to the user's computer, since they cannot contain executable code. However, cookies can be exploited to send targeted advertisements to the users, without their knowledge. This works as follows:

1. An advertising agency (say *XYZ Advertisements Limited*) contacts major Web sites and places banner ads for its corporate clients' products on their pages. It pays a fees to the site owners for this.
2. Instead of providing an image (GIF/JPEG), it provides a URL to add to each page.
3. Each URL contains a unique number in the *file* part, for example: http://www.XYZAdverts.com/07041973.gif
4. When a user visits a page P for the first time, the browser fetches the advertisement image from XYZ along with the main HTML page for the site it is visiting.
5. XYZ sends a cookie to the browser containing a unique user ID and records the relationship between this user ID and the file name.
6. Later, when the same user visits another page, the browser sees another reference to XYZ.
7. The browser sends the previous cookie to Sneaky and also fetches the current page from XYZ, as before.
8. XYZ knows that the same user has visited another Web page now.
9. It adds this reference to its database.
10. Over time, XYZ has a lot of information about the Web pages the user visits, the actions it performs, etc.
11. The *advertisement* from XYZ can be a single pixel in the same background color, making it even more difficult for the user to know that advertisements are appearing!