# Digital Signatures

Public-key encryption is more computationally intensive than symmetric one. That’s why, it is generally used to encrypt a few small blocks of information, exchanged as part of a protocol.

Digital signature is implemented using asymmetric cryptography.

Digital signature works in two ways. It can be used by the sender, C, of a document to prove that he authored the document, and it can be used by the receiver of the document as evidence that C authored it. The latter case is referred to as nonrepudiation: once you send a signed check, you cannot at a later time, deny that you authorized a money transfer. A digital signature also serves a third, very important purpose. Once a document is signed, it cannot be modified without invalidating the signature. Hence, a digital signature guarantees data integrity.

Digital signatures use a property of many asymmetric encryption algorithms – the roles of the public PC and private RC keys are symmetric:



Hence, a simple-minded signature algorithm is one in which C signs a document, M, by encrypting it with his private key, RC. If the receiver can recover meaningful information (e.g., an ASCII string) by decrypting a message using PC, the receiver can conclude that the message could have been generated only by C, since only C knows RC. This technique assumes that the receiver knows C’s public key, PC. Note that the message is not hidden. Message hiding is not the purpose of a digital signature protocol.

As far as asymmetric encryption is time consuming, some function, f, of M is computed – generally a hash – that produces a result that is considerably smaller than M itself. f(M) is also called a message digest of M. f(M) is encrypted with RC and referred to as a digital signature, which is transmitted along with M.

Thus, C sends two items, and M, to the receiver. The receiver decrypts the first item using PC and then compares the outcome with the result of applying f to the second item. If the two are the same, the receiver should be able to conclude that M could have been generated only by C. To safely allow such a conclusion, however, we must deal with some other issues.

# Digital Signatures (Cont 1)

Consider an intruder that listens to and copies a signed transmission, (,M), from C.

1. The intruder might use the signature  to sign a different message, M’, in an attempt to fool a receiver into believing that C sent M’. The intruder can succeed in this attack if it can construct M’ such that f(M)=f(M’). To prevent this attack, f is required to be a one-way function: f has the property that, given an output, y, constructing an argument, x, such that f(x)=y is computationally infeasible. For example, a one-way message digest function might produce an output string that satisfies the following properties:
2. All values in the range of f are equally likely
3. If any bit of the message is changed, every bit in the message digest has a 50% chance of changing.

Property (a) guards against the possibility of finding an M’ such that f(M)=f(M’) , because all values are equally possible. Property (b) ensures that f(M) and f(M’) are not the same simply because M and M’ are related or similar messages.

Under these conditions, it is extremely difficult for the intruder to construct a message M’ such that f(M)=f(M’) and hence to find a message, M’, to which the signature  can be attached. Furthermore, the intruder cannot forge the signature that can be used with M’, , since he does not know RC.

1. The intruder might attempt to copy and then resend a signed message a second time, in what is referred to as a replay attack. A replay attack can be dealt with by having the signer construct a timestamp and include it in M. Since the digital signature is calculated over the entire message, the intruder cannot change the timestamp. Assuming that clocks at all sites in the network are roughly synchronized, if the receiver keeps a list of the timestamps of all recently received messages and rejects arriving messages containing timestamps in the list (recall that timestamps are globally unique), it can detect a message’s second arrival. The key point here is that a particular timestamp is never used twice. Hence, a replay attack can also be dealt with if each message has a unique sequence number.

# Secure Sockets Layer Protocol: Certificates

Servers (perhaps representing business) that want to authenticate themselves to other parties as part of an Internet transaction can use a certification authority, which acts as a trusted third party.

A CA uses public-key encryption to generate certificates, which certify the association between a principal’s name (e.g., Macy) and its public key. The certificate contains (among other items) the principal’s name and public key, and it is signed with the private key of the CA. Since the CA’s public key is well known, any process in the system can determine validity of the certificate. Hence, if a client wants to communicate securely with Macy, it can encrypt a message using the public key found in a valid certificate containing the name “Macy” and be certain that only a process with knowledge of Macy’s private key will be able to decrypt the message. Certificates thus solve the problem of distributing public keys reliably, which is the key distribution problem for asymmetric encryption.

Any Internet server, S, that wants to obtain a certificate from a CA first generates a public and private key and then sends the public key, plus other information, to the CA. The CA uses various means to verify the server’s identity (perhaps by communicating with personnel at the server’s place of business by phone and ordinary mail) and then issues it a certificate containing, among other items,

* The CA’s name
* S’s name
* S’s URL
* S’s public key
* Timestamp and expiration information

The CA signs the certificate and sends the signed certificate to S; S verifies correctness of the certificate.

# SSL Protocol

The Secure Sockets Layer (SSL) protocol uses certificates to support secure communication and authentication over the Internet between a client and an Internet server (or between servers). By using certificates, SSL is able to eliminate the need for an online key server (as in Kerberos), which can be a bottleneck in transaction systems that process thousands of transactions per second.

A goal of SSL is to authenticate a server to a client. Since this is done using a certificate, each server that wants to be authenticated must first obtain a

certificate. Clients, on the other hand, are not generally registered with certification authorities and hence do not have certificates or the encryption keys associated with them. A logged-in client is typically represented by a browser, which usually does not have a private key of its own. Rather, the browser contains the public keys of all certification authorities that have made arrangements with that browser’s vendor. The browser does not actually communicate with a CA during the SSL protocol; nor does a CA know any private information about a browser.

The SSL protocol authenticates the server to the client and establishes a session key for their use. A browser engages in the SSL protocol when it connects to a server whose URL begins with https: (instead of http:), which indicates an SSL-encrypted HTTP protocol.

Assume that a browser, C, connects to a server, S, that claims to represent a particular enterprise, E (for example, Macy’s). In this case, the protocol consists of the following steps:

1. S sends C a copy of its certificate signed by the CA – in the clear
2. C validates the certificate’s signature using the CA’s public key (included in its browser) and hence knows that the public key in the certificate belongs to the enterprise named in the certificate.
3. C generates and sends to S a session key encrypted with the public key in the certificate.

Note that C, not S, generates the session key because, at this point in the protocol, C can communicate securely with S using the public key in the certificate, but S cannot communicate securely with C. Once the session key has been established, C and S can use it to exchange encrypted messages.

In many applications, S does want to ensure that it is talking to a particular client. One way to provide such authentication is for client and server to agree on a password, which the server stores and client supplies after the session key is established.

# Passport: Single Sign-On

Microsoft Passport is an Internet protocol that uses an authentication server, A, to implement single sign-on. A stores the password of each customer, C, and a symmetric encryption key, , for each server, S, that has registered for its service. It also stores a symmetric key, , for its own use. In simplified form, the protocol consists of the following steps:

1. When S wants to authenticate C, it sends a page to C’s browser that contains A’s address. The page is redirected (redirected pages are not displayed) from C to A. The effect is as if, after receiving the page, C had clicked on A’s address

Redirection can be made by, for example,

<meta http-equiv="Refresh" content="15; URL=../action/redirect.html"/> (see <http://www.w3.org/WAI/UA/TS/html401/cp0305/0305-REDIRECT.html> for more details)

1. A sends a page to C’s browser requesting C’s password
2. C enters its password and clicks the submit button. An SSL session is established between C and A, and C sends its password to A using the session key established as a part of the SSL protocol
3. A verifies that the password is correct
4. A sends a page and a cookie to C. The page states that C has been authenticated and is redirected to S. It is encrypted with , and hence S can verify that it came from A. The cookie also states that C has been authenticated. It is encrypted with  and placed on C’s browser. Its use will be explained below.
5. S sends a page to C that includes a (second) cookie to be placed on C’s browser encrypted with a key known only to S. Thus if C returns to S’s site, S can retrieve the cookie and determine that C was previously authenticated

Suppose that C later visits a different server, S’, that also offers Passport authentication, and S’ asks A to authenticate C. After step 1 of the procedure, A can retrieve the cookie it previously put in C’s browser and hence knows that C has been previously authenticated. A can now implement single sign-on. A skips steps 2 through 4 (it does not have to ask again for C’s password) and executes an abbreviated version of step 5 (it does not have to place another cookie on C’s browser).

A’s cookie is similar to a Kerberos ticket, but the password protocol does not have the extra security offered by a Kerberos authenticator and hence is subject to some of the attacks that the authenticator addresses.

For example, A leaves a cookie on C’s browser after C completes its interaction with S. If C’s interaction originates from a public terminal (for example, in public library), a subsequent user of the terminal might (perhaps inadvertently) be authenticated as C. Kerberos deals with this type of threat by requiring that the client constructs an authenticator, and this requires that the client knows information contained in the ticket (the session key, and his name). In Passport, however, the cookie can be used on behalf of a client without the client

# Passport: Single Sign-On (Cont 1)

demonstrating any knowledge of its contents. To circumvent this problem, most sites have a button that C can use to remove A’s cookie from the browser.

Cookies are files saved on a client side by a server. Cookies may be used by a server to adjust appearance of sent to a client pages according to the client’s preferences.

In particular, cookies may contain authentication information. Some samples on programming database access using ASP, taken from H.M. Deitel, P.J. Deitel, T.R.Nieto, E-Business & e-Commerce. How to program. – Prentice Hall, 2001, ISBN 0-13-028419-X, Chapter 25 Active Server Pages, pp. 806-818 are given in <http://cmpe.emu.edu.tr/chefranov/Cmpe552-Fall2016/index.htm>