WAP phones

Many WAP phones allow secure sessions to WAP gateways or WAP servers to be established, although at the time of writing WAP phones offer RSA encryption and signatures up to only 768 bits. It is expected, however, that with the introduction of WIMs, WAP phones will support keys lengths of up to 1024 bits.

Currently, there are also issues regarding end-to-end security. The WAP gateways of the telecommunication companies required for connecting WAP phones to application servers terminate the secure WTLS connection to the device at the telecommunication company and establish an SSL connection to the application server. As a work-around, WAP phones that can be configured manually can be set up to connect directly with a WAP gateway within the application provider's secure domain, but this is very inconvenient for users. The solution to this problem will be a redirection scheme that allows the application server to redirect a WAP phone that originally connects via a telecommunication company's WAP gateway to reconnect via a trusted gateway instead; in the long term, a Transport Layer Security (TLS) subset may be used instead of WTLS.

Another security issue is the over-the-air configuration capability of some WAP phones. Intended to allow telecommunication companies to change the configuration of phones in the field if required, this feature can potentially be misused by attackers. One possible attack is to send a configuration message to a phone that changes the WAP gateway to which the phone connects to a malicious one. When the user selects the WAP services after the change, the user's phone connects to the malicious gateway, which may display wrong information and obtain user IDs and passwords from the user.

For more details about WAP and WAP security, see Chapter 6.

PDAs

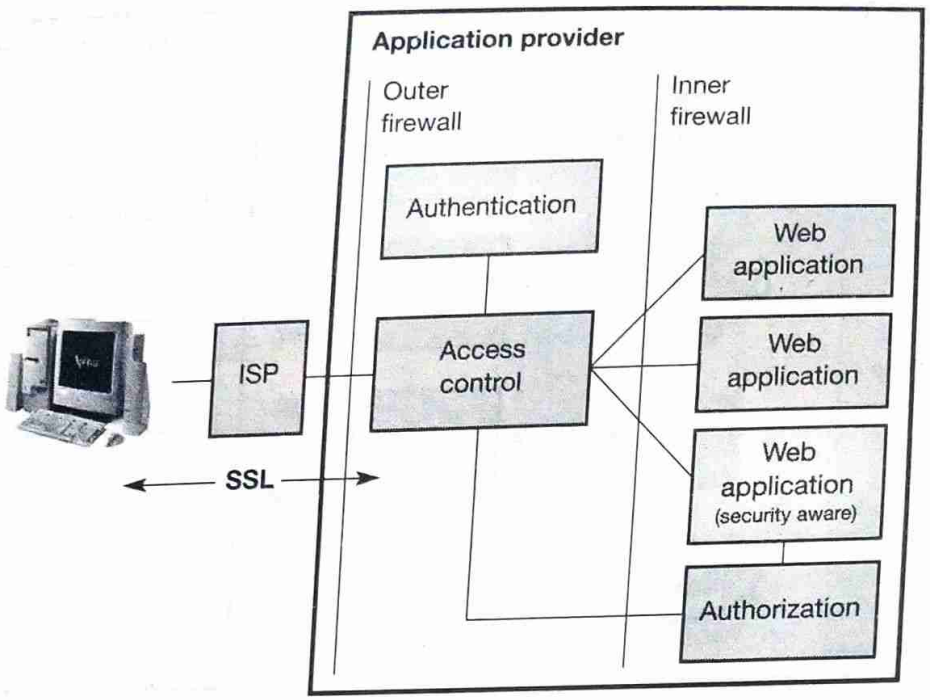
Many PDAs allow downloading and installing of arbitrary software, and thus are prone to Trojan horse attacks. PDAs that have no memory protection to isolate applications from each other are especially vulnerable, because a Trojan horse can easily access data owned by other applications.

As many PDAs do not have a powerful processor in favour of battery life, they support only relatively small key sizes for encryption of session keys and digital signatures.

4.2.3 Server-side security

Pervasive computing brings new requirements to server security. If an application provider needs to support only PC clients, things are quite easy. An applications provider can set up an outer and an inner firewall, place central access control in the 'demilitarized zone', and deploy the Web applications behind the inner firewall, as shown in Figure 4.13

Figure 4.13

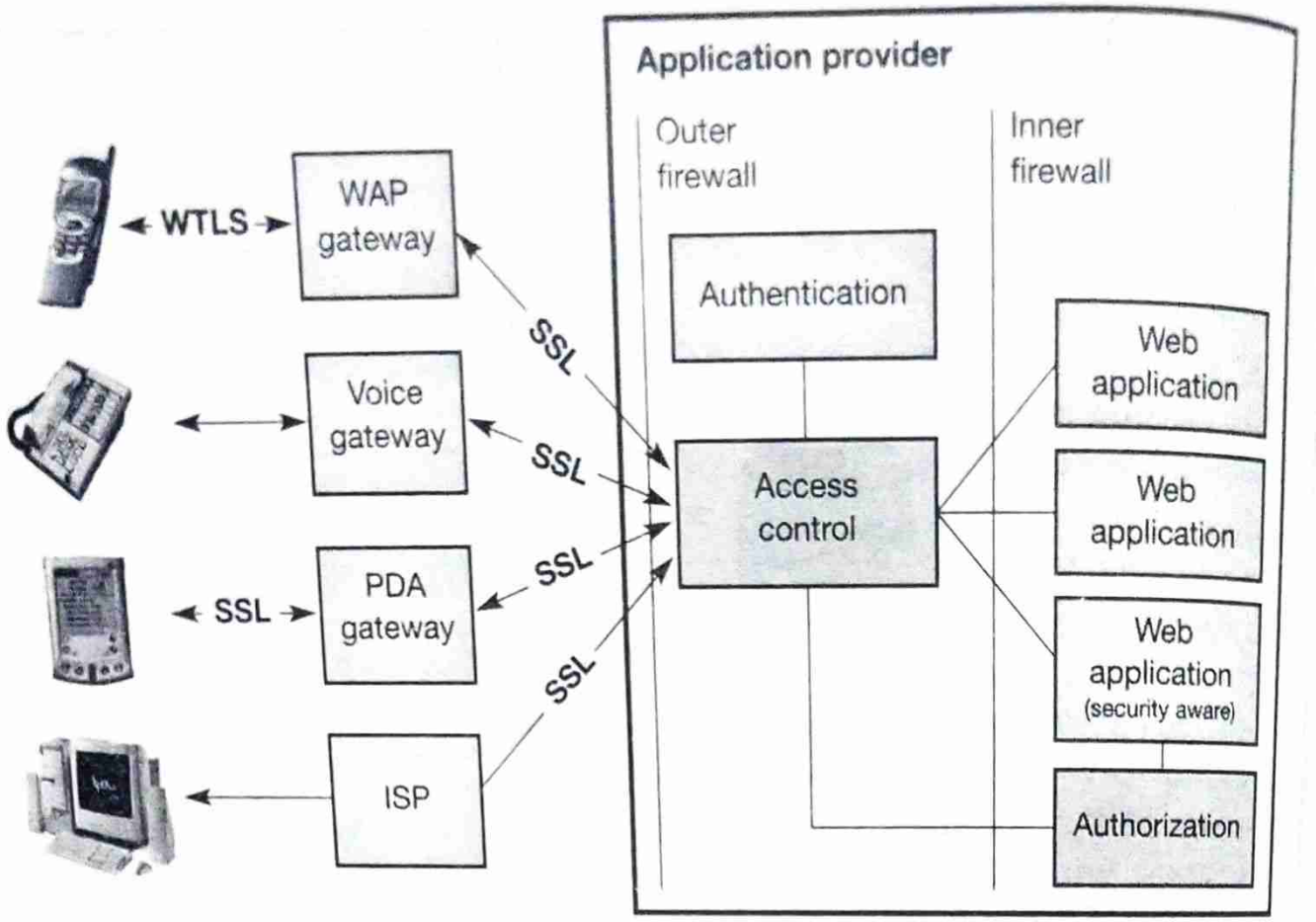


Set-up for a secure Web application

End-to-end security from the PC to the application provider's domain can easily be achieved by using the SSL protocol. After the user's PC has signed on to the ISP, the ISP acts as a pass-through for the messages exchanged between the client PC and the application provider's servers. For client authentication, form-based authentication, HTTP basic authentication, or SSL client authentication can be used. Most Web applications found in the Internet today use form-based authentication.

Additionally, to enable applications to be used from pervasive computing devices such as WAP phones, PDAs, and voice-only phones, appropriate gateways are required. If the application provider – as well as the users of the applications – trusts the gateway providers (e.g. a mobile service provider who operates a WAP gateway), the infrastructure required at the application provider's site is similar to the previous case (see Figure 4.14). Access control and authentication, however, must now be able to handle different kinds of client devices and their markup languages (e.g. when form-based authentication is used to authenticate users), and four different versions of the login form may be required: an HTML form for PC clients, a WML form for WAP clients, a VoiceXML form for voice-only phones connecting via a voice gateway, and a very simple HTML form for PDAs.

This set-up does not provide end-to-end security. In the gateways, all data exchanged between the different devices and the application provider temporarily needs to exist in the clear so that it can be converted.



A pervasive Web application using external gateways

Certificate-based client authentication supported by some devices cannot be exploited this way. For example, a WTLS connection initiated from a WAP phone is terminated at the WAP gateway, thus the application provider will get the SSL certificate of the WAP gateway, but not the original WAP certificate provided by the client itself.

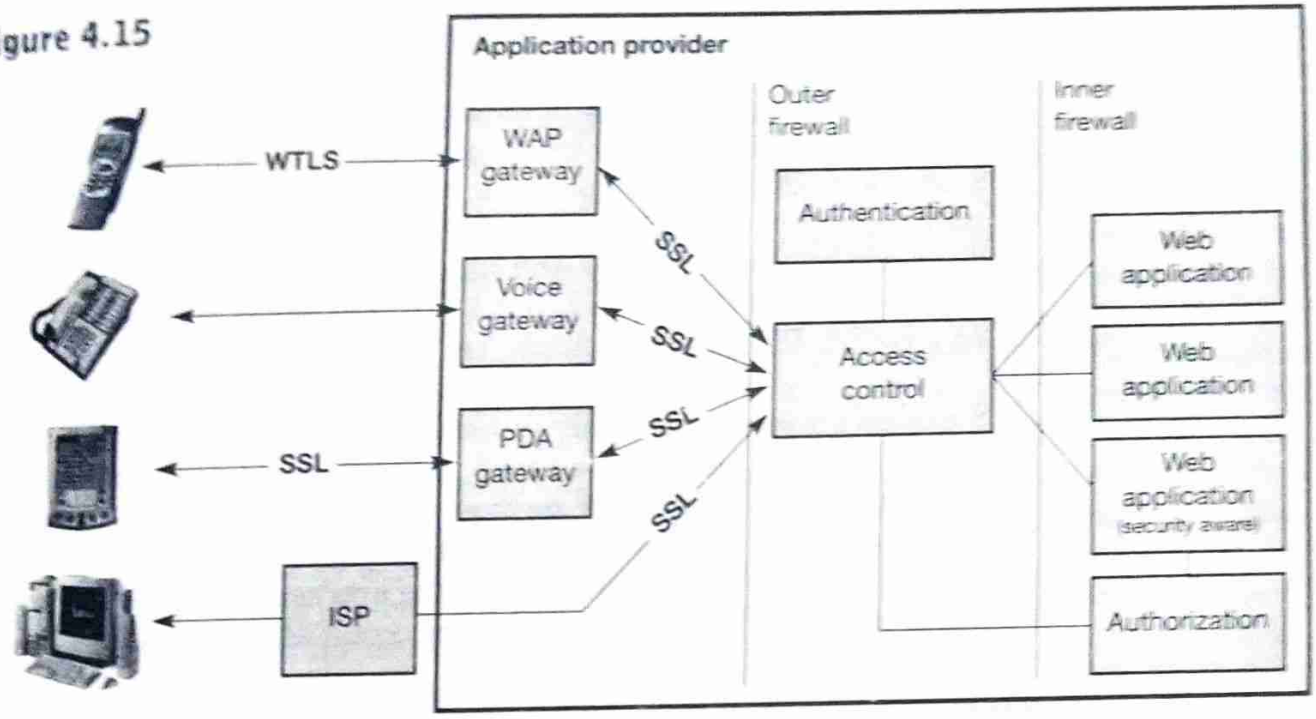
If an application provider wants to have end-to-end security, and to be able to obtain client certificates, it has to operate its own gateways, as shown in Figure 4.15. In this case, the devices connect directly to the appropriate gateways at the application provider's site.

A user with a WAP phone either dials in directly, connecting to the application provider's WAP gateway, or dials into and connects via a third-party WAP gateway and then is redirected to the local WAP gateway by the access control component. In each case, an end-to-end secure connection can be established between the WAP phone and the WAP gateway before data begin to flow.

A user with a voice phone dials in directly to the application provider's voice gateway. Over the connection between the phone and the gateway, voice is transmitted in clear. The security of communication depends on the security of the phone lines.

PDA users can dial in directly to the application provider's PDA gateway. The connection between the PDA and the PDA gateway can be secured using SSL as well as the connection between the PDA gateway and the access control server.

Figure 4.15



A pervasive Web application using own gateways

4.2.4 Cryptographic algorithms

In this section, we give an overview of the most relevant cryptographic algorithms. There are two important classes of cryptographic algorithms that we want to present – symmetric algorithms and asymmetric algorithms. While symmetric algorithms use the same key for encryption and decryption of data, asymmetric algorithms use two keys – what one key encrypts, the other one decrypts. We aim to give only a basic understanding of cryptographic algorithms. For more comprehensive and detailed information, see Schneier's book *Applied Cryptography*.[7]

Symmetric cryptographic algorithms

Symmetric cryptographic algorithms are fast and can be used to encrypt and decrypt large amounts of data. However, the fact that the same key has to be used for encryption and decryption causes a problem when symmetric algorithms are to be used to ensure privacy of communication – the key itself has to be securely transmitted up front. For information on algorithms not described here or for more details, refer to *Applied Cryptography*.[7]

Data Encryption Standard

The Data Encryption Standard (DES) was developed to protect computer and communications data, initiated by the National Bureau of Standards (NBS) and the National Institute of Standards and Technology (NIST) in 1972. The first request for proposals was issued in 1973, but none of the

submissions met the requirements. A second request was issued in 1974; this time, the NBS received a promising proposal from IBM, which became DES. The DES algorithm is a block cipher, i.e. an algorithm that divides the data to be encrypted into blocks and operates on one block at a time. DES uses a block size of 64 bits and a key size of 56 bits. DES keys are actually represented by 64 bits, but the least significant bit of each byte is used for parity checking only and ignored by the algorithm. The fundamental building block of DES is a substitution followed by a permutation, called a round. DES has 16 rounds, i.e. 16 substitutions and permutations are applied to each 64-bit block of data. Because of its repetitive nature, DES can be implemented easily in hardware. The key size of 56 bits used in the DES algorithm is quite small. Given today's powerful computers, the DES algorithm offers only mediocre security. At RSA '99, a DES challenge was announced where the 56-bit key was broken by special hardware in less than 24 h, although more than the average of 50% of the key space had to be searched by a network of many cooperating computers.

Triple Data Encryption Standard

Triple DES is based on DES but uses 112-bit keys. The key is divided into two 56-bit keys, K_1 and K_2. The data to be encrypted are encrypted under K_1, decrypted under K_2, then encrypted again under K_1. This is also known as the encrypt–decrypt–encrypt (EDE) mode. To decrypt, the cipher is decrypted using K_1, encrypted using K_2, and decrypted using K_1 again. Triple DES offers a very high level of security. Brute force attacks, which are possible against DES, are not feasible against Triple DES because the key space to be searched grows exponentially with the size of the key, i.e. finding a 112 bit key by exhaustive search takes 2^{56} times longer than finding a 56-bit key.

The Advanced Encryption Standard

The Advanced Encryption Standard (AES) will be the successor of the DES. Several algorithms have been proposed for the new standard. Of five finalists, NIST chose Rijndael as the proposed algorithm. Rijndael is a block cipher with a variable block length and key length. It allows the use of keys with a length of 128, 192, or 256 bits to encrypt blocks with a length of 128, 192, or 256 bits. All nine combinations of key length and block length are possible. It is possible to extend both block length and key length to multiples of 32 bits. Rijndael can be implemented very efficiently on a wide range of processors and in hardware.

Public-key algorithms

The basic idea that led to public-key algorithms was that keys could come in pairs of an encryption and a decryption key, and that it could be impossible to compute one key given the other. This concept was invented by Whitfield Diffie and Martin Hellman,[8] and independently by Ralph Merkle.[9]

Since then, many public-key algorithms have been proposed, many of them insecure or impractical. All public-key algorithms are very slow compared with secret-key algorithms. The well-known RSA algorithm, for example, takes about 1000 times longer than DES when implemented in hardware, and 100 times longer in software, to encrypt the same amount of data. However, public-key algorithms have a large advantage when used for ensuring privacy of communication: they use different keys for encryption and decryption. The private key may be known only to its owner and must be kept secret. It may be used for generation of digital signatures or for decrypting private information encrypted under the public key. The public key may be used for verifying digital signatures or for encrypting information. It does not need to be kept secret because it is infeasible to compute the private key from a given public key. Thus, receivers or signers of messages can post their public keys to a directory, where anybody who wants to send a message can look them up. Each entity in the network needs to store only its own private key, and a public directory can store the public keys of all entities, which is practical even in large networks.

RSA

Ron Rivest, Adi Shamir, and Leonard Adleman invented the RSA algorithm.[10,11] It is based on the difficulty of factoring the product of two large prime numbers. RSA uses key pairs, where the public key consists of a modulus m and a public exponent e, and the private key consists of the same modulus m and a private exponent d. The two keys are generated from two randomly chosen large prime numbers, p and q. To assure maximum security, the lengths of these numbers should be equal. The box below shows the computation of the numbers needed. As the problem of

RSA in detail

The modulus m is computed as the product of the two primes $m = pq$

Next, an encryption key e is chosen so that e and $(p - 1)(q - 1)$ are relatively prime.

The decryption key d is chosen so that

$ed = 1 \pmod{(p - 1)(q - 1)}$ or

$d = e^{-1} \bmod ((p - 1)(q - 1))$

Let x be the plain text with the same size as the modulus and y the cipher text.

Then the formulas for encryption and decryption are as follows:

$y = x^e \bmod m$

$x = y^d \bmod m$

More details on the mathematical background of the RSA algorithm can be found in Cormen *et al.*[12]

factoring two large prime numbers is very hard to solve, it is infeasible to compute the private key from the public key if the primes multiplied to obtain the modulus are big enough. Today's smart cards usually offer key sizes between 512 and 1024 bits. A modulus size of 512 bits is not considered secure. A modulus size of 1024 bits is considered to offer a reasonable level of security for applications such as digital signatures and encryption for documents.

Digital Signature Algorithm
NIST proposed the Digital Signature Algorithm (DSA) in 1991 for use in the Digital Signature Standard (DSS). The security of the algorithm relies on the difficulty of computing discrete logarithms in a modulus whose length defines the key size. The algorithm was designed by the NSA, which along with the proposed key size of only 512 bits led to criticism regarding its security. In 1994, the standard was issued with a variable key length from 512 to 1024 bits. For more information and the mathematics behind the algorithm, see Cormen *et al.* (1989).[12]

Elliptic curves
Elliptic curves were first proposed for use in public-key cryptosystems in 1985.[13,14] Algorithms using elliptic curves are faster than RSA or DSA and require smaller key sizes for the same level of security. Elliptic curves over the finite field $GF(2^n)$ are especially interesting, because they allow for efficient implementations. The advantages of elliptic curves make them good candidates for use on small pervasive computing devices because the computations can be conducted without powerful processors.

4.3 Device management

With millions, or soon even billions, of phones and PDAs in use, system and device management is becoming a major problem. Approximately eight million PDA units are in use at the time of writing, and 38 million are expected by 2003. The numbers for mobile phones are even more impressive: between 2003 and 2005, the number of phones deployed worldwide will exceed 1 billion. By 2004, 70% of new cellular phones and 80% of new PDAs will offer some form of access to the Internet. Meta Group predicts that by 2003, 40% of corporate users will be using wireless devices. And these numbers don't even include the future devices that will all be connected, such as cars and home appliances.

This section explains the major challenges of device management, discusses an example of device management – software distribution – and describes some approaches for solving the device-management task.

4.3.1 Device management challenges

Incorporating pervasive devices into existing business models presents some serious challenges:

- tracking the device location;
- device–user relationship (is the device used by only one user, or by several users?);
- version control of devices and software that are out in the field;
- software updates of existing devices;
- installation of new software on existing devices;
- providing secure access to device information.

Ignoring these issues will result in increasing costs as the number of devices per user is increasing. A study by Gartner Group shows that the average cost per hand-held device is approximately $2700 per device per year (including technical support, asset tracking, and synchronization).

Therefore, getting out millions or billions of connected devices is not enough. There needs to be a device-management system in place to deal with the various issues involved.

4.3.2 Software distribution

Software distribution deals with the problem of getting new software, or updates of existing software, to the devices that are out in the field. To accomplish this, a device-management system must take the following issues into account:

- *Hardware capabilities.* When downloading new software to a device, you need to know a few things about the device hardware, including processor type, available memory, and any other hardware features needed to select the appropriate software version to download (e.g. display or connectivity capabilities).
- *Hardware version management.* As pervasive computing devices are replaced in very short cycles, a device management needs to keep track of the different hardware versions to select the corresponding version of the software for downloading.
- *Software version management.* Software is changing even more rapidly than hardware. Therefore, software version management is a must for a device-management system. The software that needs to be managed includes not only application software, but also the operating system. The management system must keep track of all installed software versions on one specific device. Only when this information is available

is it possible to decide whether new software or an updated version of existing software will run without problems on this specific device.

- *Library management*. Libraries are used by different programs and therefore need special attention. The device-management system needs to know not only which libraries and versions are installed on the device, but also the version of a library needed for each program version. This can result in situations in which different versions of the same library are on one device, as different programs on the device may need different versions of that library. The device-management system should also be able to detect when a library is no longer needed on a device and can therefore be deleted.
- *Devices are not always connected*. As mobile devices are not always connected to the network, device-management systems need to keep track of devices that have already received the new or updated software, and devices that still need to get it.
- *Insecure connections*. As many wireless connections are insecure, the distributed software and data need to be protected (e.g. with encryption).
- *Unstable connections*. Connections to mobile devices can break, either for technical reasons, or because the user switches off the device. Software distribution therefore needs to take this into account and provide mechanisms for detecting incompletely transferred software and retransmiting missing parts.
- *Operating system updates*. Operating system updates are a very sensitive.task, as we do not want to have a situation in which the device does not work any more after the update. This means that there must be a mechanism in place to ensure that the device is still operational after the update. Common methods to achieve this are a rollback mechanism, in which the update fails to get back to the original operating system version, or to update only modules rather than the complete operating system.

As this list shows, software distribution for mobile devices is a complex task and requires a lot of preparation by the provider of the software distribution service.

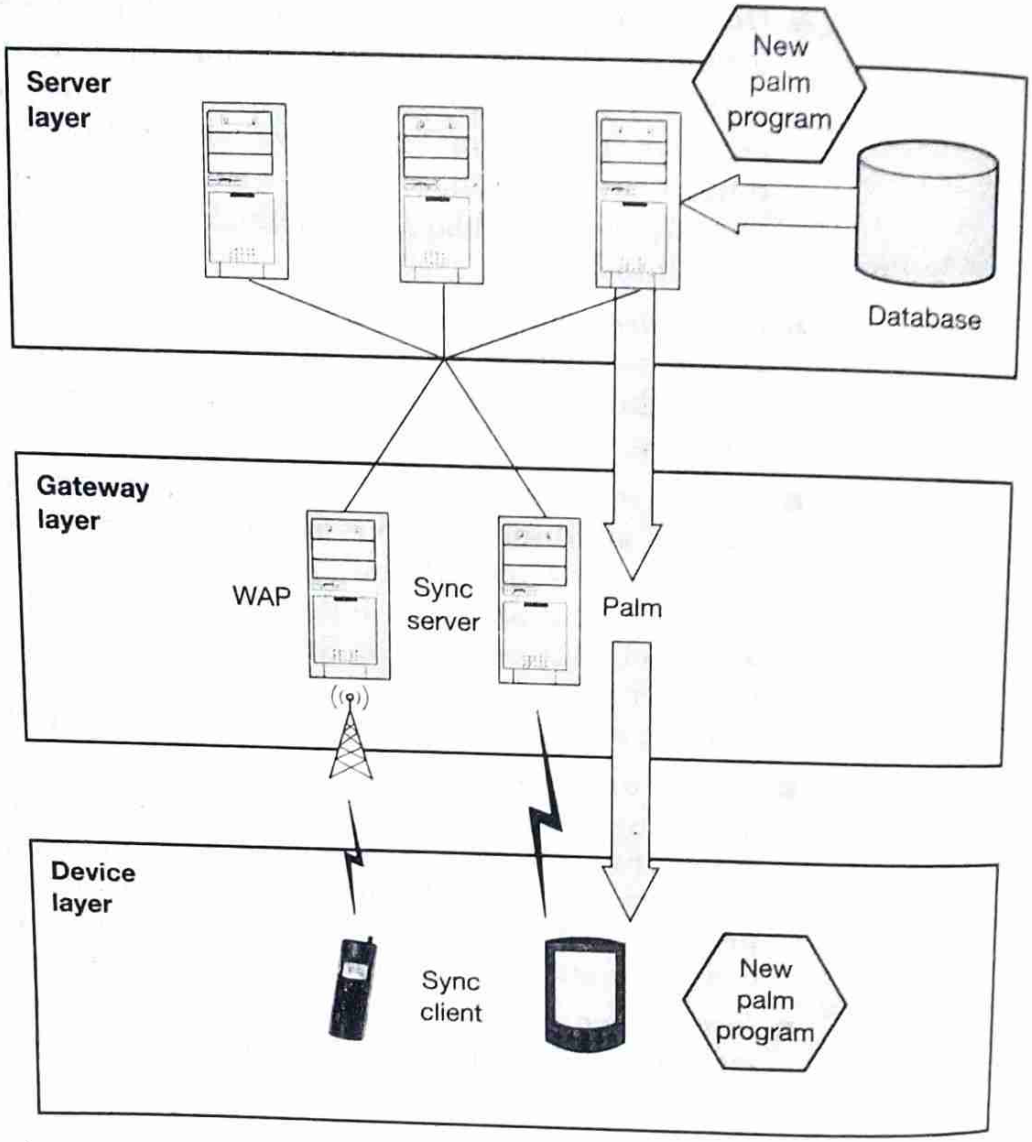
4.3.3 Approaches

There is no single solution to solve all the issues mentioned above. We need to combine several different technologies for solving the software distribution problem for pervasive devices. One basic prerequisite to achieve device management is that the devices can be separated clearly from each other (e.g. with a unique identifier). Some techniques to solve the software download problem include:

- *Hardware capabilities.* There are standards for retrieving hardware device capabilities. The most popular for pervasive computing devices is the CC/PP standard (see Chapter 6). Unfortunately, there are no pervasive computing devices available at the time of writing that implement this standard. This should change in the near future, allowing for a standardized method to retrieve the hardware capabilities.
- *Hardware and software version management.* The device-management system needs to store in a database all hardware and software versions out in the field, and all allowed combinations of hardware and software.
- *Library management.* The device-management system must also keep track of all library versions and maintain a list correlating different software versions with the corresponding library version. In addition, it must have a list for each device, in which all libraries on that device with their versions are stored. The library management should be able to detect when a library is no longer needed on a device, and initiate deletion of that library to reuse the memory on the device.
- *Devices are not always connected/unstable connections.* As devices are not always connected, and connections are unstable, the management system needs to keep track of all devices and their installed software. To achieve reliable software distribution, it can utilize transaction protocols and use rollback techniques on the device to get back to a consistent state after an unsuccessful software install.
- *Insecure connections.* Using standard authentication and encryption methods on the application layer, the device management can ensure a secure communication over insecure transport protocols.
- *Operating system updates.* Operating system updates can be achieved securely when using modular operating systems, where parts of the operating system can be exchanged at runtime.

When building a device-management system, the server side as well as the pervasive computing devices need to be included in the system design. A solution for a device management system can be seen in Figure 4.16. In this solution, the data management in the server layer is solved with databases or IT management systems, such as Hewlett-Packard's OpenView, Tivoli from IBM, or Unicenter TNG from Computer Associates. The gateway layer consists of a device gateway per device category with a synchronization engine. Finally, the devices in the device layer have a corresponding synchronization client to talk to their gateway. Synchronization can easily solve a lot of the software update problems

Figure 4.16



Device-management system example showing the installation of a new program on a Palm Pilot

4.3.4 Summary

Device management is a big problem for service providers or business units that support pervasive computing devices. If there is no efficient device-management system in place, this can dramatically increase the total cost of ownership of pervasive computing devices.

As shown in the example of software distribution, a device-management system will face a lot of challenges; it therefore needs to be designed well and to be in place before the number of devices in the field becomes unmanageable.

5 Web application concepts

In this chapter we explain how PCs can be connected to Web applications through the Internet. Although some concepts and technologies described here can be considered classical and are well known, we will revisit them briefly. In this way, they can serve as references to compare with the newer concepts and technologies presented in subsequent chapters.

We give an overview of the history of the World Wide Web (WWW), as well as the relevant concepts, protocols, and standards for communication between Web clients and servers via the Internet. One special topic we cover here is transcoding, the transformation of content to device-specific markup.

We discuss Web application security issues, and present possible solutions ranging from usage of the standard HTTPS protocol to client authentication schemes for the Internet. As well as the typical client authentication methods supported by today's browsers, we also discuss the use of smart cards for secure user authentication.

5.1 History of the World Wide Web

Although the Internet existed long before the WWW, it became popular only when a convenient, graphical way to access it became available through the WWW, and Web browsers were developed to access its information. The history of the WWW began at the European Center for Nuclear Research (CERN). CERN runs several particle accelerators across Europe, and employs large teams carrying out research in physics. The teams typically have members from various European countries who have to share a constantly changing set of documents, drawings, blueprints, etc. The researchers were originally connected via the Internet, but had no convenient way to publish information in order to share it with researchers in other locations. To allow for effective collaboration between these teams, CERN physicist Tim Berners-Lee in March 1989 made the initial proposal for a web of linked documents. A text-based prototype was operational one year later and was presented at the Hypertext '91 conference in San Antonio, Texas. The first browser with a graphical user interface, Mosaic, was developed at the National Center for

Supercomputing Applications (NCSA) and was released in February 1993. It was so popular that Marc Andreessen left the NCSA and formed a company named Netscape Communications Corporation, with the goal of developing Web software for clients and servers. They went public in 1995 with a market value of $1.5 billion.

In 1994, CERN and Massachusetts Institute of Technology (MIT) founded the World Wide Web Consortium (W3C) to develop the Web further, standardize protocols, and reach interoperability. Since 1994, hundreds of universities and companies have joined the consortium. The consortium's documents, and information on its activities, are available on its website.[1]

After the WWW had established itself as the most popular way of accessing information via the Internet, Microsoft entered the arena with its Internet Explorer. Since then, Microsoft and Netscape have fought for dominance in the browser business in the so-called browser war, frequently coming up with new versions that have additional features. Not surprisingly, Microsoft was able to capture a significant part of the market share quickly by bundling its browser with their operating systems at no additional cost.

5.2 World Wide Web architecture

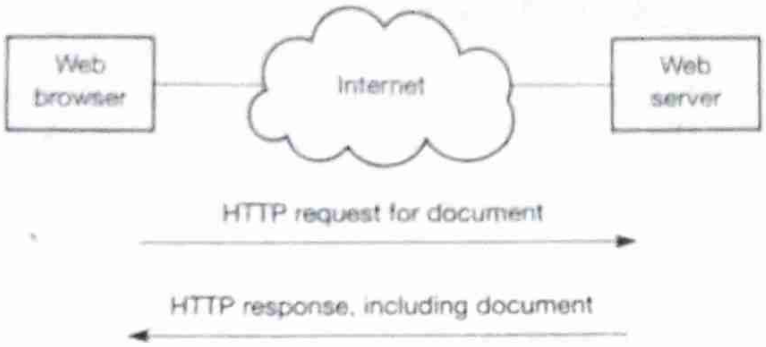
The WWW allows access to linked documents spread across an enormous number of machines connected to the Internet. From the user's perspective, the Web consists of a large number of documents that can be displayed with Web browsers. These documents can be connected to other documents by hyperlinks that establish connections to these other documents, no matter where they are located. When displaying a hyperlink, a browser marks it usually by underlining it or displaying it in a different colour. When the user clicks on a hyperlink, the browser follows the link by requesting the referred document from the referred server.

From the server's perspective, the Web is a network of millions of potential clients. Each website has one or more servers that make documents accessible to clients. When receiving a request from a client, a server retrieves the requested document from hard disk or memory, or dynamically from an application server, and sends it back with the response to that request.

The protocol used between the Web clients and Web servers is HTTP, shown in Figure 5.1.

HTTP is a simple, stateless protocol. The browser sends an HTTP request that specifies the document to be returned. To serve the request, the Web server reads the requested documents and sends it back in an HTTP response. If the document is not available, it sends back an error message. Note that usually, messages are not exchanged directly between

Figure 5.1



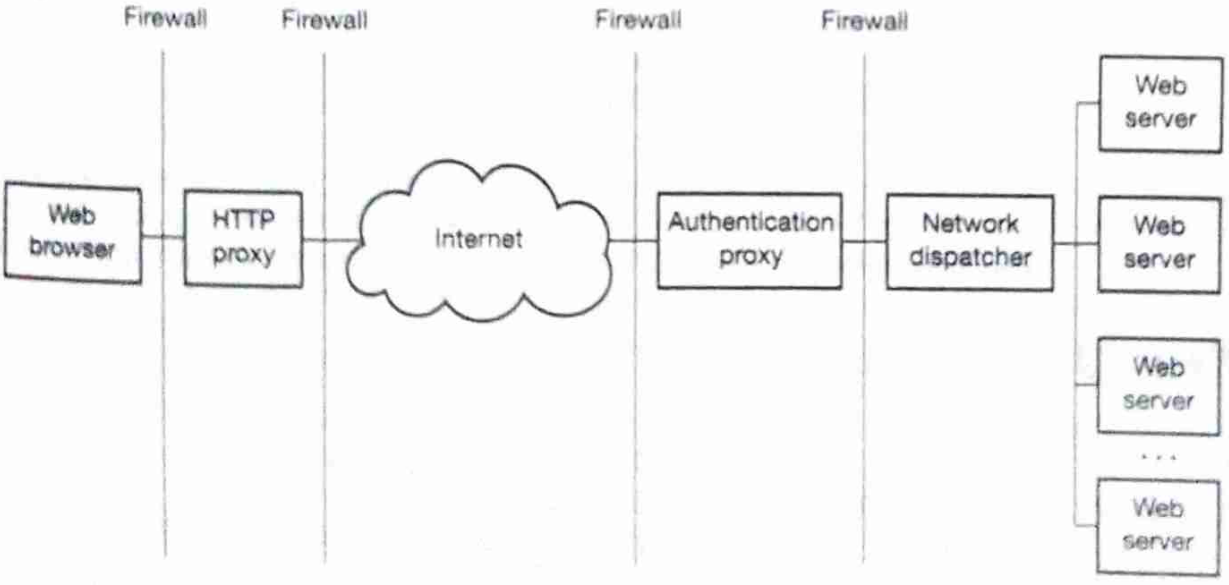
Principles of communication between a browser and a server

the client and the server. In real-life scenarios, several proxies and firewalls might be involved in the communication, as shown in Figure 5.2.

Often, the machine in which the Web browser runs does not have a direct connection to the Internet, but accesses the Internet via a proxy and one or more firewalls. Also, many servers do not have a direct Internet connection. They may be located behind one or more firewalls and a proxy for authentication of clients or caching. Websites that have to handle a large number of hits often take advantage of a network dispatcher for load balancing between several Web server instances.

Although HTTP does not have restrictions on the types of documents that may be transmitted, most documents on the WWW use HTML, which was developed specifically for the Web.

Figure 5.2



Real-life example of communication between a client and a server

5.3 Protocols

In this section we give a brief overview of the most important protocols in the Internet, as shown in the protocol stack depicted in Figure 5.3. We'll start with a brief introduction to the network layer and transport layer protocols TCP/IP, followed by information on the security protocols SSL and TLS and the application layer protocol HTTP.

5.3.1 TCP/IP

The Internet has two main transport layer protocols: the connection-oriented TCP and the connectionless user datagram protocol (UDP). We focus on TCP here, as it is the basis of HTTP, which we will discuss in subsequent sections. TCP provides a reliable end-to-end byte stream over an unreliable internetwork. As internetworks consist of different parts that may have very different bandwidths, topologies, packet sizes, and other parameters, TCP was designed to dynamically adapt to these differences and tolerate many kinds of failures.

5.3.2 SSL and TLS

SSL is a protocol for secure connections via the Internet on top of TCP/IP. It was originally developed by Netscape. Over time, it evolved through several versions to SSL 3.0, supported by today's software, and finally into TLS, the successor of SSL standardized in the Internet Engineering Task Force (IETF). This section will discuss TLS 1.0.[2] The primary goal of TLS is to provide privacy and data integrity of messages exchanged between two communicating parties over an untrusted network, like the Internet. In addition, TLS also allows for mutual authentication of the communicating parties.

The TLS protocol consists of several layers. The lowest layer is the TLS Record Protocol, which assures privacy and reliability of connections and is used for encapsulating higher-level protocols, such as the TLS Handshake Protocol. This protocol provides connection security, which ensures that a peer's identity can be authenticated using public-key cryptography, and that the negotiation of a shared secret is secure and reliable.

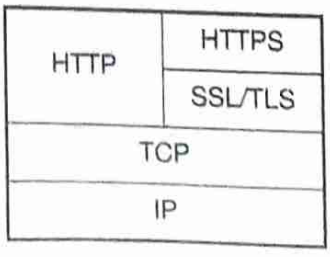


Figure 5.3 *Protocol stack*

When TLS is used to secure client–server communications, in most cases the handshake protocol only conducts server authentication. During the handshake protocol, the server transmits its certificates to the client. The client validates the server certificates and extracts the public key from one of the certificates. This key is used to encrypt the session key for further communications. TLS allows for optional client authentication. During the handshake protocol, the server can send a certificate request that requires the client to send its certificate and a certificate verify message to the server. The server can authenticate the client by validating the obtained certificate and using the public key from that certificate to verify the client's signature contained in the certificate verify message.

5.3.3 HTTP

HTTP is a generic, stateless application level protocol.[3] HTTP can be used for many tasks beyond its original use for transmitting hypertext. Examples of other tasks are use as name servers, in distributed object management systems, and in service-oriented architectures as a basis for the SOAP protocol (see Chapter 9). HTTP supports typing and negotiation of data representations, such that systems can be implemented independently of the data they transfer. HTTP defines requests that are sent from user agents to servers and responses that are sent from servers back to user agents.

Requests

HTTP requests include a request line, headers, and, optionally, a message body, e.g. data to be stored or processed at the server. The request line includes a method, a uniform resource identifier (URI) identifying the resource on which to apply the request, and the HTTP version as shown in Example 5.1.

xample 5.1 The request line

```
GET http://www.w3.org/pub/WWW/TheProject.html HTTP/1.1
```

HTTP defines eight methods:

- *GET* is used to retrieve the file identified by the request URI.
- *POST* requests that the server accepts the data enclosed in the request as a new subordinate of the resource specified by the request URI. It is intended to be used for functions such as annotation of existing resources, posting messages to bulletin boards, and providing data such as the result of submitting a form.

- *HEAD* allows a user agent to obtain a resource's header without the message body.
- *PUT* requests to store data enclosed in the request under the request URI.
- *DELETE* requests that the server delete the resource identified by the request URI.
- *OPTIONS* is used in requests for information about the communication options available on the request response chain identified by the request's URI.
- *TRACE* can be used to invoke a remote loop-back of the request message. It allows a client to see what is being received at the server's side and to trace the request chain between client and server by examining the Via header fields.
- *CONNECT* is reserved by the specification for use with proxies that can dynamically switch to being a tunnel.

To allow clients to pass additional information about requests or the clients itself, an HTTP request can contain request header fields, e.g. `Accept`, `Accept-Charset`, `Accept-Encoding`, `Accept-Language`, `Authorization`, `Expect`, `From`, `Host`, `Proxy-Authorization`, `Range`, `Referrer`, or `User-Agent`. The `Accept` and `User-Agent` fields are especially important with regard to pervasive computing, as they can be used by a server to determine the client capabilities and deliver content in an appropriate form.

HTTP requests may optionally include a message body. The presence of a message body in a request is signalled by the inclusion of `Content-Length` and `Transfer-Encoding` header fields.

Responses

For each incoming request, the servers send back a response that consists of a status line and optionally a message body, e.g. a Web page or data returned by a Web application. The status line consists of the HTTP version, a status code, and its associated textual phrase. The first digit of the status code defines the response's class:

- *1xx – informational:* the request was received and is being processed.
- *2xx – success:* the action was received, understood, and accepted. The code for success is 200 (OK).
- *3xx – redirection:* further action is required in order to complete the request.
- *4xx – client error:* the request contains bad syntax or cannot be fulfilled. Some examples of client errors are 401 (Unauthorized), 403 (Forbidden), 404 (Not Found), and 405 (Method Not Allowed).

- *5xx – server error:* the server failed to fulfill an apparently valid request. Some examples of server errors are 500 (Internal Server Error) and 503 (Service Unavailable).

5.3.4 HTTPS

HTTPS means using the HTTP protocol over a secure connection instead of a plain transport connection (see Figure 5.3). Current practice is to layer HTTP over SSL, distinguishing secured traffic from insecure traffic by using port 443 instead of port 80. In the future, HTTP will be layered over TLS similarly, as described in RFC 2818 released by the IETF.

5.4 Transcoding

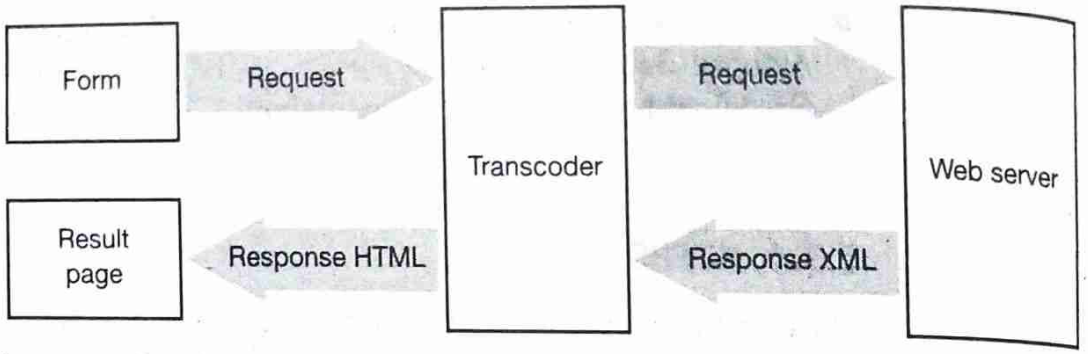
There are many situations in which existing Web applications have to be enabled for pervasive computing devices. It is the expectation that any type of information or service that can be accessed with a Web browser should also be accessible to ever-new waves of diverse pervasive devices. Obviously, content and use need to be adapted to the requirements and constraints of these devices to optimize the user experience – a need that was not been foreseen when the initial application was deployed. One approach for this is transcoding.

Transcoding means transforming certain input formats to different output formats, e.g. transforming content from one markup language to another, transforming images with a high resolution to a low resolution representation, or repaginating a large page into several smaller pages. Transcoders are typically capable of dealing with any kind of input, and are able to generate any kind of output. However, transcoding works best if the input is structured data, for example XML, and becomes very tedious and error prone when markup that is intended for presentation has to be screen-scraped and transformed into other types of markup.

Technically speaking, transcoding performs post-processing of content generated by a Web server in one of two places: in an HTTP proxy that intercepts the HTTP data stream to apply transcoding as required, or within the application server itself. If the Web server produces structured data rather than presentation markup, this black-box approach allows for a clean separation of data as generated by the Web server and presentation generated by transcoding (Figure 5.4)

These methods both have advantages and disadvantages. Deployment of the transcoder in the application server itself allows for use of SSL encryption, transparency for clients, and selective transcoding of particular content subsets, which are not possible when a transcoder is set-up as an HTTP proxy. On the other hand, the HTTP proxy set-up can be used with

Figure 5.4



A transcoder intercepting communication between a client and a Web server

any Web server, while deployment of transcoding within the application server might easily lead to dependencies on particular implementation details of the application server.

The general concept of transcoding does not limit the markup languages of both input and output to any particular set of languages. However, transcoder implementations typically provide built-in support for HTML and XML, while other formats will often require custom code. The decision as to which transformation should be applied to the input content can be based on the type of markup language of the input, the device type and its particular constraints, additional HTTP header fields, network constraints, user preferences, and organizational access policies. Some of this information is provided as part of every HTTP request; the rest is based on configuration of the transcoder.

For HTML input, transcoders are usually capable of dropping or replacing information from the HTML input, e.g. dropping, downsizing, or dithering embedded images, replacing them with a link for separate download, or replacing tables with ordered lists. This is useful if the user agent that requested the HTML document is capable of displaying only a certain subset of the HTML level that the document was authored for. Creating output in other markup languages, e.g. WML, that will contain only a certain information subset from HTML input is usually not possible through a straightforward transformation. To achieve this, decisions must be made by the transcoder regarding which document elements to transform into output and which elements to drop. Some transcoders provide a set of APIs that help in writing code to specify which parts of data in an HTML documents are relevant and how to create output in another markup language. As HTML is focused on presentation rather than structured data, these data need to be extracted from the HTML document through page-specific code, which is tied to one Web page at one point of time – if the Web page changes, the custom code to transcode this page might also have to be changed.

On the other hand, XML input contains tagged data that can be parsed and searched for data elements using XML parsers. Thus, transcoders do not need to provide any particular functionality to handle XML docu-

ments, but rather provide capabilities to select, load, and execute Extensible Stylesheet Language (XSL) stylesheets (see Chapter 9) based on device and user agent types. This is the most flexible approach in that it works for any XML instance, and allows for any kind of transformation that can be performed with the XSL-T programming language. XML is the preferable input format for transcoding.

5.4.1 Transcoding v. device-specific content generation

To a certain degree, transcoding allows existing applications to be used with new devices, whereas device-specific content generation using JSPs is something that has to be implemented as a part of an application itself.

Device-specific content generation is the method of choice when an application is implemented from scratch, or when at least access is available to the back-end system or database on which an existing application relies. Compared with transcoding, this method provides superior performance and scalability, and even allows implementation of different dialog flows optimized for individual devices.

Transcoding can be used to enable an existing application that may not be modified for pervasive computing, or for processing data provided by content syndicators. However, existing applications often do not deliver desirable XML input to the transcoder, so screen-scraping code that is specific to a page at one point of time has to be written. Transcoding to various combinations of screen sizes, markup languages, and information subsets is possible in a generic way, e.g. by application of stylesheets, only when applications provide output in structured data formats that allow for effective post-processing, such as XML.

A good example for transcoding is rendering of information in a portal for different devices. Now that many content syndicators provide news, weather, stock quotes, and other information in XML formats, portals can integrate those data with the respective look and feel for each device simply by supplying different stylesheets for transcoding.

5.5 Client authentication via the Internet

While early Internet applications did not need know user identities, today's e-business applications need to know the identity of users in order to be able to perform business transactions. The required level of assurance of the user's identity depends on the security requirements of Web applications. For non-sensitive Web applications, such as portals, email services etc., simple authentication mechanisms are usually sufficient to indicate the user's identity. Sensitive applications providing high-value transactions on the other side require real proof of user identities. In the following sections we present user-authentication methods for the

Internet, listed in increasing level of security and assurance of the user's identity. We explain smart-card-based authentication, the method that provides the highest level of security, in particular detail.

5.5.1 Basic and digest access authentication

Basic authentication is a simple authentication protocol that has been defined as an extension of the HTTP 1.1 protocol in RFC 2617.[1] Origin servers as well as proxies can use basic authentication to authenticate Web clients.

When a Web client requests a protected resource on an origin server, the origin server returns an error message with the error code `401 (Unauthorized)` and one or more challenges to the client, as shown in Example 5.2.

Example 5.2 The origin server's error message

```
WWW-Authenticate: Basic realm="sample"
```

The client prompts the user for credentials and resends the original request with an additional `Authorization` header field. To perform basic authorization, the client sends the user identification and password, separated by a single colon as a base64 encoded string, as shown in Example 5.3.

Example 5.3 Basic authorization

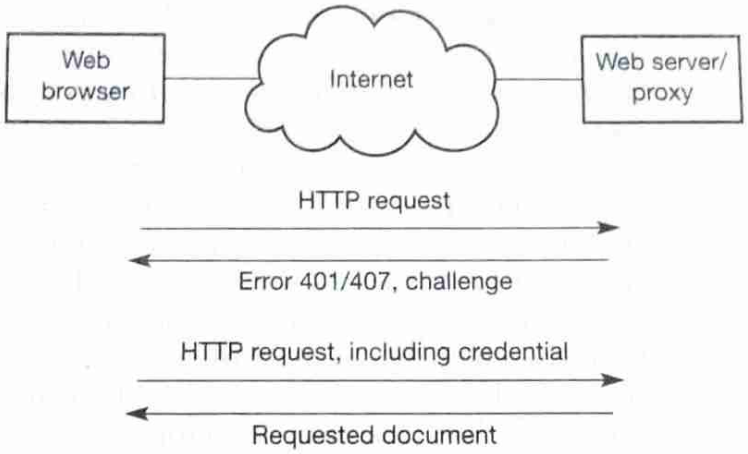
```
Authorization: Basic QWxhZGRpbjpvcGVuIHNlc2FtZQ==
```

Similarly, when a client requests a resource that is protected by a proxy, the proxy returns an error message with the error code `407 (Proxy Authentication Required)` and one or more challenges. The client resends the original request, including an additional `Proxy-Authorization` field. Figure 5.5 shows the messages exchanged during the authentication protocol.

When using basic authentication, the user identification as well as the password are transmitted in base64-encoded clear text. As a result, an attacker may eavesdrop on the communication to obtain passwords that could be used for later access to the protected resources.

To overcome this obvious weakness, the digest access authentication scheme was developed. Clients receive a nonce as a challenge from the

Figure 5.5



Basic authentication

server and send back a message digest over the username, the password, the given nonce value, the HTTP method, and the requested URI. By default, the MD5 algorithm is used to compute the message digest. As only a digest is transmitted over the communication line, and a nonce is included in computation of the digest, it is not possible to obtain the user's password by eavesdropping on the communication or to perform a successful replay attack. This protocol still does not provide the security of Kerberos or client-side private-key systems. It is vulnerable to man-in-the-middle attacks, unless communication between the client and server is secured by encryption.

5.5.2 Form-based authentication

When using basic or digest access authentication, the window displayed by the browser to obtain the required credentials from the user disturbs the user interface flow of a Web application. To avoid this, many applications use form-based authentication in order to give a better-looking interface.

Instead of a simple error message, the server or proxy sends a page with a form that prompts the user to enter their credentials, usually user identification and password. When the user enters the data and presses the form's submit button, the credentials are posted to the server, where they are verified. If this is successful, the user is allowed to access the protected resources. Similarly to basic authentication, form-based authentication is insecure unless communication is protected by encryption.

5.5.3 SSL/TLS client authentication with client certificates

SSL/TLS authentication using client certificates and private keys on the

client side provides a higher level of security than the previously described mechanisms. As public-key cryptography is employed, no secret authentication credentials need to be stored on the server. The server needs only a public key and a directory of registered users.

To authenticate a user, the server sends a random challenge to the client. The client has to sign the challenge using a private key, and sends back the signature together with its client certificate. The server validates the certificate, extracts the public key from the certificate, and uses it to verify the signature over the random challenge. If the signature is valid, then the client may access the protected resources. This method can be considered secure against eavesdropping and stealing of information from the server, but it is prone to Trojan horse attacks that aim to obtain the private key stored on the client.

5.5.4 Authentication using smart cards

As the Internet is used increasingly as a platform for e-business transactions, security becomes a primary issue for Web applications. Many Web applications are too sensitive to be secured appropriately using software-only security mechanisms like the ones described above. To provide a sufficient level of protection for these applications, smart cards can be used to securely authenticate users, and to secure individual transactions through digital signatures. Smart cards are credit-card-sized tokens with a tamper-resistant chip that typically includes a microprocessor, and a small amount of RAM and EEPROM storage. For communication, smart cards use contacts, a contactless communication unit, or a combination of both. Applications cannot simply read or write data on a smart card; they need to communicate with the smart card through application protocol data units (APDUs). When the smart card receives a request APDU, it processes it and replies with a response APDU.

In this section, we explain how smart cards can be integrated with Web applications, and show how secure authentication via the Internet can be achieved using either asymmetric or symmetric cryptographic algorithms on smart cards. We give an overview of the predominant smart card types, and present two important interfaces for smart card applications: PC/SC and the OpenCard Framework (OCF).

Application integration

There are various ways to implement authentication using a smart card; we describe two of these in more detail. One way is to integrate the smart card into the standard TLS/SSL protocol; the other is to use custom pairs of authentication applets and servlets.

TLS/SSL client authentication using a smart card

A smart card can be used for client authentication within the TLS/SSL pro-

tocol by making it accessible to the browser through an appropriate interface, e.g. PKCS#11 for Netscape Communicator or CAPI for Microsoft Internet Explorer. To enable all entities involved in a business application, the users have to be equipped with a smart card, a smart card reader, and software to enable their browser to access the smart card via the reader, i.e. an appropriate CAPI or PKCS#11 provider. The servers hosting the application must be set up to perform TLS/SSL client authentication, and must be connected to an appropriate public-key infrastructure. Only special smart cards that support exactly the cryptographic operations required for TLS/SSL client authentication can be used.

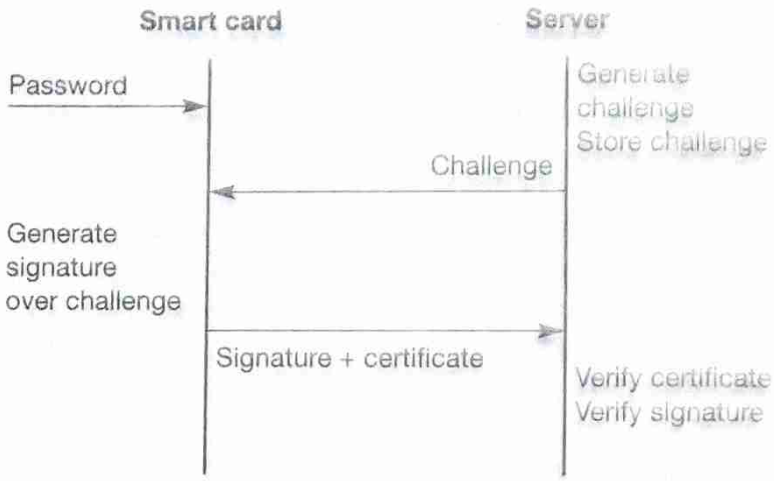
Smart-card-based authentication using an applet and a servlet
Frequently, smart cards are used that do not provide the functions needed for client authentication according to the TLS/SSL protocol. Examples are low-cost smart cards that support only symmetric cryptographic algorithms and thus are not able to perform the public-key algorithms to generate the required digital signatures. In these cases, an authentication protocol based on the capabilities of a particular smart card has to be implemented. A very simple and convenient way to do so is to implement an authentication applet/servlet pair.

Public- v. secret-key-based authentication
In recent years, smart card technology has advanced quickly. It has now reached a state where smart cards integrate easily into public-key infrastructures. Many of today's smart cards provide memory of up to 32 KB to store keys, certificates and information and they have crypto-coprocessors that allow them to generate digital signatures using RSA or DSA algorithms with key lengths of up to 1024 bits or even 2048 bits. However, many applications still use low-cost smart cards that support only symmetric cryptographic algorithms.

Authentication using public-key smart cards
Figure 5.6 shows how smart cards that support public-key cryptography can be used to perform authentication through a challenge–response protocol. The server gives a random challenge to the smart card. The smart card uses its private key to generate a digital signature over the challenge. The digital signature and the certificate associated with the smart card's private key are sent to the server. The server verifies the certificate and then uses the public key contained in the certificate to verify the signature.

Authentication using non-public-key smart cards
Figure 5.7 shows how authentication can be performed using smart cards that do not support public-key cryptography. The server gives a random challenge to the smart card and requests a signature over a card ID and the challenge. Often, a password provided by the user has to be given to the smart card before a signature can be obtained, to ensure that someone

Figure 5.6



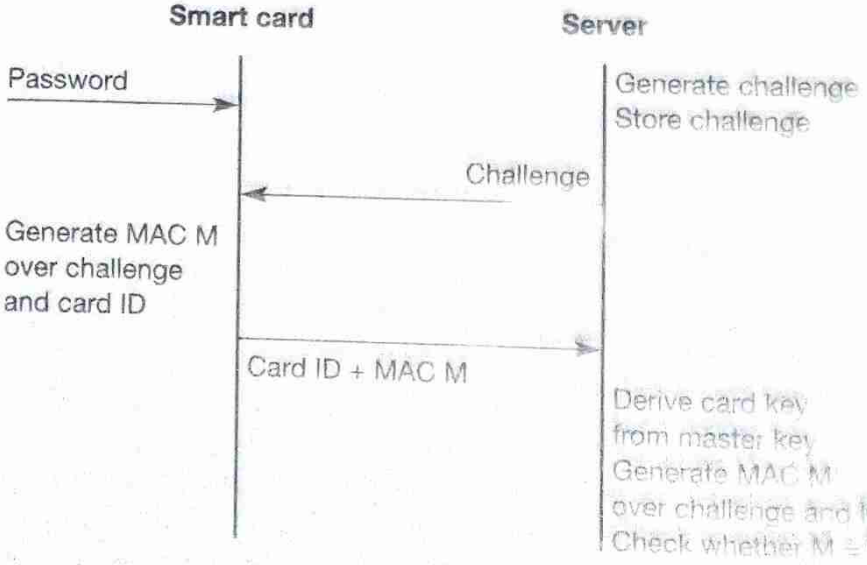
Authentication protocol for public-key smart cards

who steals or finds the card cannot abuse it. The smart card uses a card individual key to generate a message authentication code (MAC) over the card ID, the challenge is obtained from the server, and the ID and the MAC are sent back to the server. The server uses the card ID to derive the card individual key from a master key, and uses that card individual key to verify the MAC sent from the card.

Types of smart card

In recent years, a number makes and types of smart cards have come to market. We can identify three major categories: simple file-system-oriented smart cards without public-key capability, advanced file-system smart cards with public-key capability, and programmable smart cards such as JavaCards and Windows-powered smart cards.

Figure 5.7



Authentication protocol for non-public-key smart cards

Simple file-system smart cards
File system smart cards provide a file system whereby reading and writing of files can be protected by various methods. These cards support only symmetric cryptographic algorithms, such as DES or Triple DES. To use a simple file-system smart card for authentication, a file containing the card ID can be created on the card, and the access conditions for that file can be set so that it can be read with a MAC after a challenge has been provided. This allows a protocol like that shown in Figure 5.7 to be run.

File-system cards with public-key cryptography
File-system cards with public-key cryptography support can store private keys and associated certificates. Key pairs are usually created in the card, and the private key never leaves the card. The private key is used internally only, for generating digital signatures or decrypting session keys or small amounts of data. To use file-system cards with public-key support for authentication, a private key and an associated certificate must be present in the card. This allows a protocol like that shown in Figure 5.6 to be run.

JavaCard
Smart cards that conform to the JavaCard specification[5] can host several applets. Applets are applications that reside on the card and offer an APDU interface to off-card applications. The JavaCard allows off-card applications to select an applet on the card by specifying the applet's application ID. After that, they can send a sequence of APDUs to the selected applet. Applets are implemented using a subset of the Java programming language, relying on Java libraries tailored for use in smart cards. For more information on the JavaCard architecture and JavaCard programming, see Chen (2000).[6] To use a JavaCard for authentication, the card must contain an applet that exposes an appropriate APDU interface to the external world, e.g. a command that can be parameterized with a challenge, returning a digital signature over the challenge and a command to obtain certificates stored in the card. This functionality allows the kind of protocol shown in Figure 5.6 to be run.

Windows-powered smart cards
The Windows-powered smart card allows implementation of custom commands. It is possible to implement commands that use functions from the card's internal crypto library to provide a function for generating digital signatures. Storing and reading of certificates is possible by default. This functionality also allows for PKI authentication protocols as shown in Figure 5.6.

Smart card APIs

In this section, we present two APIs for smart card applications: the PC/SC interface for native applications running on the Microsoft

Windows operating systems and the OpenCard Framework for Java applications and applets. The need for these standard interfaces came from a variety of different smart-card readers and smart cards, all of which were delivered with their own access software with no common API. As a result, smart-card applications had to include code that was dependent on the API of a particular smart-card reader manufacturer. Because of this, it was hard to migrate applications to other readers or allow for use of different readers. Below, we explain how the PC/SC and OpenCard Framework APIs have solved these problems to allow for smart-card reader and smart-card interoperability.

PC/SC

PC/SC is the standard interface for native smart-card-aware applications on Windows operating systems. The interface is included in Windows ME, Windows 2000 and future Windows versions. It has been defined by the PC/SC workgroup, which included Apple, Bull, Gemplus, Hewlett Packard, IBM, Infineon, Intel, Microsoft, Schlumberger, Sun Microsystems, and Toshiba. The workgroup's mission was to promote a standard specification to ensure that smart cards, smart-card readers, and computers made by different manufacturers worked together, and to facilitate the development of smart-card applications for PCs and other computing platforms. One specific goal was to specify common PC programming interfaces and control mechanisms. In December 1997, the PC/SC Workgroup released the PC/SC Specification 1.0,[7] defining the architecture and key interfaces for using smart cards from applications on PCs. Figure 5.8 shows an overview of the PC/SC architecture.

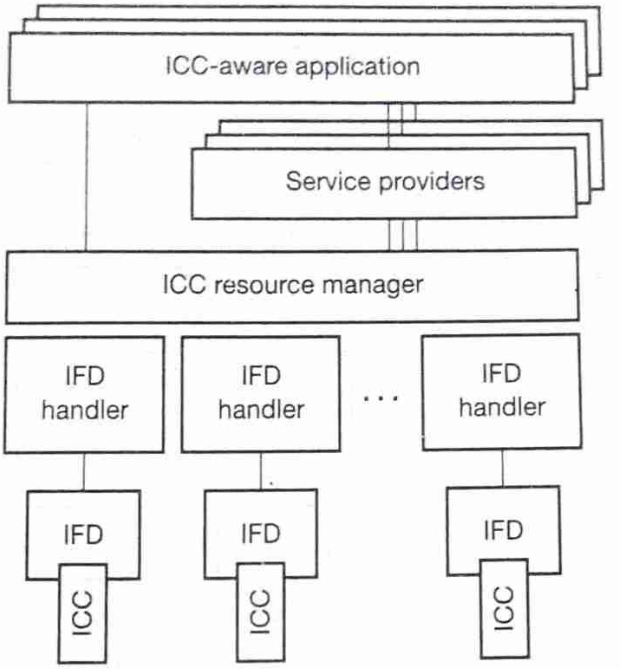
Smart cards (*ICCs* in PC/SC terminology) are accessed by applications on the PC via a smart-card reader (*IFD* peripheral devices in PC/SC terminology). There may be many smart-card readers per system, e.g. those connected via serial line, those integrated in the keyboard, and PC-card-based smart-card readers. Associated with each smart-card reader is an IFD handler, which is usually implemented as a device driver that provides a standard interface as defined in Part 3 of the PC/SC specification.

The ICC resource manager provides system-level service. It manages smart-card readers and inserted smart cards, controls shared access to these devices, and supports transaction management primitives.

The service providers provide applications with a high-level interface mapped to specific smart-cards. The PC/SC specification defines common interfaces for services, such as authentication, file access, and cryptography, and defines how to define extensions for domain-specific requirements.

Applications written to the PC/SC architecture typically make use of the resource manager and specific service providers, e.g. by waiting for card insertion, obtaining a service provider for the inserted smart card from the resource manager, using the service provider to have the smart card perform certain functions, and finally freeing the resources to allow other applications to use the card thereafter.

Figure 5.8



PC/SC architecture overview

The OpenCard Framework

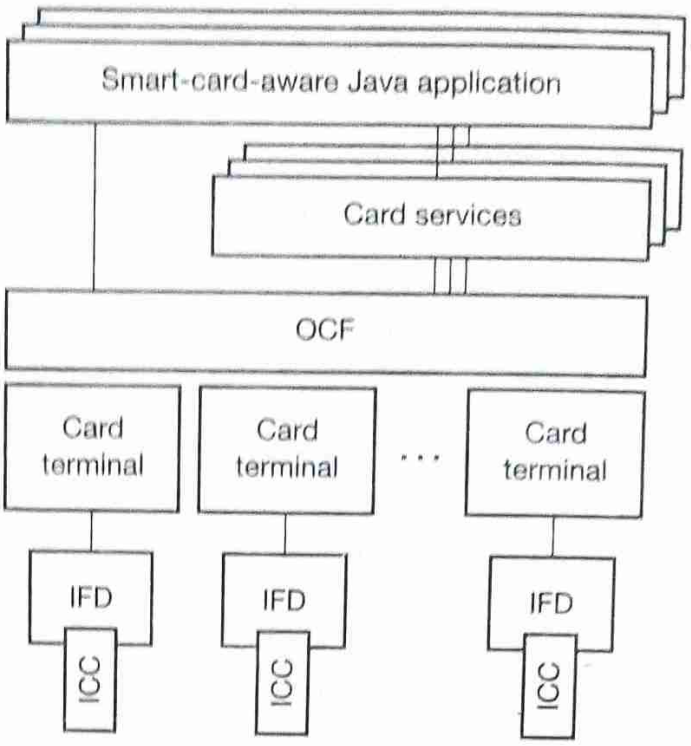
The OCF has become the standard interface for smart-card applications written in Java. In 1997, IBM, Sun, Netscape and others founded the OpenCard Consortium to establish the OCF as a de facto standard for accessing smart cards from Java. In 2000, the OpenCard Consortium released OCF Version 1.2 and OpenCard for Embedded Devices 1.2.

OCF allows smart card applications to be implemented in Java independent of the card terminal used to access the smart card and independent of the smart card used (Figure 5.9).

To achieve independence from particular card terminals, the Consortium defined a card terminal interface with functions required to access a smart card, e.g. resetting the card, getting the answer to reset, sending an application protocol data unit to the card, and getting the response. There are OpenCard card terminal drivers for virtually all PC smart-card readers on the market. A pure java card terminal can be implemented, or the reader can be accessed via a generic PC/SC card terminal for OpenCard that provides a bridge to the PC/SC card terminal interface.

To achieve independence from particular cards, card services are used. Card-service interfaces can be defined for particular sets of smart-card functions. Two interfaces that are defined in the OCF itself are the File Access Interface and the Signature Interface. The File Access Interface allows files on a smart card to be accessed. The Signature interface allows digital signatures to be generated. Once a card service interface has been defined, various card service implementations that implement the interface can be developed for different cards.

Figure 5.9



The OCF

The main role of the OCF is on the client side of Web applications, running in a Java applet. The most advantageous way to deploy the OCF in such an application is to install the OCF and the required card-terminal classes locally on the client, adding the OpenCard JAR (Java Archive) files and executables to the browser's paths. The card services to be used should be packaged with the applet JAR file that is deployed on the application server, so that they can be updated easily without changing client installations.

The OCF is available from the OCF website.[8] The website also provides documentation, user guides, links to papers and books about the OCF, and the latest news and announcements from the OpenCard Consortium. For comprehensive information on developing smart-card applications in Java, see Hansmann *et al.* (2000).[9]

References

1. World Wide Web Consortium: http://www.w3.org
2. Dierks, T. and Allen, C. (1999) 'The TLS Protocol Version 1.0'. IETF. http://www.ietf.org/rfc/rfc2246.txt
3. Fielding, R., Gettys, J.C., Mogul, J.C., Frystyk, H., Masinter, L., Leach, P., Berners-Lee, T. (1999) 'Hypertext Transfer Protocol – HTTP/1.1'. IETF. http://www.ietf.org/rfc/rfc2616.txt

6 WAP and beyond

WAP has become a synonym for new, wireless Internet services. This chapter describes the basics of WAP e-business implementation, device characteristics, protocol stack, security issues, products, and tools currently available on the market.

6.1 Introduction

Mobile communication services provide limited data rates and lower reliability compared with switched-line networks. Compared with modern desktop computers, mobile devices provide only a constrained application environment tailored to small, low weight devices with low-energy consumption, powered by small, rechargeable batteries. Mobile users demand devices that are easy to use and are adapted to the mobile environment. The standard PC methods to access and manage Internet applications cannot be mapped to the mobile environment because too many user interactions and lengthy data transfers would cause customer dissatisfaction.

Therefore, mobile equipment manufacturers founded the Wireless Application Protocol Forum (WAP Forum) in 1997 to define an architecture that extends the Internet technology for mobile devices. Founding members were Ericsson, Motorola, Nokia, and Openwave Systems, (formerly known as Phone.com, formerly known as Unwired Planet). Today, the WAP Forum comprises over 200 members ranging from network operators over mobile device suppliers to content developers and many IT companies.

The WAP architecture defines an optimized protocol stack for communication over wireless lines, an application environment for mobile phone applications, a content description language, and a miniature browser interface. The ability to work with a wide variety of devices is a key feature of the WAP architecture. The mobile user interfaces are typically limited:

- *Small display.* A standard mobile phone display has a size of 96 × 65 pixels. A PDA display is slightly larger at 160 × 160 pixels, but today's desktop PCs feature screen sizes of 1024 × 768 pixels and larger.

- *Restricted input capability*. Mobile phones offer a keyboard with 12 keys for data input, which makes entering text a time-consuming process, e.g. entering the character 'z' requires four keystrokes. Dictionary-assisted input methods such as T9 or Octave help to make entering text more convenient. The pen-based character input of PDAs allows trained users to enter data faster, but it does not match the capabilities of a full-size PC keyboard.
- *Limited memory and processing power*. A typical mobile phone has 8–32 MB of RAM and a 16-bit digital signal processor CPU running at about 10–20 MHz, while desktop computers come with 128 MB of RAM and 1-GHz processors.
- *Low-speed network connections with high latency*. Today, a typical GSM channel for WAP in Europe offers 9.6 kbps compared with 56 kbps of an analog PC modem or even 768 kbps with asymmetric digital subscriber line (ADSL).

These limitations have influenced the architecture at various levels. The replacement of HTML pages (typical 10 kB) with highly condensed WML pages, called cards, reduces the overhead of content presentation and reflects the limited display capabilities of mobile devices. A compact binary encoding of the data stream minimizes the data flow between device and server. The protocol and the browser are designed to run on very small, embedded systems with subsecond response time.

Within a short time, the WAP architecture has grown into a complex specification with many overlaps with the global Internet architecture. Some of the basic assumptions, such as limited data rate and client processing power, are no longer valid for the high end of mobile devices. Therefore, many solution providers, but also device and equipment manufacturers, favor a rapid merging of WAP and Internet technology into a mobile Internet architecture. However, WAP technology is reasonably mature, is supported by major industry players, and will exploit the improved speed of GPRS and UMTS systems. Therefore, the WAP architecture and systems built on this foundation will play a major role in the pervasive computing industry and will be embraced by the global mobile Internet architecture.

6.2 Components of the WAP architecture

The WAP architecture is similar to the Open System Interconnect (OSI) Reference Model defined by the International Organization for Standardization (ISO). The layered model describes how information flows from a Web application to the physical interface of the mobile device up to an application residing on a mobile phone. Each layer performs only

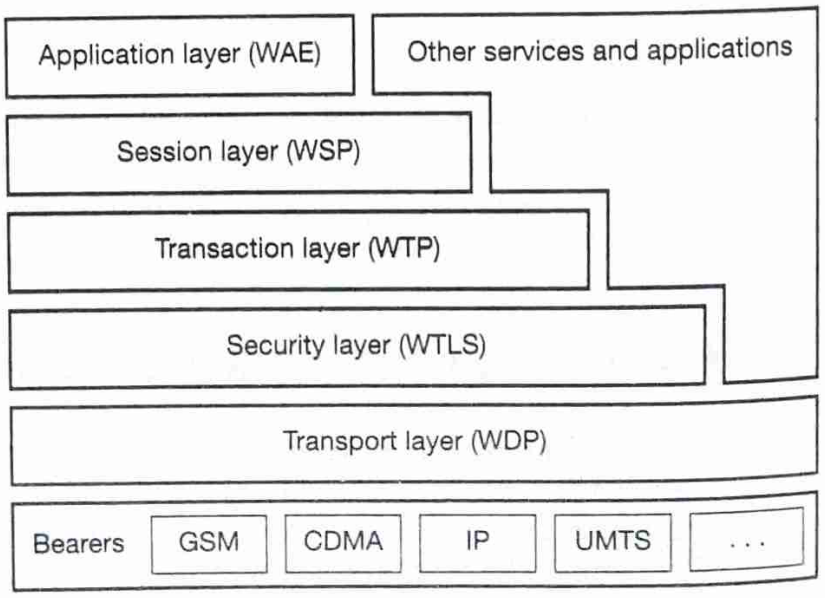
one specific network function, e.g. provides transactional services. Each layer is independent and can be implemented without affecting the other layers. A typical WAP application uses only functions of the application layer, which itself uses the layers beneath it. Figure 6.1 illustrates the layers of the WAP architecture of a mobile phone.

6.2.1 Bearers

A bearer is the low-level transport mechanism for network messages. The WAP protocol stack is designed to operate with a variety of bearer services, with emphasis on low-speed mobile communication channels. One bearer service, for example, is an HSCSD connection with a minimum of 9.6 kbps. Short messages can also be used as a bearer for WAP, but speed is limited. GPRS and UMTS will also enable faster, packet-switched bearer services for WAP applications.

Packet-switched bearer services are much better suited than circuit-switched services for mobile devices because they can provide reliable services in an inherent, unreliable mobile environment. They also support the typical 'on demand' requests of mobile users better and faster than circuit-switched bearer services. Major mobile networks have been GPRS-enabled since the beginning of 2001. The gating factor for service introduction will probably be the billing model and billing system to support packet-switched operation on a large scale unless flat-fee models are introduced.

Figure 6.1



WAP architecture (client)

6.2.2 Transport layer – Wireless Datagram Protocol

The Wireless Datagram Protocol (WDP)[1] operates above the bearer services and provides a connectionless unreliable datagram service similar to UDP, transporting data from a sender to a receiver. The WDP layer provides the upper layers (security, transaction, and session) with a uniform interface to operate independently of the underlying bearer services. WDP offers simultaneous communication from the upper layers to the bearer services.

Errors during transmission at the WDP layer can be communicated via the Wireless Control Message Protocol (WCMP), similar to the Internet Control Message Protocol in the IP world. A typical WCMP message would be 'receiver cannot be reached'. The WCMP interface also provides an independent control channel for diagnostics and administration.

6.2.3 Wireless Transport Layer Security

WTLS[2] specifies a framework for secure connections, using some protocol elements from the Internet TLS protocol. WTLS offers base cryptographic services for WAP applications that can, for example, be used for transferring sensitive data between the device and a server. WTLS provides data integrity, privacy, and client and server authentication.

Before data exchange, a WTLS session has to be established by performing an initial key exchange and negotiation of the cryptographic algorithm to use (see Chapter 4). WTLS gives several options for key exchange, including RSA, Diffie-Hellman (DH) key exchange, or elliptic curve cryptography (ECC). Privacy of communication can be secured using the Data Encryption Standard (DES) or the International Data Encryption Algorithm (IDEA).

6.2.4 Wireless Transaction Protocol

WTP[3] provides a reliable data-transport mechanism. WTP in a WAP stack can be considered the equivalent of the TCP layer in the IP stack, but it is optimized for low bandwidth. WTP offers three classes of transaction services:

- *class 0* provides unreliable one-way messages without confirmation of messages, e.g. single request within an existing wireless connection;
- *class 1* provides reliable one-way message without result messages, e.g. push services like SMS, where no response is expected;
- *class 2* provides reliable two-way request–response messages, e.g. every request is answered with a response, like a confirmation or the result of a query.

6.2.5 Wireless session protocol

The Wireless Session Protocol (WSP)[4] provides connection-oriented services based on WTP and connectionless services based on WTLS or WDP. It supports HTTP 1.1 functionality and semantics in a binary-encoded format to minimize data transfer to the mobile phone.

One of WSP's prime features is to hold a shared state between the client and the WAP gateway (see WAP Infrastructure, below). To provide this feature in an HTTP environment, the WAP gateway typically maps the WSP states to cookies stored in the gateway. Thus, the WAP gateway emulates the preferred solution used by HTTP servers.

6.2.6 Wireless Application Environment

The Wireless Application Environment (WAE)[5] combines Web and mobile phone technologies. It provides a network-neutral application environment and permits a high degree of device independence by using the WAP protocol stack.

The WAE contains a microbrowser that is capable of displaying WML pages and executing WMLScript, a scripting language similar to JavaScript. The microbrowser also includes an additional wireless telephony application interface (WTAI) with telephony functions allowing call control, network text messaging, and offering a phone book interface.

6.3 WAP infrastructure

In order to run a WAP application, a complex infrastructure consisting of a mobile client, a public land mobile network such as GSM, a public switched telephony network such as ISDN, a WAP gateway, an IP network, and a WAP application server is required. Additionally, there may be system components such as WAP portals, proxy servers, routers, and firewalls. Several service providers will be engaged in providing the actual service providing to consumers requiring sophisticated service-level and billing agreements. The WAP-specific parts of this chain are described below.

The programming model of the WWW defines a direct connection between the client and the application server, as shown in Figure 6.2. More detailed information about the Web programming model can be found in Chapter 5.

WAP introduces an additional component between the client and the application server: the WAP gateway (see Figure 6.3). The WAP gateway performs two main tasks:

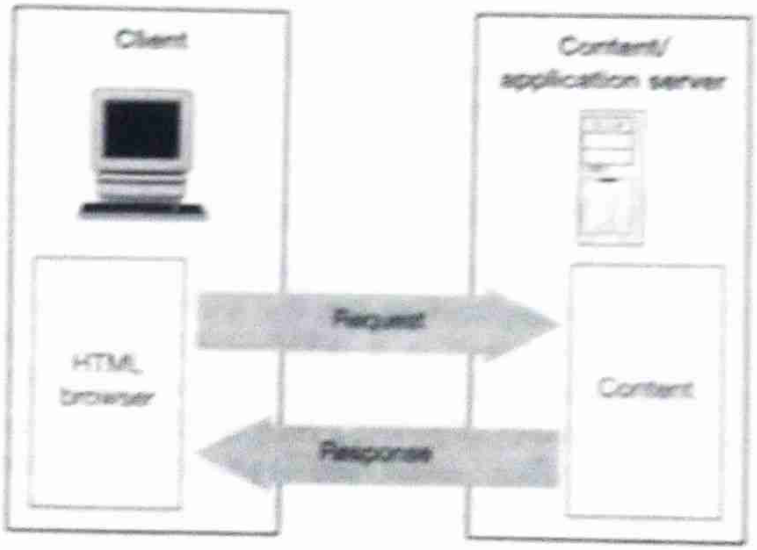
- *Protocol conversion*. The WAP gateway is a protocol intermediary between the client and the server. The communication between the

client and the WAP gateway is performed with WAP protocols, and the communication between the WAP gateway and the server is based on HTTP protocols.

- *Content encoding.* Data transferred between the client and the WAP gateway are binary encoded in order to minimize data transfer.

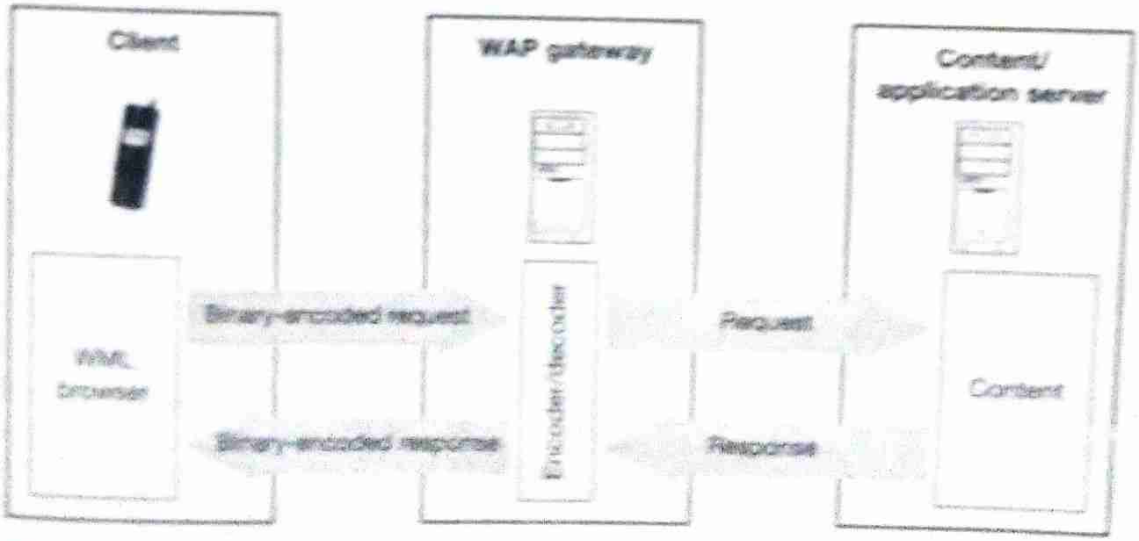
All requests from the phone are sent to the WAP gateway. The WAP gateway decodes the binary messages from WSP, transforms them to HTTP, and sends them to the selected application server. The host name or IP address of the application server is part of the URL given in the

Figure 6.2



The WWW programming model

Figure 6.3



The WAP programming model

request from the phone. The application server prepares the response and sends it to the WAP gateway, where it is encoded in the gateway and forwarded to the phone.

Figure 6.4 illustrates the protocol stacks inside a WAP gateway and the flow of a request and response pair through the stacks.

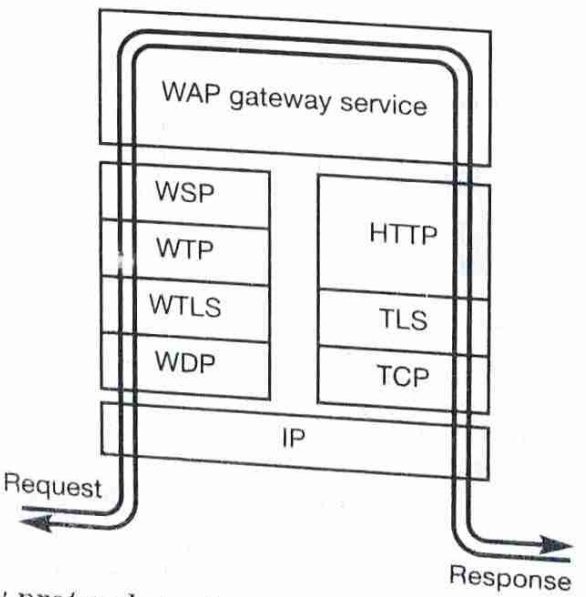
This path through the protocol stacks in combination with the conversion from WML to binary WML is computing intensive. Therefore, the hardware running the WAP gateways should be very powerful in order to avoid long delays.

The devices participating in a WAP communication and the associated protocol stacks are shown in Figure 6.5. In this scenario, the mobile phone is connected via the point-to-point protocol (PPP) to a dial-in router of a telecommunication provider. The dial-in router sends all requests from the mobile phone to the WAP gateway. The WAP gateway receives the incoming WAP requests and forwards them to the server via the HTTP protocol.

6.3.1 WAP profile

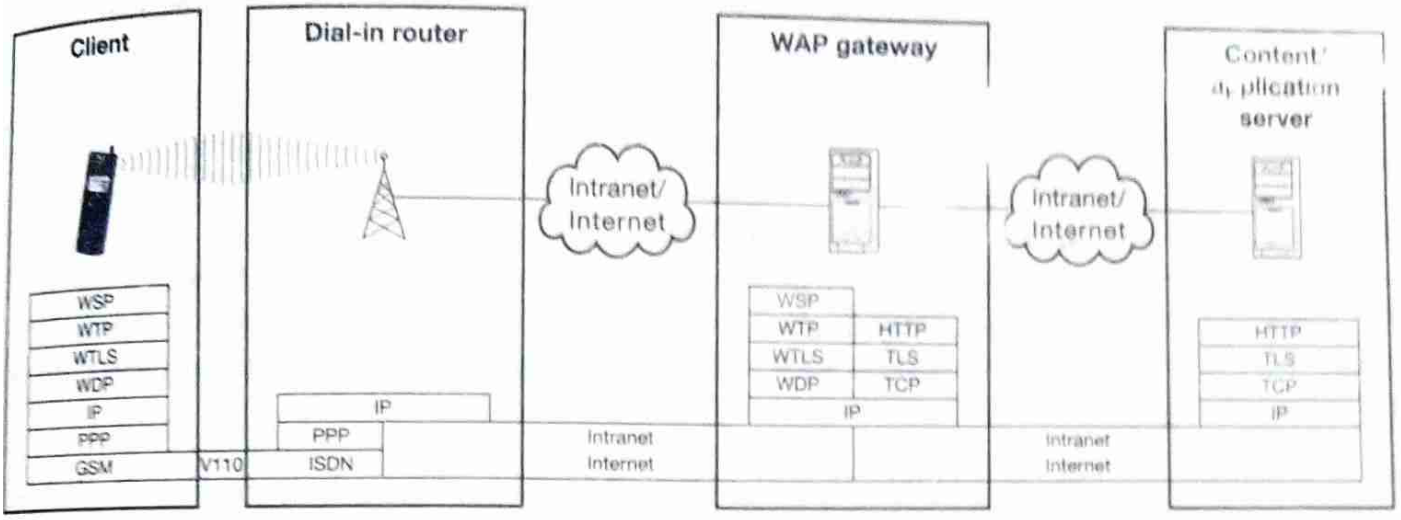
A WAP browser requires a set of parameters to build a connection to an Internet application like the scenario shown in Figure 6.5. These parameters are stored in a WAP profile on the mobile client, much like the parameters of an IP connection are stored in a browser profile on a PC. The number of profiles that can be stored in a mobile phone is limited to three or five – some mobile phones have been delivered to customers even with a single fixed profile. Frequently, users do not switch their profiles at all, so they are restricted to one WAP gateway of their service provider.

Figure 6.4



The WAP gateway protocol stack

Figure 6.5



Sample WAP infrastructure

The WAP profile contains the following elements:

- *Homepage*. The URL address of the Internet start page (format *http://x.yy.z.xyz/homepagexyz.wml*).
- *Connection Type*. Defines the communication protocol between the mobile phone and the dial-in router as either continuous (switched line) or temporary (packet switched).
- *Connection Security*. Set to either ON or OFF. A WTLS session is established between the mobile phone and the WAP gateway when the setting is ON.
- *Bearer*. Defines the bearer service used for data transfer. When 'SMS' is selected, the server number and service number must be supplied. The data-transfer mode, e.g. HSCSD, is used in GSM systems and the following parameters must be specified.
- *Dial-up number*. Extension of dial-in server to which the WAP gateway is attached.
- *IP Address*. The IP address of the WAP gateway
- *Authentication type*. Defines whether a security certificate is to be used for client authentication. Setting is 'normal' or 'secure' with certificate. Certificates can be requested from a service provider.
- *Data call type*. Defines the signalling between the mobile phone and the dial-in server. Analog signalling is generally slower than ISDN signalling, but it is required in some countries where access only via analog lines is possible.

- *Data call speed*. Defines the communication speed between the phone and the network; usually 9600 or 14400 Baud (higher rates are expected in the future).
- *User name*. A generic or dedicated user name. Access to the network and the WAP gateway of an Internet provider or a corporation will require positive user identification through a user-specific name and user authentication through a password.
- *Password*. Secret password for access to the network to which the WAP gateway is attached. User name and password are used only for network access with WAP phones and/or PCs, and are not passed on to the WAP gateway.

Most mobile phone users will have problems in entering all of these parameters correctly. A single faulty character in the profile will disable the WAP service. Problem determination is complex because no automatic diagnosis tools can be used before a connection with the WAP gateway is established. Therefore, some mobile phone manufacturers support over-the-air provisioning (OTP) – the setting of profiles via SMS. The user requests a profile by sending a specific SMS message to the service provider. Then, the profile is sent in a response message and must be activated by the user.

The use of WAP profiles was designed according to the PC browser model. However, the introduction of the WAP gateway in the communication path between the client and the Internet application server has caused some infrastructure problems. Companies may choose to install a private WAP gateway, in order to overcome the inherent security problems, but they must force all users to install a special profile for access. This may work for employees, but may not work for customers who are not willing to switch profiles whenever they want to access a specific service.

WAP portals offered by mobile service providers, industry associations, or Internet providers may provide a partial solution to this problem that is typical of any non-IP-based access, such as WAP and voice. In addition to the basic connectivity, portals also provide personalization, customization services, and security frameworks for their customers, and a public key infrastructure for content providers. The WAP forum is addressing some of these problem areas with an extension of the WAP security architecture, but it will take several years before the new functions are implemented in the majority of mobile phones. Therefore, WAP system designs will exploit various schemes used in the IP world to secure the communication between client and application provider.

6.4 WAP security issues

Overall, WAP provides a couple of advanced security features that are not available in the IP environment. Over-the-air communication privacy is ensured by encryption, and mobile subscriber ISDN numbers are not exposed. The GSM SIM provides good identification and authentication. Typical IP infrastructures do not use this type of security at the client side. However, the introduction of a WAP gateway in the information flow between client and server creates potential security leaks unless the WAP gateway is operated in a secure environment and trusted by all participants.

Some security issues have been identified within the current WAP architecture:

- *End-to-end security*. The current WTLS architecture does not support end-to-end security. The communication link between the mobile device and the WAP gateway is secure, as is the link from the WAP gateway to the server. However, all messages that go from the mobile device to the server or vice versa are decrypted by the WAP gateway, and then encrypted again for transmission to the server. This means all messages are available in clear text within the WAP gateway. As a result, the user and the application provider must trust the operator of the WAP gateway.
- *Missing secure authentication*. To allow for highly secure authentication to servers, generation of digital signatures must be performed in tamper resistant devices, such as smart cards, which are not part of the current WAP specification. To overcome this issue, the WAP Forum proposed a WAP identification module (WIM) specification. WIM is a device for storing private keys and performing user authentication in a WTLS session set-up. In mobile phones, SIMs with extended functionality may be used as WIMs.
- *Unauthenticated OTP*. Many mobile phones can be configured remotely using OTP. When configuration data are received via the OTP mechanism, the originator cannot be authenticated via a cryptographic mechanism. This means that a user has no proof of the origin of the message and may end up accepting malicious configuration messages accidentally.
- *Missing public key infrastructure*. The WAP Forum proposed a WAP Public Key Infrastructure (WPKI). WPKI is a specification of client and WAP gateway certificate requests based on PKCS#7. This allows the exchange of public keys in a defined manner.

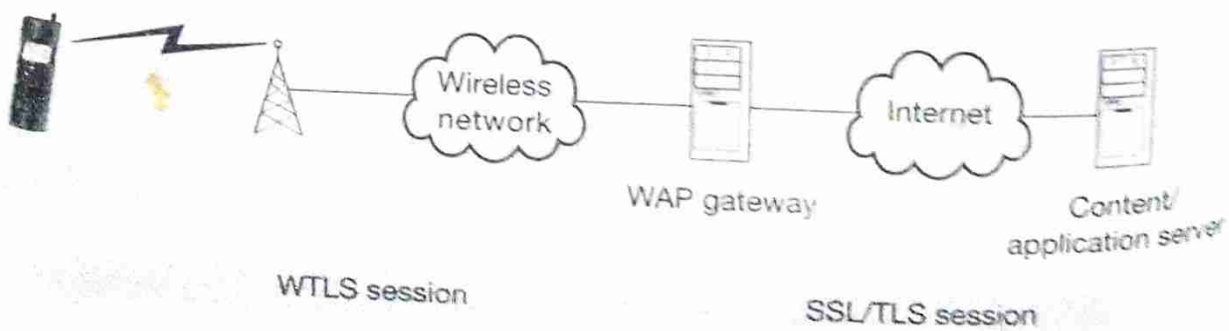
6.4.1 WAP session security

Missing end-to-end security is one of the major problems of the current WAP architecture. The lack of such security requires the user and the application provider to trust the operator of the WAP gateway, since the operator has the ability to intercept all messages in clear. Figure 6.6 illustrates the secure parts of the connection.

A WTLS session exists between the client and the WAP gateway, and an SSL or TLS session exists between the WAP gateway and the server. When the client sends an encrypted message to the server, the WAP gateway decrypts the message, encrypts it according to SSL/TLS, and forwards it to the server. This means that each message exchanged between the client and the server exists at least temporarily in clear text on the WAP gateway. Obviously, the same is true for the response sent back by the server. For example, in a banking applica[illegible] such as account information and PIN/TAN codes, exchanged between the user and the bank are exposed at the WAP gateway. Unless the bank [illegible]ates its own WAP gateway, the gateway operator will be a telecommunication company, which is unlikely to be trusted by a bank. As a result, some banks have already set up their own WAP gateways to be able to achieve an appropriate level of security for their applications. The downside of this approach, however, is that customers need to add a configuration to their phones that enables them to access directly the bank's WAP gateway instead of going through the preconfigured gateway of the telecommunication company.

The WAP Forum proposed two approaches to resolve this issue. One approach is to create a subset of the TLS protocol that is appropriate for use with small mobile devices (Figure 6.7). This would enable end-to-end security without decrypting and re-encrypting the messages in the WAP gateway, which would also remove load from the WAP gateway.

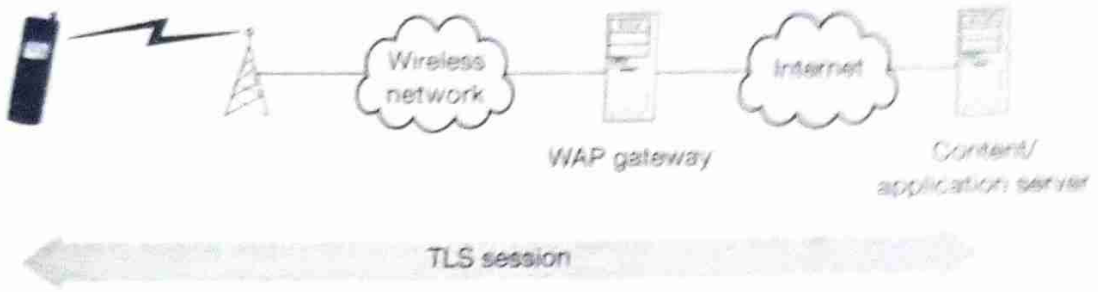
The other approach is to redirect sessions that require secure communication from the WAP gateway, installed at the mobile service provider location, to a trusted gateway, e.g. a WAP gateway installed at the bank itself (Figure 6.8). To do so, the server sends a redirection message to the



igure 6.6

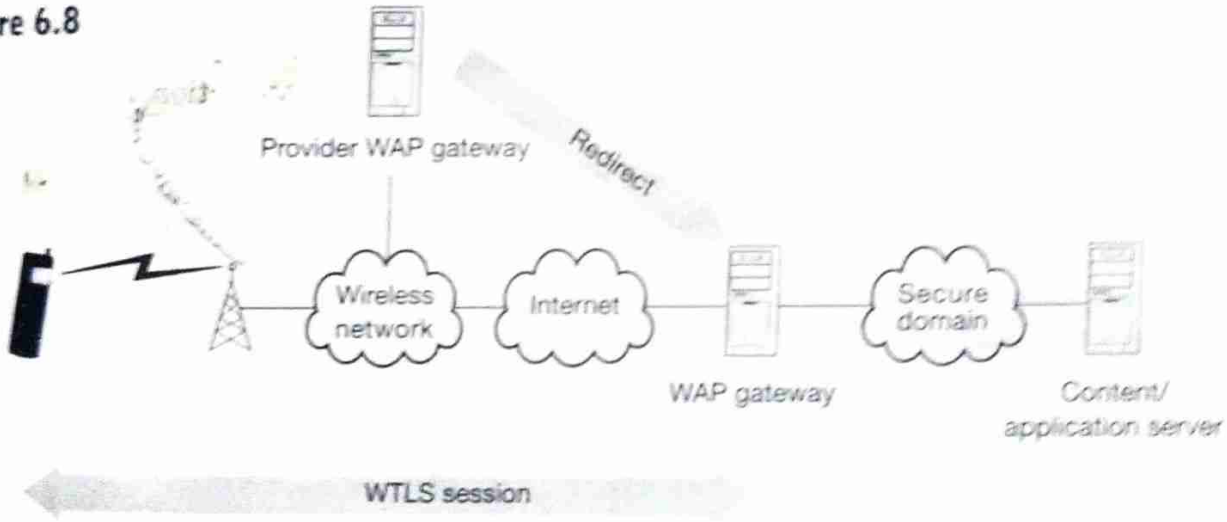
WAP session security

Figure 6.7



TLS end-to-end security

Figure 6.8



Redirection to a secure WAP gateway

mobile device in order to cause the subsequent requests to the server to be sent through the secure WAP gateway. However, a redirection message is required from the banking application, which increases the response time for the transaction.

6.5 Wireless Markup Language

HTML and JavaScript are designed for desktop computers, and are far too complex for today's mobile devices. Several subsets of HTML have been defined to accommodate less powerful devices than PCs. However, none of these proposals has the kind of broad support from the industry that WML[6] and WMLScript[7] defined by the WAP Forum for mobile devices have received.

6.5.1 Markup

WML is an XML-based markup language designed especially for small devices. Compared with HTML, WML introduces new features suited for use with mobile devices but lacks many of HTML's features.

- An application can transmit multiple pages (called cards) simultaneously in a single transmission unit called a deck.
- WML has the concept of events, e.g. a timer can execute tasks such as displaying a new page after a certain period of time.
- WML can preserve the content of variables between different WML pages (cards). This means that WML, unlike HTML, is not stateless. For example, a variable that is used in one card can be shared with another card because variables are not cleared when a new card is loaded.

Example 6.1 shows a WML deck. A deck is the unit that is transferred from the application or content server to the mobile device.

Example 6.1 **A WML deck**

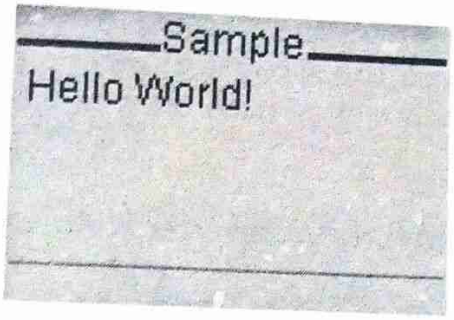
```
<?xml version="1.0"?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN"
"http://www.wapforum.org/DTD/wml_1.1.xml">

<wml>
 <card id="card1" title="Sample">
  <p>
   Hello World!
  </p>
 </card>
</wml>
```

In WML, a single page is called a card. Cards and decks are defined in WML to reduce client/server interaction and therefore to reduce the response time for user interactions. Figure 6.9 shows the result of the example above.

6.5.2 WMLScript

WML contains the scripting language WMLScript for performing certain tasks on the mobile device, such as validating user input. WMLScript is derived from ECMAScript, which itself is derived from JavaScript.

ure 6.9



The result of entering the WML Script in Example 6.1

Variables in WML are type less. They can hold a null value or any sequence of characters.

WMLScript Version 1.3 contains the following standard libraries:[8]

- The Lang library provides base functions for WMLScript, such as `min`, `max`, and `random`.
- The optional Float library contains floating-point functions, such as `floor`, `round`, and `pow`.
- The String library contains string-manipulation functions, such as `subString` and `length`.
- The URL library provides functions for handling absolute and relative URLs.
- The WMLBrowser library provides functions for manipulating the associated WML context, such as loading a new page or updating the display. It also contains functions such as `setVar` to set the value of a WML variable.
- The Dialogs library contains user interface functions to prompt for input or confirming an action.

Specific implementations of WMLScript may contain more libraries than required by the WAP standard. These libraries may contain crypto functions, phone book access, or dialling services.

Example

The WMLScript shown as Examples 6.2 and 6.3 is an example of a simple temperature converter. The user enters a temperature and selects the target measurement unit (degrees Celsius or degrees Fahrenheit). The example consists of two parts – a WML deck containing the markup, and the WMLScript containing the program code used for converting the temperature. The markup code displays an entry field for temperature input and two links for measurement unit selection. After entering the temperature and selecting a link, the function `convert` in the WMLScript code is executed. The function sets the variable `conversion` of the WML deck

6.6 WAP push

WAP push is a service that allows information to be sent to a mobile device without previous user interaction.

The WAP programming model introduced above describes a set-up in which all interaction is triggered from the client. The client sends a request to the server and gets a response for each request. This is called 'pull technology'. In contrast is 'push technology', in which data are sent to the client that do not form an answer to an explicit request. Examples of a push message are the result of a tennis game, or stock quotes that are sent to the mobile device from a news agency.

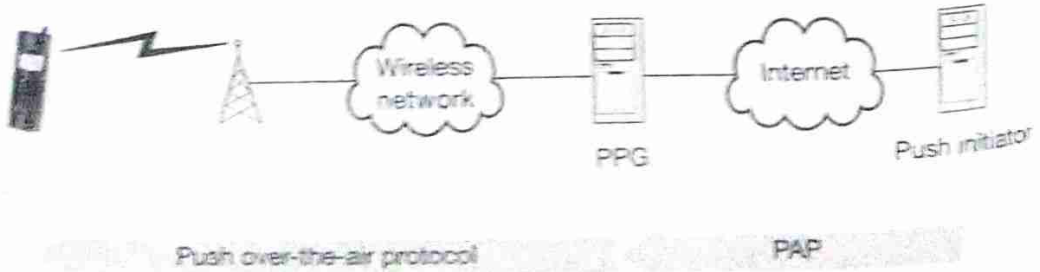
A push operation is triggered from a push initiator by sending a push message to the push proxy gateway using the Push Access Protocol (PAP).[9] This is used to deliver content with standard Internet technology to a push proxy gateway (PPG).[10] The PPG delivers the message to the mobile device using the push over-the-air protocol. Depending on the capabilities of the PPG, a notification about the outcome of the push operation is sent to the push initiator. The set-up of a push operation is shown in Figure 6.14.

6.6.1 Push Access Protocol

PAP carries an XML-formatted document over the Internet. PAP uses, but is not limited to, HTTP as a transport mechanism. PAP offers the following services:

- *Push submission* sends a message to a mobile device.
- A *confirmation notification* is sent to the push initiator when the push message is delivered to the mobile device, or when a delivery failure occurs.
- A *push cancellation* is sent from the push initiator to the PPG when a previously submitted push submission should be cancelled.
- A *status query* response contains the status of a push submission.

Figure 6.14



Push message flow

- A *client capabilities query* request returns the capabilities of a mobile device.

6.6.2 The push over-the-air protocol

The push over-the-air (OTA[11]) protocol is a protocol layer for transmitting push messages over the WSP. It is the part of the push framework that is responsible for exchanging content between the user agent and the PPG.

6.7 Products

The market for mobile phones is basically split into two segments. The low-end segment provides basic voice and SMS services. Emphasis is on ease of use, innovative design, and small device size. The high-end segment ranges from basic WAP devices to wireless PDAs. WAP gateways are developed by mobile equipment providers and IT infrastructure providers exploiting their investments in IP protocol stacks. In this section, we describe a selection of mobile phones, WAP gateways, and WAP simulator tools that are available on the market today.

6.7.1 WAP phones

The number of WAP phone models is increasing steadily, and prices are falling. In this section we present some of the most common phones in use today. It can be accepted that WAP will become a standard feature for all GSM mobile phones.

Nokia 7110

The Nokia 7110 (Figure 6.15) was one of the first WAP phones available, and most applications have been developed and tested with this phone. It is a dual-band (GSM 900 and GSM 1800) phone, and supports data rates of up to 14.4 kbps. It has a display with 96 × 65 pixels, which is the equivalent of four lines with 25 characters each. The maximum deck size is just over 1400 bytes.

The Nokia 7110 supports five configuration profiles. The built-in microbrowser ignores all emphasis tags (`big`, `small`, `em`, `strong`, `b`, `i`, `u`). Tables are supported, but every cell appears in a new line.

The Siemens 35 series

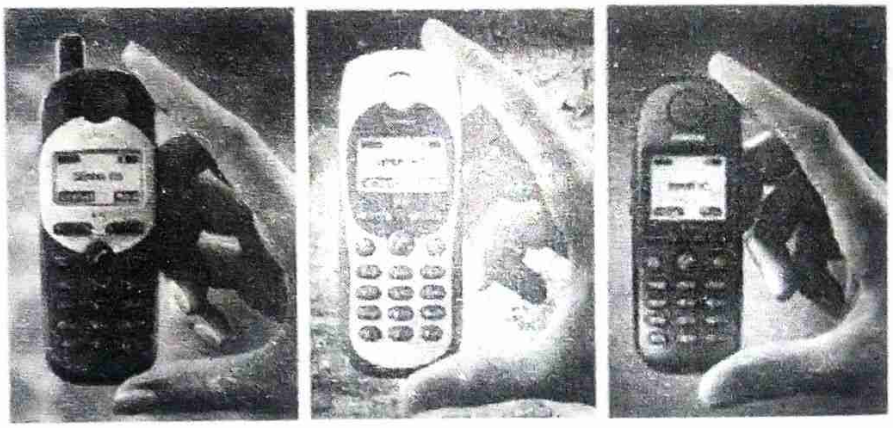
The current Siemens 35 series consists of three GSM mobile phones (Figure 6.16) and an organizer (Figure 6.17).

Figure 6.15

Nokia 7110

Courtesy of Nokia

Figure 6.16



The Siemens 35 series phones

Courtesy of Siemens

Figure 6.17

Siemens IC35 Organizer

Courtesy of Siemens

The C35i is designed for consumers. The M35i is designed for outdoor use, featuring a five-line display and reminder list. The S35i is designed for business use. The features of the phone are an IrDA link, a seven-line display, a calendar, voice dialling, and an organizer with calendar, appointments, and alarm list.

All phones are dual band (GSM 1800 and GSM 1900) and use the UP.Browser WAP 1.1 browser from Openwave Systems. The WAP browser supports suppression of the head and status lines by pressing the #key several times. This increases the number of lines available for WAP browsing (up to seven lines with the S35i phone and up to five lines with the C35i and M35i phones). All Siemens phones provide five configuration profiles. The deck size limit of all Siemens 35 series devices is 2 kB.

The Siemens Unifier (IC35) basically is a PDA with an integrated WAP 1.1 browser. The IC35 connects to a mobile phone via the IrDA interface or a serial cable. The IC35 supports three configuration profiles.

The IC35 offers the following applications: appointment calendar, address book, task list, text editor, pocket calculator, euro converter, and telephone manager. It has an IrDA interface as well as a serial interface, offers a slot for multimedia cards, and has a smart card slot.

One of the most interesting applications for the Unifier is mobile banking. The additional smart card slot allows automatic starting of the banking application and connection set-up as soon as a banking smart card is inserted.

Ericsson R380

The Ericsson R380 (Figure 6.18) is a mobile phone with an integrated organizer. It is a dual-band (GSM 900 and GSM 1800) phone, and supports data rates up to 9.6 KBps. Its touch screen has a size of 360 × 120 pixels, and supports pen input. The R320 2.0 WAP browser supports WAP 1.1.

The integrated organizer is based on the EPOC operating system and features typical PDA applications, including address book, calendar, notepad, and support for synchronization with PC applications.

ure 6.18

The Ericsson R380
Courtesy of Ericsson

6.7.2 WAP gateways

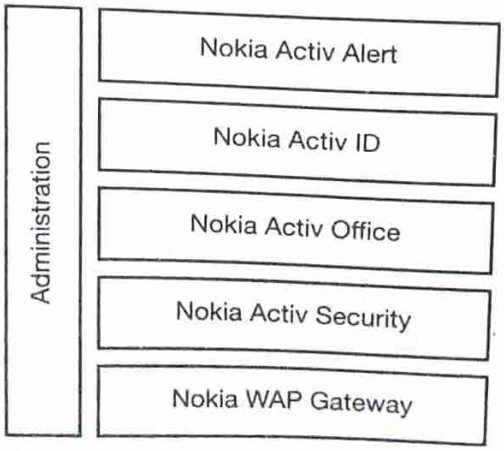
Wireless service providers or Internet providers offer Internet access with mobile phones through WAP gateways. Enterprises or banks may also install WAP gateways in order to prevent exposure of private data to other parties. For example, a bank may prefer to operate their own WAP gateway to lower the risk of somebody listening to the communication between the customer's mobile phone and the application server of the bank. Enterprises may install their own WAP gateways in their intranets because the security policy may not allow traffic from the wireless operator's network into the company's intranet. Typically, access to a WAP gateway requires a specific configuration profile, including dial-in number, protocol settings, optional user identification and password, WAP gateway IP address, and home page URL, much like the profile used for Internet access. However, mobile phones limit the number of profiles that can be stored and managed. Customers may also not be willing to switch profiles. Therefore, the number of WAP gateways easily accessible by a customer is rather limited.

WAP gateways are available from several sources, including an open-source implementation named Kandel. In this section we describe products from Nokia and Openwave Systems to give two examples.

Nokia Activ Server

The Nokia Activ Server 2.0[12] consists of a WAP gateway, a Web server for static content, and a servlet engine. It is a Java application available for Windows NT/2000, HP-UX, and Sun Solaris. The following plug-ins, shown in Figure 6.19, are available for the Activ Server:

Figure 6.19



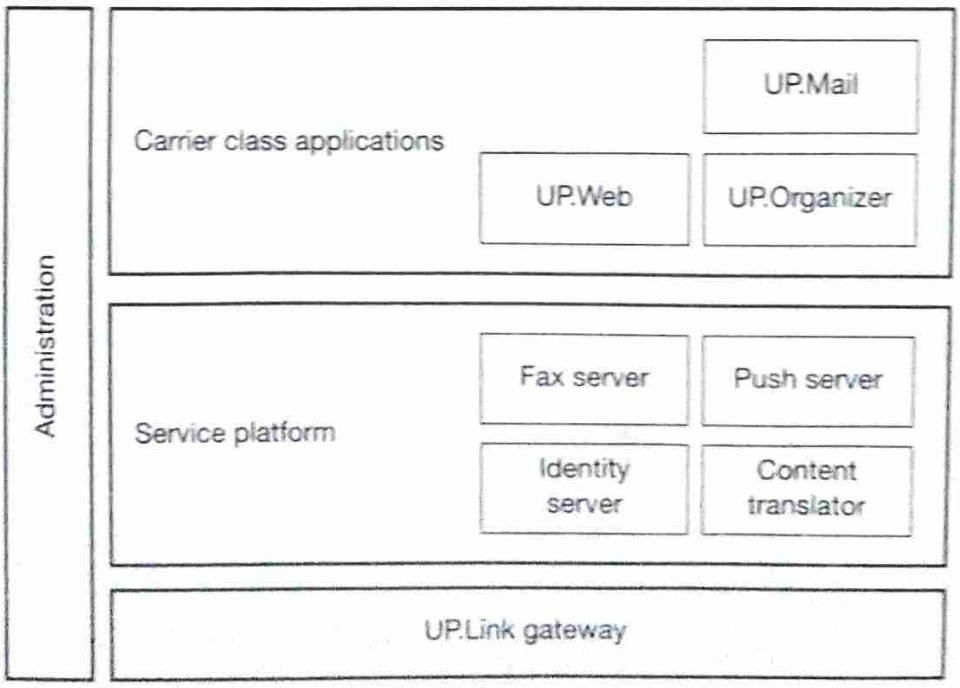
The Nokia Activ Server components

- *Nokia Activ Security* is a plug-in for the Nokia Activ Server offering WAP 1.1 compliant server authentication and data encryption. Activ Security is available with 56-bit and 128-bit key-length encryption. It supports SSL connections from the WAP server to the Web applications. The Security Pack is available for the same platforms as the Nokia Activ Server.
- *Nokia Activ ID* is a plug-in that enables personalized WAP applications based on the MSISDN (telephone) number recognition.
- *Nokia Activ Office* is a plug-in for PIM applications. It offers WAP email connectors for POP3, IMAP4, Lotus Notes, and Microsoft Exchange (via IMAP4) mail servers. The plug-in also supports replicating notifications of calendar entries to the mobile phones.
- *Nokia Activ Alert* provides an API for WAP push applications. It allows notifications or information to be sent to the mobile phone.

Openwave Systems WAP gateway

The WAP gateway from Openwave Systems[13] is called UP.Link server (Figure 6.20). It consists of the UP.Link WAP gateway, the UP.Link WAP enhanced services, and the UP.Admin administration interface. The UP.Link WAP gateway provides the protocol translation, security, activity tracking, and administration tools needed to implement a wireless Internet solution. It supports multiple air-link standards, including CDMA, cellular digital packet data (CDPD), GSM, integrated digital enchanced network (iDEN), personal digital cellular (PDC), personal

gure 6.20



The UP.Link server components

handyphone system (PHS), and TDMA. Packet, circuit and SMS bearers are supported. The UP.Link enhanced services features a content translator, a push server, a fax server, and an identity server. The UP.Link administration component provides a Web-based administration system for the UP.Link server.

6.7.3 WAP simulator tools

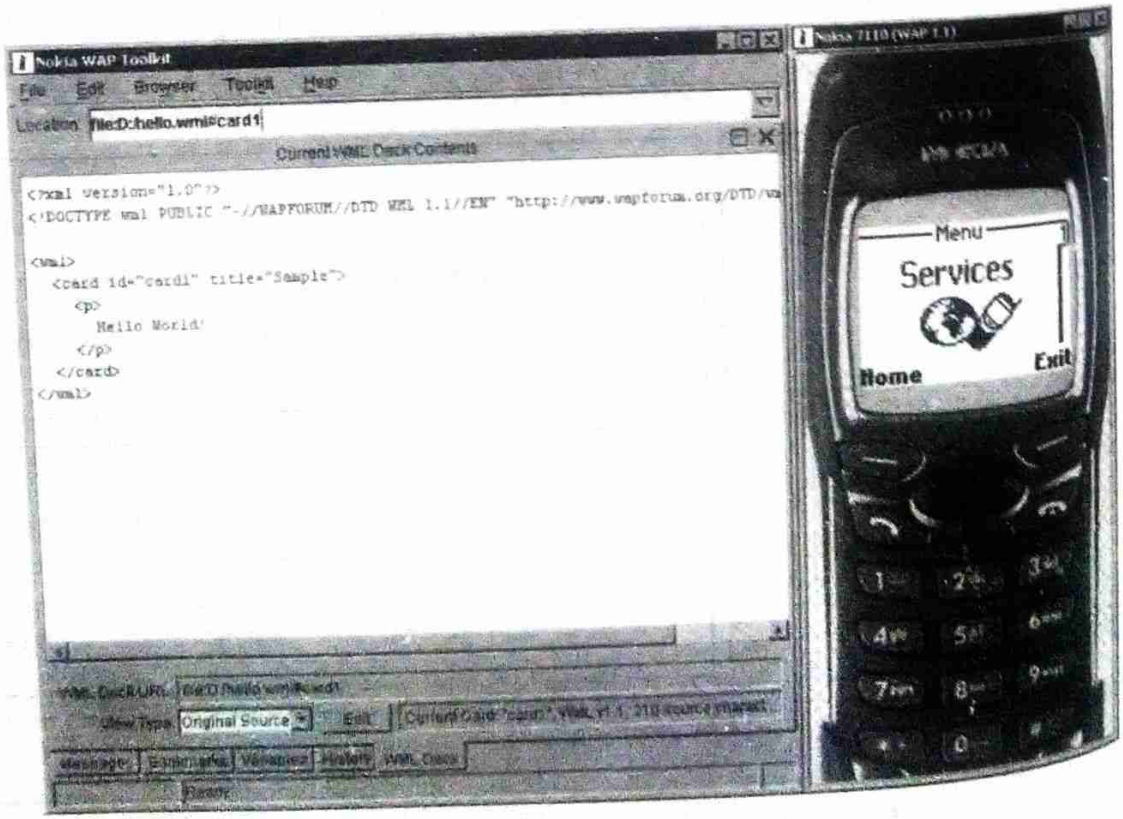
Simulator tools are used as a substitute for real WAP phones while developing WAP applications. Simulators typically display an image of the real phone and provide the same functionalities. They also offer debugging features, like the ability to decode binary WML deck content back to source code, or a tool for examination of the device cache. However, simulators may not represent all specific features of an actual WAP browser implementation, and may not emulate the behavior of a specific communication network. Therefore, applications must also be tested on real devices in real-life environment.

WAP simulator tools are available from many vendors. In this section, we describe the Nokia WAP Toolkit 2.0 as an example.

The Nokia WAP Toolkit 2.0

The Nokia WAP Toolkit (Figure 6.21) is an environment for creating and testing WML and WMLScript applications. It runs in Windows as a Java

Figure 6.21



Nokia WAP Toolkit in 7110 emulation mode

application. It includes a graphics editor for WBMP images, and allows GIF and JPEG images to be converted to WBMP. The Nokia WAP Toolkit has two simulation modes:

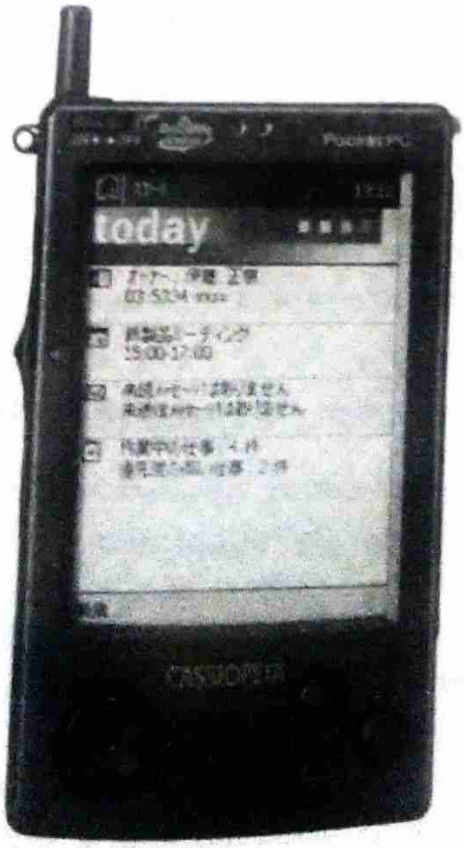
- The generic WAP phone emulation supports WAP 1.2 functionality, including WML, WMLScript, and WAP push services.
- The Nokia 7110 emulation mode is an exact simulation of the Nokia 7110 mobile phone.

6.8 i-Mode

i-Mode is an Internet service for mobile devices that is currently available in Japan, and will be available in the future in the USA and Europe. A typical device is shown in Figure 6.22. It is well accepted, with several million subscribers. It allows users to browse specially formatted websites and to receive email using their mobile phones. 3G i-mode phones are offering a display size of about 24 × 10 characters. i-mode uses a packet-switched bearer service with 9.6 kbps that allows for permanent connections. The monthly fee depends on the number of packets exchanged.

A phone is tied to a service provider; changing a service provider requires a new phone to be purchased. The i-mode gateway is always installed at the provider's site. Mail is stored at the server or in the phone, depending on the service provider business model. The size of each mail is limited to a certain size, e.g. 500 characters. Longer mails are truncated.

Figure 6.22



i-Mode device

The devices on the market today allow Internet pages to be displayed in the S-JIS character set with GIF images; some phones even allow animated GIF. The number of supported HTML tags is different for each different device type. Simple devices support the following HTML 1.0 tags: <!-- -->, <A>, <BASE>, <BLOCKQUOTE>, <BODY>,
, <CENTER>, <DIR>, <DD>, <DIV>, <FORM>, <HEAD>, <H>, <HR>, <HTML>, <IMG>, <INPUT>, <LI>, <MENU>, <OL>, <OPTION>, <P>, <PLAINTEXT>, <PRE>, <SELECT>, <TEXTAREA>, <TITLE>, and <UL>.

New i-mode phones support SSL end-to-end security without client certificates. Root certificates of the well-known trust centers are stored in the phones. The user can disable these certificates partially or completely.

A Java Micro Edition VM (CLDC, see Chapter 3) is also integrated in new phone models. Java applications must implement a specific base class defined by the VM, because Java applets or AWT are not included in the class library. The functions of the VM are very limited, for example only one application can be executed at one time, and access to the phonebook is disabled due to security reasons.

i-mode phones do not support cookies, which means a special cookie proxy is necessary to support complex Web applications.

7 Voice technology

Automated speech recognition enables computers to convert spoken language to text. Speech is the natural communication method for humans, and therefore is a very convenient user interface for applications. At the same time, speech recognition is a difficult task for computers. Although speech recognition has been researched since the 1950s, commercial continuous speech-recognition systems with an acceptable recognition rate became available only a few years ago.

Within the next five years or so, pervasive devices will have enough memory and processing power to use speech recognition as a user interface method. This will have dramatic impact, especially on the consumer market. Today, the user interface of most video cassette recorders (VCRs) and mobile phones is so complex that the average user utilizes only a small part of the devices' features, because they do not know, or do not remember, how to use all the available features. This will change with voice-based user interfaces, because it will be a simple task for the user to tell the device what they want it to do.

Speech recognition may also start a new era in the pervasive computing space. Mobile phones have now reached a form factor where they can get no smaller because of the space needed for the keyboard and the display. Devices will become unusable if the size of the displays and keyboards is reduced further. With speech recognition, these devices will shrink even more, and could even be integrated into watches, jewellery, or clothing.

In this chapter, we will cover the basics of speech recognition. We will describe two important standards in the area of voice-based user interfaces: VoiceXML, a markup language for speech, and Java Speech, a Java API that provides speech capability to Java applications. Typical speech applications, such as speech recognition on the PC, speech recognition over a telephone line, and text-to-speech translation, will then be covered. Finally, we take a closer look at speech recognition in pervasive computing, and at security when using speech as user interface method.

7.1 Basics of speech recognition

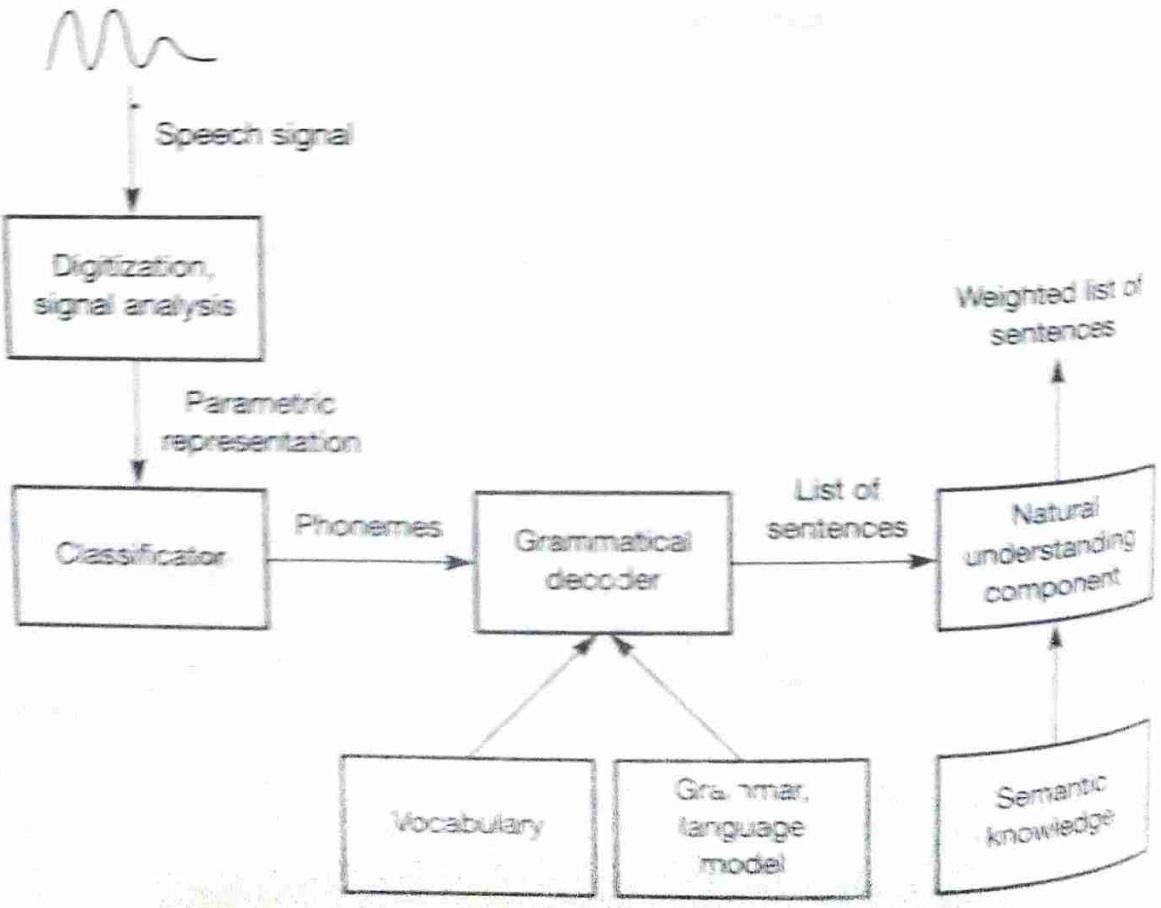
Speech recognition is a wide, complex field, and it is beyond the scope of this book to cover it in depth. However, we will give a short overview of the problems of speech recognition, and the approaches and techniques used to solve them.

7.1.1 Speech-recognition techniques

Just as written language has letters as units, speech recognition can be based on phonemes or words. The speech signal is transformed into frequency space representation for further processing. Only vowels generate resonance frequencies in the speech signal; consonants are produced by turbulent airflow in the human speech system, and therefore show up as noise in the frequency space of the speech signal. These resonance frequencies, or Eigenfrequencies, are called formants. From formants, the speech-recognition system can derive phonemes via signal analysis, using pattern-recognition algorithms.[1,2]

Figure 7.1 gives an overview of a continuous speech-recognition system. The system receives the digitized speech signal as input, and transforms it into the frequency space (e.g. by applying a Fourier transformation). The

Figure 7.1



Overview of a continuous speech-recognition system

following signal analysis calculates a feature vector representing the spectral distribution of the speech signal for a given time frame. This parametric representation of the speech signal is given to the classificator. The classificator uses pattern-recognition heuristics to generate phonemes out of the spectral vector (e.g. with techniques called segmentation and labelling). The grammatical decoder uses the vocabulary and grammar rules or language models (e.g. bi- or trigrams) to construct a list of possible sentences for the received phoneme sequence. A grammar or language model can reduce the number of possibilities by magnitudes, as not all words in a vocabulary are allowed at all places within a sentence. Finally, the natural understanding component takes this list and sorts out the most probable words based on semantic knowledge.

The recognition rate measures the accuracy of speech-recognition systems when analyzing speech. A word-recognition rate of 90% means that on average, one out of ten words is recognized wrongly.

7.1.2 Challenges of speech recognition

As mentioned above, a lot of research has been carried out since the 1950s to achieve speech-recognition systems with recognition rates that are acceptable for everyday use. These systems are still not available for the generic case of speaker-independent and task-independent speech recognition. This is due to the fact that speech recognition has some serious challenges.

The recognition accuracy is constrained by the following problems:

- *Isolated, connected, and continuous speech*. Isolated word recognition is one of the simpler problems, because isolated words are clearly separated and distinguished from each other. Continuous speech recognition, however, is very difficult because the word boundary for a spoken word is not defined clearly, poor articulation of words creates ambiguities, and stronger coarticulation (simultaneously produced adjacent vowels and consonants) causes problems. The worst case is spontaneous continuous speech, because the spoken sentences often are grammatically incorrect and may contain hesitations or press noises such as 'hmmm'. In connected word recognition, it is difficult to recognize the word boundaries, but there are no fill words as in continuous speech.
- *Vocabulary size*. The recognition accuracy depends on the vocabulary size. The vocabulary size varies inversely with the speech-recognition accuracy and efficiency, as more words introduce more ambiguities and require more time to process.
- *Speaker dependent/speaker independent*. Speech-recognition systems trained and used by a single speaker perform much better than

systems dealing with different speakers. This is due to the fact that a system used by only one speaker can be trained quickly and efficiently for that particular speaker.

- *Task and language constraints*. Speech recognition can be much simplified if the system uses limited grammar and dictionaries. For example, an airline information system can restrict its grammar to, say, 1000 possible combinations of words. Since this increases the recognition speed and accuracy dramatically, almost all speaker-independent speech-recognition systems with a word-recognition rate higher than 70% are using this technique.
- *Environmental noise*. Environmental noise can have a large impact on the recognition rate. Critical environments for speech-recognition systems are offices, public places, cars, trains, factories, etc. However, speech-recognition systems can be adapted through training and filtering to special environmental noise. In-car speech-recognition systems, for example, are using these techniques.
- *Channel quality*. The quality of the speech channel depends mainly on the microphone. Different microphones produce different speech signals. If the channel uses a digital line, such as GSM or UMTS, the digital processing of these channels also has an effect on the recognition rate. For more information about GSM and UMTS, see Chapter 4.
- *Line and digitization noise*. Regular analog lines may introduce systematic distortions, e.g. echoes during overseas calls, or damping of high frequencies. Digitization and short interrupts on digital lines (e.g. GSM, UMTS) also create systematic distortions.
- *Vocabulary ambiguity*. Words like 'show' and 'snow' sound similar, and it can be difficult for speech-recognition systems to recognize the correct word. Humans use context information to solve this problem. For example, it is easy for humans to decide that the sentence must be 'show me your house' and not 'snow me your house', but for a computer this is a complex task.

With so many problems to solve, speech-recognition systems that have a recognition rate of 99% for continuous speech, are speaker independent, and support a large, context-free vocabulary are far in the future. Nevertheless, speech-recognition do systems exist that achieve word recognition rates of over 95%, with a vocabulary of more then 60 000 words, after training by the speaker, for example from IBM or Lernout & Hauspie. However, even a rate of 95% still means that every twentieth word is recognized wrongly and must be corrected manually. With continuous improvements of hardware support and speech-recognition algorithms, speech recognition will increase usability and will be applied in new areas.

7.2 Voice standards

When using speech recognition in the computing world, there are two important standards to consider. The first is VoiceXML, which can be used in a Web environment together with a voice browser to make the Web application accessible through the telephone. The second is the Java Speech API, which enables Java programs to use speech as the user interface method.

7.2.1 VoiceXML

VoiceXML is a markup language for speech that is based on XML and enables Web applications for voice-accessed devices. Version 1.0 was developed by AT&T, IBM, Lucent Technologies, and Motorola, and has been submitted to the W3C for standardization.

The VoiceXML 1.0 specification provides a high-level programming interface to speech and telephony applications.[3] VoiceXML standardization has the following goals:

- To simplify creation and delivery of Web-based, personalized, interactive voice–response services.
- To enable phone and voice access to integrated call-center databases, information, and services on websites and company intranets.
- To enable new voice-capable devices and appliances.

A VoiceXML application is a collection of dialogs, which are equivalent to WML or HTML pages. The two basic dialog types are `forms`, which are used for presenting and gathering information, and `menus`, which offer the choices to the user. An example of a VoiceXML file is given in Example 7.1.

In Example 7.1, the user hears an introduction ("Welcome ...") and then a menu, where he or she can chose one option. If the system recognizes one of the keywords 'Menu', 'Order', or 'Exit', it processes the corresponding JSP or VoiceXML page. The recognition rate at this stage will be pretty good because the system must choose from a small list of only three keywords, which all sound quite different. To get even better recognition rates, a grammar can be specified with `<grammar ...>` (see the next section for how to specify speech grammars). VoiceXML allows a standard reply for unrecognized voice input to be defined. In Example 7.1, this is done with two levels: the first responds with the short text "Please say Menu, Order or Exit", whereas after the second unrecognized input the long version is used. For more VoiceXML examples, see Chapter 15.

e 7.1 Example of VoiceXML

```
<?xml version="1.0"?>
<vxml version="1.0">
 <form>
   <block>
    Welcome to Uncle Enzo's Pizza Service.
    <goto next="#main" />
   </block>
  </form>
  <menu id="main" scope="document">
   <prompt>Main Menu: <enumerate/></prompt>
   <choice next="/PCVShop/Shop/VoiceMenu.jsp">Menu
    </choice>
    <choice next="/PVCShop/Shop/VoiceOrder.jsp
      ">Order</choice>
   <choice next="/PVCShop/Shop/Exit.vxml ">Exit</choice>
   <prompt timeout="4s">What do you like to do? </prompt>
   <grammar src="/PVCShop/Shop/VoiceMenu.gram"/>
   <catch event="nomatch noinput help" count="1">
    Please say Menu, Order or Exit.
   </catch>
   <catch event="nomatch noinput help" count="2">
    Uncle Enzo's Pizza ordering service allows you to
order via the telephone. Say Menu to hear the menu, say
Order to create an order, say Exit or hang up to cancel.
   </catch>
  </menu>
</vxml>
```

7.2.2 Java Speech API

To use speech recognition and text-to-speech transformation from Java programs in a consistent manner, Sun, Apple, AT&T, Dragon Systems, IBM, Novell, Philips, and Texas Instruments developed the Java Speech API.[4] The API supports two core speech technologies: speech recognition and speech synthesis (text-to-speech).

The Java Speech API separates the speech-recognition and text-to-speech engines from the application. This allows for an easy exchange of the underlying speech engine without changing the application. The application can use the new engine and benefit from better recognition rates. It also allows building applications to be built in which customers

Some implementations of the Java Speech API are IBM's Speech for Java, Lernout & Hauspie's TTS, and Festival, developed at the University of Edinburgh.

Components of the Java Speech API

The Java Speech API consists of the following packages:

- *javax.speech* defines an abstract software representation of a speech engine. A speech engine can deal with either speech input or speech output. Therefore, speech recognizers and text-to-speech systems are both examples of speech engines. Speaker-verification systems and speaker-identification systems are also speech engines, but they are not currently supported through the Java Speech API.
- *javax.speech.synthesis* defines the API for text-to-speech engines. It is possible to customize the text-to-speech engine via the Java Speech Markup Language (JSML) API. JSML provides cross-platform control of speech synthesizers.
- *javax.speech.recognition* defines the API for speech-recognition engines. It allows for grammar-based speech recognition, and is a finite state machine. The grammar has to be in the Java Speech Grammar Format (JSGF), which provides cross-platform control of the speech recognizers.

Example 7.2 pronounces the string 'Hello World' in English. It uses the simple `speakPlainText` method to generate the speech output. In addition to `speakPlainText`, the Java Speech API provides a much more flexible method for generating speech output: `speak`. This requires input in the JSML.

Example 7.3 shows how to recognize "Hello World". Grammar in the JSGF is needed. The trivial grammar shown has a single public rule called `<sentence>` that defines what a user may say. Valid sentences with this grammar are `hello world`, `hi there`, or `hello computer`.

The Java program for our example could look like the code shown in Example 7.4.

The main method sets up the speech recognizer, assigns the grammar, and adds the `HelloWorld` class as listener. As `HelloWorld` is subclassed from `ResultAdapter`, the method resultAccepted is called from the speech recognition engine, if a spoken sentence matches the given grammar. The implementation of our `resultAccepted` method gets the best recognition match for the spoken text with `r.getBestTokens` and prints the recognized words to the standard output stream.