J Comput Virol (2006) 2:87–97
DOI 10.1007/s11416-006-0017-x

ORIGINAL PAPER

End-to-end security implementation for mobile devices using TLS protocol

Baris Kayayurt · Tugkan Tuglular

Received: 13 January 2006 / Revised: 11 February 2006 / Accepted: 28 March 2006 / Published online: 11 July 2006

Abstract End-to-end security has been an emerging need for mobile devices with the widespread use of personal digital assistants and mobile phones. Transport Layer Security Protocol (TLS) is an end-to-end security protocol that is commonly used on the Internet, together with its predecessor, SSL protocol. By implementing TLS protocol in the mobile world, the advantage of the proven security model of this protocol can be utilized. The main design goals of mobile end-to-end security protocol are maintainability and extensibility. Cryptographic operations are performed with a free library, Bouncy Castle Cryptography Package. The object oriented architecture of proposed end-to-end security protocol implementation makes the replacement of this library with another cryptography package easier. The implementation has been experimented with different cases, which represent use of different cryptographic algorithms.

1 Introduction

Current trends in mobile applications have been the enterprise-style applications that needed high-capacity, network-connected devices. Financial applications like banking and stock trading are common examples of these kinds of applications. However, some deficiencies prevent its acceptance in e-commerce applications [12]. With more and more network connected applications, security has became one of the most popular concept in mobile community. Mobile security deals mainly with two issues: security of the physical device and its contents and security of the data in network communication.

The communication of data in mobile devices is provided by the mobile networks. Mobile networks are open to many kinds of attacks. The open data communication in these networks may cause attacks against the secrecy, integrity and authenticity of data. Many vendors have proposed solutions against these security vulnerabilities. Most of these solutions are vendor-specific proprietary products or libraries.

Jøsang and Sanderud [6] investigated the aspects of mobile networks and found that it is both harder and easier to implement communication security as compared to the Internet for example. They concluded that communication between mobile and fixed networks create particular problems regarding security protocol design. Data security problems can be minimized through the implementation of end-to-end security in this proxy-based environment.

Many mobile networks have proxy-based architecture. In this architecture, security between the mobile device and the proxy server is provided with a solution and the security between the proxy server and the destination server is provided with another solution. WTLS, announced as the security solution of WAP protocol, is such a protocol. There are two problems with proxy-based solutions. First, it decreases performance. As the data is decoded and reencoded at proxy server, it may cause latency. Second, regardless of using WTLS, a malicious operator can eavesdrop and tamper with the

B. Kayayurt (✉)
Havelsan, Eskisehir Yolu 7. Km.,
06520 Ankara, Turkey
e-mail: kayayurt@bornova.ege.edu.tr

T. Tuglular
Department of Computer Engineering,
Izmir Institute of Technology,
Gulbahce Koyu, Urla, Izmir, Turkey
e-mail: tugkantuglular@iyte.edu.tr

data [10]. These attacks may be a threat to the data security between the period it is decoded and encoded.

The alternative to proxy based security solutions is end-to-end security. End-to-end security refers to the securely encoding of data at the source host and decoding at the destination host. The data will not travel unencoded at any part of the communication. Transport Layer Security Protocol (TLS) and its predecessor Secure Sockets Layer (SSL) protocols are end-to-end security solutions that are commonly used in the wired world [1]. There are a number of reasons to use these protocols in the mobile world:

- TLS and SSL protocols have been used for many years, so their security is tested by the community and accepted as secure enough to be used by financial data communication.
- TLS and SSL protocols are common in wired world and may be accepted as the security protocol of Internet. Using these in mobile applications will make the integration between mobile applications and Internet easier.
- TLS and SSL have open specifications and many implementations. It may be relatively easy to implement these for specific needs.

Both protocols are known by their resource consuming nature and the idea that they are not suitable for mobile networks. The end-to-end security solution presented in this paper is implemented for mobile devices using Java platform in order to show that it is possible and efficient to use TLS in mobile networks. Kwon et al. [8] worked on the same problem and published an end-to-end security model that utilizes transport layer security to overcome WAP security problems. However, implementation of their model and related experiments with resulting values cannot be found.

In this paper, only transport layer security for mobile devices has been discussed. Other security issues are out of scope of this paper. Any implementer of the transport layer security for mobile devices should be aware of the security weaknesses that the J2ME environment contains [4].

This paper presents a solution to end-to-end security needs of mobile devices, such as PDAs and mobile phones. After a brief explanation of mobile architectures is given, the proposed architecture for end-to-end security of mobile devices is described. The next section mentions the implementation details of the architecture; the development environment and some implementation issues. Then the experiment enviroment and experiments are explained together with their results. The paper ends with a discussion of the results of experiments.

2 Mobile architectures

Mobile Java applications run on mobile devices, mostly mobile phones and Personal Digital Assistant (PDA) devices. Whatever the device is, the application platform has a unique architecture. This architecture consists of the device's architecture (hardware, OS, JVM, etc.), network connection architecture that connects the mobile device to the outside world and the application architecture. The mobile security protocol developed will run on the architecture explained below.

2.1 Device architecture

Today, many cell phones and PDAs support Mobile Information Device Profile (MIDP) technology. Usually, cell phones have built-in support for either MIDP 1.0 or 2.0; and PDAs (mostly PALM based PDAs) support MIDP after installing the KVM (Kilobyte Virtual Machine) specific for these devices. A variety of devices from different vendors have J2ME (Java 2 Micro Edition) MIDP support [7]. In order to support this kind of flexibility and customization need, J2ME architecture is designed to be modular and scalable. Figure 1 shows the general architecture of a J2ME MIDP device with its layered approach, based on [13]. In this architecture, the host operating system communicates with the device hardware and provides services for the Java Virtual Machine (JVM) layer. JVM layer is the implementation of a Java virtual machine that is customized for a particular device's host operating system and supports a particular J2ME configuration. The configuration layer defines the minimum set of Java virtual machine features and Java class libraries available on a particular category of devices. The profile layer defines the minimum set of Application Programming Interfaces (APIs) available on a particular family of devices. The layer where Java applications are written is the Profile Layer. Mobile device operating systems control the mobile device hard-

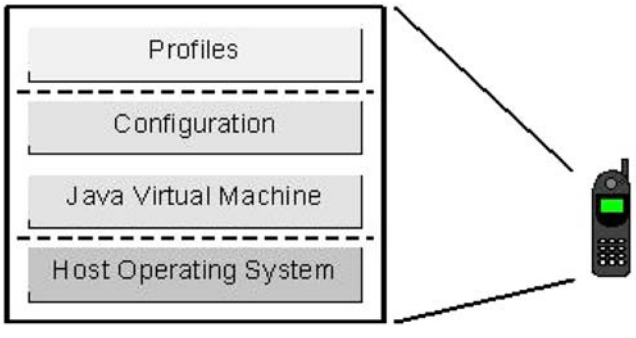

Fig. 1 J2ME device architecture [13]

ware and provide services for the JVM layer. Although MIDP applications run on top of a JVM and are independent of the operating system, the implementation of the JVM is customized for the operating system. So the operating systems capabilities define the MIDP application capabilities. The most common operating systems in MIDP supporting devices are PALM OS and Symbian OS.

As in desktop, Java applications on mobile devices run on top of a virtual machine called JVM. The virtual machine is implemented in native code and may directly use operating system services. The VM of mobile devices are different from J2SE JVM because of the limited resources and device architectures of mobile devices. The K Virtual Machine (KVM) is a highly portable JVM designed from the ground up for small memory, limited-resource and network-connected devices such as cellular phones, pagers and personal organizers [13].

Configuration Layer defines the minimum set of Java virtual machine features and Java class libraries available on a particular "category" of devices. Connected, Limited Configuration Layer (CLDC) is the configuration specified for small, resource constraint mobile devices. CLDC specification defines Java language and virtual machine features, core libraries, input/output, networking, security and internationalization [14]. CLDC configuration does not address application lifecycle management (installation, launching, deletion), user interface, event handling and high level application model. These features are addressed by profiles implemented on top of the CLDC.

Profile Layer defines the minimum set of APIs available on a particular family of devices. Profiles are implemented on top of configurations. A device can support multiple profiles. MIDP (see [15,16]) is the most common profile implemented in all J2ME profiles. MIDP defines a very limited API because of size and performance reasons. However, it can be extended with various optional packages. Bluetooth (JSR 82), Web services (JSR 172), wireless messaging (JSR 120), multimedia (JSR 135), and database connectivity are some of the optional packages developed for CLDC/MIDP environments. Device manufacturers may or may not include optional packages in their device implementations.

2.2 Network connection architecture

A Wireless Local Area Network (WLAN) is a flexible data communications system that can either replace or extend a wired LAN to provide added functionality [5]. WLANs use Radio Frequency (RF) to transmit and receive data over the air without cables. Data is superimposed onto a radio wave through a process called modulation, and this carrier wave then acts as the transmission medium, taking the place of a wire. WLANs use the 2.4 Gigahertz (GHz) frequency band. WLANs offer all the features of traditional Ethernet or Token Ring networks with the addition of increased network infrastructure flexibility. This flexibility makes wireless networks an important alternative to wired-networks in small areas. WLANs provide a communication medium for wireless mobile devices (PALM, e.g.) in small areas like a building.

An alternatively new technology in mobile device communication is Bluetooth. Bluetooth provides short-range wireless connectivity over radio-frequency technology that uses the 2.4 GHz Industrial-Scientific-Medic al (ISM) band [9]. Bluetooth lets mobile devices communicate with each other up to 10 m range and 1 Mbit/s speed. The IEEE has designated its version of Bluetooth with 20 Mbit/s speed, as the IEEE 802.15 standard.

GPRS (General Packet Radio Service) is an enhancement of core GSM (Global System for Mobile Communications) networks that allows the rapid transfer of data bundled into packets, separate from voice or data call circuits [17]. GPRS is a 2.5 Generation (2.5G) technology that is the last stone before the coming 3G networks that will give high speed access allowing live video. GPRS data is transmitted in packets, up to the speed of 20 or 30 Kbps though the theoretical maximum speed is 171.2 Kbps. GPRS set-up time is short and connection is always on.

2.3 Application architecture

Client/server architecture is the main architecture for mobile network applications. In this architecture, a network-aware application residing on a wireless device, client, connects with back-end applications and servers behind a firewall or proxy gateway over a wireless network and Internet and corresponding communication protocols. Figure 2 shows the client/server architecture of a mobile application.

In Figure 2, the wireless devices can be mobile phones, PDAs or two-way pagers. They have the mobile applications on top of MIDP and communicate with an antenna over a wireless communication protocol. These protocols can be Wireless-LAN on a small area or GPRS on a wide area. The antenna is directly connected to a wired network like Internet, which connects the mobile device to the back-end server systems.

Mobile applications typically use HTTP as the application-level communication protocol as it is common and can pass firewalls. HTTP is the mandatory protocol in MIDP 1.0 [15] and MIDP 2.0 specifications [16].

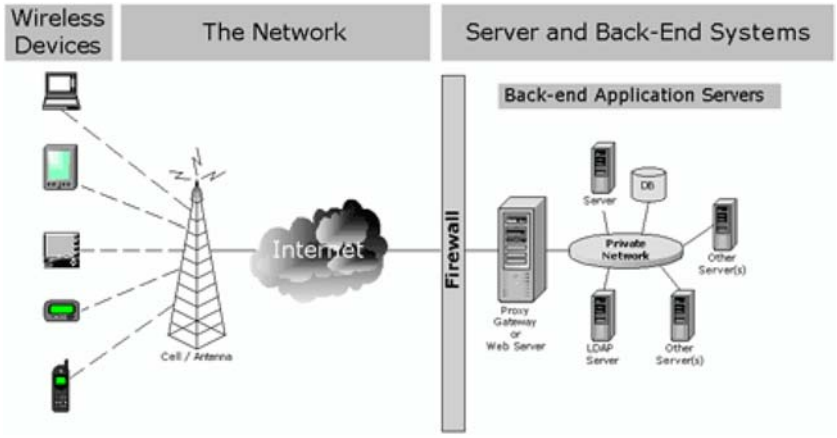

Fig. 2 Environment of a typical networked wireless application [11]

With MIDP 2.0, low-level socket APIs can also be used to directly communicate with the server application below the application level.

TLS may provide an application-level end-to-end security on top of sockets in this architecture. The TLS implementation developed in this thesis runs on top of pure sockets and provides an application level security between a MIDP application and server back-end application. The MIDP version of the developed TLS protocol communicates with the J2SE version of the protocol and sends object data over the secure connection. The mobile device always runs the client version of TLS protocol and the back-end server always runs the server version of the TLS protocol. Peer To Peer Architecture is an alternative and new architecture for wireless devices. In this architecture, the two devices communicate directly with each other. This communication may be the direct communication of the devices over a communication medium like Bluetooth or it may be an application level communication with the two mobile applications communicating over a central server.

The client side of the application is the same as the Client/Server Architecture. It uses either HTTP or low-level socket APIs, either stream-based or datagram-based, for network communication. The difference is in the server side. In this architecture, the server is also a mobile device. MIDP 2.0 has an API for server sockets that may be used for this purpose. The TLS implementation to work in this architecture needs both client and server versions of TLS to work in the mobile device.

This may be impractical for two reasons. First, resource constraints of mobile devices may cause performance problems for server side implementation of the TLS protocol. Second, server sockets are not commonly implemented in MIDP devices. In spite of these deficiencies, the TLS implementation presented in this paper can be used in peer-to-peer architectures for experimental purposes.

3 Design of mobile devices end-to-end security architecture

The security protocol presented in this paper is an end-to-end security protocol based on TLS 1.0 specifications and adopted to work on J2ME mobile devices as well as standard Java VMs. The protocol implementation itself is also an application, although high-level applications may use it through its APIs to transmit data and objects securely.

The security protocol architecture developed covers both TLS protocol architecture and the necessary APIs architecture. TLS protocol architecture is based on TLS 1.0 specification and has some additions and subtractions. The necessary APIs architecture involves cryptography model classes that abstract the real implementations of cryptography packages; socket classes that abstract TCP or UDP based socket implementations and the serialization of object data; XML Serializer architecture that is a standalone API incorporated into the protocol implementation. The main design issues taken into consideration during the definition of the architecture of protocol implementation are J2ME compatibility, mobile adaptation, secure object transmission, full abstraction and complete solution.

The main architecture of the mobile end-to-end security protocol is based on the TLS 1.0 protocol specification [3]. TLS is a protocol that consists of several layers of protocols as shown in Figure 3. At the bottom of these layers, there is the TLS Record Protocol. The Record Protocol is responsible for taking messages to be transmitted, fragmenting into blocks, optionally compressing, applying MAC to protect data integrity, encrypting the block and transmitting the result to higher level clients. Received data is then decrypted; applied MAC is verified to protect data integrity; optionally decompressed; reassembled and then delivered to higher level clients by the Record Protocol. TLS Record Protocol

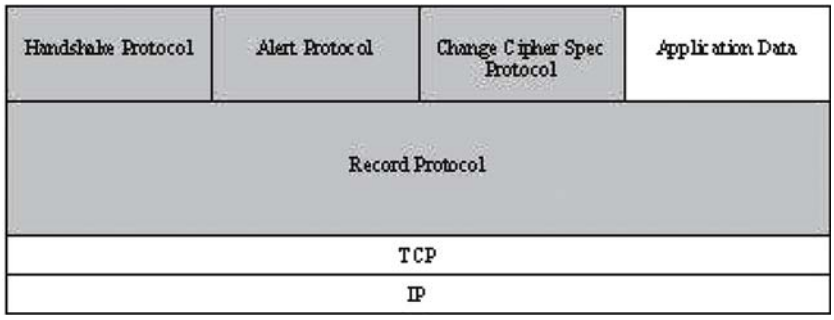

Fig. 3 The layered structure of TLS protocol

blocks are transported over a reliable transport protocol like TCP. There are four clients of record protocol: TLS Handshake Protocol, TLS Alert Protocol, TLS Change Cipher Spec Protocol and application data. TLS 1.0 Specification allows new record protocol clients to be supported by the record protocol. TLS Handshake Protocol is used to allow peers to authenticate each other and to exchange cryptographic parameters that are used in TLS Protocol. TLS Handshake is performed at the beginning of a session before the application data is transmitted or received. The Change Cipher Spec Protocol is used to start to use new keys and encryption methods that the Handshake Protocol has established. TLS Alert Protocol is used to send warning and fatal level errors that could occur in TLS session. After the handshake is performed, application data is taken by Record Layer and sent securely to the other peer.

In our model, the Record Layer behavior is implemented in the class RecordLayerImpl and the Handshake Layer behavior is implemented in the class HandshakeLayerImpl. This class has two instances of class RecordLayerImpl; one for current state and one for pending state. Both HandshakeLayerImpl and RecordLayerImpl classes are transparent of the underlying socket implementations and talk to the peer classes in both sides of the communication.

The classes TLSClientSocket and TLSServerSocketListener are the only classes needed to be known by the applications that will use this implementation of the protocol. An application that needs to be the client in the secure communication must use the class TLSClientSocket; an application that needs to be the server in the secure communication must use the class TLSServerSocketListener. The class TLSServerSocket is used to handle a secure connection session in the server-side as a separate thread. An application may use the class TLSSocketFactory to obtain either a TLSClientSocket class instance or a TLSServerSocketListener class instance.

The mobile secure application is designed as the protocol implementation separated from the underlying communication model. This is achieved by the interface TLSSocketImpl. This interface defines the operations needed for sending and receiving of secure data and implemented by the client and server socket classes that abstract the communication details from the protocol details. Thus, the communication of data can be changed without changing any protocol dependent code. The class TLSTCPSocketImpl is the client-side implementation of the interface TLSSocketImpl as TCP based stream socket and the class TLSTCPServerSocketImpl is the server-side implementation of the interface TLSSocketImpl as TCP based stream server socket. The class TLSUDPSocketImpl is the client-side implementation of the interface TLSSocketImpl as UDP based datagram socket and the class TLSUDPServerSocketImpl is the server-side implementation of the interface TLSSocketImpl as UDP based datagram socket. Figure 4 shows the main object model of the mobile end-to-end security protocol.

The class HandshakeLayerImpl defines operations and attributes needed to perform Handshake Layer behavior and to send and receive application-level data after the establishment of a secure session. Each secure session has one instance of class HandshakeLayerImpl and the class instances at both side of the communication talk to each other. While the Handshake Layer in TLS Protocol communicates with the Record Layer to send and receive data, the HandshakeLayerImpl class uses RecordLayerImpl class to send and receive data.

The class RecordLayerImpl defines operations and attributes needed to perform Record Layer behavior. The responsibilities of RecordLayerImpl class are to send and receive application and Handshake Layer data after encoding and decoding; generate new TLS keys according to the exchanged security parameters; copy security parameters from pending state to current state and activate new TLS keys for read side and write side. In TLS 1.0, the Record Layer performs encryption, MAC and compression functions according to the algorithms exchanged during handshake procedure. The Record Layer architecture of the mobile protocol is modeled to support this need in an extensible way. This is

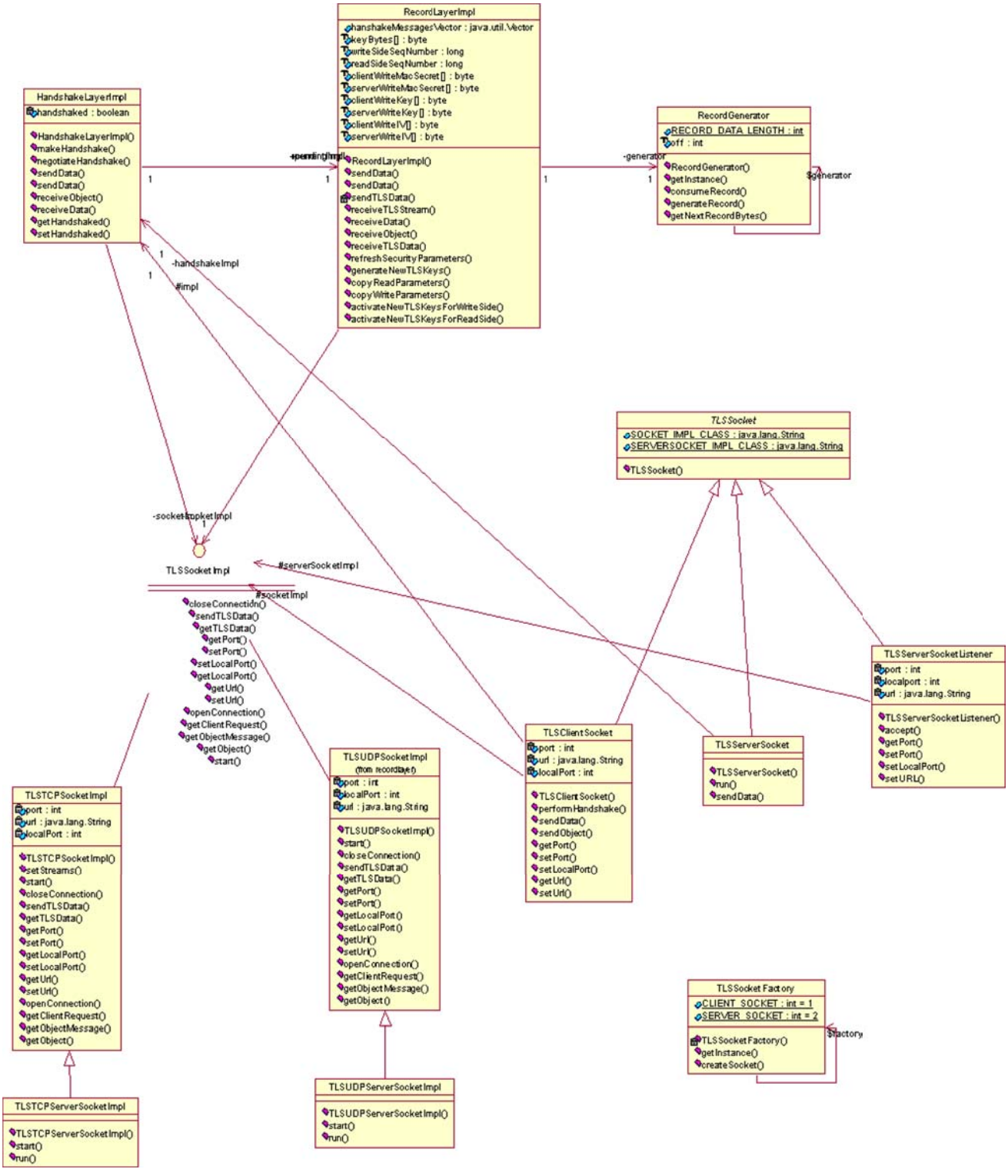

Fig. 4 Main object model of the mobile end-to-end security protocol

achieved with the use of the interfaces TLSEncryption, TLSMac and TLSCompression. These interfaces define the base methods to perform encryption, decryption, adding MAC, verifying MAC, compression and decompression. Algorithms are implemented in the classes implementing these interfaces. This method eases the addition of a new algorithm to the protocol implementation.

4 Implementation of mobile devices end-to-end security architecture

The mobile end-to-end security protocol is implemented in Java to be compliant to both J2ME and J2SE platforms. The development environment for mobile end-to-end security protocol is JBuilder 8.0 Enterprise MobileSet 3 Integrated Development Environment. The development environment was chosen because it has an integrated MIDP application development support. A project may be compiled with MIDP libraries by changing the project JDK to J2MEWTK from the project properties window. JBuilder IDE was also chosen for its high performance, code completion feature and user-friendly interface.

JBuilder IDE uses Sun Microsystem's J2ME Wireless Toolkit for MIDP development. J2ME Wireless Toolkit (J2MEWTK) is a toolkit that can be used separately or plugged into a development IDE. It provides development tools and emulators for MIDP application development. The 2.1 version of J2MEWTK is used in the development and in the experiements of the developed protocol for mobile environments. J2MEWTK 2.1 supports MIDP 2.0, CLDC 1.1, optional Wireless Messaging API (JSR 120), Mobile Media API (JSR 135) and Web Services Access for J2ME API (JSR 172).

J2ME Wireless Toolkit has the tool Ktoolbar that manages MIDP application development. Ktoolbar has menu options for creating a new MIDP application, opening an existing one, compiling and running the application with the set up MIDP version, changing the J2MEWTK and application preferences.

J2MEWTK uses device emulators to run MIDP applications. Device emulators are software implementations of mobile devices to test applications before the real deployment. Device emulators provide a faster and easier development for mobile applications. Most mobile phones and PALM devices have their emulator software. Mobile phone emulators are distributed by telephone vendors freely. They have the same operating system and visual interface with the original telephone. PALM emulators emulate PALM OS. A ROM file is taken from a real PALM device with Hotsync connection and loaded into the emulator software. This gives all the features of the PALM device to the emulator software including program installation and uninstallation. Some examples are also run on original PALM M100 PDAs and Nokia 6600 mobile phones.

Java has been divided into three platforms: J2EE for server side development, J2SE for standard desktop applications and J2ME for small, embedded devices. J2ME has its own configurations and profiles. Each configuration and profile has its own library APIs. The main target Java platform for the mobile security protocol is J2ME platform. The configuration is CLDC and the profile is MIDP. This fact caused CLDC/MIDP APIs to be used in the implementation of the protocol.

Most CLDC/MIDP language APIs are also valid in J2SE platform, that makes the developed protocol also compliant to this platform. However, because of the limits of mobile devices, network connection library has been changed. J2ME uses Generic Connection Framework (GCF) for all kinds of connections including network and file connections. This brings a lighter connection API when compared with the heavy network API of J2SE platform. J2ME version of the protocol uses GCF and the J2SE version uses java.net package for network connections.

The security protocol implementation needs secure network connections to send and receive data between peers. Network connection implementations are abstracted from business logic classes and appear in socket classes that are used by other classes.

J2ME uses GCF and J2SE uses java.net package network connection classes to provide network connections. The protocol implementation may use TCP and UDP sockets and server sockets for network connections between client and server. The socket connection type is determined by the constants in class TLSSocket.

The Bouncy Castle Crypto package is a Java implementation of cryptographic algorithms. The package is organised so that it contains a light-weight API suitable for use in any environment (including the newly released J2ME) with the additional infrastructure to conform the algorithms to the JCE framework [2]. The Bouncy Castle Crypto package can be downloaded from `http://www.bouncycastle.org/` for free. This software is distributed under a license based on the MIT X Consortium license.

5 Experiments and discussion

The mobile end-to-end security protocol was implemented mainly to be used in mobile devices, PDAs and mobile phones, for example. The implemented protocol library is loaded on a real mobile phone to show that it can run on a resource-constraint environment and establish network connection on a real wireless network. Mobile end-to-end security protocol was designed and implemented to have a reliable and maintainable protocol implementation that would have acceptable performance results. This section mentions the experiments performed to measure the performance averages of the protocol implementation. It presents the scope of experiments, the platform experiments were performed,

different experiment cases prepared and the evaluation of the experiment results. The result values of those experiments are shown as well.

5.1 Scope of the experiments

The mobile end-to-end security protocol experiments involve the performance runs of both the client and server modes of the protocol with different cipher suites. Experiments cover both J2SE (as server) and J2ME (as client) platforms. The secure connection will be established with different cipher suites and TCP is used as the underlying network connection protocol.

The experiments of the protocol are divided into two main sections:

- Experiment of the handshake procedure.
- Experiment of the application level object transmission and receiving.

As the handshake procedure has many steps, it is divided into different time spans. In this way, it is aimed to measure the real performance of the steps and to define bottlenecks in the protocol implementation. Some operations will only be available in specific cipher suites. Table 1 shows the operations in both server and client modes of the protocol. Experiments were performed by using different TLS cipher suites. Server side of the program supports all cipher suites available to the mobile security protocol implementation. Each experimental client supports only one cipher suite and requests the handshake to be performed with that cipher suite when it sends ClientHello message to to the server. Thus, the connection will be established with that cipher suite. Table 2 shows cipher suites used in the protocol experiments. The first cipher suite in Table 2 is non-ephemeral, that means asymmetric keys are not generated at run-time. The other cipher suites in Table 2 are ephemeral, that means asymmetric keys are generated at run-time. Experiment results are organized to compare ephemeral cipher suite results with each other. The other comparison given is between non-ephemeral RSA and ephemeral RSA.

Table 1 Mobile security protocol operations used in experiments

Operation	Server	Client
Public key signing	◇	
Public key verification		◇
Key pair generation	◇	◇
Pre-master secret generation	◇	◇
Master secret generation	◇	◇
TLS keys generation	◇	◇
All handshake	◇	◇

The experiments were performed according to the following scenario:

- A server listening to secure connection requests.
- A client requesting a connection with the server.
- Performance of the handshake procedure.
- Sending of an example object from client to server over the secure connection established. The example object is called Person that has the private attributes id, name and surname.

5.2 Mobile device experiment platform configuration

Mobile end-to-end security protocol was designed and implemented to be run on J2ME CLDC/MIDP platforms including cellular phones and PALM PDAs. The experiments with a real mobile device are performed by running the experimental client program on a mobile phone and server program on a desktop PC. The mobile phone is a Nokia 6600 smart phone with ARM9 104 MHz CPU, 6 MB of storage memory, GPRS and Bluetooth connectivity and Symbian OS 7.0s operating system with MIDP 2.0 support. The network connection between the mobile phone and the desktop PC is provided with the GPRS connection provided by Turkcell mobile service carrier. The transport protocol used in the experiments is TCP.

Mobile device experiments were performed only on J2SE-J2ME architecture mentioned in section 5.1. Experimental programs run ten times and the highest (max), lowest (min) and average time durations are measured.

5.3 Mobile device experiment results

Mobile device experiments were performed according to the configuration explained above. There was no problem in establishing communication with GPRS and all experiments succeeded. Figures 6 and 7 (see Appendix) show the secure communication time span durations when the experimental client program runs on a Nokia 6600 mobile phone, the server program runs on J2SE platform and transport protocol between is TCP. All values presented in the tables are milliseconds (ms). “0” value in a table means that the time duration is below milliseconds. N/A value in a table means that the operation is not applicable in the cipher suite.

Figure 5 shows the comparison chart of average client total handshake times of the experimental cipher suites when the client runs on a Nokia 6600 mobile phone and the server runs on J2SE platform and the transport

Table 2 Cipher suites used in protocol experiments

Cipher suite	Authentication and key exchange algorithm	Symmetric algorithm	Hash algorithm
TLS_RSA_WITH_AES_CBC_SHA	RSA	AES	SHA
TLS_RSA_EXPORT_WITH_RC2_CBC_40_MD5	RSA_EXPORT	RC2	MD5
TLS_DHE_RSA_WITH_DES_CBC_SHA	DHE_RSA	DES	SHA
TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA	ECDHE_ECDSA	AES	SHA

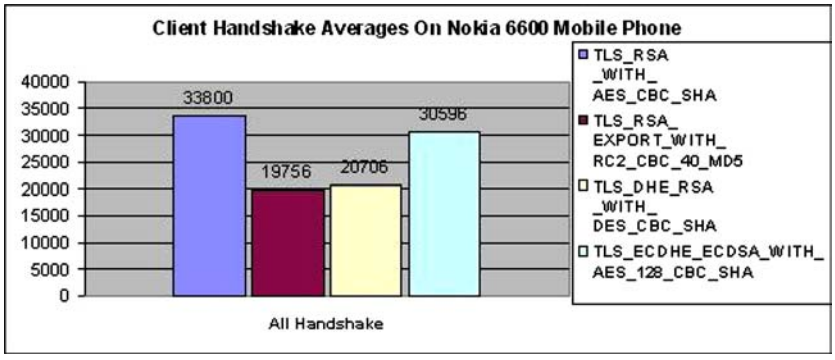

Fig. 5 Client average handshake times on Nokia 6600 mobile phone

protocol between is TCP. The results of the experiments performed were shown in tables given in Appendix. The handshake procedure is divided into operations. Each operation is an important step in the handshake that was expected to take a measurable time. The experiments were repeated for a definite number for each case and the resulting average, max and min values were noted. Experiment results will be evaluated according to the following criteria:

- Defined operations;
- Running platform;
- Network protocol;
- Cipher suite.

All the average, min and max values increase in the J2SE-J2ME experiment results shown in Figures 6 and 7. This architecture includes a client in J2ME platform and a server in J2SE platform and expected to be widely used in real life. The lowest average total handshake value in this architecture is 19756 milliseconds of TLS_RSA_EXPORT_WITH_RC2_CBC_40_MD5 cipher suite. The highest average total handshake value is 33,800 milliseconds of TLS_RSA_WITH_AES_CBC _SHA cipher suite. Considering ephemeral, the highest average total handshake value is 30,596 milliseconds of TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA cipher suite. This shows that Elliptic Curve Diffie -Hellman operations have a poor performance on J2ME platform. These results are shown in the chart in Figure 5.

6 Conclusion

End-to-end security is an emerging need for mobile devices. Banking, military and other enterprise applications need more and more security to run on mobile devices. This project aimed at developing an extensible end-to-end security protocol implementation that could be used by both mobile and desktop applications. TLS protocol that is commonly used on the Internet was chosen as the base of the developed protocol implementation.

The proposed protocol was designed and implemented. The implementation was run with different experiments and the time span of operations were measured. The protocol implementation run on a real Nokia 6600 mobile phone and established a secure connection with a server computer connected to the Internet over GPRS. All the experiments were successfully completed, which indicated that the protocol was properly designed and implemented with respect to specifications. The implementation guarantees the security of any transmission at most as much as TLS. The implementation did not include optional TLS specifications like client authentication, session resumption and compression.

Appendix

Experimental results

Fig. 6 Cryptographic operation durations in the mobile device experiments for ephemeral

Time Span	TLS_RSA_EXPORT_WITH_RC2_CBC_40_MD5 (ms)			TLS_DHE_RSA_WITH_DES_CBC_SHA (ms)			TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA (ms)		
	Avg.	Max	Min	Avg.	Max	Min	Avg.	Max	Min
Client									
Public key verification	3021	5562	2672	3271	5781	2938	8251	11156	7640
Premaster secret generation	322	359	282	17	31	0	2061	2235	1938
Master secret generation	129	281	78	120	265	63	148	250	79
Key pair generation	N/A	N/A	N/A	303	625	234	2369	2547	2219
TLS keys generation	798	4625	359	804	4610	359	841	4671	390
All Handshake	**19756**	**45125**	**16156**	**20706**	**42766**	**17000**	**30596**	**68906**	**25625**
Time Span	**TLS_RSA_EXPORT_WITH_RC2_CBC_40_MD5 (ms)**			**TLS_DHE_RSA_WITH_DES_CBC_SHA (ms)**			**TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA (ms)**		
	Avg.	Max	Min	Avg.	Max	Min	Avg.	Max	Min
Server									
Public key signing	43	121	20	42	90	30	72	160	40
Key pair generation	811	2673	210	1743	4767	30	N/A	N/A	N/A
Premaster secret generation	26	31	20	1	10	0	17	115	11
Master secret generation	8	60	0	8	60	0	5	50	0
TLS keys generation	10	60	0	42	421	0	10	60	0
All Handshake	**11106**	**22943**	**9243**	**12060**	**23194**	**9764**	**20364**	**32717**	**18256**

Fig. 7 Cryptographic operation durations in the mobile device experiments for RSA ciphers

Time Span	TLS_RSA_WITH_AES_CBC_SHA (ms)			TLS_RSA_EXPORT_WITH_RC2_CBC_40_MD5 (ms)		
	Avg.	Max	Min	Avg.	Max	Min
Client						
Public key verification	N/A	N/A	N/A	3021	5562	2672
Premaster secret generation	600	1781	578	322	359	282
Master secret generation	167	312	125	129	281	78
Key pair generation	N/A	N/A	N/A	N/A	N/A	N/A
TLS keys generation	801	4234	390	798	4625	359
All Handshake	**33800**	**82375**	**13907**	**19756**	**45125**	**16156**
Time Span	**TLS_RSA_WITH_AES_CBC_SHA (ms)**			**TLS_RSA_EXPORT_WITH_RC2_CBC_40_MD5 (ms)**		
	Avg.	Max	Min	Avg.	Max	Min
Server						
Public key signing	N/A	N/A	N/A	43	121	20
Key pair generation	N/A	N/A	N/A	811	2673	210
Premaster secret generation	55	120	30	26	31	20
Master secret generation	7	50	0	8	60	0
TLS keys generation	8	60	0	10	60	0
All Handshake	**19657**	**64893**	**6840**	**11106**	**22943**	**9243**

References

1. Agarwal, A.K., Gill, J.S., Wang, W.: An experimental study on wireless security protocols over mobile IP networks. In: IEEE Proceedings of 60th Vehicular Technology Conference (2004)
2. BouncyCastle: Bouncy Castle Documentation. (2006) http://www.bouncycastle.org/documentation.html
3. Dierks, T., Allen, C.: The TLS protocol version 1.0, IETF RFC 2246 (1999)
4. Reynaud-Plantey, D.: J2ME low level security: implementation versus specification. (2005) http://prdownloads.sourceforge.net/tinapoc/Reynaud_J2ME.pdf?download
5. Intel IEEE 802.11b high rate wireless local area networks, Intel Corporation (2000)
6. Jøsang, A., Sanderud, G.: Security in mobile communications: challenges and opportunities. In: Australasian Information Security Workshop (AISW2003), Australia (2003)
7. Knudsen, J.: Introduction to Wireless Java Technologies White Paper, Sun Microsystems Inc. (2001)
8. Kwon, E., Cho, Y., Chae, K.: Integrated transport layer security: end-to-end security model between WTLS and TLS. In: IEEE Proceedings of 15th International Conference on Information Networking, pp. 65–71 (2001)
9. Mahmoud, Q.H.: Wireless Application Programming with J2ME and Bluetooth. (2003) http://developers.sun.com/techtopics/mobility/midp/articles/bluetooth1/
10. Mynttinen, J.: End-to-end security of mobile data in GSM. Tik-110.501 Seminar on Network Security. Helsinki University of Technology (2000)
11. Ortiz, E.: The Complexity of Developing Mobile Networked Data Services, J2ME Wireless Connection Wizard For Sun ONE Studio. (2006) http://developers.sun.com/techtopics/mobility/midp/articles/wizard/index.html
12. Soriano, M., Ponce, D.: A security and usability proposal for mobile electronic commerce. IEEE Commun Mag, 40(8) 62–67 (2002)
13. Sun Microsystems: Java 2 Platform Micro Edition (J2ME) Technology for Creating Mobile Devices White Paper, Sun Microsystems Inc (2000)
14. Sun Microsystems: Connected, Limited Device Configuration 1.0a Specification, Sun Microsystems Inc. (2000)
15. Sun Microsystems: JSR-000037 Mobile Information Device Profile (MIDP) 1.0 Specification, Sun Microsystems Inc. (2000)
16. Sun Microsystems: JSR-000118 Mobile Information Device Profile 2.0 Specification, Sun Microsystems Inc. (2002)
17. Symbian: Symbian on GPRS (2001). (2001) http://www.symbian.com/technology/standard-gprs.html