BUILD A FRAMEWORK TO OPTIMIZE M-COMMERCE SECURITY

K. SRIDHAR
Research scholar, CSE Dept., JNTUH Prof,
Hyderabad, Telangana
Email:sridhark529reddy@gmail.com

DR. D. SURESH BABU
Supervisor, Assoc. Prof, CSE Dept.
JNTUCEJ, Warangal,
Telangana

DR.T.VENUGOAPL
CO- Supervisor, Assoc.
Sulthanapoor, Medak,
Telangana

ABSTRACT:

Mobile commerce (m-commerce) is as long as industrial services those area unit accessible by victimization mobile devices, PDA, etc. the most benefits of such services area unit their high handiness, independence of physical location and time. However the move to make a wireless version of net suggests that a brand new set of issues. Like the prevailing fastened net, the most important downside is security. Even though the very fact that operators area unit asserting or rolling out Wireless Applications Protocols (WAP), I-mode and java-based info, the platforms have opened security holes.

This paper aims to present some suggestions to enhance m-commerce security and limit the m-commerce drawbacks. These suggestions associated with the subsequent functional: End-to-End Transport Layer Security by Java a pair of small edition/ mobile info device profile (J2ME/MIDP). victimization J2ME/MIDP to mobile communication overcome the safety challenges Janus-faced with WAP technology, however securing the XML messages transferred between the movable and therefore the server would offer high level of integrity for the information itself not for the physical association.

Keywords: *mobile commerce, wireless applications protocol, wireless transport layer security.*

1. INTRODUCTION

Mobile commerce (m-commerce) is providing commercial services that are accessible by using mobile devices, typically a mobile phone. The main advantages of such services are their high availability, independence of physical location and time. Yet the move to create a wireless version of internet means a new set of problems. As with the existing fixed internet, the biggest problem is security. Despite the fact that operators are announcing or rolling out Wireless Applications Protocols (WAP), I-mode and java-based information, the platforms have gaping security holes.

2. This research aims to present some suggestions to improve **m-commerce** security and limit the m-commerce drawbacks. These suggestions related to the following functional: End-to-End Transport Layer Security by Java 2 micro edition/ mobile information device profile (J2ME/MIDP). Using J2ME/MIDP to mobile communication overcome the security challenges faced with WAP technology, but securing the XML messages transferred between the mobile phone and the server would give high level of integrity for the data itself not for the physical connection.

3. There are several different ways of defining mobile commerce. Some consider it to involve monetary value where as the others term it to provide services. The more general definition of m-commerce refers to the access to the Internet via a mobile device, such as a cell phone or a Personal Digital Assistance (PDA). M-commerce is termed as using a mobile device for business transactions on the Internet that involve the transfer of money.

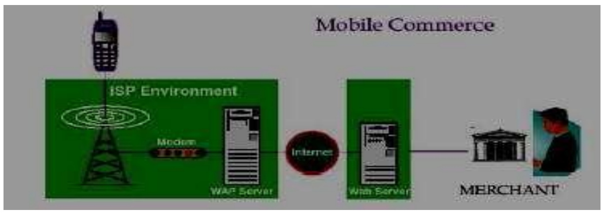


Figure 4.1: M-Commerce transaction

Typical m-commerce transaction would look like the one depicted in figure (1) where a mobile device is equipped with a Wireless Application Protocol (WAP) that connects to the Internet. The figure shows a cell phone that accesses the nearest access point. The data from WAP device is transferred to a WAP server, which then transfers it to the Internet and finally reaches the merchant [1, 2].

1.2 SECURITY FOR MOBILE COMMUNICATIONS

WAP Technology for Secure Mobile Communications [3, 4]:
The Wireless Transport Layer Security (WTLS) implements many features to insure secure data transmissions thereby protecting users, network and service operators as well as the functionality of the upper layers of the WAP stack. The WTLS provides the integrity, confidentially, authentication and detect and reject data that is corrupt or otherwise unfit for admittance into the application system. The WTLS Record Protocol is a layered protocol that accepts raw data from the upper layers to be transmitted and applies the selected compression and encryption algorithms to the data. Moreover, the Record Protocol takes care of the data integrity and authentication. Received data is decrypted, verified and decompressed and then handed to the higher layers. The Record Protocol is divided into four protocol clients. The application protocol is not described here, since it is the interface for the upper layers.
(a) The Change Cipher Spec Protocol - The Change Cipher Spec is sent to peer either by the client or the server. When the Change Cipher Spec message arrives, the sender of the message sets the current write state to the pending state and the receiver also sets the current read state to the pending state. The Change Cipher Spec message is sent during the handshake phase after the security parameters have been agreed on.
(b) The Alert Protocol - The Record Protocol also provides a content type of alert messages. There are three types of alert messages: warning, critical, and fatal. Alert messages are sent using the current secure state, i.e. compressed and encrypted, or under null cipher spec, i.e. without compression or encryption. Both fatal and critical messages result in the termination of the secure connection but the failed connection may be used to establish a new secure connection if the message was critical but not if it was fatal. Error handling in the WTLS is based on the alert messages.

(C) The Handshake Protocol - All the security related parameters are agreed on during the handshake. These parameters include attributes such as used protocol versions, used cryptographic algorithms, information on the use of authentication and public key techniques to generate a shared secret.

Java Technology for Secure Mobile Communications [5, 6]: With Java 2 Platform, Micro Edition (J2ME) There are two relevant JSRs (Java Specification Requests) in relation to cryptography and secure m-commerce according to the Java community process program: Mobile Information Device Profile, MIDP and Security and Trust Services API for J2ME. MIDP is a specification of a security framework for Java applications designed to run within the MIDP Java environment. MIDP uses a Java Virtual Machine of reduced complexity designed specifically for mobile devices. MIDP specifies how a signed Java application can be verified to belong to a domain defined by a root certificate and an associated policy file. The policy file specifies the capabilities of Java applications within that domain.

II THE PROPOSED FRAMEWORK:

J2ME provides several levels of security, such as class loader, byte code verifier, and security manager. These security levels protect client systems from unreliable programs. The security advantages of J2ME over WAP are end-to-end security, less use of network and content-based encryption.

End-to-end security: J2ME supports end-to-end encryption, authentication, and verification. In WAP, a request from a wireless device is encrypted in WTLS and this request needs to be decrypted as Transport Layer Security (TLS) data. While this conversion takes place, the data is unencrypted making it highly vulnerable. J2ME does not need a gateway between the device and the server. This allows J2ME to provide end-to-end security. There is no conversion of data from WTLS to TLS, thereby eliminating the chance of the data being unencrypted at any point of time.

Less use of network: J2ME allows data to be processed locally, unlike WAP that needs to connect to the network for any kind of data processing. This feature in J2ME in turn reduces the possibility of data loss or theft.

Content-based encryption: J2ME applications process data before sending it across a network. A J2ME application can set the security policy based on the content.

The proposed suggestions:

The proposed suggestion aim to secure XML messaging between a J2ME/MIDP wireless front end and a JSP page back end. XML digital signature technology can help to implement lightweight and flexible security solutions for wireless Web services applications. XML is becoming a major data exchange protocol in the world of Web services. XML messages that drive Web services often need to go through multiple intermediaries before they reach destinations. So, it is important that we secure the communication content from end to end. The best way to do it is to ship an XML document and its security information (such as signatures, digests, keys, and so on.) altogether as a single document. Handling the XML digital signature in MIDP applications IBM alpha Works develops a Java package called XML Security Suite, which supports the latest XML digital signature specification. As known, to handle XML digital signatures, the wireless devices being used need to support the following functions:

Read and write data from/to an XML document and Sign the message and verify the signature. These functions require a cryptography API that is not part of the current MIDP specification.

The bouncy castle crypto APIs is an open source, lightweight cryptography package for the Java platform. It supports a large number of cryptography algorithms and provides an implementation for JCE. Because Bouncy Castle is designed to be lightweight, it can run from J2SE to J2ME (including MIDP) platforms. It is the only complete cryptography package that runs on MIDP. Together XML digital signature specification and the usage of several different Bouncy Castle key generators, encoding engines, digital signature singers, and a digest engine. Section below will illustrate the signing documents on the server side, encoding and transporting documents in secure XML format, and verifying documents on the client side.

Example:

One of the most familiar examples is the elliptic curve, an elliptical curve DSA signature example In the ECDSASigUtil class, you first define the elliptical curve model you plan to use, as in code shown below:

private static BigInteger q = new BigInteger("6277101735386680763835789 4232076664 16083908700390324961279"); private static BigInteger a = new BigInteger("fffffffffffffffffffffffffffffffeffffff fffffffffc", 16); private static BigInteger b = new

BigInteger("64210519e59c80e70fa7e9ab72 243049feb8 deecc146b9b1", 16); private static BigInteger n = new BigInteger("6277101735386680763835789 4231760590 137671947731828422840 81"); private static byte [] G = Hex.decode("03188da80eb03090f67cbf20e b43a18800f4 ff0afd82ff1012")

The ECDSASigUtil.generateKeys() method generates random key pairs. This step is normally done offline by a central certificate authority.

// Get a secure random source. SecureRandomsr = new SecureRandom(); ECCurve.Fp curve = new ECCurve.Fp(q, a, b); ECDomainParametersECDomPara = new ECDomainParameters(curve, curve.decodePoint(G), n); ECKeyGenerationParametersECKeyGenPara = newECKeyGenerationParameters(ECDom Para, sr); ECKeyPairGeneratorECKeyPairGen = new ECKeyPairGenerator(); ECKeyPairGen.init(ECKeyGenPara); AsymmetricCipherKeyPairkeyPair = ECKeyPairGen.generateKeyPair(); privKey = (ECPrivateKeyParameters) keyPair.getPrivate(); pubKey = (ECPublicKeyParameters) keyPair.getPublic();

The public key is characterized by a parameter Q, and it is retrieved by the pubKey .getQ() method. To avoid confusion with the model parameter q, you use QQ in the method and XML element names for capital Q. code below show the methods in the ECDSAUtil class. These methods retrieve the model and key parameters, which are necessary to reconstruct the public key object.

// public key specific field

public static String getQQ() throws Exception { return (new String(Base64.encode(pubKey.getQ().getEncoded())));} // Key parameter fields. Could also be retrieved from pubKey.

public static String getQ() throws Exception { return (new String(Base64.encode(q.toByteArray())));}

public static String getA() throws Exception { return (new String(Base64.encode(a.toByteArray())));}

public static String getB() throws Exception { return (new String(Base64.encode(b.toByteArray())));}

public static String getN() throws Exception { return (new String(Base64.encode(n.toByteArray())));}

public static String getG() throws Exception { return (new String(Base64.encode(G)));}

Using the generated private key, the utility class ECDSASigUtil can get a two-part DSA signature, R and S, from a digest:

static public String [] getSignature (String digest) throws Exception { // Sign

ECDSASigner signer = new ECDSASigner(); signer.init(true, privKey);

BigInteger [] sigArray = signer.generateSignature(digest.getBytes()) ;

String [] result = new String [2]; // Signature R

result[0] = new String(Base64.encode(sigArray[0].toByteArray()));

// Signature S

result[1] = new String(Base64.encode(sigArray[1].toByteArray())); return result;

}

The server encodes the digest, signature, and key parameters into ASCII text form and embed the text in XML digital signature format. As in the retrieval method name, the public key parameter Q is noted as QQ to differentiate it from the key parameter q, in the corresponding XML element, as shown below:

```
<SignedMesg>
<mesg>Hello World</mesg><Signature>
<SignedInfo>
<SignatureMethod
Algorithm="http://www.w3.org/2000/09/xm
ldsig#dsa-sha1" />
<DigestValue>Ck1VqNd45QIvq3AZd8XYQ
LvEhtA=</
DigestValue>

</SignedInfo>
<SignatureValue>
<R>NK/EIL2lrbFFCThnEuYlUWzh6IEfM
sts</R>
<S>AMeJDecKWrQO6Eeehl3het+FlDDL4
IedCA==</S>
</SignatureValue>
<KeyInfo>
<KeyValue>
<ECKeyValue>
<QQ>AwCiF5uG+DII/x1XTq84fLm4eGN2
fED1PYc=<
/QQ>
<Q>AP////////////////////7//////////w==</Q>
<A>AP////////////////////7//////////A==</A>
<B>ZCEFGeWcgOcPp+mrciQwSf643uzB
Rrmx</B>
<N>AP///////////////5ne+DYUa8mxtNIoMQ=
=</N>
<G>AxiNqA6wMJD2fL8g60OhiAD0/wr9g
v8QEg==</
G>
</ECKeyValue>
</KeyValue>
</KeyInfo>
</Signature>
</SignedMesg>
```

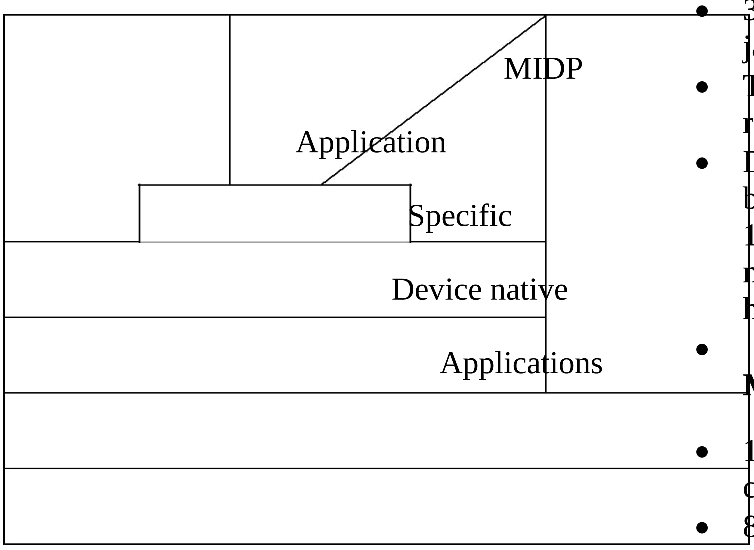
The verification MIDP application parses the digest, key parameters, and signature out of the XML document, reconstructs the public key, and uses the method shown below to validate the signature:

```
static public boolean verify (String digest,
String sig_r, String sig_s,
String key_q, String key_a, String key_b,
String key_n, String key_G, String key_Q )
{
BigInteger q = new BigInteger(
Base64.decode(key_q) );
BigInteger a = new BigInteger(
Base64.decode(key_a) );
BigInteger b = new BigInteger(
Base64.decode(key_b) );
BigInteger n = new BigInteger(
Base64.decode(key_n) );
byte [] G = Base64.decode(key_G); byte [] Q
= Base64.decode(key_Q);
BigInteger r = new BigInteger(
Base64.decode(sig_r) ); BigInteger s = new
BigInteger( Base64.decode(sig_s) );
ECCurve.Fp curve = new ECCurve.Fp(q, a,
b); ECDomainParametersECDomPara =
new ECDomainParameters(
curve, curve.decodePoint(G), n );
ECPublicKeyParameterspubKey = new
ECPublicKeyParameters(

curve.decodePoint(Q), ECDomPara ); //
Verify

ECDSASigner signer = new
ECDSASigner(); signer.init( false, pubKey
);

boolean result = signer.verifySignature(
digest.getBytes(), r, s );

return result;

}
```

III EXPERIMENTAL WOK:

J2ME makes a new generation of wireless application, such as multi-user internet games, mobile commerce, and enterprise client/server applications. Possible on cell phones an two way pagers, MIDP, connected limited device configuration (CLDC) and K virtual machine (KVM) form the foundation for developing wireless java applications.

So, the architecture of j2me for the work presented in this research would be as in figure (5) below



The architecture can be categorized into five layers. From the bottom up, they are as follows:
MID hardware layer: refers to cell phones (for Example, Motorola's iDEN3000) or two-way pagers (for example, RIM blackberry 950).

- Native system software layer: contain the native operating system and libraries provided by the device manufacture.
  KVM layer: provides the runtime environment for java applications.
  CLDC layer: provides core java API for wireless applications.
- MIDP layer: provides the GUI libraries, persistent storage libraries, networking libraries, and timer classes.
- Since wireless devices have meet certain criteria to be able to support J2ME. To run KVM efficiently with CLDC libraries, devices must have at least:160 KB to 512KB of total memory budget available for the java plat form. A 16 bit or 32 bit processor with 25MHz speed.
- Low power consumption, often operating with battery power.
- Connectivity to some kind of network, often with a wireless, intermittent connection and with limited (often 9600bps) bandwidth.
- 128 KB of non-volatile memory available for the java virtual machine and CLDC libraries.
- 32KB of volatile memory available for the java runtime and object memory.
- The MIDP impose the following requirements on hardware:
- Display: screen-size: 96x54.Display depth: 1 bit. Pixel shape (aspect ratio): approximately 1:1 One or more of the following user input mechanisms: one handed keyboard, two handed keyboard, or touch screen.
-

Memory:

- 128 KB of non-volatile memory for MIDP components.
- 8 KB of non-volatile memory for application-created persistent data.

32 KB of volatile memory for java runtime (for example the java heap).

Networking: Two - way, wireless, possibility intermittent, with limited bandwidth.

IV CONCLUSION

Always there is no perfect secure system, especially in m-commerce since the mobile communication system and all its applications still in the childhood level. But also the mobile wireless hackers and crackers still in the same level. Using J2ME/MIDP to mobile Communication overcome the security challenges faced with WAP technology, since there is no decryption and encryption performed in the WAP proxy server. With java technology the data formatted and transmitted with XML language, so the XML messages transferred between the mobile phone and the server over the WAP. By using the XML digital signature with java technology this provide high level of integrity for the data itself not for the physical connection, in other world it will provide end-to-end

ACKNOWLEDGMENT

We would like to thank to all the faculty members of department of computer science and my friends for their good wishes, their helping hand and constructive criticism which led the successful completion of this paper.
We thank all those who directly and indirectly helped us in this regard.

REFERENCES

[1] A. Fouratiet al [2002]: A SET Based Approach to Secure the Payment inMobile Commerce. In Proceedings of 27th Annual IEEE Conference on Local Computer Networks (LCN'02), Tampa, Florida

[2] Anurag Kumar jain et al. [2012]: Addressing Security and Privacy Risks Mobile applications. IEEE Computer society.

[3] ArunKumar Gangula et al. [2013]: Survey on Mobile Computing Security. IEEE Computer Society.

[4] Ashok K Talukder ET AL. [2005]: Mobile Computing. TaTa McGraw Hill Education, January.

[5] B. S. Yee [1994]: Using Secure Coprocessor, PhD thesis, Carnegie Mellon University.

[6] Bernaard menezes [2015]: Network security and cryptography. CENGAGE Learning,second edition.

[7]C. Boyd et al. [2001]: Curve Based Password Authenticated Key Exchange Protocols. LNCS Vol. 2119, pp. 487-501.

[8] C. Boyd et al. [2001]: Elliptic Curve Based Password Authenticated Key Exchange Protocols. LNCS Vol. 2119, pp. 487-501.

[9] CUI Jian-qi et al.[2007]: New secure mobile Electronic commerce solution based on WA. Application Research of Computers Vol.24

[10] Dharma prakash agrawal et al. [2015]: Introduction to Wireless and Mobile Systems. Third Edition, Cengage Learning USA

[11] Feng Tian et al. [2009]: Application and Research of Mobile E-commerce security based on WPKI. IEEE International Conference on Information Assurance and Security.