Building a Universal Secure E-Mail System

Shankara K Narayandas

University of Louisiana at Lafayette
The Center for Advanced Computer Studies; P.O. Box 43922, Lafayette, LA 70504-3922
337-482-1040
shankar@cacs.louisiana.edu

Chung-Huang Yang

National Kaohsiung Normal University
Graduate Institute of Information and Computer Education; 116, Ho-Ping 1st Road, Kaohsiung 802, TAIWAN
886-935-803789
chyang@computer.org

T.R.N. Rao

University of Louisiana at Lafayette
The Center for Advanced Computer Studies; P.O. Box 44330, Lafayette, LA 70504-4330
337-482-6877
trn@cacs.louisiana.edu

ABSTRACT

E-mail security is the subject of utmost importance today. In-spite of having secure e-mail standards like S/MIME, PEM and PGP, the fraction of users adopting these standards is small. The existing standards are not easily adaptable to different platforms. In this paper we propose a Universal Secure E-Mail System (SEMS) built to be compatible with different platforms. In SEMS we bring together the Certification Authority (CA), the Key Distribution Center (KDC) and the smart card technology in attaining data confidentiality, authentication, data integrity and non-repudiation. SEMS is built using Java. The underlying core SEMS is customized to fit to different environments. This paper presents different versions namely, the desktop version, the web-based version and the J2ME based version to run on a PDA or a cell phone.

INTRODUCTION

Electronic mail, since its inception in the late 70's is the most widely used internet based application to date. By the end of 80's, a formal model for receiving and sending e-mail was proposed. The protocol for sending e-mail was called Simple Mail Transfer Protocol (SMTP). The protocol for receiving e-mail was called Post Office Protocol (POP). The objective of SMTP is to transfer mail reliably and efficiently. The intent of the POP is to allow a user's workstation to access e-mail from a mailbox server.

When the SMTP and the POP protocol were first developed, the internet community was so small that the need for a secure infrastructure was not an issue of importance. But since e-mail is today the mostly widely used web-based application, used by almost everyone, the need for security in these models has become an issue of increasing importance.

Secure e-mail standards like S/MIME, PEM and PGP, that were designed to work by having smart software only at the source and destination have been in existence for more than a decade now. But still most of the e-mail today travels unencrypted due to several reasons like complexity of the infrastructure required and lack of adequate user friendliness.

We propose here an infrastructure called Secure E-Mail System (SEMS) that provides a fast, reliable and user-friendly system that enables e-mail users use both the secret-key technology and the public-key infrastructure. The system developed in Java (Sun Microsystems Inc. http://www.java.com/en), provides an encryption and decryption mechanism for sending/receiving e-mail using the secret-key technology with/without requiring user intervention. The public key infrastructure (PKI) (Adams, 1999) is mainly used to enable users to digitally sign/verify their e-mail and to exchange secret-keys. The public key infrastructure is provided by OpenSSL (The OPENSSL Project. http://www.openssl.org/), an open source software.

In this system we use the stream cipher crypto engine described in (Premasathian, 2002) in addition to other algorithms like AES (Daemen and Rijmen, 2002), TDES (NIST, 1999), for encryption and decryption. Stream ciphers are useful in situations where transmission errors are highly probable; stream ciphers are advantageous because they have no error propagation (Menezes, 1997).

The underlying core SEMS is customized to fit to different environments. The desktop version is the name given for application that needs to be installed before it could be executed. This program runs with privileges like any other installed desktop application. The web-based version is for those transient users who need to get access to their e-mail at different locations on different computers. The PDA/Cell phone version runs on J2ME (Java 2 Platform, Micro Edition) enabled cell phones and PDAs.

The paper is organized as follows. Section 2 describes the architectural details of the core SEMS. We describe the different versions available for different platforms and the implementation specifics of those systems in Section 3. Section 4 describes about adapting the underlying core SEMS to different web-based applications followed by conclusion in Section 5.

DEVELOPING A UNIVERSAL SEMS

In SEMS we adopt several components like Certification Authority (CA), the Key Distribution Center (KDC) (Menezes, 1997), stream cipher based crypto-engine, technologies like smart card technology and the Java programming concepts like trusted applets, applet to server communication and J2ME.

The SEMS basic Encryption and Decryption component is shown in Figure 1. The advantage in using Universal SEMS, in addition to satisfying all the security requirements is interoperability. For example, a user will be able to send an encrypted e-mail using a desktop version and will be able to decrypt using the J2ME based PDA version.

CA Server

Public-key infrastructures (PKI) are comprised of supporting services that are needed for using public-key technologies on a large scale. A public-key certification system works by having a certification authority (CA) for the generation and management (application, storage, renewal, revocation, and inquiry) of public-key certificates. Both CA and PKI are crucial parts of many secure network applications, such as, secure e-mail, online banking, and internet stock trading.

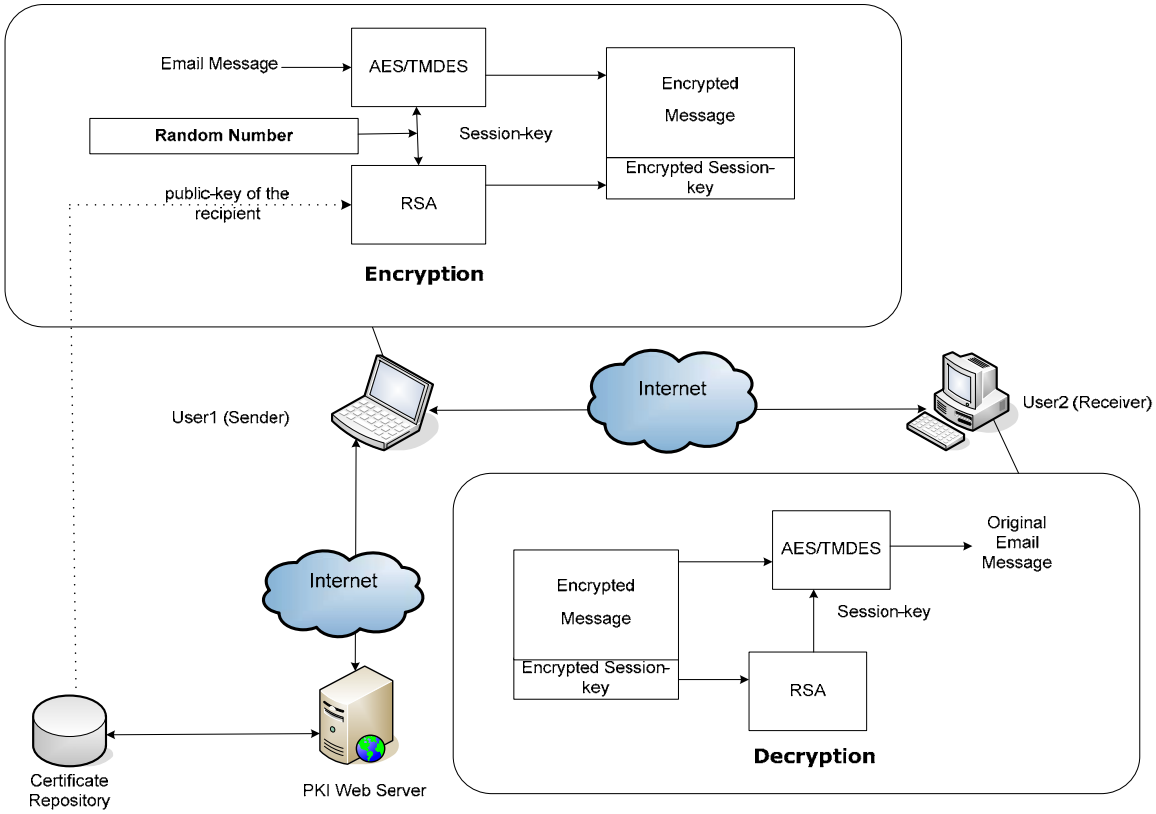

Figure 1. SEMS Cryptographic Operations

The OpenCA PKI Development Project is a collaborative effort to develop a robust, full-featured and open source CA implementing the most used protocols with full-strength cryptography(The OpenCA Group. http://www.openca.org/).

We use OpenCA in order to build the Certification CA Server for SEMS. OpenCA is built over an apache server. Users will be able to request and receive their certificates through internet. OpenCA uses OpenSSL for certification handling. OpenSSL uses X.509 Version 3 certificate format (Chokhani, 1999), which is the current international standard. Certificate contains information related to user's name, public key of the user, validity, issuer details, and so on. The information of the user is digitally signed by the private key of the certifying authority. The algorithms adopted for CA are RSA (RSA Security Inc. www.rsasecurity.com) digital signature algorithm and SHA1 (NIST, 2002) algorithm. SHA1 is used to calculate message digests associated with the signatures. The Certification process is shown in Figure 2.

CA's private key needs to be stronger and is usually of 2048-4096 bits; the validity of the user certificates depends on it. CA node is required to be stand alone with "offline" communication only with the Registration Authority (RA). This is done manually using portable storage devices like floppy disks, USB memories etc. RA is a node between users and CA. Users request for their certificates through internet. The administrator at RA after validating the user information sends it to CA for creating a certificate for the user. Once the certificate has been created, the user is informed about the availability of the requested certificate. The user downloads it through the internet.

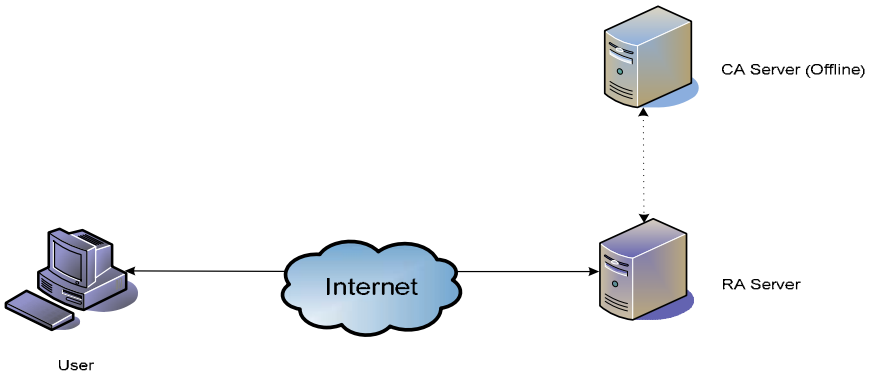

Figure 2. Certification Process

Key Distribution Center (KDC)

Key Distribution Center (KDC) in general is a system that acts as a trusted secret-key repository. It is required when users employ the symmetric-key technology for faster performance. It implements a key distribution protocol to provide keys, usually, session-keys, to two or more entities that wish to communicate securely. A typical scenario where a KDC distributes session-keys to two users, Alice and Bob, is shown in Figure 3.

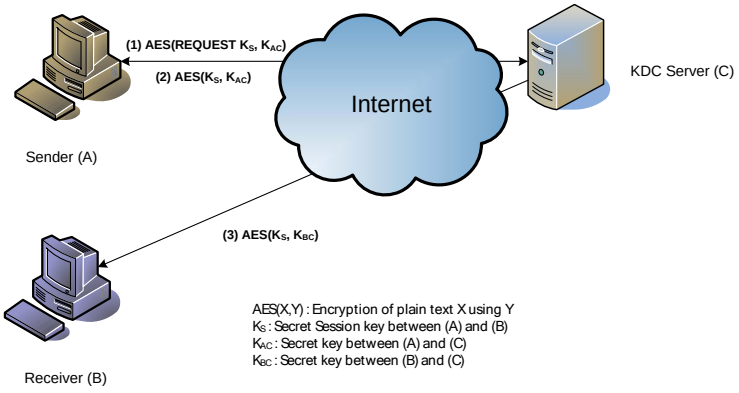

Figure 3. Key Distribution Center

In order for the KDC system to be integrated with SEMS, we set it to be on the same machine as the existing CA server. The existing CA is setup on an Apache server. The *php* (The PHP Group.

http://www.php.net) programming language interpreter is added on top of the existing apache server and all the required KDC programs are written in *php* language. The program on the client side uses the http connection in order to communicate with the KDC server.

Components of SEMS Client

The Secure E-Mail System client is architecturally divided into different components. The functionality of each component is briefly described below. The interaction between the components of the SEMS is shown in Figure 4.

- ***Log-In*** **Component:** The *Log-In* component handles the specifics related to user authentication. Once the *SEMS* application has been launched a *login* screen will appear. The user logs in by typing his/her *Login-Id* and *password*, which are then verified and authenticated by the *SEMS* server. In addition, the user may also be able to use his/her smart card for added protection.
- ***Inbox*** **Component:** The *Inbox* component is for retrieving and displaying the e-mail headers. It retrieves them from the corresponding mail server (POP/IMAP).
- ***Compose/Write*** **Component:** With *Compose/Write* component, we will be able to compose and transmit an e-mail message. It has options for encrypting and *digitally* signing the e-mail message and attachments. It sends e-mail using the SMTP server.
- ***Read*** **Component:** The *Read* component handles the job of retrieving and displaying the e-mail message and the attachments. It has options for decrypting and verifying the *digitally* signed e-mail message and attachments. It retrieves the e-mail message from the corresponding mail server (POP/IMAP).
- ***Secure Address Book (SAB)*** **Component:** The *SAB* component handles the details related to storing/retrieving e-mails, names, the corresponding *secret-keys* and *certificates*. The information stored in SAB is encrypted with the *user-password*.

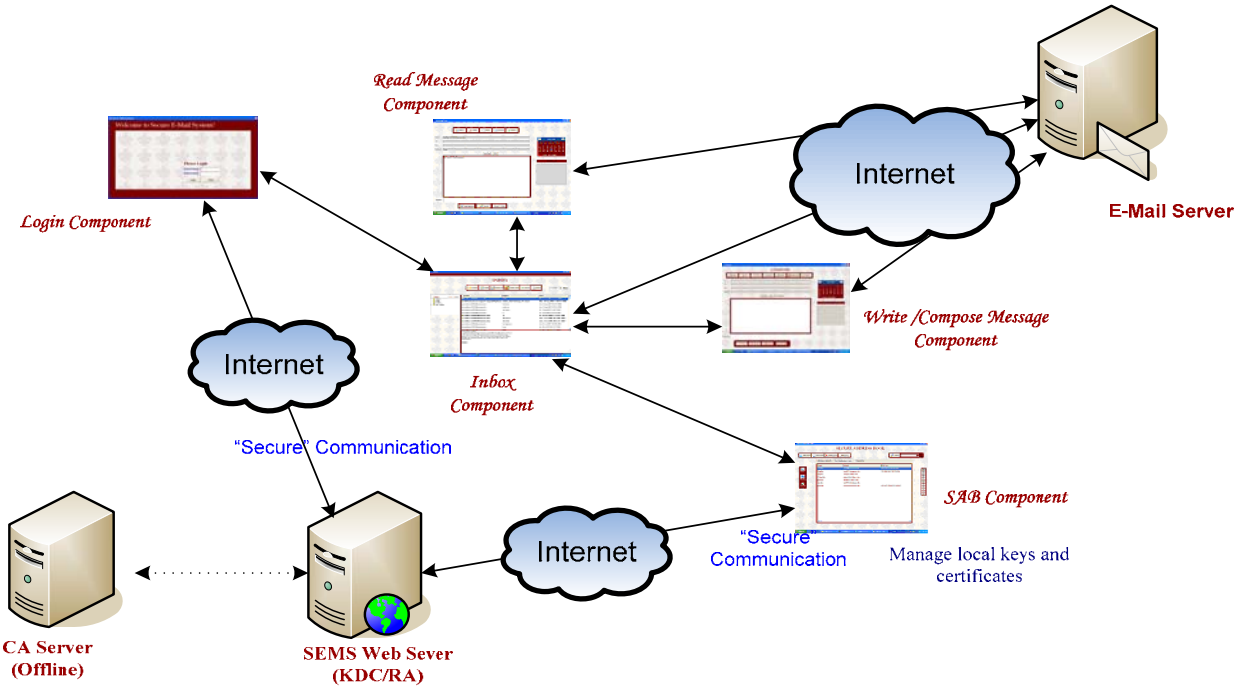

Figure 4. Interaction between different components of the SEMS

Features of SEMS

SEMS features are listed in Table 1.

128-bit Encryption	Supports 128-bit AES, Dynamic key based triple-DES (Encryption that changes the key in every block) and the stream cipher based algorithm
Message Integrity Checks	SEMS provides message integrity check (MIC) with minimal overhead
Time Stamps	Each encrypted message has its own time stamp. The time stamp is used in the encryption and decryption and it will corrupt the decryption if it is modified

Table 1: Features of SEMS

Dynamic-key encryption means using a new key for each succeeding block of encryption. The idea is to have only one *plaintext-ciphertext* pair encrypted with the same key, thus making it secure enough against the known plaintext attack. In Dynamic-Key Encryption technique, the key is generated by the Dynamic Key Generator (DKG) as follows:

$$KD(I) = DKG(X), \text{ and} \quad (1)$$
$$C(I) = E_{KD(I)}\,(M(I)), \quad (2)$$

where KD(I) is the encryption key for the message block *I*, and *X* is the input of the dynamic key generator. C(*I*) is the cipher-text obtained by encrypting message block *I*, M(*I*), using key KD(*I*) (Premasathian, 2002).

SEMS users can choose to encrypt messages using either the block cipher (AES, Triple-Modified-DES) or the stream ciphers algorithms. The Triple-Modified-DES based crypto engine provides dynamic key encryption with additional features, such as key updating, and message integrity check (MIC). Since stream cipher algorithms run faster compared to block cipher algorithm, it can be used to encrypt a large message or a large attachment. Key update is not allowed when a message is encrypted using stream ciphers since K (new) cannot be extracted from KD(*n*-1) (Premasathian, 2002). Although sending a message using stream ciphers does not allow an update key, the user still can choose the key from the SAB to encrypt messages using stream ciphers. Stream cipher algorithm is written as described in (Premasathian, 2002) which was developed by modifying the Zeng-Yang-Rao stream cipher algorithm (Zeng et al., 1990).

SEMS allows users to encrypt a message partially. The user can specify the part to be encrypted by having the strings "BEGIN ENCRYPT" and "END ENCRYPT", all in uppercase, to indicate the starting and ending locations of the message part. There can be more than one part in the message.

SEMS also provides Message Integrity Check (MIC) that requires a minimal overhead. SEMS uses the three characters "EOM" in the last decrypted block to check the integrity of the message. If "EOM" does not appear, a return request is sent.

VERSIONS OF SEMS

The underlying core SEMS is customized to fit to different environments. We have different versions available, as a desktop version, a web-based version and a J2ME based version to run on a PDA or a cell phone.

SEMS as a desktop application

The desktop version is the name given to the application that runs independently on a computer desktop. The application needs to be installed before it could be run. The installation package is created using the *InstallShield* software package (Macrovision Corporation. http://www.installshield.com/). The application is developed in Java and hence is multiplatform compatible. Smart card devices integrate with the desktop version easily and are used for authentication in addition to password based authentication. The desktop version is developed as a complete package. It runs independently with out relying on any e-mail client applications like Microsoft outlook. It includes all the features of the core SEMS and is designed to work in a user-friendly manner.

SEMS as a web based application

The web-based SEMS is the version developed to execute in a browser like Microsoft IE or Mozilla Firefox. The architecture of the web-based SEMS is shown in Figure 5.

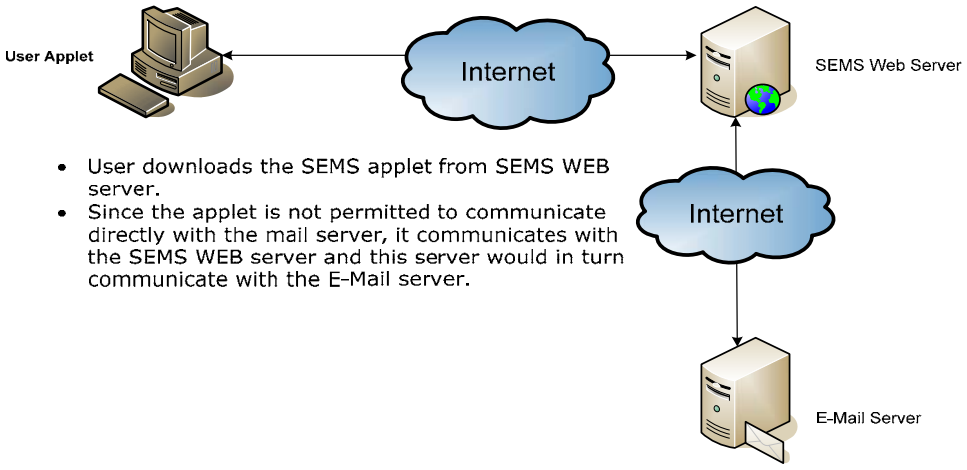

Figure 5. Web-Based SEMS

The architecture is designed taking into consideration, the different constraints being imposed on applets. The basic requirement for our system was to execute all the encryption programs at the client side. In order to achieve this, we need a model where the programs are downloaded onto the browser and execute. Some of the solutions that could be adopted would be either based on Java Applets or Microsoft's Activex components (Microsoft Corporation. http://www.microsoft.com). We adopted the Java applet based architecture for multiplatform compatibility and its interoperability with other versions of SEMS. Applets cannot communicate with any other server except the server it has been downloaded from. So here our applet communicates with the SEMS server which in turn communicates with the mail server. The applet and the SEMS server communicate using the Applet to Servlet communication technology. Using *trusted applets* we will be able to access the local drive of the user. These applets will be signed by a trusted CA. Also these *trusted applets* will be able to access the smart card readers for smart card based authentication.

SEMS for Mobile cellphones/PDAs

MOBILE-SEMS is a J2ME based application that while incorporating the basic features of SEMS, facilitates the user to access his/her e-mail on a J2ME enabled cell-phone or a PDA. It allows users to send encrypted messages and decrypt the received encrypted messages. The current package of MOBILE-SEMS has being tested on PCS Phone Handspring Treo® of SPRINT. The architecture of the MOBILE-SEMS is similar to that of the web-based SEMS. The application running on the cell-phone or a PDA is a J2ME based MIDlet (Mobile Information Device Profile application). This MIDlet communicates with the servlets on the SEMS server. These servlets send only the required information to the MIDlet thereby reducing the network load. For Example, if the user requests for e-mail headers, the corresponding servlet sends only five (or less) e-mail headers separated by some chosen delimiters, instead of sending all the e-mail header information.

The architecture of the MOBILE-SEMS is depicted in Figure 6.

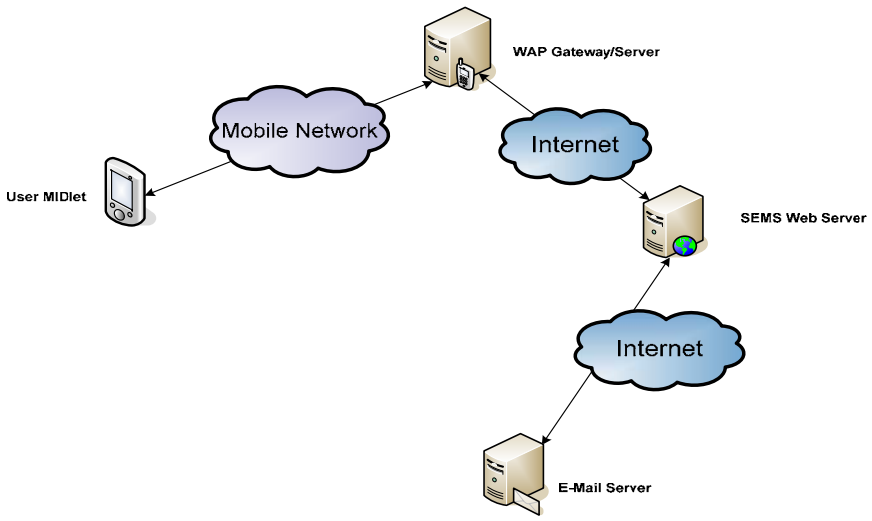

Figure 6. MOBILE SEMS Architecture.

The SEMS MIDlet communicates with the SEMS server and this server would in turn communicate with the mail server.

APPLYING SEMS TECHNOLOGY TO OTHER WEB-BASED APPLICATIONS

The SEMS infrastructure was successfully adopted for ilisting.com, a web based system that provides with a real alternative to the traditional brokerage model for selling and buying a home (Ilisting.com. www.ilisting.com). The website was already fully functional except for the security requirements. We need to develop a module that would fit in the existing application. The module to be developed would then satisfy the security requirements for ilisting.com. The requirements were:

1. All the data that travels between the client machine (browser) and the server must be guaranteed to be secure.
2. The system guarantees that the signature the customer generates is non-forgeable and non-repudiable.

All the encryptions must occur on the client machine before the text gets transmitted over to the server side. In order to achieve this, the following semi-Java based architecture was adopted. The JavaScript function is invoked for the key establishment process (Using Diffie-Hellman Algorithm (Rescorla, 1999)) to initiate. This function internally invokes the embedded Java Applet. This Java Applet communicates with the ASP script on the server machine in establishing the cryptographic secret key. The ASP script calls the Java Application on the server machine for the required cryptographic support. The Java Application is installed as a COM component in order to be accessed from the ASP script. The DSS specified Digital Signature Algorithm (DSA) (NIST, 1999) was used in computing and verifying digital signatures.
SEMS infrastructure could also be adopted for other internet-based applications. Peer-peer applications and instant messaging applications are to mention a few.

A wide variety of applications have the scope of incorporating the MOBILE SEMS based architecture. Applications that are usually controlled by a computer server could be provided with a remote access in a secure way with MOBILE SEMS. Remote access to applications controlling X-10 based (Midkiff, 2002) devices in a secure way is one such example.

CONCLUSION

We have described here the implementation specifics of the secure e-mail system infrastructure built to fit different environments. The Java based SEMS application is developed to run on a variety of operating systems like Windows, Linux, MAC-OS, and Solaris. The Java applet web-based system is designed for users who need access to their e-mail on machines located at different places, i.e. for those who are traveling or who relocate frequently. A J2ME based version described here is designed to run on a PDA or a cell phone. Finally we have presented the flexibility of the SEMS infrastructure by adapting it to other web-based applications.

REFERENCES

Adams, C. and Lloyd, S. (1999) Understanding Public-Key Infrastructure, Macmillan Technical Publishing

Chokhani, S. (1999) Internet X.509 Public Key Infrastructure Certificate Policy and Certification Practices Framework, RFC 2527

Daemen, J. and Rijmen, V. (2002) The Design of Rijndael: AES - The Advanced Encryption Standard, Springer-Verlag

Kaufman, C., Perlman, R. and Speciner, M. (2002) Network Security: Private Communication in a Public World, 2nd edition, New Jersey: Prentice Hall

Menezes, A. J., Oorschot, P. C. V. and Vanstone, S. A. (1997) Handbook of Applied Cryptography, , Boca Raton: CRC Press

Midkiff, S.F. (2002) Mobile Computing “Killer App” Competition, *IEEE Pervasive Computing*, 1(3), 101-104

National Institute of Standards and Technology (NIST) (1999) FIPS PUB 46-3, ‘Data Encryption Standard’, U.S. Department of Commerce, Gaithersburg, MD

National Institute of Standards and Technology (NIST) (2000) FIPS PUB 186-2, ‘Digital Signature Standard’, U.S. Department of Commerce, Gaithersburg, MD

National Institute of Standards and Technology (NIST) (2002) FIPS PUB 180-2, ‘Secure Hash Standard’, U.S. Department of Commerce, Gaithersburg, MD

Premasathian, N. (2002) Design and Analysis of Dynamic Key-driven Crypto Engine, PhD dissertation, University of Louisiana, Lafayette

Rao, T.R.N. and Premasathian, N. (2000) Properties and Parameters of Block Ciphers for E-mail Applications, *IEEE 32nd Southeastern Symposium on System Theory*

Rescorla, E. (1999) Diffie-Hellman Key Agreement Method, RFC 2631

Zeng, K., Yang, C. H. and Rao, T. R. N. (1990) Large primes in stream cipher cryptography, *Advances in Cryptology - AUSCRYPT '90*, Vol. 453 of *LNCS*, Sydney, Australia: Springer-Verlag, 194-205

Zimmermann, P. (1995) The Official PGP User's Guide, MIT Press