IDENTITY - BASED ENCRYPTION



Presented to :

**Mr. Ramesh Babu Battula**  
 (Assistant Professor)

By : -

Himanshu 2012UCP1554

Peeyush Yadav 2012UCP1687

Ankit Bisla 2012UCP1684

Dated : May 10,2015

Abstract

This report describes an implementation of Secure Sockets Language(SSL) protocol for secure communications over the internet.

Current SSL standard is the most generally used cryptographic protocol,but this implementation uses Identity Based Encryption(IBE) which eliminates the requirement of certificates of server side.

This new system, called IBE-SSL, involves the use of a private key generator (PKG) to create a private key for the server. The server can then use its private key to decrypt any messages sent to it by a client using the server’s DNS name as a public key.

The report includes the IBE system and its method of securely generating keys and the encryption and decryption functions. Then an overview of the IBE-SSL system implementation is given.The system includes a sample private key generator, as well as test client and server.

Introduction

Purpose

The purpose of this project is to demonstrate that traditional SSL can be supplanted with identity-based encryption to eliminate the need of site certificates. Code for the proof-of-concept is given in Java.

Background

Throughout history there has been a race in the field of cryptology: cryptographers strive to find new ciphers that are increasingly hard to break, and cryptanalysts work to break the ciphers. In traditional (symmetric) ciphers, there is a single key that allows the sender to encrypt a message, and the recipient must use this same key to decrypt the message. Most of these ciphers were hard to break in their time, but a large problem remained: the single key. If two parties were unsure about the security of their connection, they might think to encrypt their messages using cryptography. However, they first had to meet and exchange a common key in order to decrypt each other’s messages. This was a problem if the pair could not meet in person.

Cryptographers Ron Rivest, Adi Shamir and Leonard Adleman developed a revolutionary new method in 1977. Their method uses two keys: a privatekey known only to the owner and a public key that can be given to anyone. The system is based on a complicated mathematical formula involving large prime numbers and exponentiation. When someone wishes to send a message to a person, he or she must find that person’s public key and encrypt the message.

The recipient then decrypts the message with his or her private key and receives the plaintext. When online retailers appeared on the World Wide Web, experts realized that this new form of cryptography (commonly called public-key cryptography) could allow these businesses to conduct secure transactions. This new algorithm, called Secure Sockets Language (SSL), quickly made its way into most web browsers.

In an SSL system, the server generates its own public and private key pair and then publishes the public key to the Internet. Anyone wanting to transfer sensitive data can then use the server’s public key to encrypt the data. However, if a hacker compromised the server and replaced the server’s public key with the hacker’s public key, then the hacker could intercept incoming ciphertext messages and decrypt them. To avoid this, the creators of SSL introduced certificates. One of the unique properties about the public / private key pair is that a person can encrypt a message with his or her private key, and then anyone can get the person’s public key and decrypt the message.Since the person’s public key decrypts the message, the corresponding private key must have encrypted it, so anyone who decrypts the message knows that the person sent it. This process is known as “signing.”

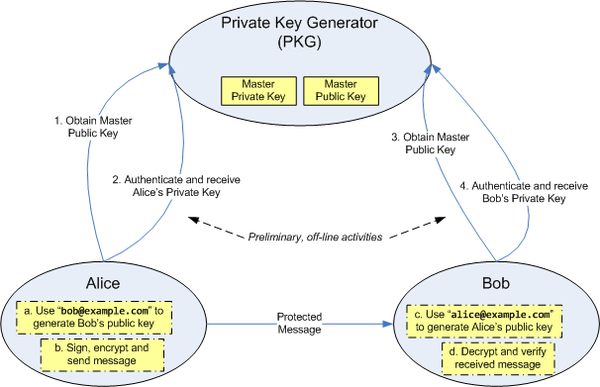
In SSL, a trusted party (called a certificate authority or CA) will issue a certificate for a server. Essentially, the certificate is the public key of the server encrypted with the private key of the signer. Web browsers have the public key of the CA embedded in their code and these keys are implicitly trusted on SSL’s root-level certificate trust model. Since web browsers (and users) trust that the CA has not given away its private key, they can decrypt the server’s public key by decrypting with the CA’s public key with the assurance that a hacker has not compromised the system security.

The Problem

In the most common example, suppose that someone wants to buy a product from an online retailer. With traditional SSL, the web browser connects to the server and downloads the certificate. It checks to see that it has been signed by a trusted party, and extracts the public key of the server. It then encrypts data with the public key and sends it to the server, which decrypts the data and completes the transaction. The main disadvantage to this scheme is that certificates rely on the user to manage them, and most web users are not even aware of SSL technology, much less certificates. Certificates can also be revoked, but most users do not check the status of the certificate before transferring secure data.

Solution

If SSL used the IBE system instead, there would only be the need for one certificate: the master certificate embedded in web browsers for the PKG. The browser would then encrypt data with the URL of the website (e.g. “amazon.com”) and send the encrypted text to the server. The server would get the encrypted message from the client and decrypt it with its private key that it has received from the PKG. There is no need for the server to have its own certificate: it merely needs theprivate key generated from the PKG to decrypt any message sent to it. This process is shown graphically Figure .



ID Based Encryption: Offline and Online Steps

Scope

In this project, we have developed a method of using IBE in SSL to eliminate the need for certificates. We have provided proof-of-concept code to set up the PKG, for the server to connect to the PKG and request its private key, and for the client to encrypt with the server’s public key and connect to the server.

IBE-SSL Implementation

In this section, we will outline the steps taken to show that the IBE system can be applied to a traditional SSL implementation. While SSL is traditionally performed implicitly in modern browsers, we have chosen to write demonstration programs to show a proof of concept. Main functions of the implementation are described below:

Private key Generator Methods

PKG will generate the public key and private key corresponding to a ID which is an “Domain Name” using RSA. How RSA is used to generate public key and private key for a “Domain Name” is shown below.

1. For generating public key corresponding to ID using RSA

**public** BigInteger get\_public\_key(){

phi = p.subtract(BigInteger.***ONE***).multiply(q.subtract(BigInteger.***ONE***));

e = BigInteger.*valueOf*(public\_key\_temp);

**while** (phi.gcd(e).compareTo(BigInteger.*valueOf*(1)) != 0 ) {

e = e.divide(phi.gcd(e));

}

public\_key = e;

get\_private\_key();

**return** public\_key;

}

Where p and q are Big PRIME NUMBERS,

public\_key\_temp = Math.abs(ID.hashCode())

and phi = (p-1)\*(q-1).

In this method we find a number which is relatively prime to phi by continuously diving public\_key\_temp by gcd(public\_key\_temp,phi) ,until gcd(public\_key\_temp,phi) = 1,and select it as an public key.

1. For generating private key corresponding to ID using RSA

**public** BigInteger get\_private\_key(){

d = public\_key.modInverse(phi);

private\_key = extendedEuclid(public\_key,(**this**.p.subtract(BigInteger.***ONE***)).multiply(**this**.q.subtract(BigInteger.***ONE***)));

**return** private\_key;

}

We calculated the modular inverse of public key with respect to phi and select it as an private key.

1. Extended Euclidian Method

**public** BigInteger extendedEuclid(BigInteger a, BigInteger b) {

BigInteger x = BigInteger.*valueOf*(1);

y = BigInteger.*valueOf*(0);

BigInteger xLast = BigInteger.*valueOf*(0);

yLast = BigInteger.*valueOf*(0);

BigInteger q, r, m, n;

**while**(a.compareTo(BigInteger.*valueOf*(0)) != 0) {

q = b.divide(a);

r = b.remainder(a);

m = xLast.subtract(q.multiply(x));

n = yLast.subtract(q.multiply(y));

xLast = x;

yLast = y;

x = m;

y = n;

b = a;

a = r;

}

**if**(xLast.compareTo(BigInteger.*valueOf*(0))<0)

xLast = xLast.add((**this**.p.subtract(BigInteger.***ONE***)).multiply(**this**.q.subtract(BigInteger.***ONE***)));

**return** xLast;

}

This method is used to find the modular inverse of a number with respect to a number which is relative prime to it.

Client Methods

We take ID of user (sender) ,ID of the the receiver ,and MESSAGE which is to be send . Then receive the public key corresponding to the ID of receiver and ‘n’ which is p\*q from PKG. And then encrypt the message using this public key by RSA.Then store the encpyted message along with USER ID in a file named “EncryptedMessage.txt”.

Encrpyt message using C = Me mod n where 0 ≤ M < n.

**private** **byte** [] messageEncrypt(**byte** [] message){

byte [] EncryptMessage = (**new** BigInteger(message)).modPow(Public\_key, n).toByteArray();

**return** EncryptMessage;

}

Server Methods

We take ID of user (Receiver) and private key corresponding to his ID is provided by PKG. Encrypted messages along with sender ID from the file “EncryptedMessage.txt” is taken. After this decrypt the messages using private key by RSA and show it to USER along with sender’s ID.

Decrpyt message using M = Cd mod n.

**public** **byte**[] decryptMessage(**byte**[] encrptedmessage) {

**byte**[] message = (**new** BigInteger(message)).modPow(private\_key, n).toByteArray();

**return** message;

}

This function will return message in byte form ,so we have to convert it into string form.

IBE vs. PKI

* IBE has no Certificates and Certificate management

No certificate server

No certificate lookups for the client

No certificate (or key) revocation, CRLs, OCSP etc.

Instead, IBE uses short-lived keys. PKI can’t do this because this would compound lookup problem

* PKI requires pre-enrollment

In PKI, recipient must generate key pair before sender can encrypt message

IBE is Ad-Hoc capable, a sender can send message at any time

* IBE eliminates encryption key recovery/escrow server

Most PKI applications require access to private keys  
(e.g. Lost keys, Financial Audit, Virus Filtering etc.)

Key server can generate any key on the fly

IBE and PKI – Strengths and Weaknesses

Public Key Infrastructure (PKI)

* Expensive to deploy and run
* Requires pre-enrollment
  + Issuing certificates
* Works well for authentication
* Can be made highly secure through smart cards

Where to use PKI

* Inside the organization
* For maximum security/high cost deployments
* Mainlyauthentication and signing

Identity-Based Encryption

* Ad-hoc capable
  + requires no pre-enrollment
  + software only
* Powerful for encryption
  + no key-lookup
  + revocation is easy

Content scanning easy

Where to use IBE

* Inside or outside the organization
* For any level of security
* Where encryption/ privacy is important