432 Information Security
Homework 3

Ed Felten

October 15, 2008

Contact Bill Zeller with comments or questions at wzeller@princeton.edu

1 Setup

You will be using the same setup that you used for Assignment 1. You should download and use the new version of our code, as some parts of it have changed. You should not be reusing files from Assignment 1. However, you may replace the **MySSHShell.py** file we provide with the file you edited in Assignment 1. The files you edited in Assignment 2 are not compatible with this version due to minor changes we made.

2 Code Structure

- PythonSSH
  - Client (client specific code)
    - ClientCommandTransport.py
    - MyClientCommandTransport.py
    - MyClient.py (fully functional SSH client. You can run this directly)
  - Server (server specific code)
    - ServerCommandTransport.py
    - MyServerCommandTransport.py
    - MyServer.py (fully functional SSH server. You can run this directly)
    - SSHShell.py
    - MySSHShell.py (You can replace this file with your submission from assignment 1)
  - Fun
  - SSHUtil.py
  - PacketUtil.py

- IntUtil.py
- **DiffieHellman.py**
- **Random.py**
- **RSA.py**

(You will be submitting the files in **bold**)

3 The Assignment

The goal of this assignment is to implement the RSA algorithm, the Diffie-Hellman algorithm and a random number generator. These are used in the SSH Transport Layer[1].

SSH uses these in the following ways:

- Diffie-Hellman is used to create a shared secret known only by the client and the server.
- RSA is used to verify the server. The server signs the "exchange hash" and sends this to the client along with the public key. The client checks the "exchange hash" signature which verifies that the owner of the public key signed the message. It is essential for the public key to match the expected public key. This is why you're asked to verify that the fingerprint of the public key matches some known value.
- Random numbers are used to generate Diffie-Hellman secrets, RSA keys and other random values throughout the system.

3.1 What To Do

You will be implementing a variety of functions in a variety of different files. To make this easier, we've created unit tests for each file you'll be editing. For example, when editing `RSA.py` you can run `python RSA.py` directly, which will run the unit tests in the file. If no errors are returned, your code has passed the unit tests. This means you *do not* need to start the server and client each time you run your code. If your code passes all of the unit tests in all of the files, you should be able to connect your client to the server. The assignment is considered complete if you can log into your SSH server and enter commands in the shell.

We will answer questions about the API, but we expect you to implement the algorithms on your own.

- DiffieHellman.py

  Specific instructions can be found in the function definitions

  - DiffieHellmanClient::Initialize(self, randBits)

[1]The SSH Transport Layer is defined in RFC4253

  - DiffieHellmanClient::ComputeSharedSecret(self, f)
  - DiffieHellmanServer::Initialize(self, randBits)
  - DiffieHellmanServer::ComputeSharedSecret(self, e)

- Random

  Specific instructions can be found in the function definitions

  - Random::RandBytes(self, bytes)
  - Random::Rand(self)
  - Random::RandRange(self,a,b)
  - Random::RandDecimal(self)

- RSA

  Specific instructions can be found in the function definitions

  - RSA::GenerateKeys(p, q, bits)
  - RSA::GeneratePandQ(bits)
  - RSA::EncryptInt(message, e, n)
  - RSA::DecryptInt(ciphertext, d, n)
  - RSA::SignInt(message, key, n)
  - RSA::VerifyInt(signed, key, n)
  - RSAUtil::GetPrime(bits)
  - RSAUtil::ExtendedEuclidean(a,b)
  - RSAUtil::gcd(p,q)
  - RSAUtil::MillerRabin(n, k = 50)
  - RSAUtil::MillerRabinWitness(a,n)

A Miller-Rabin

We provide pseudo-code for the Miller-Rabin primality test.

Algorithm 1 MillerRabinWitness(a, n)

let $n-1 = 2^t u$, where $t \geq 1$ and u is odd
$x_0 \leftarrow$ MODULAREXPONENTIATION(a,u,n)
for $i = 1$ to t **do**
    $x_i \leftarrow x_{i-1}^2 \pmod{n}$
    if $x_i = 1$ and $x_{i-1} \neq 1$ and $x_{i-1} \neq n-1$ **then**
        return TRUE
    end if
end for
if $x_i \neq 1$ **then**
    return TRUE
end if
return FALSE

Algorithm 2 MillerRabin(n, k)

for $j = 1$ to k **do**
    $a \leftarrow$ RANDOM(1,$n-1$)
    if MILLERRABINWITNESS(a,n) **then**
        return FALSE
    end if
end for
return TRUE

More details can be found in Introduction to Algorithms (Cormen, Leiserson, Rivest, and Stein) on pg. 890-896 and other sources (Wikipedia, etc).