Project 2: Secure Messaging

Demonstrate by Tuesday, October 12

Worth 100 points

The secure messaging protocol we designed in class used the following sequences for sending and receiving:

Connection Initiator (Sender)	Connection Receiver
Generate Public/Private key pair	
Send public key	Receive public key
	Generate AES key
	Encrypt AES key with public key
Receive encrypted AES key	Send encrypted AES key
Decrypt AES key using private key	
Send messages using AES key as desired	

"Send" here is up to your interpretation. I recommend using TCP sockets, since these allow sending of arbitrary data between two parties. Designation of "server" and "client" is pretty arbitrary for this system, but to use TCP, one side must listen for a connection, and the other must initiate the connection. A brief demo will be given in class.

If you would rather not use TCP, any method of exchanging data between the two parties will work. For example, writing data to a file which will be sent using some other means (Facebook, web server, on a USB drive taped to the bottom of a park bench, etc).

Here are some suggestions regarding libraries for various programming languages.

- Python: pycryptodome supports both AES and RSA. There are other libraries that do one or the other, but this looks easiest. There is also a cryptographic services module, but it does not seem to support AES as of 2015. Note that pycryptodome is mostly compatible with the old pycrypto package, so tutorials will be close.
- C++: crypto++ looks good, but I have no experience with it. Supports AES and RSA (and many others).
- BASH script: You can use BASH if you want. ssh-keygen generates RSA keys, aespipe does aes encryption, and openssl can encrypt using RSA. It is also possible to run unix commands in Python, which is often easier than BASH scripting. nc can be used to send data over the network.
- Any other language: Any reasonably common language will have RSA and AES libraries. Java, C#, Objective C, etc. should be fine, just search for libraries. You're welcome to use Oberon or Squeak or something else obscure, but the easiest solution for obscure languages will probably be running existing AES and RSA programs on the command line using `system` or some equivalent.

Since your program will probably be complicated to run and use, you must demonstrate it to me. Unlike the other projects assigned in CS475, this project may be done collaboratively, provided that each student takes responsibility for an entire end of the communication. This limits the maximum group size to 2. So in this scenario, one student will be the sender, and the other the receiver. You are also welcome to handle both ends of the communication yourself, thus avoiding group work completely. We'll reserve a bit of class time for this on October 12.