Implementation of Cryptographic Algorithms

Assigned: Nov. 7, 2018
Due: Nov. 21, 2018
Version: 1.0.1

In this assignment, you will gain better understanding of cryptography by implementing a simplified version of Diffie-Hellman (DH) public-key encryption and then by using public keys generated by the software, PGP. In the first part of the assignment we will extend the client and server written for Project 1. Now the server process will send its public key to a client process. The client process will then use the public key to send a short session key to the server. The server will decrypt that key, which will eventually be used in a symmetric encryption implementation. The second part of the assignment involves you generating public and private key pairs to send messages to each other with confidentiality.

Part A: Simplified Diffie-Hellman Public Key Encryption

In class we discussed the RSA public-key encryption algorithm. The Diffie-Hellman public-key encryption algorithm is an alternative key exchange algorithm that is used by protocols such as IPSec for communicating parties to agree on a shared key. The DH algorithm makes use of a large prime number *p* and another large number, *g* that is less than *p*. Both *p* and *g* are made public (so that an attacker would know them). In DH, Alice and Bob each independently choose secret keys, S_A and S_B, respectively. Alice then computes her public key, T_A, by raising *g* to S_A and then taking *mod p*. Bob similarly computes his own public key T_B by raising g to S_B and then taking mod *p*. Alice and Bob then exchange their public keys over the Internet. Alice then calculates the shared secret key S by raising T_B to S_A and then taking mod *p*, i.e., ${T_B}^{S_A} \bmod p$. Similarly, Bob calculates the shared secret key S' by raising T_A to S_B and then taking mod p, i.e., ${T_A}^{S_B} \bmod p$.

In this assignment we will simulate the existence of two parties, a client and a server, that use public key encryption to exchange a session key that will be used to encrypt data using the simplified AES algorithm. Your server will create a string containing its name (e.g., "Server of John A. Smith") and then begin accepting connections from clients. Your client should first accept two integers from the keyboard--one a prime number, p, between 257 and 1021, and the other, g, called a generator that is strictly less than the prime. Your program will ensure that these numbers satisfy the relationship that $g^i \bmod p$ can generate all integers between 1 and $(p-1)$ for all i between 1 and $(p-1)$. Some pairs of numbers (g, p) for which this is true include (3, 257), (5, 743), (11, 769), (5, 907), etc. Once you have collected these numbers and they are valid, open a TCP socket to your server and send a message containing the string "110 Hello" and then wait for a server reply. The server should reply to your message with the string "110 Hello". After the server acknowledges the client's "Hello", the client should send a packet with the string "110 Generator: " concatenated with the generator integer, which in turn

is concatenated with the string ", Prime: " followed by the prime. The server will acknowledge this message with the string "111 Generator and Prime Rcvd". The client and server will then separately compute their private and public keys by calling the appropriate methods, which are provided for you. The client should then send a packet with the string "120 PubKey " concatenated with the client's public key. The server will acknowledge this packet with the string "120 PubKey " followed by the server's public key. Next, the client will generate a random integer, a nonce, and encrypt it with the server's public key. The client will send a packet with the string "130 Ciphertext " followed by the encrypted nonce. The server will extract the encrypted nonce, decrypt it, subtract 5 from the number and then encrypt the transformed nonce with the client's public key. The server will send a packet with the string "130 Ciphertext " followed by the transformed and encrypted nonce. Once the client receives this message, it should print out a status message of "150 OK" if the absolute value of the difference between the decrypted nonce and the original nonce is 5. Otherwise you should print "400 OK" to the console. At this point, both the client and the server should print out g, p, their respective public keys, their respective private keys, and the other process's public key. If the status message is "150 OK," the client can then request that the server rolls all five dice with a "200 Roll Dice" message. The program operation can continue as was specified for Project 1.

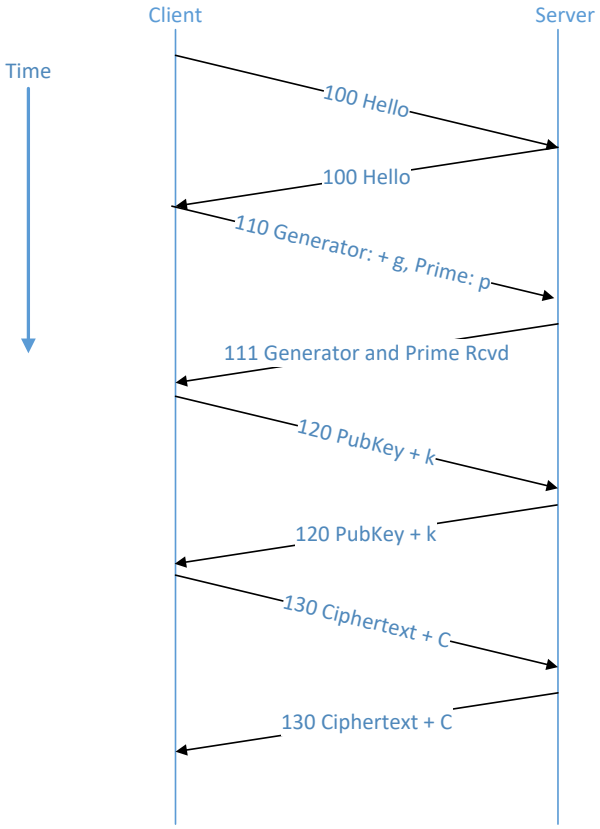

Figure 1: Messages exchanged for cryptography project

Implementation notes

1. Use the modular_exponentiation function, whose pseudocode is specified below to take the modular exponentiation of any integer.

2. For debugging purposes you should print out all the messages received from the server. Make sure to document clearly in your code any assumptions you make about the input and encryption algorithm.

Modular Exponentiation Algorithm

```
modular_Exponentiation(a,b,n)

c ← 0
d ← 1
//let < bk, bk-1,..., b0> be the binary representation of b
for i← k downto 0
   do c ← 2c
      d ← (d*d) mod n
      if bi == 1
         c ← c+1
         d ← (d*a) mod n
return d
```

Part B: Using PGP

The objective of this part is for you to become familiar with PGP and its use for secure communications. You may either use PGP on the lab computers under Linux, or you may download PGP for your personal computer. The information posted at https://help.ubuntu.com/community/GnuPrivacyGuardHowto is a helpful tutorial on using PGP on an Ubuntu machine. Please carry out the following tasks:

1. Generate a pair of keys. Export your public key into a .asc file.
2. Upload your public key to the MIT pgp keyserver, pgp.mit.edu.
3. Get at least two classmates to sign your public key, and sign at least two classmates' public keys. Ensure they upload the signed key to the MIT keyserver. For each key you signed, explain the process you used to obtain, verify, and sign the key.
4. Use PGP to send a secure e-mail message to one of the persons with whom you exchanged public keys.

What to turn in:

- A copy of your public key.
- An explanation of the process used to obtain, verify, and sign the key.
- A screenshot of the encrypted email message including the header.

Part C: Using a one-time pad (Optional: 10 extra-credit points)

Add the message "140 One Time Pad: <encryptedkey>", where the value in <encryptedkey> is going to be a secret key that is encrypted with a one-time pad. The other party will prompt the user for the one-time pad value, and then decrypt the value in <encryptedkey> and print it to the console. In your report

comment on the use of the one-time pad versus the Diffie-Hellman key exchange.
NB: You will not receive the extra-credit points if your solution to part A is non-functional.

Rules

- **The project is designed to be solved independently**.
- **You may not share submitted code with anyone.** You may discuss the assignment requirements or your solutions away from a computer and without sharing code, but you should not discuss the detailed nature of your solution. If you develop any tests for this assignment, you may share your test code with anyone in the class. Please **do not put any code from this project** in a public repository.

What to turn in

1. A single zip|tar archive containing all the source files for your implementation, a PDF document with your tests and program documentation, and a copy of your public key. Your PDF document should also contain the screenshot for Part B. Your archive must unzip to a directory named with your student ID, and all of your files must be in that directory.
2. You will hand in the code for the client and server implementations along with screen shots of a terminal window, verifying that your programs actually carry out the computation that is specified.
3. For Part A, a separate (typed) document of a page or so, describing the overall program design, a verbal description of "how it works," and design tradeoffs considered and made. Also describe possible improvements and extensions to your program (and sketch how they might be made).
4. A separate description of the tests you ran on your programs to convince yourself that they are indeed correct. Also describe any cases for which your programs are known not to work correctly.

Grading

- For Part A:
    - Program listing
        - Works correctly as specified in assignment sheet – 35 points
        - Contains relevant in-line documentation – 5 points
    - Design document
        - Description – 5 points
        - Thoroughness of test cases – 5 points
- For Part B:
    - Key generation - 5 points
    - Screen shots of encrypted email messages – 5 points.