CS 105 – Lab # 11 – Modern Cryptography

Today, we will look at two problems.

- Diffie-Hellman key exchange: how two people can agree on a common key without sending it to each other.
- The RSA cipher system.

Log into the computer as usual. It's a good idea to have two terminal windows open so that you can see them side by side on the screen. Create a new folder called lab11 to hold today's work. Make sure that your current working directory is lab11 before continuing. In other words, when you type the `pwd` command, the system should tell you that you are inside lab11.

Part 1 – Diffie-Hellman key exchange

In class you learned about the Diffie-Hellman-Merkle key exchange protocol. It can be used to let two people come up with a shared secret key without sending the key itself over the network.

You are going to create a Pascal version of the key exchange algorithm. Use nano to create a Pascal program. Here is the code below, with some pieces left out for you to finish. All of the variables that you need have been declared for you.

```
(* dh.pas - Diffie-Hellman key exchange.
 * The formulas use variables a, A, b and B.  But in Pascal,
 * variable names are not case sensitive.  So, these numbers
 * will be called SecretA, BigA, SecretB, BigB.
 *)
program dh;

var
  p, q, SecretA, BigA, SecretB, BigB, AliceKey, BobKey : integer;

(* This function computes a^b mod c. *)
function PowerMod(a, b, c : integer) : integer;
  var i : integer;

  begin
    PowerMod := 1;
    for i := 1 to b do
      PowerMod := (Powermod * a) mod c;
```

```
  end;

begin
  (* 1.  Ask the user to enter p, q and a.
   *     Remind the user that p needs to be prime.
   *)



  (* 2.  Alice computes A = q^a mod p. *)
  BigA := ________________;

  writeln('Alice sends to Bob the number BigA = ', BigA);
  writeln('and p = ', p);
  writeln('and q = ', q);

  (* 3.  Ask user for Bob secret number b. *)



  (* 4.  Bob computes B = q^b mod p and K = A^b mod p. *)
  BigB := ________________________;
  BobKey := ________________________;

  writeln('Bob sends to Alice the number BigB = ', BigB);


  (* 5.  Alice can compute K = B^a mod p. *)
  AliceKey := ______________________________;


  writeln('Let''s compare keys.');
  writeln('Alice''s key = ', AliceKey);
  writeln('Bob''s key = ', BobKey);
  if AliceKey = BobKey then
    writeln('The keys match, as expected!')
  else
    writeln('Something is wrong.  The keys don''t match.');

end.
```

Save, compile, and run your program. Run the program using the example input we tried in class. Run your program a few more times with different input data. In the space below, give an example set of input values that are suitable for creating a secure key:

p = _____, q = _____, a = _____, b = _____, and resulting key = _____

Next, give an example set of input values that obey the instructions of the algorithm but produce a poor result for the key (e.g. easy for an eavesdropper to guess).

p = _____, q = _____, a = _____, b = _____, and resulting key = _____

Part 2 – RSA encryption

Symmetric key cryptography (where everyone has the same key) is fun and easy. However, it can pose some problems. How comfortable are you about sharing a secret key with someone you never met? What if your communication link with the other party is difficult or unreliable? You might not have time to set up a secret key. In the end, you may have to resort to setting up a unique key for each transaction.

Asymmetric cryptography attempts to address these concerns.

Asymmetric cryptography allows two different parties to create private and public keys and transfer information over the Internet securely. Have you ever noticed the padlock symbol on your Web browser when you access certain sites, or how some URLs start with https instead of http? This indicates that your connection is secure and it implements asymmetric cryptography. The most famous asymmetric encryption algorithm in use is RSA, which you implement next.

The RSA algorithm begins with the choice of two prime numbers p and q. RSA is a "one-way" function, one that can only be reverse engineered by someone with knowledge of the numbers p and q. Each function can be personalized by choosing p and q, which multiply to give N. The function allows everybody to encrypt messages to a particular person by using that person's choice of N, but only the intended recipient can decrypt the message because that person is the only one who knows p and q, and hence the only person who knows the decryption key d.

Use nano to create a new Pascal program called rsa.pas. The code is given below, with some pieces that you need to fill in. All the variables that you need have already been declared.

```
(* rsa.pas - Practice the RSA cipher. *)
program rsa;

var
  (* We may want to use int32 if the input values are large. *)
  p, q, M, N, i, x, y, z, e, d : integer;

(* This function computes a^b mod c. *)
function PowerMod(a, b, c : integer) : integer;
  var i : integer;

  begin
    PowerMod := 1;
    for i := 1 to b do
      PowerMod := (Powermod * a) mod c;
  end;

(* A helper function that allows us to determine
 * if two integers are relatively prime.  The only
 * positive integer that divides both a and b is 1.
 *)
function RelativelyPrime(a, b : integer) : boolean;
  var
    i, lesser : integer;
    FoundCommonDivisor : boolean;

  begin
    (* 1.  Determine the lesser of a and b, and put it in
     * the variable lesser.  Use an if-then-else statement.
     *)




    (* 2.  For all i from 2 up to the lesser, see if this number
     * i divides both a and b.  If so, we have found that
     * the numbers are NOT relatively prime.
     * Finish the if-statement below.
     *)
    FoundCommonDivisor := false;
    i := 2;
    while (i <= lesser) and (FoundCommonDivisor = false) do
      begin
        if _______________________________ then
```

```
          FoundCommonDivisor := true;
        i := i + 1;
      end;

    if FoundCommonDivisor = true then
      RelativelyPrime := false
    else
      RelativelyPrime := true;
  end;

(* This is the main program. *)
begin
  (* 3.  Ask the user for two secret prime numbers, p and q. *)




  (* 4.  Compute N = pq and M = (p-1)(q-1). *)




  (* 5.  Choose public encryption key e, which is < M and
   * relatively prime to M.  Let's show the user a list
   * of valid candidates from which to choose.
   * Finish the if-statement with a call to RelativelyPrime.
   *)
  writeln('Here is a list of possible values for e:');
  for i := 1 to M - 1 do
    if _____________________ then
      write(i, ' ');
  writeln();

  write('Which value would you like for public key e?  ');
  readln(e);


  (* 6.  The message is x.  Sender transmits y = x^e mod N.
   * Finish the assignment statement for y.  *)
  write('Enter numerical message x:  ');
  readln(x);
```

```
  y := ____________________;
  writeln('The message you send is y = ', y);


  (* 7.  Choose private decryption d, where ed mod M = 1.
   * Let's show the user a list of possible choices for d.
   * Complete the if-statement condition.
   *)
  writeln('Here is a list of possible values for d:');
  for i := 1 to M - 1 do
    if ________________ then
      write(i, ' ');
  writeln();

  write('Which value would you like for private key d?  ');
  readln(d);

  (* 8.  Recipient computes z = y^d mod N.  z should equal x.
   * Complete the assignment statement for z.
   *)
  z := ____________________;

  writeln('Recipient computes z = ', z);

  if x = z then
    writeln('Good news!  As expected, x and z match.')
  else
    writeln('Uh-oh, something went wrong.');
end.
```

Save, compile, and run your program. Verify that the program runs correctly.

It turns out that there is a limit to how large x can be when we run this program. Can you find an example value of x for which the algorithm fails? Holding all other input constant, what is the largest value of x that seems to work for the RSA algorithm? Give your input data below: