CS 346 – Lab 13 – Cryptography

Today we will work on simple implementations of two classic cryptography problems: the Diffie-Hellman key exchange mechanism, and the RSA encryption algorithm. Create a directory called lab13 and do all your work there.

You will need to create a source file in your favorite programming language. This will contain various functions that will emulate the routine operations we need to perform in Diffie-Hellman and RSA. Your program should have this organization:

- powerMod(int a, int b, int c) – compute the value of a^b mod c.
- testPowerMod() – Interactively obtain values of a, b, c, and then print the result from powerMod().

- diffieHellman(int p, int q, int a, int b) – Solve an instance of the Diffie-Hellman problem. Verify the two keys are equal.
- testDiffieHellman() – Get interactive input for the Diffie-Hellman problem and invoke diffieHellman().

- diffieHellmanEavesdropper(int p, int q, int A, int B) – Try to compute either the value of a or b in order to compute k. This function should determine how difficult the task is, such as counting the number of trial and error cases needed to determine the key.
- testDiffieHellmanEavesdropper() – Interactively obtain the four parameters to run the eavesdropper experiment.

- relativelyPrime(int a, int b) – Return true if a and b are relatively prime. Will be used by RSA().
- RSA(int p, int q, int x) – Given primes p and q and a numerical message x, determine the numerical messages sent by both parties.
- testRSA() – Interactively obtain parameters necessary for RSA.

- Main program to run the four "test" functions. See the example I/O at the end of the handout as a guide.

Please work on this lab in three phases. You may want to review your class notes on how we do calculations for Diffie-Hellman and RSA. Your program should have sufficient and clear diagnostic output to convince the user that the program is working correctly.

1. Implement powerMod() and testPowerMod(). Your powerMod() implementation should be computationally efficient. In particular, its computational complexity should not be linear in the exponent b. In other words, don't write a loop that has b iterations. An elegant solution would be to write this function recursively. Hint: $a^{21} = a^{11} * a^{10}$ is more efficient than $a^{20} * a^1$. Also, you need to take the mod at various points to make sure your calculations do not overflow.

2. Implement the Diffie-Hellman functions. Refer to handout. Compute the value of K (the shared key) two ways: first using the sender's information which makes use of the "secret" variable a, and then using the receiver's information which makes use of the value of b. Verify that these two key values match. Finally, implement an eavesdropper function that uses the values of p, q, A and B to try to guess the value of either a or b. Once it knows the value of a or b, it can compute the shared key using the appropriate formula.

What values of p, q, a, b make the eavesdropper work really hard? ________________

____

3. Finally implement the RSA functions. Think carefully how you want to determine the values of e and d. For example, how can we determine if two numbers are relatively prime? Finally, RSA() should display the messages y and z that get sent. Verify that x = z.

Here is an example run of the program:

```
Testing a**b % c.  Enter value for a:  22

Enter value for b:  29

Enter value for c:  37
a ** b mod c equals  19

Testing Diffie-Hellman.  Enter prime number p:  29
The remaining input parameters should be less than  29

Enter a number q:  15

Enter Alice's secret number a:  21

Enter Bob's secret number b:  6
Diffie Hellman.  Secret parameters a =  21 and b =  6
Computed values A =  12 and B =  5
K computed by Alice =  28
K computed by Bob  =  28
The keys match.

Okay, eavesdropper, what is the prime number p?  29

What is the value of q?  15

What is the value of A?  12

What is the value of B?  5

Eve believes the key is  28
The process took  21  iterations of trial and error.

RSA requires two prime numbers p and q.

Enter p:  31

Enter q:  41

Please enter the plaintext number x:  12
Here is a list of possible values you can choose for
the public encryption key e.
```

```
1 7 11 13 17 19 23 29 31 37 41 43 47 49 53 59 61 67 71 73 77 79 83 89 91 97 101
103 107 109 113 119 121 127 131 133 137 139 143 149 151 157 161 163 167 169 173
179 181 187 191 193 197 199 203 209 211 217 221 223 227 229 233 239 241 247 251
253 257 259 263 269 271 277 281 283 287 289 293 299 301 307 311 313 317 319 323
329 331 337 341 343 347 349 353 359 361 367 371 373 377 379 383 389 391 397 401
403 407 409 413 419 421 427 431 433 437 439 443 449 451 457 461 463 467 469 473
479 481 487 491 493 497 499 503 509 511 517 521 523 527 529 533 539 541 547 551
553 557 559 563 569 571 577 581 583 587 589 593 599 601 607 611 613 617 619 623
629 631 637 641 643 647 649 653 659 661 667 671 673 677 679 683 689 691 697 701
703 707 709 713 719 721 727 731 733 737 739 743 749 751 757 761 763 767 769 773
779 781 787 791 793 797 799 803 809 811 817 821 823 827 829 833 839 841 847 851
853 857 859 863 869 871 877 881 883 887 889 893 899 901 907 911 913 917 919 923
929 931 937 941 943 947 949 953 959 961 967 971 973 977 979 983 989 991 997
1001 1003 1007 1009 1013 1019 1021 1027 1031 1033 1037 1039 1043 1049 1051 1057
1061 1063 1067 1069 1073 1079 1081 1087 1091 1093 1097 1099 1103 1109 1111 1117
1121 1123 1127 1129 1133 1139 1141 1147 1151 1153 1157 1159 1163 1169 1171 1177
1181 1183 1187 1189 1193 1199

Which value would you like for e?  7
The ciphertext y =  1047
Your decryption key is d =  343
The recipient's plaintext z =  12
The plaintext matches the original!
```