For Problem2

fastexp.m:

function output = fastexp(a,n,k)

%input a, n, k, output = a^k mod n

%convert k to base 2 representation

array = [];

while k > 0

bit = mod(k,2);

quotient = floor(k/2);

array = [array, bit];

k = quotient;

end

len = length(array);

array\_mod = [];

ans = a;

%compute a^(2^i) for each i

for i = 1:len

ans = mod(ans, n);

array\_mod = [array\_mod, ans];

ans = ans^2;

end

%output = array\_mod;

%compute the final answer

output = 1;

for j = 1:len

if array(j) == 1

output = output\*array\_mod(j);

output = mod(output, n);

end

end

Problem2.txt:

fastexp(5,17,2631)

ans =

10

fastexp(5,17,1)

ans =

For Problem2

5

fastexp(5,17,2)

ans =

8

fastexp(5,17,4)

ans =

13

fastexp(5,17,8)

ans =

16

fastexp(5,17,7749)

ans =

14

fastexp(12,17,7749)

ans =

3

fastexp(15,17,1000)

ans =

1

fastexp(1,17,1000)

ans =

For Problem2

1

fastexp(1,17,1318)

ans =

1

fastexp(9,17,1318)

ans =

4

diary off

For Problem4

solversa.m:

function m = solversa(n,e,c)

%inputs public RSA key n,e and ciphertext c

%outputs plaintext m

y = sqrt(n);

%first find p,q that factorizes n

p = 1;

q = 1;

%factorizing n into p,q

%we only need to test first sqrt(n) elements to make

%the algorithm faster

for i = 2:y

if mod(n,i) == 0

p = i;

q = n/p;

break;

end

end

%then compute phi(n)

phi = (p-1)\*(q-1);

%find the decryption key

d = inverse(e,phi);

%decrypt c

m = fastexp(c, n, d);

end

Helper methods:

extendedeuclid.m:

function output = extendedeuclid(a,b)

%we assume input a > b, if a < b, we swap them in the beginning

areal=a;

breal=b;

if a < b

temp=a;

areal=b;

breal=temp;

end;

%initializing our matrix

output=[];

A=[];

For Problem4

Q=[];

X=[];

Y=[];

A(1)=areal;

A(2)=breal;

Q(1)=0;

X(1)=1;

X(2)=0;

Y(1)=0;

Y(2)=1;

i=2;

%do euclid algorithm until A(i) is 0

while A(i) > 0

Q(i)=floor(A(i-1)/A(i));

A(i+1)=A(i-1)-Q(i)\*A(i);

X(i+1)=X(i-1)+Q(i)\*X(i);

Y(i+1)=Y(i-1)+Q(i)\*Y(i);

i=i+1;

end

%since in the end, i is 1 greater than the last i recorded in the matrix,

%and in matlab index 1 is actually index 0, these effects cancal out when

%we are deciding the signs of X and Y in the end

%the first column of output is gcd(a,b), second column is x, and third is y

output=[output;A(i-1)];

output=[output;(-1)^(i)\*X(i-1)];

output=[output;(-1)^(i+1)\*Y(i-1)];

end

inverse.m:

function output = inverse(a,n)

%We assume the input a and n are relatively prime

%Since xa = qn + 1, we find xa - qn = 1

%we use extended euclid algorithm as subroutine

%we don't care what q is since we only need x

%if a<n, extendedeuclid will give us temp(3) as x

temp = extendedeuclid(a,n);

output=mod(temp(2),n);

if a<n

output=mod(temp(3),n);

end

if temp(1) == a || temp(1) == n

output=0;

For Problem4

end

if a == 0 || n == 0

output=0;

end

end

Problem4.txt:  
solversa(8439833,5711029,62472)

ans =

2345678

fastexp(2345678, 8439833, 5711029)

ans =

62472

diary off

For Problem5

squareroots.m:

function output = squareroots(c,p,q)

%input two distinct primes p and q, which are both 3 mod 4

%input c

%output four squareroots m of pq such that m^2 = c mod p\*q

output = [];

%n vector is always [p,q]

posp = fastexp(c,p,(p+1)/4);

negp = p-posp;

posq = fastexp(c,q,(q+1)/4);

negq = q-posq;

%get the vectors

n = [p,q];

b1 = [posp,posq];

b2 = [negp,posq];

b3 = [posp,negq];

b4 = [negp,negq];

%use chinese remainder theorem to compute square roots

m1 = crt(n,b1);

output = [output, m1];

m2 = crt(n,b2);

output = [output, m2];

m3 = crt(n,b3);

output = [output, m3];

m4 = crt(n,b4);

output = [output, m4];

end

Helper method:

crt.m:

function output = crt(n,b)

%takes input vectors n and b, returns x

%which is congruent to each respective entry of b modulo

%the respective entry of n

%x is nonnegative and less than the product of entries

%of n

ln = length(n);

lb = length(b);

%check if two vectors are of different length

if ln ~= lb

output='two vectors do not have equal length!';

return

end

For Problem5

%check if input vectors have length less than 5

%if ln < 5

% output='vector length is shorter than 5!';

% return

%end

smalln = 1;

%check if entries in n are nonzero and pairwise relatively prime

%in the same time compute their product

for i=1:ln

if n(i) == 0

output='entry in n cannot be 0!';

return

end

for j=1:i-1

temp = extendedeuclid(n(j),n(i));

if temp(1) ~= 1

output='entries in n are not pairwise relatively prime!';

return

end

end

smalln = smalln \* n(i);

end

x = 0;

%computes x

for k=1:ln

bign = smalln/n(k);

bigninv = inverse(bign,n(k));

x = x + b(k) \* bign \* bigninv;

end

output = mod(x,smalln);

For Problem5

Problem5.txt:

squareroots(17,19,59)

ans =

500 146 975 621

n = factor(9353881)

n =

2999 3119

squareroots(6245706,n(1),n(2))

ans =

1443540 1234567 8119314 7910341

diary off