Algorithm of Password Authenticated Key Exchange by Juggling:-

***Code Snippets for basic functions and structures***

**User Defined Power Function:**

//Power function

//result=x^y

long int powdef(long int x,long int y)

{

long int i,res=1;

for(i=1;i<=y;i++)

{

res=res\*x;

}

return res;

}

**Hash Array:**

//Hash Array to store the hashed key

int hash[10];

**Jpake Parameters Structures:**

//JPake Parameters to be used

struct JPakeParameters

{

long int p;

long int g;

};

**Jpake User Structures:**

//JPake User structure variables

struct JPakeUser

{

//Must be unique

const char \*name;

//3 for Alice 7 for Bob

long int base;

//The shared secret

long int secret;

//The calculated(shared) Secret

long int key;

//Alice's x1 or Bob's x3

long int xa;

//Alice's x2 or Bob's x4

long int xb;

//Alice's g^x3 or Bob's g^x1

long int stp1c;

//Alice's g^x4 or Bob's g^x2

long int stp1d;

//Alice's A or Bob's B

long int stp2;

//Variables for passing the Knowledge Proofs

long int r1,c1,s1;

long int r2,c2,s2;

long int r3,c3,s3;

//For the session key after hashing

long int sessionk;

};

**Initializing Parameters:**

//Initializing the JPake Parameters

void JPakeParameters\_Init(struct JPakeParameters \*param)

{

int i;

for(i=0;i<10;i++)

hash[i]=-999;

//prime number(p)

printf("\nEnter a prime number :- ");

scanf("%d",&param->p);

//generator(g)

printf("\nEnter a generator :- ");

scanf("%d",&param->g);

}

**Changing the number in decimal to the given base:**

//Changing the base for the JPake User Parameters for proper displaying

void changebase(long int no,long int base)

{

long int i,b,x=0;

char stringhold[20];

//Converting from decimal to the given base

if(no==0)

{

stringhold[x]=(char)(no+48);

x++;

}

else

{

while(no!=0)

{

b=no%base;

no=no/base;

if(b>=0 && b<=9)

{

stringhold[x]=(char)(b+48);

x++;

}

if(b>9 && b<16)

{

stringhold[x]=(char)(b+55);

x++;

}

}

}

stringhold[x]='\0';

for(i=x-1;i>=0;i--)

printf("%c",stringhold[i]);

}

**Displaying the user structure:**

//Displaying the JPake User variables

void show(struct JPakeUser \*user)

{

/\*printf("\nName\t:\t%s",user->name);

printf("\nBase\t:\t%d",user->base);

printf("\nSecret\t:\t");

changebase(user->secret,user->base);

printf("\nKey\t:\t");

changebase(user->key,user->base);

printf("\nxa\t:\t");

changebase(user->xa,user->base);

printf("\nxb\t:\t");

changebase(user->xb,user->base);

printf("\nstp1c\t:\t");

changebase(user->stp1c,user->base);

printf("\nstp1d\t:\t");

changebase(user->stp1d,user->base);

printf("\nstp2\t:\t");

changebase(user->stp2,user->base);

printf("\n");\*/

printf("\nName\t:\t%s\nBase\t:\t%ld\nSecret\t:\t%ld\nKey\t:\t%ld\nxa\t:\t%ld\nxb\t:\t%ld\nstp1c\t:\t%ld\nstp1d\t:\t%ld\nstp2\t:\t%ld\nsession key\t:\t%ld\nhash value\t:\t%ld\n",

user->name,user->base,user->secret,user->key,user->xa,user->xb,user->stp1c,user->stp1d,user->stp2,user->sessionk,hash[user->sessionk]);

}

**Generating the random number:**

//To calculate random numbers starting from the given number

long int genrand(long int n)

{

long int x;

//Defining the starting point

srand(n\*rand());

//Generating the random number

x=rand();

return x;

}

Step 1 :- Instantiating JPakeParameters and JPakeUsers and Initializing their variables.

**Code Snippet:**

//Declaring the structure variables for the parameters and the uers

struct JPakeParameters param;

struct JPakeUser alice,bob;

//Initializing the parameters

JPakeParameters\_Init(&param);

//Initializing the variables for user Alice

alice.name = "Alice";

alice.base=5;

alice.key=0;

//Initializing the variables for user Bob

bob.name = "Bob";

bob.base=7;

bob.key=0;

Step 2 :- Generating the secret key(≠ 0) randomly using the genrand(n)function for both the users and initialising them.

**Code Snippet:**

//The secret key for the users are created and shared by both the users

alice.secret=genrand(1)%10;

bob.secret=alice.secret;

Step 3 :- Selecting the secret values of x1 and x2 for Alice and the secret values of x3 and x4 for Bob by the genknp(&user) function, which generates the values at random, where x1,x3 ЄR Zq and x2,x4 ЄR Z\*q and x2,x4 ≠ 0.

**Code Snippet:**

**From Main:**

//Generate secret values for Alice and Bob

//Alice's x1 and x2

genknp(&alice);

//Bob's x3 and x4

genknp(&bob);

**Function Declaration:**

//To select the secret values for the users

//Alice selects x1 and x2

//Bob selects x3 and x4

void genknp(struct JPakeUser \*user)

{

//If user is Alice the x1 and x2 are selected

if(strcmp(user->name,"Alice"))

{

do

{

user->xa=genrand(3)%10;

user->xb=genrand(5)%10;

}while(user->xa!=0 && user->xb!=0);

}

//Else If user is Bob the x3 and x4 are selected

else if(strcmp(user->name,"Bob"))

{

do

{

user->xa=genrand(13)%10;

user->xb=genrand(17)%10;

}while(user->xa!=0 && user->xb!=0);

}

}

Step 4 :- Sending Step 1 for both Alice and Bob. Alice sends gx1, gx2 and knowledge proofs for x1 and x2. Bob sends gx3, gx4 and knowledge proofs for x3 and x4.

1. Function sendstep1 (&from,&to,&param) sends gx1, gx2 for Alice and sends gx3, gx4 for Bob.
2. Function zkp(from,to,param,step number) sends knowledge proofs for x1 and x2 of Alice and knowledge proofs for x3 and x4 of Bob.
3. Function prover(from,to,param,step number,variable order) calculates the knowledge proofs for x1 and x2 of Alice or knowledge proofs for x3 and x4 of Bob by generating the commitment(r) and receiving the challenge(c) from verifier and then sends s=r+c\*x, where x Є { x1, x2, x3, x4}.
4. Function verifier(u,step number,variable order,0,0,0,0,0) calculates the challenge for the given x, where x Є { x1, x2, x3, x4}.

**Code Snippet:**

**From Main:**

//Step 1 send

sendstep1(&alice,&bob,&param);

sendstep1(&bob,&alice,&param);

**Function sendstep1 declaration:**

//Step 1 Send

//Alice sends g^x1 and g^x2 and zero knowlwdge proofs of x1 and x2 to Bob

//Bob sends g^x3 and g^x4 and zero knowlwdge proofs of x3 and x4 to Alice

void sendstep1(struct JPakeUser \*from,struct JPakeUser \*to,struct JPakeParameters \*param)

{

long int f=1;

//Alice sends g^x1 to Bob and Bob sends g^x3 to Alice

to->stp1c=powdef(param->g,from->xa);

//Alice sends g^x2 to Bob and Bob sends g^x4 to Alice

to->stp1d=powdef(param->g,from->xb);

//Alice sends knowledge proofs of x1 and x2 to Bob and Bob sends knowledge proofs of x3 and x4 to Alice

f=zkp(from,to,param,1);

}

**Function zkp declaration:**

//zkp(Alice or Bob,parameter,send step)

long int zkp(struct JPakeUser \*from,struct JPakeUser \*to,struct JPakeParameters \*param,long int ss)

{

long int f1=1,f2=1,u;

//It is step 1 send then calculating the knowlwdge proofs of x1 and x2 for Alice or the knowlwdge proofs of x3 and x4 for Bob

if(ss==1)

{

//Knowledge proof for x1 of Alice or knowledge proof for x3 of Bob

prover(from,to,param,1,1);

//Knowledge proof for x1 of Bob or knowledge proof for x3 of Alice

prover(from,to,param,1,2);

return 0;

}

}

**Function prover declaration:**

//prover(Alice or Bob,parameters,send step,substep)

void prover(struct JPakeUser \*from,struct JPakeUser \*to,struct JPakeParameters \*param,long int ss,long int sss)

{

long int u,t,r,c,s,y,f;

if(strcmp(from->name,"Alice"))

u=1;

else if(strcmp(from->name,"Bob"))

u=2;

//Calculating Knowledge Proofs for step 1 send

if(ss==1)

{

//Knowledge proof of x1 for Alice or Knowledge proof of x3 for Bob

if(sss==1)

{

if(u==1)

r=genrand(7);

else if(u==2)

r=genrand(11);

//Random number for Commitment by prover

to->r1=r%10;

//Random number which is the challange by the verifier

to->c1=verifier(u,1,1,0,0,0,0,0);

//The last message by the prover s=r+c\*x

to->s1=to->r1+to->c1\*from->xa;

}

}

**Function verifier declaration:**

//verifier(Alice or Bob,send step,verification step,generator,r,c,s=r+c^x,y=g^x)

long int verifier(long int user,long int ss,long int sss,long int g,long int r,long int c,long int s,long int y)

{

long int t1,t2,t;

//For send step 1

if(ss==1)

{

//Challenge of x1 for Alice or Challenge of x3 for Bob

if(sss==1)

{

if(user==1)

c=genrand(29);

else if(user==2)

c=genrand(31);

}

//Challenge of x2 for Alice or Challenge of x4 for Bob

else if(sss==2)

{

if(user==1)

c=genrand(37);

else if(user==2)

c=genrand(41);

}

c=c%10;

return c;

}

}

Step 5 :- Verifying Step 1 for both Alice and Bob. Alice and Bob verify the received knowledge proofs, and also checks gx2 , gx4 ≠ 1.

1. Function verifysendstep1(&from,&to,&param) verifies if the sending step 1 is successful.
2. Function zkp(from,to,param,step number) verifies knowledge proofs for x1 and x2 of Alice and knowledge proofs for x3 and x4 of Bob.
3. Function verifier(u,step number,variable order,generator,commitment,challenge,s,x) verifies the knowledge proof for the given x, where x Є { x1, x2, x3, x4}.

**Code Snippet:**

**From Main:**

//Verify step 1 send

verifysendstep1(&alice,&bob,&param);

verifysendstep1(&bob,&alice,&param);

**Function verifysendstep1 declaration:**

//Verify step 1 send

//Verifies the step 1 send to check if the transmission was successful

void verifysendstep1(struct JPakeUser \*from,struct JPakeUser \*to,struct JPakeParameters \*param)

{

long int f=1;

//Verifying the sending of knowledge proofs of x1 and x2 by Alice to Bob or knowledge proofs of x3 and x4 by Bob to Alice

f=zkp(from,to,param,2);

if(f2==1)

printf("\nStep 1 successful\n");

else

{

printf("\nStep 1 Fail\nExitting\n");

exit(0);

}

}

**Function zkp declaration:**

//zkp(Alice or Bob,parameter,send step)

long int zkp(struct JPakeUser \*from,struct JPakeUser \*to,struct JPakeParameters \*param,long int ss)

{

long int f1=1,f2=1,u;

//It is the verification of step 1 send then calling verifier to verify if g^s=t\*y^c for Alice or Bob

else if(ss==2)

{

if(strcmp(from->name,"Alice"))

u=1;

else if(strcmp(from->name,"Bob"))

u=2;

//Verifying the knowledge proof of x1 for Alice or x3 for Bob

f1=verifier(u,2,0,param->g,to->r1,to->c1,to->s1,to->stp1c);

//Verifying the knowledge proof of x4 for Alice or x4 for Bob

f2=verifier(u,2,0,param->g,to->r2,to->c2,to->s2,to->stp1d);

if(f1==1 && f2==1)

return 1;

}

}

**Function verifier declaration:**

//verifier(Alice or Bob,send step,verification step,generator,r,c,s=r+c^x,y=g^x)

long int verifier(long int user,long int ss,long int sss,long int g,long int r,long int c,long int s,long int y)

{

long int t1,t2,t;

//For step 1 send verification

else if(ss==2)

{

//Checking if g^=t\*y^c of x1 and x2 for Alice and Checking if g^=t\*y^c of x3 and x4 for Bob

t1=powdef(g,s);

t=powdef(g,r);

t2=powdef(y,c);

t2=t2\*t;

if(t1==t2)

return 1;

}

}

Step 6 :- Sending Step 2 for both Alice and Bob. Alice sends A=g(x1+x3+x4)\*x2\*secret and knowledge proof x2\*secret. Bob sends B=g(x1+x2+x3)\*x4\*secret and knowledge proof for x4\*secret.

1. Function sendstep2 (&from,&to,&param) sends A=g(x1+x3+x4)\*x2\*secret for Alice and sends B=g(x1+x2+x3)\*x4\*secret for Bob.
2. Function zkp(from,to,param,step number) sends knowledge proof for x2\*secret of Alice and knowledge proof for x4\*secret of Bob.
3. Function prover(from,to,param,step number,variable order) calculates the knowledge proof for x2\*secret of Alice or knowledge proof for x4\*secret of Bob by generating the commitment(r) and receiving the challenge(c) from verifier and then sends s=r+c\*x, where x Є {x2\*secret, x4\*secret}.
4. Function verifier(u,step number,variable order, 0,0,0,0,0) calculates the challenge for the given x, where x Є {x2\*secret, x4\*secret}.

**Code Snippet:**

**From Main:**

//Step 2 send

sendstep2(&alice,&bob,&param);

sendstep2(&bob,&alice,&param);

**Function sendstep2 declaration:**

//Step 2 Send

//Alice sends A=g^{(x1+x3+x4)\*x2\*secret} and zero knowlwdge proof of x2\*secret to Bob

//Bob sends B=g^{(x1+x2+x3)\*x4\*secret} and zero knowlwdge proof of x4\*secret to Alice

void sendstep2(struct JPakeUser \*from,struct JPakeUser \*to,struct JPakeParameters \*param)

{

long int f=1;

//Alice sends A=g^{(x1+x3+x4)\*x2\*secret} to Bob and Bob sends B=g^{(x1+x2+x3)\*x4\*secret} to Alice

to->stp2=(powdef(param->g,from->xa)\*from->stp1c\*from->stp1d);

to->stp2=powdef(to->stp2,from->xb\*from->secret)%10;

//Alice sends knowledge proof of x2\*secret to Bob and Bob sends knowledge proof of x4\*secret to Alice

f=zkp(from,to,param,3);

}

**Function zkp declaration:**

//zkp(Alice or Bob,parameter,send step)

long int zkp(struct JPakeUser \*from,struct JPakeUser \*to,struct JPakeParameters \*param,long int ss)

{

long int f1=1,f2=1,u;

//It is step 2 send then calculating the knowlwdge proof of x2\*secret for Alice or the knowlwdge proof of x4\*secret for Bob

else if(ss==3)

{

//Knowlwdge proof of x2\*secret for Alice or knowlwdge proof of x4\*secret for Bob

prover(from,to,param,3,0);

return 0;

}

}

**Function prover declaration:**

//prover(Alice or Bob,parameters,send step,substep)

void prover(struct JPakeUser \*from,struct JPakeUser \*to,struct JPakeParameters \*param,long int ss,long int sss)

{

long int u,t,r,c,s,y,f;

if(strcmp(from->name,"Alice"))

u=1;

else if(strcmp(from->name,"Bob"))

u=2;

//Calculating Knowledge Proofs for step 2 send

//Knowledge proof of x2\*secret for Alice or Knowledge proof of x4\*secret for Bob

else if(ss=3)

{

if(u==1)

r=genrand(19);

else if(u==2)

r=genrand(23);

//Random number for Commitment by prover

to->r3=r%10;

//Random number which is the challange by the verifier

to->c3=verifier(u,3,0,0,0,0,0,0);

//The last message by the prover s=r+c\*x

to->s3=to->r3+to->c3\*from->xb\*from->secret;

}

}

**Function verifier declaration:**

//verifier(Alice or Bob,send step,verification step,generator,r,c,s=r+c^x,y=g^x)

long int verifier(long int user,long int ss,long int sss,long int g,long int r,long int c,long int s,long int y)

{

long int t1,t2,t;

//For send step 2

//Challenge of x2\*secret for Alice or Challenge of x4\*secret for Bob

else if(ss==3)

{

if(user==1)

c=genrand(51);

else if(user==2)

c=genrand(53);

c=c%10;

return c;

}

}

Step 7 :- Verifying Step 2 for both Alice and Bob. Alice and Bob verify the received knowledge proof.

1. Function verifysendstep2(&from,&to,&param) verifies if the sending step 2 is successful.
2. Function zkp(from,to,param,step number) verifies knowledge proof for x2\*secret of Alice and knowledge proofs for x4\*secret of Bob.
3. Function verifier(u,step number,variable order,generator,commitment,challenge,s,x) verifies the knowledge proof for the given x, where x Є {x2\*secret, x4\*secret}.

**Code Snippet:**

**From Main:**

//Verify step 2 send

verifysendstep2(&alice,&bob,&param);

verifysendstep2(&bob,&alice,&param);

**Function verifysendstep2 declaration:**

//Verify step 2 send

//Verifies the step 2 send to check if the transmission was successful

void verifysendstep2(struct JPakeUser \*from,struct JPakeUser \*to,struct JPakeParameters \*param)

{

long int f=1;

//Verifying the sending of knowledge proof of x2\*secret by Alice to Bob or knowledge proof of x4\*secret by Bob to Alice

f=zkp(from,to,param,4);

if(f==1)

printf("\nStep 2 successful\n");

else

{

printf("\nStep 2 Fail\nExitting\n");

//exit(0);

}

}

**Function zkp declaration:**

//zkp(Alice or Bob,parameter,send step)

long int zkp(struct JPakeUser \*from,struct JPakeUser \*to,struct JPakeParameters \*param,long int ss)

{

long int f1=1,f2=1,u;

//It is the verification of step 2 send then calling verifier to verify if g^s=t\*y^c for Alice or Bob

else if(ss==4)

{

if(strcmp(from->name,"Alice"))

u=1;

else if(strcmp(from->name,"Bob"))

u=2;

//Verifying the knowledge proof of x2\*secret for Alice or x4\*secret for Bob

f1=verifier(u,4,0,param->g,to->r3,to->c3,to->s3,(from->xb\*from->secret));

if(f1==1)

return 1;

}

}

**Function verifier declaration:**

//verifier(Alice or Bob,send step,verification step,generator,r,c,s=r+c^x,y=g^x)

long int verifier(long int user,long int ss,long int sss,long int g,long int r,long int c,long int s,long int y)

{

long int t1,t2,t;

//For step 2 send verification

else if(ss==4)

{

//Checking if g^=t\*y^c of x2\*secret for Alice and Checking if g^=t\*y^c of x4\*secret for Bob

t1=powdef(g,s);

t=powdef(g,r);

t2=powdef(y,c);

t2=t2\*t;

if(t1==t2)

return 1;

}

}

Step 8 :- If the sending in two steps (sendstep1 and send step2) are successful then the keys are computed. compute\_key(&user,&param) computes K=(B/gx2\*x4\*secret)x2= g(x1+x3)\*x2\*x4·secret for Alice and K=(A/gx2\*x4\*secret)x4= g(x1+x3)\*x2\*x4·secret for Bob.

**Code Snippet:**

**From Main:**

//Compute keys

compute\_key(&alice,&param);

compute\_key(&bob,&param);

**Function compute\_key declaration:**

//Computeing the keys for Alice and Bob after step 1 send and step 2 send are successful

void compute\_key(struct JPakeUser \*user,struct JPakeParameters \*param)

{

float temp;

//For Alice K=(B/g^{x2\*x4\*secret})^x2

//For Bob K=(A/g^{x2\*x4\*secret})^x4

temp=powdef(user->stp1d,(user->xb\*user->secret));

temp=user->stp2/temp;

user->key=powdef(temp,user->xb);

}

Step 9 :- Performs hashing on the computed keys for the users and maps them in the hash table.

**Code Snippet:**

**From Main:**

//Generating session key

hashing(&alice);

hashing(&bob);

**Function hashing declaration:**

//Hashing the computed keys to their positions in the hash table

void hashing(struct JPakeUser \*user)

{

long int k=user->key,i,s;

//Using sum of digits until it is between 0 and 10

while(k>10)

{

s=0;

while(k>0)

{

s=s+k%10;

k=k/10;

}

k=s;

}

//Finding the appropriate hashed position in the hash table

do

{

if(hash[k]==-999)

{

hash[k]=user->key;

user->sessionk=k;

break;

}

else

k++;

}while(1);

}

Step 10 :- Successful Termination.

**From Main:**

//Displaying the attributes of Alice and Bob

show(&alice);

show(&bob);

Algorithm of Deffie-Hellman Key Exchange:-

***Code Snippets for basic functions and structures***

**Variables needed for the program:**

long int x=0,y=0,g,p;

**User Defined Power Function:**

//Power function

//result=x^y

long int powdef(long int x,long int y)

{

long int i,res=1;

for(i=1;i<=y;i++)

{

res=res\*x;

}

return res;

}

**Function to generate the private keys of the users:**

//For generating the private keys

long int gen\_no(long int g,long int p,long int x)

{

long int r;

long int z;

z = pow(g,x);

if(g!=1)

r = z % p;

else

r=1;

return(r);

}

Step 1 :- The prime number and the generator are initialized and the choice of the user, whether he wants to perform the key exchange by Normal Exchange of key or Key Exchange infilterated by attacker.

**Code Snippet for Main:**

int main()

{

int ch;

//prime number(p)

p=11;//23

//generator(g)

g=7;//5

printf("\nPrime number p = %ld",p);

printf("\nGenerator g = %ld",g);

//Performing the key exchange depending on user's choice of Normal Exchange of key or Key Exchange infilterated by attacker

do

{

printf("\n\nDEFFIE-HELLMAN KEY EXCHANGE:-");

printf("\n1. Normal Exchange of key.");

printf("\n2. Key Exchange infilterated by attacker.");

printf("\n3. Exit.");

printf("\nEnter your choice (1-3):");

scanf("%d",&ch);

switch(ch)

{

case 1:

user\_a();

break;

case 2:

attacker();

break;

case 3:

exit(0);

default:

printf("Invalid Choice!");

break;

}

}while(1);

return 0;

}

Step 2 :- For choice 1 of user, user\_a() is invoked with x=0 and y=0. The private key of User A is entered and the public key is generated (x = gen\_no(g,p,a) ) and since it is a normal key exchange user\_b() is invoked. The private key of User B is entered and the public key is generated (y = gen\_no(g,p,b) ). Then common private key is calculated by User B (sb = gen\_no(x,p,b) ) and common private key is calculated by User A (sa = gen\_no(y,p,a) ).

**Code Snippet for User A:**

//To take user A's private key and generate the public key

void user\_a()

{

long int a,sa;

do

{

//Private key of user A(6,6)

printf("\nEnter User A's private key :- ");

scanf("%ld",&a);

if(a<p)

break;

else

printf("\nReenter");

}while(1);

//Public key for user A

x = gen\_no(g,p,a);

printf("\nPublic key for User A: %ld",x);

//Start actions of User B

if(y==0)

user\_b();

//Private key for both the users

sa = gen\_no(y,p,a);

printf("\nCommon Private Key as calculated by User A: %ld",sa);

}

**Code Snippet for User B:**

//To take user B's private key and generate the public key

void user\_b()

{

long int b,sb;

do

{

//Private key of user B(9,15)

printf("\nEnter User B's private key :- ");

scanf("%ld",&b);

if(b<p)

break;

else

printf("\nReenter");

}while(1);

//Public key for user B

y = gen\_no(g,p,b);

printf("\nPublic key for User B: %ld",y);

//Private key for both the users

sb = gen\_no(x,p,b);

printf("\nCommon Private Key as calculated by User B: %ld",sb);

}

Step 3 :- For choice 2 of user, attacker() is invoked with x=0 and y=0. The private key of User A is entered and the private key of User B is entered through the attacker. The Attacker generates the public key to be transmitted to User A and invokes user\_a(). The private key entered by User A and the public key generated by User A and the common private key generated by User A are now known to the Attacker. Then the does the same thing for User B. Here to User A the Attacker poses as User B and to User B the Attacker poses as User A.

**Code Snippet:**

//To take user A's private key and generate the public key

void attacker()

{

long int c,d,sa,sb;

//For User A

printf("\nEnter the 1st private key for attacker:");

scanf("%ld",&c);

//For User B

printf("\nEnter the 2nd private key for attacker:");

scanf("%ld",&d);

//Generating Public Key to be transmitted to User A

y=gen\_no(g,p,c);

user\_a();

//Calculating common private key between User A and Attacker

sa= gen\_no(x,p,c);

printf("\nCommon Private Key as calculated for User A by Attacker: %ld",sa);

//Generating Public Key to be transmitted to User B

x=gen\_no(g,p,d);

user\_b();

//Calculating common private key between User B and Attacker

sb=gen\_no(y,p,d);

printf("\nCommon Private Key as calculated for User B by Attacker: %ld",sb);

}