CNS record

▼ RSA block cipher asymmetric

- asymmetric cipher

```
#include <stdio.h>
#include <stdlib.h>

int modfun(int a, int n, int b)
{
    if (b == 1)
    {
        return a % n;
    }
    else
    {
        return ((a % n) * modfun(a, n, b - 1)) % n;
    }
}

int gcd(int a, int b)
{
    if (a == 0)
    {
        return b;
    }
    else if (b == 0)
    {
        return a;
    }
    else if (a == b)
    {
        return a;
    }
    else if (a > b)
        return gcd(a - b, b);
    else
        return gcd(a, b - a);
}

void main()
{
    int a, b, n, e, d, phi, m;
    int em, dm;
    printf("enter two prime numbers\n");
    scanf("%d %d", &a, &b);
    n = a * b;
    phi=(a-1)*(b-1);
    printf("%d", phi);
    printf("\nenter e value: ");
    scanf("%d", &e);
    if (0 < e < n && gcd(e, phi) == 1)
    {
        for (int i = 2; i < n; i++)
        {
            if ((e * i) % phi == 1)
            {
                printf("d value is: %d", i);
                d = i;
                break;
            }
        }
        printf("\nKU {%d %d}\n", e, n);
        printf("KR {%d %d}", d, n);

        printf("\nEnter message\n");
        scanf("%d", &m);
        em = modfun(m, n, e);
        dm = modfun(em, n, d);

        printf("encrypted message %d\n", em);
        printf("decrypted message %d\n", dm % n);
    }
    else
    {
        printf("invalid e value");
    }
}
```

▼ MD5

- produces 128 bit message digests

1. padding 64 bit less than multiple of 512
2. append [len mod64)]
3. dividing , we will get the multiples of 512
4. initializing the variables
5. takes 512 bits and produces 5*32=128 bits message digest.

```
import java.security.*;

class MD5 {
    public static void main(String[] a) {
        // TODO code application logic here
        try {
            MessageDigest md = MessageDigest.getInstance("MD5");
            System.out.println("Message digest object info: ");
            System.out.println(" Algorithm = " + md.getAlgorithm());
            System.out.println(" Provider = " + md.getProvider());
            System.out.println(" ToString = " + md.toString());
            String input = "";
            md.update(input.getBytes());
            byte[] output = md.digest();
            System.out.println();
            System.out.println("MD5(\"" + input + "\") = " + bytesToHex(output));

            input = "abc";
            md.update(input.getBytes());
            output = md.digest();
            System.out.println();
            System.out.println("MD5(\"" + input + "\") = " + bytesToHex(output));
        }

        catch (Exception e) {
            System.out.println("Exception: " + e);
        }
    }
    public static String bytesToHex(byte[] b) {
        char hexDigit[] = { '0', '1', '2', '3', '4', '5', '6', '7', '8', '9', 'A', 'B', 'C', 'D', 'E', 'F' };
        StringBuffer    buf = new StringBuffer();
        for (int j = 0; j < b.length; j++) {
            buf.append(hexDigit[(b[j] >> 4) & 0x0f]);
            buf.append(hexDigit[b[j] & 0x0f]);
        }
        return buf.toString();
    }
}
```

SHA1

similar to MD5 , but produces 160 bits

```
import java.security.MessageDigest;

public class SHA1 {
    public static void main(String[] a) {
        try {
            MessageDigest md = MessageDigest.getInstance("SHA1");
            System.out.println("Message digest object info: ");
            System.out.println(" Algorithm = " + md.getAlgorithm());
            System.out.println(" Provider = " + md.getProvider());
            System.out.println(" ToString = " + md.toString());
            String input = "";
            md.update(input.getBytes());
            byte[] output = md.digest();
            System.out.println();
            System.out.println("SHA1(\"" + input + "\") = " + bytesToHex(output));

            input = "abc";
            md.update(input.getBytes());
            output = md.digest();
            System.out.println();
            System.out.println("SHA1(\"" + input + "\") = " + bytesToHex(output));

        } catch (Exception e) {
            System.out.println("Exception: " + e);
        }
    }

    public static String bytesToHex(byte[] b) {
        char hexDigit[] = { '0', '1', '2', '3', '4', '5', '6', '7', '8', '9', 'A', 'B', 'C', 'D', 'E', 'F' };
        StringBuffer buf = new StringBuffer();
        for (int j = 0; j < b.length; j++) {
            buf.append(hexDigit[(b[j] >> 4) & 0x0f]);
            buf.append(hexDigit[b[j] & 0x0f]);
        }
        return buf.toString();
    }
}
```

BlowFish

```
import javax.crypto.*;
import javax.crypto.spec.*;
```

```
public class BlowFish   {
    public static void main(String[] args) throws Exception {

        KeyGenerator kgen = KeyGenerator.getInstance("Blowfish");
        Cipher cipher = Cipher.getInstance("Blowfish");
        SecretKey skey = kgen.generateKey();
        byte[] raw = skey.getEncoded();
        new SecretKeySpec(raw, "Blowfish");
        cipher.init(Cipher.ENCRYPT_MODE, skey);
        String inputText =new String("abcd");
        byte[] encrypted = cipher.doFinal(inputText.getBytes());
        cipher.init(Cipher.DECRYPT_MODE, skey);
        byte[] decrypted = cipher.doFinal(encrypted);
        System.out.println("DeCRYPTED "+ decrypted );
        System.exit(0);
    }
}
```

Diffie-Hellman

```
import java.util.Scanner;

public class DHKE {
    public static void main(String[] args) {
        int q, alpha_picked, xa, xb, ya, yb, ka, kb, index = 0;
        int alpha[] = new int[100];
        System.out.println("Enter the prime : ");
        try (Scanner sc = new Scanner(System.in)) {
            q = sc.nextInt();
            for (int i = 2; i < q; i++) {
                int alpharnot[] = new int[q];
                for (int j = 1; j <= q; j++) {
                    alpharnot[j - 1] = (int) ((java.lang.Math.pow(i, j)) % q);
                    int c = 0;
                    for (int k = 0; k < q; k++) {
                        for (int p = k + 1; p < q; p++) {
                            if (alpharnot[k] == alpharnot[p]) {
                                c++;
                            }
                        }
                    }
                    if (c == 0) {
                        alpha[index] = i;
                        index++;
                    }
                }
            }
            for (int i = 0; i < index; i++) {
                System.out.println("Primitive root is  : " + alpha[i]);
            }
            System.out.println("Select one of the root : ");
            alpha_picked = sc.nextInt();
            System.out.println("Select  Xa: ");
            xa = sc.nextInt();
            System.out.println("Select  Xb: ");
            xb = sc.nextInt();
        }
        ya = (int) ((java.lang.Math.pow(alpha_picked, xa)) % q);
        yb = (int) ((java.lang.Math.pow(alpha_picked, xb)) % q);
        ka = (int) ((java.lang.Math.pow(yb, xa)) % q);
        kb = (int) ((java.lang.Math.pow(ya, xb)) % q);
        System.out.println("Ka: " + ka + " Kb :" + kb);
        if (ka == kb) {
            System.out.println("Keys are same");
        }
    }
}
```