CSE 4000

Weekly presentation Date: 2-11-23

Mazharul Islam – 180710

Fermat's Little Theorem:

$$\boldsymbol{p^q = 1 \bmod m}$$

Here p, q prime number, and m is co-prime of p, q;

Using the formula we can compute modulus of Big-Integers.

Implementation code:

```
import java.math.BigInteger;
public class modInv {
    // Java program to find modular
// inverse of 'a' under modulo m
// using Fermat's little theorem.
// This program works only if m is prime.
   // BigInteger k2 = new BigInteger("1");
        static BigInteger __gcd(BigInteger a, BigInteger b)
        {

            if (b.intValue()==0) {
                return a;
            }
            else {
                return __gcd(b, a.mod(b));
            }
        }
        // To compute x^y under modulo m
        static BigInteger power(BigInteger x, BigInteger y, BigInteger m)
        {
            BigInteger k = new BigInteger("1");
            if (y.intValue()==0)
                return k;
            BigInteger k1 = new BigInteger("2");
            BigInteger p = power(x, y.divide(k1), m).mod(m);
            p = (p.multiply(p)).mod(m);

            return (y.mod(k1).intValue()==0) ? p : (x.multiply(p).mod(m));
        }
        // Function to find modular
        // inverse of 'a' under modulo m
        // Assumption: m is prime
        static void modInverse(BigInteger a, BigInteger m)
        {
           BigInteger k3 = new BigInteger("2");
            if (!(__gcd(a, m).intValue()==1))
                System.out.print("Inverse doesn't exist");

            else {

                // If a and m are relatively prime, then
                // modulo inverse is a^(m-2) mode m
```

```
                System.out.print(
                        "Modular multiplicative inverse is "
                                + power(a, m.subtract(k3), m));
            }
        }
        // Driver code
        public static void main(String[] args)
        {
            BigInteger a =new BigInteger("11") ;
            BigInteger m =new BigInteger("13");
            modInverse(a, m);
        }
    }
```

Now the main drive code is:

```
// Press Shift twice to open the Search Everywhere dialog and type `show
whitespaces`,
// then press Enter. You can now see whitespace characters in your code.
import java.math.BigInteger;
import java.math.BigInteger.*;
import java.security.SecureRandom;
import java.util.Random;

public class Main {

    static BigInteger __gcd(BigInteger a, BigInteger b)
    {

        if (b.intValue()==0) {
            return a;
        }
        else {
            return __gcd(b, a.mod(b));
        }
    }

    // To compute x^y under modulo m
    static BigInteger power(BigInteger x, BigInteger y, BigInteger m)
    {
        BigInteger k = new BigInteger("1");
        if (y.intValue()==0)
            return k;
        BigInteger k1 = new BigInteger("2");
        BigInteger p = power(x, y.divide(k1), m).mod(m);
        p = (p.multiply(p)).mod(m);

        return (y.mod(k1).intValue()==0) ? p : (x.multiply(p).mod(m));
    }

    // Function to find modular
    // inverse of 'a' under modulo m
    // Assumption: m is prime
    static void modInverse(BigInteger a, BigInteger m)
    {
        BigInteger k3 = new BigInteger("2");
        if (!(__gcd(a, m).intValue()==1))
            System.out.print("Inverse doesn't exist");
```

```java
        else {

            // If a and m are relatively prime, then
            // modulo inverse is a^(m-2) mode m
            System.out.print(
                    "Modular multiplicative inverse is "
                            + power(a, m.subtract(k3), m));
        }
    }
    void modPow(BigInteger p, BigInteger q, BigInteger m)
    {
        modInverse(p, m);
    }
    public static void main(String[] args) {
        int bit_length = 2048;
        Random rand = new SecureRandom();
        BigInteger p = BigInteger.probablePrime(bit_length, rand);
        //BigInteger p = new BigInteger("11");
        System.out.println("P : " + p);
        BigInteger q = BigInteger.probablePrime(bit_length/2, rand);
        //BigInteger q = new BigInteger("7");
        System.out.println("Q : "+q);
        BigInteger m = p.multiply(q);
        System.out.println("M : "+m);
        BigInteger r = BigInteger.probablePrime(bit_length/2, rand);
        //BigInteger r = new BigInteger("2");
        System.out.println("R : "+r);
        //System.out.println("Message : "+msg);
       // System.out.println("Cipher : "+"25");
        //System.out.println("Plaintext : "+"3");
        BigInteger msg = new BigInteger("3");
        System.out.println("Message : "+msg);

        // Encryption
        BigInteger cipher = (msg.add(r.multiply(p.modPow(q,m))));
       // cipher = cipher.mod(m);
        System.out.println("Cipher = " +cipher);
//public BigInteger decrypt(BigInteger p)
        // Decryption
        msg = cipher.mod(p);
        System.out.println("message = " + msg);
    }

}
```

Output:

P :
176957951863583411923117123056557819517042506457167911617018769570954430609011429085862775206875145820279101036564947930721498609750205449651396960199202351947000208161218818952903888673297433843651002045034641839534209003914783594194483583138117424137814369617056259985344309339315867351311934906699090615032262943949922264718469267512291260877627451444802958260776653082953903332780031209421973355357058582665420139877144560716138981940777172074810699247692449060662616365144834714725263277603412047113365959617657616390712744786529260870127475203702824856069543523978269596448640438509496129537114333971661596241 53

Q :
1219549240095076029932902408897564912818482450074934019220451168621634601859100065039912430430349634368477217346930446542434717962980754990518964824733389526591416612961953212630402750997590357094004964287335710030145363314898821762336725356608406238543079260822762296584387217460459927616168095575358386749633

M :
2158089357240141932156915859376490962990945497805887871800571833273678922691693125654223507895226115187992557369922119831256561181556761196039489543078869341513007476724011700428785643823723316591871779045195694701374680628823803573947322307550403207648887370830424187801852792833574447127382981335968770209470245579164973038868355312478984077861114079597611189473159591153665576836535303977636856696978054604063590555723049529252162737906338419464076949802375123087551822183452023187072637647780719224085895419771060994587080142448256160581281928431806761668808899478010374655257058687519825814597252176958715895424559298826912184046374525219587032503756175015611825891649420798508613766994599966560913779650848027162181938493831582177741242235640943530267192700684062804995955113519297041991065659857867869663514082731957050554150257497385117906401577140218219407192582822084821851975368401955017854199039122370493611181339

R :
1514615967701331218143740557662870993369086209076659144128497248585222719285462961074461352415818498794828560135947986953829410969389066495034574511192515589470607647318827530503688669466475000755195064421179834039298328848074417711629559779952429074888919221280538783917051783044492863305830393329145672124433

Message : 3

Cipher =
2680233395043069774757997298049527327512901602105304956410315089440513264581630028381998134562494301799359106874621056148982449876115198109084640819063551788552324666457577347626713667772179449815640809334816854507981025494806940630727059180376188832544186391956791956110978469286996548735442233925346410369254969505933502826608692413324039639894441640572899597252393456490268785593265137994506652480583502947106654925685290727996875013892124763208971985046551201323735416488474175421842246536139180075413997442157762276345081486495304364197558212849573843863266094947219738456362827505032507601673720523408861888350744055309644731300310001677596212614441392656964485673950309710400278408821292806555204090699049932029901795601051769167118472298542113303299534908357122386467120580749917644088565984351973437255954638169374649817114688033633652226787888449526955402435463135852120746502669780413758361825644488949185784935782

message = 3

Process finished with exit code 0

According to Gorti[1], this is the most efficient encryption algorithm for mobile Ad-hoc network against blackhole attack. Let's compare with some other famous encryption algorithm.

Message size in bits	Elgamal	MMH	EHES
250	89	21	9
500	105	21	11
1000	142	22	8

Table 1: [1] Encryption process time of Schemes µSec with key size 512 bit increase in the encryption key size

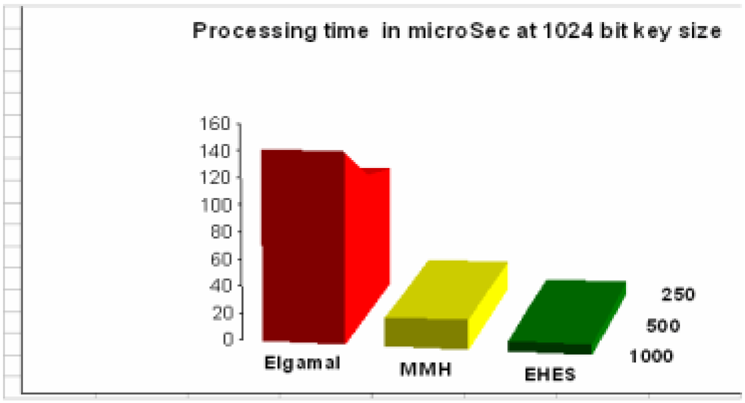

Figure 1: Processing time of schemes in µSec at 1024 bits key size

Next week plan:

1) Visualization of Blackhole attack using AODV and AOMDV routing protocol against Blackhole attacks

References:

[1] armar PV, Padhar SB, Patel SN, Bhatt NI, Jhaveri RH. Survey of various homo morphic encryption algorithms and schemes. Int J Comput Appl 2014;91:26– 32. doi:10.5120/15902-5081.