Assignment No: 10A

Title of the Assignment: RSA Algorithm

Problem statement: Write a Java/C/C++/Python program to implement RSA algorithm.

Objective:

- To understand the key generation using RSA Cryptographic technique.
- To implement encryption/decryption using RSA.

Theory:

RSA is a cryptosystem for public-key encryption, and is widely used for securing sensitive data, particularly when being sent over an insecure network such as the Internet.

RSA was first described in 1977 by Ron Rivest, Adi Shamir and Leonard Adleman of the Massachusetts Institute of Technology. Public-key cryptography, also known as asymmetric cryptography, uses two different but mathematically linked keys, one public and one private. The public key can be shared with everyone, whereas the private key must be kept secret. In RSA cryptography, both the public and the private keys can encrypt a message; the opposite key from the one used to encrypt a message is used to decrypt it. This attribute is one reason why RSA has become the most widely used asymmetric algorithm: It provides a method of assuring the confidentiality, integrity, authenticity and non-reputability of electronic communications and data storage.

Many protocols like SSH, OpenPGP, S/MIME, and SSL/TLS rely on RSA for encryption and digital signature functions. It is also used in software programs -- browsers are an obvious example, which need to establish a secure connection over an insecure network like the Internet or validate a digital signature. RSA signature verification is one of the most commonly performed operations in IT.

Explaining RSA's popularity

RSA derives its security from the difficulty of factoring large integers that are the product of two large prime numbers. Multiplying these two numbers is easy, but determining the original prime numbers from the total -- factoring -- is considered infeasible due to the time it would take even using today’s super computers.

The public and the private key-generation algorithm is the most complex part of RSA cryptography. Two large prime numbers, *p* and *q*, are generated using the Rabin-Miller primality test algorithm. A modulus *n* is calculated by multiplying *p* and *q*. This number is used by both the public and private keys and provides the link between them. Its length, usually expressed in bits, is called the key length. The public key consists of the modulus *n*,and a public exponent, *e*, which is normally set at 65537, as it's a prime number that is not too large. The *e* figure doesn't have to be a secretly selected prime number as the public key is shared with everyone. The private key consists of the modulus *n* and the private exponent *d*, which is calculated using the Extended Euclidean algorithm to find the multiplicative inverse with respect to the totient of *n*.

A simple, worked example

Alice generates her RSA keys by selecting two primes: p=11 and q=13. The modulus $n=p\times q$=143. The totient of n $\phi(n)=(p-1)x(q-1)=120$. She chooses 7 for her RSA public key *e*and calculates her RSA private key using the Extended Euclidean Algorithm which gives her 103.

Bob wants to send Alice an encrypted message *M* so he obtains her RSA public key (*n*, e) which in this example is (143, 7). His plaintext message is just the number 9 and is encrypted into ciphertext *C* as follows:

$M^e \bmod n = 9^7 \bmod 143 = 48 = C$

When Alice receives Bob’s message she decrypts it by using her RSA private key (*d*, *n*) as follows:

$C^d \bmod n = 48^{103} \bmod 143 = 9 = M$

To use RSA keys to digitally sign a message, Alice would create a hash or message digest of her message to Bob, encrypt the hash value with her RSA private key and add it to the message. Bob can then verify that the message has been sent by Alice and has not been altered by decrypting the hash value with her public key. If this value matches the hash of the original message, then only Alice could have sent it (authentication and non-repudiation) and the message is exactly as she wrote it (integrity). Alice could, of course, encrypt her message with Bob's RSA public key (confidentiality) before sending it to Bob. A digital certificate contains information that identifies the certificate's owner and also contains the owner's public key. Certificates are signed by the certificate authority that issues them, and can simplify the process of obtaining public keys and verifying the owner.

Security of RSA

As discussed, the security of RSA relies on the computational difficulty of factoring large integers. As computing power increases and more efficient factoring algorithms are discovered, the ability to factor larger and larger numbers also increases. Encryption strength is directly tied to key size, and doubling key length delivers an exponential increase in strength, although it does impair performance. RSA keys are typically 1024- or 2048-bits long, but experts believe that 1024-bit keys could be broken in the near future, which is why government and industry are moving to a minimum key length of 2048-bits. Barring an unforeseen breakthrough in quantum computing, it should be many years before longer keys are required, but elliptic curve cryptography is gaining favor with many security experts as an alternative to RSA for implementing public-key cryptography. It can create faster, smaller and more efficient cryptographic keys. Much of today's hardware and software is ECC-ready and its popularity is likely to grow as it can deliver equivalent security with lower computing power and battery resource usage, making it more suitable for mobile apps than RSA. Finally, a team of researchers which included Adi Shamir, a co-inventor of RSA, has successfully determined a 4096-bit RSA key using acoustic cryptanalysis, however any encryption algorithm is vulnerable to this type of attack.

RSA algorithm is asymmetric cryptography algorithm. Asymmetric actually means that it works on two different keys i.e. **Public Key** and **Private Key.** As the name describes that the Public Key is given to everyone and Private key is kept private.

An example of asymmetric cryptography :

1. A client (for example browser) sends its public key to the server and requests for some data.
2. The server encrypts the data using client's public key and sends the encrypted data.
3. Client receives this data and decrypts it.

Since this is asymmetric, nobody else except browser can decrypt the data even if a third party has public key of browser.

The idea! The idea of RSA is based on the fact that it is difficult to factorize a large integer. The public key consists of two numbers where one number is multiplication of two large prime numbers. And private key is also derived from the same two prime numbers. So if somebody can factorize the large number, the private key is compromised. Therefore encryption strength totally lies on the key size and if we double or triple the key size, the strength of encryption increases exponentially. RSA keys can be typically 1024 or 2048 bits long, but experts believe that 1024 bit keys could be broken in the near future. But till now it seems to be an infeasible task.

Let us learn the mechanism behind RSA algorithm :

>> Generating Public Key :

- Select two prime no's. Suppose **P = 53 and Q = 59**.

Now First part of the Public key : **n = P*Q = 3127**.

- We also need a small exponent say **e** :

But e Must be

- An integer.
- Not be a factor of n.
- **1 < e < Φ(n)** [Φ(n) is discussed below],
- Let us now consider it to be equal to 3.

- Our Public Key is made of n and e

>> Generating Private Key :

- We need to calculate Φ(n) :

Such that **Φ(n) = (P-1)(Q-1)**
so, Φ(n) = 3016

- Now calculate Private Key, **d** :

d = (k*Φ(n) + 1) / e for some integer k
For k = 2, value of d is 2011.

Now we are ready with our – Public Key (n = 3127 and e = 3) and Private Key(d = 2011)

Now we will encrypt **"HI"** :

- Convert letters to numbers : H = 8 and I = 9

- Thus **Encrypted Data c = 89^e mod n**.

Thus our Encrypted Data comes out to be 1394

Now we will decrypt **1394** :

- **Decrypted Data = c^d mod n**.

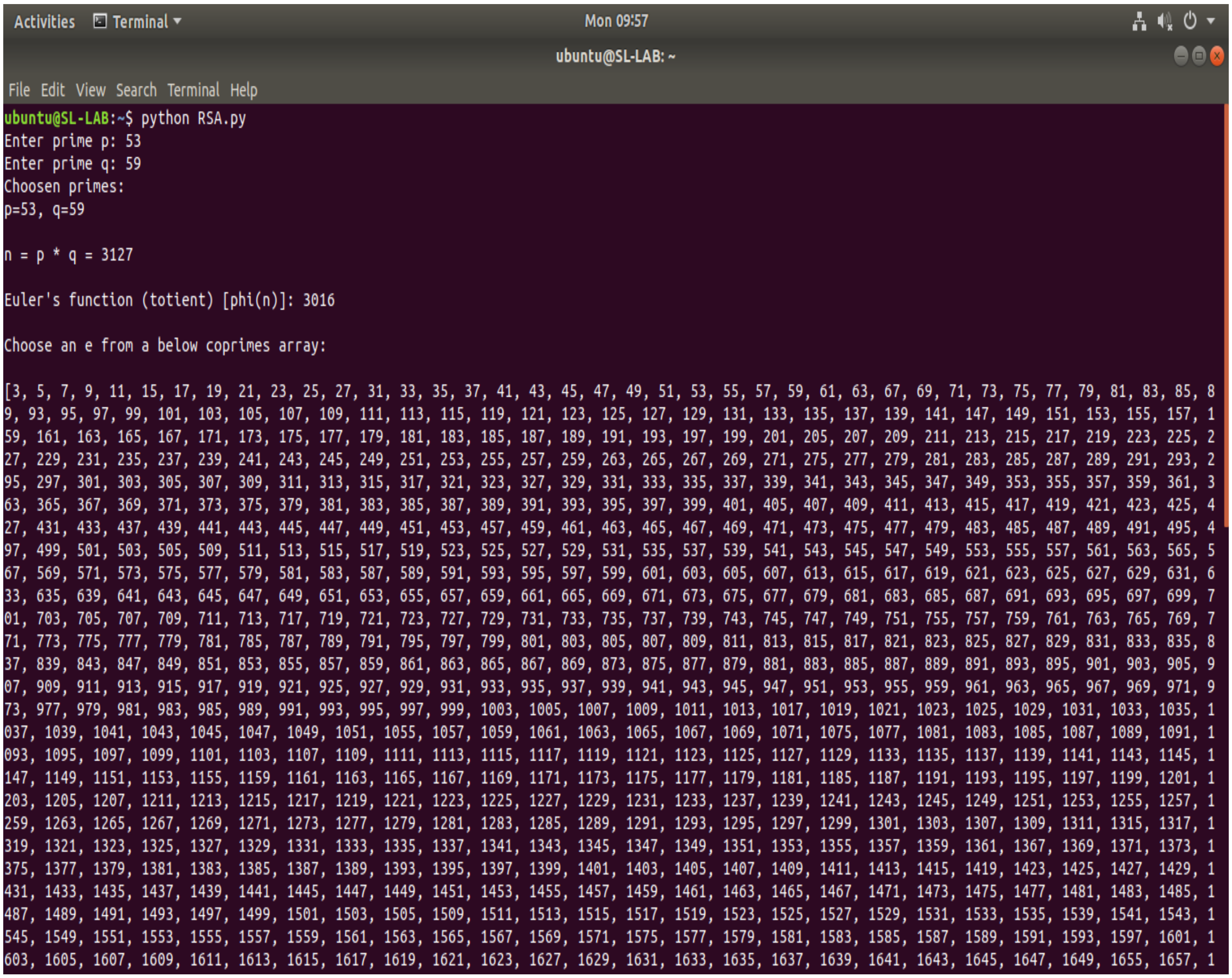
Thus our Encrypted Data comes out to be 89
8 = H and I = 9 i.e. "HI".

RSA Program :

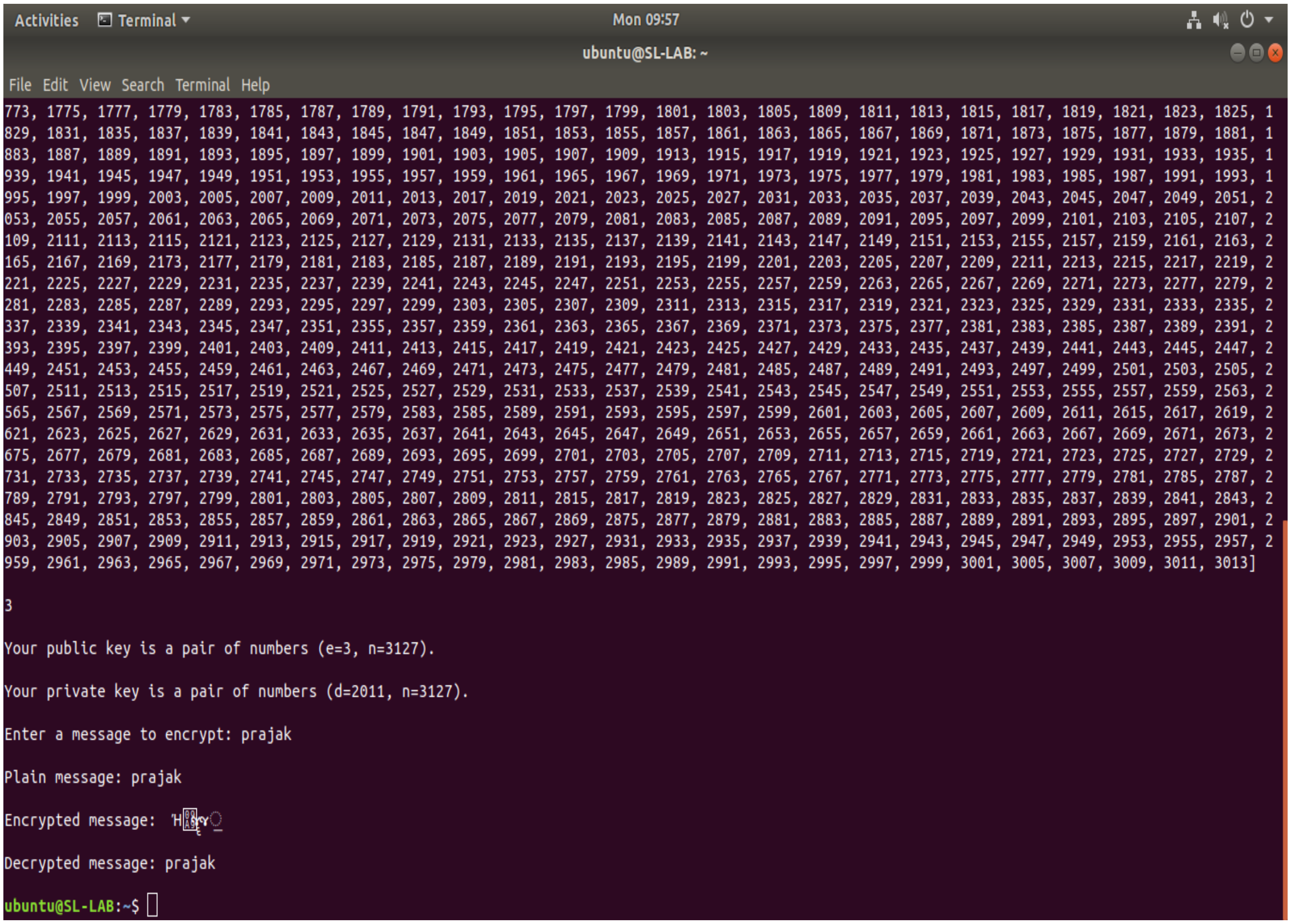
```
def gcd(a, b): # calculates GCD of a and d
    while b != 0:
        c = a % b
        a = b
        b = c
    return a


def modinv(a, m): # calculates modulo inverse of a for mod m
    for x in range(1, m):
        if (a * x) % m == 1:
            return x
    return None


def coprimes(a): # calculates all possible co-prime numbers with a
    l = []
    for x in range(2, a):
        if gcd(a, x) == 1 and modinv(x, phi) != None:
            l.append(x)
    for x in l:
        if x == modinv(x, phi):
            l.remove(x)
    return l

def encrypt_block(m): # encrypts a single block
    c = m ** e % n
    return c


def decrypt_block(c): # decrypts a single block
    m = c ** d % n
    return m


def encrypt_string(s): # applies encryption
    return ''.join([chr(encrypt_block(ord(x))) for x in list(s)])
```

```
def decrypt_string(s): # applies decryption
    return ''.join([chr(decrypt_block(ord(x))) for x in list(s)])

if __name__ == "__main__":
    p = int(input('Enter prime p: '))
    q = int(input('Enter prime q: '))

    print("Choosen primes:\np=" + str(p) + ", q=" + str(q) + "\n")

    n = p * q
    print("n = p * q = " + str(n) + "\n")

    phi = (p - 1) * (q - 1)
    print("Euler's function (totient) [phi(n)]: " + str(phi) + "\n")

    print("Choose an e from a below coprimes array:\n")
    print(str(coprimes(phi)) + "\n")
    e = int(input())

    d = modinv(e, phi) # calculates the decryption key d

    print("\nYour public key is a pair of numbers (e=" + str(e) + ", n=" +
str(n) + ").\n")
    print("Your private key is a pair of numbers (d=" + str(d) + ", n=" + str(n)
+ ").\n")

    s = input("Enter a message to encrypt: ")
    print("\nPlain message: " + s + "\n")
    enc = encrypt_string(s)
    print("Encrypted message: ", enc, "\n")
    dec = decrypt_string(enc)
    print("Decrypted message: " + dec + "\n")
```

Output:



```
ubuntu@SL-LAB:~$ python RSA.py
Enter prime p: 53
Enter prime q: 59
Choosen primes:
p=53, q=59

n = p * q = 3127

Euler's function (totient) [phi(n)]: 3016

Choose an e from a below coprimes array:

[3, 5, 7, 9, 11, 15, 17, 19, 21, 23, 25, 27, 31, 33, 35, 37, 41, 43, 45, 47, 49, 51, 53, 55, 57, 59, 61, 63, 67, 69, 71, 73, 75, 77, 79, 81, 83, 85, 8
9, 93, 95, 97, 99, 101, 103, 105, 107, 109, 111, 113, 115, 119, 121, 123, 125, 127, 129, 131, 133, 135, 137, 139, 141, 147, 149, 151, 153, 155, 157, 1
59, 161, 163, 165, 167, 171, 173, 175, 177, 179, 181, 183, 185, 187, 189, 191, 193, 197, 199, 201, 205, 207, 209, 211, 213, 215, 217, 219, 223, 225, 2
27, 229, 231, 235, 237, 239, 241, 243, 245, 249, 251, 253, 255, 257, 259, 263, 265, 267, 269, 271, 275, 277, 279, 281, 283, 285, 287, 289, 291, 293, 2
95, 297, 301, 303, 305, 307, 309, 311, 313, 315, 317, 321, 323, 327, 329, 331, 333, 335, 337, 339, 341, 343, 345, 347, 349, 353, 355, 357, 359, 361, 3
63, 365, 367, 369, 371, 373, 375, 379, 381, 383, 385, 387, 389, 391, 393, 395, 397, 399, 401, 405, 407, 409, 411, 413, 415, 417, 419, 421, 423, 425, 4
27, 431, 433, 437, 439, 441, 443, 445, 447, 449, 451, 453, 457, 459, 461, 463, 465, 467, 469, 471, 473, 475, 477, 479, 483, 485, 487, 489, 491, 495, 4
97, 499, 501, 503, 505, 509, 511, 513, 515, 517, 519, 523, 525, 527, 529, 531, 535, 537, 539, 541, 543, 545, 547, 549, 553, 555, 557, 561, 563, 565, 5
67, 569, 571, 573, 575, 577, 579, 581, 583, 587, 589, 591, 593, 595, 597, 599, 601, 603, 605, 607, 613, 615, 617, 619, 621, 623, 625, 627, 629, 631, 6
33, 635, 639, 641, 643, 645, 647, 649, 651, 653, 655, 657, 659, 661, 665, 669, 671, 673, 675, 677, 679, 681, 683, 685, 687, 691, 693, 695, 697, 699, 7
01, 703, 705, 707, 709, 711, 713, 717, 719, 721, 723, 727, 729, 731, 733, 735, 737, 739, 743, 745, 747, 749, 751, 755, 757, 759, 761, 763, 765, 769, 7
71, 773, 775, 777, 779, 781, 785, 787, 789, 791, 795, 797, 799, 801, 803, 805, 807, 809, 811, 813, 815, 817, 821, 823, 825, 827, 829, 831, 833, 835, 8
37, 839, 843, 847, 849, 851, 853, 855, 857, 859, 861, 863, 865, 867, 869, 873, 875, 877, 879, 881, 883, 885, 887, 889, 891, 893, 895, 901, 903, 905, 9
07, 909, 911, 913, 915, 917, 919, 921, 925, 927, 929, 931, 933, 935, 937, 939, 941, 943, 945, 947, 951, 953, 955, 959, 961, 963, 965, 967, 969, 971, 9
73, 977, 979, 981, 983, 985, 989, 991, 993, 995, 997, 999, 1003, 1005, 1007, 1009, 1011, 1013, 1017, 1019, 1021, 1023, 1025, 1029, 1031, 1033, 1035, 1
037, 1039, 1041, 1043, 1045, 1047, 1049, 1051, 1055, 1057, 1059, 1061, 1063, 1065, 1067, 1069, 1071, 1075, 1077, 1081, 1083, 1085, 1087, 1089, 1091, 1
093, 1095, 1097, 1099, 1101, 1103, 1107, 1109, 1111, 1113, 1115, 1117, 1119, 1121, 1123, 1125, 1127, 1129, 1133, 1135, 1137, 1139, 1141, 1143, 1145, 1
147, 1149, 1151, 1153, 1155, 1159, 1161, 1163, 1165, 1167, 1169, 1171, 1173, 1175, 1177, 1179, 1181, 1185, 1187, 1191, 1193, 1195, 1197, 1199, 1201, 1
203, 1205, 1207, 1211, 1213, 1215, 1217, 1219, 1221, 1223, 1225, 1227, 1229, 1231, 1233, 1237, 1239, 1241, 1243, 1245, 1249, 1251, 1253, 1255, 1257, 1
259, 1263, 1265, 1267, 1269, 1271, 1273, 1277, 1279, 1281, 1283, 1285, 1289, 1291, 1293, 1295, 1297, 1299, 1301, 1303, 1307, 1309, 1311, 1315, 1317, 1
319, 1321, 1323, 1325, 1327, 1329, 1331, 1333, 1335, 1337, 1341, 1343, 1345, 1347, 1349, 1351, 1353, 1355, 1357, 1359, 1361, 1367, 1369, 1371, 1373, 1
375, 1377, 1379, 1381, 1383, 1385, 1387, 1389, 1393, 1395, 1397, 1399, 1401, 1403, 1405, 1407, 1409, 1411, 1413, 1415, 1419, 1423, 1425, 1427, 1429, 1
431, 1433, 1435, 1437, 1439, 1441, 1445, 1447, 1449, 1451, 1453, 1455, 1457, 1459, 1461, 1463, 1465, 1467, 1471, 1473, 1475, 1477, 1481, 1483, 1485, 1
487, 1489, 1491, 1493, 1497, 1499, 1501, 1503, 1505, 1509, 1511, 1513, 1515, 1517, 1519, 1523, 1525, 1527, 1529, 1531, 1533, 1535, 1539, 1541, 1543, 1
545, 1549, 1551, 1553, 1555, 1557, 1559, 1561, 1563, 1565, 1567, 1569, 1571, 1575, 1577, 1579, 1581, 1583, 1585, 1587, 1589, 1591, 1593, 1597, 1601, 1
603, 1605, 1607, 1609, 1611, 1613, 1615, 1617, 1619, 1621, 1623, 1627, 1629, 1631, 1633, 1635, 1637, 1639, 1641, 1643, 1645, 1647, 1649, 1655, 1657, 1
```


```
773, 1775, 1777, 1779, 1783, 1785, 1787, 1789, 1791, 1793, 1795, 1797, 1799, 1801, 1803, 1805, 1809, 1811, 1813, 1815, 1817, 1819, 1821, 1823, 1825, 1
829, 1831, 1835, 1837, 1839, 1841, 1843, 1845, 1847, 1849, 1851, 1853, 1855, 1857, 1861, 1863, 1865, 1867, 1869, 1871, 1873, 1875, 1877, 1879, 1881, 1
883, 1887, 1889, 1891, 1893, 1895, 1897, 1899, 1901, 1903, 1905, 1907, 1909, 1913, 1915, 1917, 1919, 1921, 1923, 1925, 1927, 1929, 1931, 1933, 1935, 1
939, 1941, 1945, 1947, 1949, 1951, 1953, 1955, 1957, 1959, 1961, 1965, 1967, 1969, 1971, 1973, 1975, 1977, 1979, 1981, 1983, 1985, 1987, 1991, 1993, 1
995, 1997, 1999, 2003, 2005, 2007, 2009, 2011, 2013, 2017, 2019, 2021, 2023, 2025, 2027, 2031, 2033, 2035, 2037, 2039, 2043, 2045, 2047, 2049, 2051, 2
053, 2055, 2057, 2061, 2063, 2065, 2069, 2071, 2073, 2075, 2077, 2079, 2081, 2083, 2085, 2087, 2089, 2091, 2095, 2097, 2099, 2101, 2103, 2105, 2107, 2
109, 2111, 2113, 2115, 2121, 2123, 2125, 2127, 2129, 2131, 2133, 2135, 2137, 2139, 2141, 2143, 2147, 2149, 2151, 2153, 2155, 2157, 2159, 2161, 2163, 2
165, 2167, 2169, 2173, 2177, 2179, 2181, 2183, 2185, 2187, 2189, 2191, 2193, 2195, 2199, 2201, 2203, 2205, 2207, 2209, 2211, 2213, 2215, 2217, 2219, 2
221, 2225, 2227, 2229, 2231, 2235, 2237, 2239, 2241, 2243, 2245, 2247, 2251, 2253, 2255, 2257, 2259, 2263, 2265, 2267, 2269, 2271, 2273, 2277, 2279, 2
281, 2283, 2285, 2287, 2289, 2293, 2295, 2297, 2299, 2303, 2305, 2307, 2309, 2311, 2313, 2315, 2317, 2319, 2321, 2323, 2325, 2329, 2331, 2333, 2335, 2
337, 2339, 2341, 2343, 2345, 2347, 2351, 2355, 2357, 2359, 2361, 2363, 2365, 2367, 2369, 2371, 2373, 2375, 2377, 2381, 2383, 2385, 2387, 2389, 2391, 2
393, 2395, 2397, 2399, 2401, 2403, 2409, 2411, 2413, 2415, 2417, 2419, 2421, 2423, 2425, 2427, 2429, 2433, 2435, 2437, 2439, 2441, 2443, 2445, 2447, 2
449, 2451, 2453, 2455, 2459, 2461, 2463, 2467, 2469, 2471, 2473, 2475, 2477, 2479, 2481, 2485, 2487, 2489, 2491, 2493, 2497, 2499, 2501, 2503, 2505, 2
507, 2511, 2513, 2515, 2517, 2519, 2521, 2525, 2527, 2529, 2531, 2533, 2537, 2539, 2541, 2543, 2545, 2547, 2549, 2551, 2553, 2555, 2557, 2559, 2563, 2
565, 2567, 2569, 2571, 2573, 2575, 2577, 2579, 2583, 2585, 2589, 2591, 2593, 2595, 2597, 2599, 2601, 2603, 2605, 2607, 2609, 2611, 2615, 2617, 2619, 2
621, 2623, 2625, 2627, 2629, 2631, 2633, 2635, 2637, 2641, 2643, 2645, 2647, 2649, 2651, 2653, 2655, 2657, 2659, 2661, 2663, 2667, 2669, 2671, 2673, 2
675, 2677, 2679, 2681, 2683, 2685, 2687, 2689, 2693, 2695, 2699, 2701, 2703, 2705, 2707, 2709, 2711, 2713, 2715, 2719, 2721, 2723, 2725, 2727, 2729, 2
731, 2733, 2735, 2737, 2739, 2741, 2745, 2747, 2749, 2751, 2753, 2757, 2759, 2761, 2763, 2765, 2767, 2771, 2773, 2775, 2777, 2779, 2781, 2785, 2787, 2
789, 2791, 2793, 2797, 2799, 2801, 2803, 2805, 2807, 2809, 2811, 2815, 2817, 2819, 2823, 2825, 2827, 2829, 2831, 2833, 2835, 2837, 2839, 2841, 2843, 2
845, 2849, 2851, 2853, 2855, 2857, 2859, 2861, 2863, 2865, 2867, 2869, 2875, 2877, 2879, 2881, 2883, 2885, 2887, 2889, 2891, 2893, 2895, 2897, 2901, 2
903, 2905, 2907, 2909, 2911, 2913, 2915, 2917, 2919, 2921, 2923, 2927, 2931, 2933, 2935, 2937, 2939, 2941, 2943, 2945, 2947, 2949, 2953, 2955, 2957, 2
959, 2961, 2963, 2965, 2967, 2969, 2971, 2973, 2975, 2979, 2981, 2983, 2985, 2989, 2991, 2993, 2995, 2997, 2999, 3001, 3005, 3007, 3009, 3011, 3013]

3

Your public key is a pair of numbers (e=3, n=3127).

Your private key is a pair of numbers (d=2011, n=3127).

Enter a message to encrypt: prajak

Plain message: prajak

Encrypted message:  'H[illegible]

Decrypted message: prajak

ubuntu@SL-LAB:~$
```

Conclusion:

We have implemented RSA algorithm using Python.

Oral questions:

1. What is the real crux of RSA?

2. Discuss the security of RSA.
3. L:ist the security protocols that uses RSA encryption algorithm.
4. What requirements must RSA fulfill to be a secure algorithm?
5. Using the RSA algorithm, if a small number of repeated encodings give back the plaintext, what is the likely cause?
6. Describe the advantages of asymmetric-key cryptography over symmetric-key cryptography.
7. In RSA, each user has a public key e, and private key, d. Suppose Bob leaks his private key. Rather than generating a new modulus, he decides to generate a new public and a new private key. Is this safe?
8. List the security services achieved by RSA cryptographic algorithm.
9. Discuss the factorization attack with respect to RSA.
10. What are the security vulnerabilities of RSA?
11. How Is RSA Used For Authentication In Practice?