Example of the RSA Algorithm

Crash course in modulo arithmetic with Pari

1. Define a number mod 24:
```
(09:27) gp > a = Mod(5, 24)
    %1 = Mod(5, 24)
```

2. Define another one:
```
(09:28) gp > b = Mod(11, 24)
    %2 = Mod(11, 24)
```

3. We can add them:
```
(09:28) gp > a * b
    %3 = Mod(7, 24)
```

4. ... multiply:
```
(09:28) gp > a + b
    %4 = Mod(16, 24)
```

5. ... take inverse:
```
(09:28) gp > 1/a
    %5 = Mod(5, 24)
```

6. ... if possible, that is:
```
(09:28) gp > 1/(a+b)
    *** impossible inverse modulo:  Mod(8, 24).
```

7. This is how to get a “usual” number back:
```
(09:43) gp > lift (a-b)
    %6 = 12
```

Preparation of the RSA data (Receiver side)

8. Find a random **LARGE** prime number:
```
(09:28) gp > p = precprime(random(10^100))
    %7 = 72630520263309287726720073181289266279220282194494505829690299022161
    00445447520540359470447610370331
```

9. ... and another one:
```
(09:36) gp > q = precprime(random(10^100))
    %8 = 59531874687060904897504991500535808745752512354891691036536183975314
    50761504106381126226420803762109
```

10. Part of the keys, all calculations will be done mod n:
```
(09:36) gp > n = p * q
    %9 = 43238310307713663152626975586165863035763396194413441466153480992484
    3908526224769896556053175453288490491672782389361184848551882069356798917
    77898161410498766420920797865802667984297850830287515588079
```

11. Helping constant:
```
(09:36) gp > m = lcm(p-1, q-1)
    %10 = 7206385051285610525437829264360977172627232699068906911025580165414
    0651421037461649426008862575548059721546305336528093437247866707380295657
    49740537131802023962103508385765952726229394188903183575940
```

12. This is going to be a **public key**
```
(09:36) gp > r = random(10^30)
    %11 = 556734754038906146893491637226
```

13. It must be relatively prime with m:
```
(09:37) gp > gcd(r, m)
    %12 = 2
```

14. Another try:
```
(09:37) gp > r = random(10^40)
    %13 = 8164154045487638680805646584601153678131
```

15. ... fine now:
```
(09:37) gp > gcd(r, m)
    %14 = 1
```

16. And this is the corresponding **private key**
```
(09:37) gp > s = lift(1 / Mod(r, m))
    %15 = 6442767185184458325552853081485590819967021981560067729213608872604
    1417175025265087812792330987010742407156548165071458511016503988352035938
    14840888846631624616779242107929197757094637519271543308491
```

17. Check that $rs \equiv 1 \bmod m$:
```
(09:37) gp > r * s % m
    %16 = 1
```

18. Send the public key pair (n, r) to the Sender.

Encrypting the data (Sender side)

19. This is going to be our message:
```
(09:37) gp > M1 = 98723847682187283765098299879
    %17 = 98723847682187283765098299879
```

20. Encrypt it, that is make $R_1 = M_1^r \bmod n$:
```
(09:38) gp > R1 = lift(Mod(M1, n)^r)
    %18 = 80626700610115267140056678324946859121962548434454256758279386453983
    51953775686009932634800704211491995735768989295246680760296934092219964377
    35978426495472147386915330330267322923632498505349180273 8
```

21. Send the encrypted message R_1 to the Receiver.

Decrypting the data (Receiver side)

22. Decrypted message should be $A_1 = R_1^s \bmod n$:
```
(09:39) gp > A1 = lift(Mod(R1, n)^s)
    %19 = 98723847682187283765098299879
```

23. Check the result:
```
(09:39) gp > A1 - M1
    %20 = 0
```

24. Repeat the process once more. Here's a new message:
```
(09:39) gp > M2 = random(10^50)
    %21 = 15003568416585151247272618283690698009974349861232
```

25. ... encryption:
```
(09:39) gp > R2 = lift(Mod(M2, n)^r)
    %22 = 15176819958773309008621681896061018536965567386204887677949877964162
    70639564654401242849191607206470036059679919564719085277903601087500910976
    369189519028872021703142262739984863665477746257048045381 3
```

26. ... decryption:
```
(09:39) gp > A2 = lift(Mod(R2, n)^s)
    %23 = 15003568416585151247272618283690698009974349861232
```

27. ... checking the answer:
```
(09:39) gp > A2 - M2
    %24 = 0
```