RSA Encryption Scheme

The following is a demonstration of RSA public key encryption. Alice makes the following preparations so that others will be able to send her secure messages over an open communication channel. She chooses two large primes p and q at random. She sets $n = p\,q$ and computes the value of $\phi(n) = (p-1)(q-1)$.

```
p=NextPrime[RandomInteger[{1,10^50}]];  Print["p = ",p]
q=NextPrime[RandomInteger[{1,10^50}]];  Print["q = ",q]
n=p*q;  Print["n = pq = ",n]
phin=(p-1)*(q-1);  Print[phi,"(n) = ",phin]
```

p = 97 057 500 264 268 002 225 242 768 399 993 153 216 795 359 838 153

q = 11 655 474 417 508 749 124 994 185 123 315 787 079 242 883 010 493

n = pq =
1 131 251 211 357 524 357 501 585 396 202 651 850 228 686 763 380 537 149 023 068 789 155 594 566 675 746 312 712 - 490 825 880 739 429

phi(n) =
1 131 251 211 357 524 357 501 585 396 202 651 850 228 686 763 380 428 436 048 387 012 404 244 329 722 223 003 772 - 194 787 637 890 784

Alice chooses a random number $e < n$. She checks that gcd $(n, \phi(n)) = 1$. (If this condition fails, she finds a new value of e that works.)

```
e=RandomInteger[{1,n}];  While[GCD[e,phin]≠1, Print[e," fails"]; e=RandomInteger[{1,n}]];  Print
```

520 701 938 428 561 212 152 877 804 445 021 992 252 640 927 943 400 413 453 805 483 347 818 164 151 445 255 713 798 - 048 753 774 966 fails

1 119 304 586 142 527 664 949 455 583 019 205 985 148 281 079 274 987 147 166 372 423 543 502 405 800 572 193 269 - 428 234 867 737 916 fails

47 158 469 020 137 022 753 718 559 838 789 774 704 070 439 063 613 992 019 242 759 903 939 162 494 543 408 652 453 - 000 318 515 518 fails

741 100 298 609 214 423 847 133 753 136 755 869 133 204 417 775 059 597 507 076 877 446 025 963 081 160 051 951 818 - 112 518 764 584 fails

183 033 519 733 464 198 470 628 491 104 706 325 671 338 332 506 573 739 751 158 156 004 221 931 149 207 406 668 738 - 471 886 604 198 fails

e =
723 714 857 551 705 973 787 893 648 042 547 244 424 553 610 755 277 880 975 860 374 982 586 385 352 503 116 584 - 061 145 640 338 623

Alice sets d equal to the inverse of e mod $\phi(n)$ (which her computer finds using Euclid's Algorithm). This is her decryption key.

```
d=ModularInverse[e,phin];  Print["d = ",d]
```

```
d =
 28 867 380 583 421 789 276 539 287 913 823 022 577 090 657 437 397 426 761 593 300 951 992 720 182 438 129 064 405 -
   588 680 440 671
```

Alice publishes the values of n and e (her public encryption key). She keeps the other values (p, q, $\phi(n)$ and d) secret.

Bob wants to send Alice an encrypted message over an insecure channel, which only she can decrypt. His message, “SEND MONEY”, is first translated into an integer m using A=01, B=02, ..., Z=26, blank=27, giving

```
m=19051404271315140525;  Print["m = ",m]
```

```
m = 19 051 404 271 315 140 525
```

(If m is larger than n, he must split up this string of digits into pieces, each of which is smaller than n, and encrypt each piece of the message separately.) Bob looks up Alice’s public key (n, e) posted on her website. He encrypts his message m by raising it to the power e mod n, which his computer evaluates using fast modular exponentiation.

```
m1=PowerMod[m,e,n];  Print["m1 = m^e (mod n) = ",m1]
```

```
m1 = m^e (mod n) =
 601 676 881 621 439 677 800 852 525 881 647 716 613 298 280 508 276 430 701 381 095 526 243 143 796 073 316 949 -
   582 897 328 716 028
```

Bob sends the resulting encrypted message m_1 to Alice over the insecure communication channel.

Alice receives the encrypted message m_1. To decrypt it, she raises m_1 to the power d mod n, also using modular exponentiation:

In[2]:=
```
m2=PowerMod[m1,d,n];  Print["m2 = m1^d (mod n) = ",m2]
```

```
m2 = m1^d (mod n) = 19 051 404 271 315 140 525
```

The resulting integer m_2 which she obtains, coincides with Bob’s original message m.