3-pass protocol, digital version

- Bob, who wants to receive something securely (and potentially anonymously), chooses a large prime p and publishes it.
- Alice encodes her message as a number $m < p$ and also picks an exponent a, $0 < a < p$ with $\gcd(a, p-1) = 1$. She posts m^a mod p.
- Bob picks his own exponent b, $0 < b < p$ with $\gcd(b, p-1) = 1$ and publishes m^{ab}.
- Alice knows the multiplicative inverse of a mod $(p-1)$ and so she is able to post m^b mod p.
- Bob then uses the multiplicative inverse of b mod $(p-1)$ to determine m.

Primality testing, Miller-Rabin

Lemma

If $x^2 \equiv y^2 \bmod n$ and $x \not\equiv \pm y \bmod n$ then n is composite. Moreover, $\gcd(n, x+y)$ and $\gcd(n, x-y)$ are non-trivial factors of n.

- Here is the Miller-Rabin algorithm for determining primality probabilistically:
- Suppose n is odd and greater than 9. Write $n - 1 = 2^k m$ where m is odd.
- Now pick $a < n$ randomly and compute a series b_i for $i = 0, \ldots, k-1$ as follows:

$$b_0 \equiv a^m \bmod n, b_1 \equiv b_0^2 \bmod n, \ldots, b_i \equiv b_{i-1}^2 \bmod n, \ldots$$

- We will guess that n is prime if $b_0 \equiv \pm 1 \bmod n$ or if $b_i \equiv -1 \bmod n$ for any i. Otherwise we will say that n is composite.

Primality testing, Miller-Rabin, cont'd

- Claim: If we say n is composite we will be correct.
- There is at most a 25% chance that if we say n is prime then we will be wrong.
- If we repeat this test many times for randomly chosen a and we always get the answer "prime" then with high probability n is prime.

Primality testing, Agarwal-Kayal-Saxena

- There is a primality test which is completely deterministic and polynomial in the number of digits of the given number.
- The problem is that its known run-time is order n^6 although 6 is not known to be best possible.
- In practice, if one needs to determine primality for a given number, one uses the probabilistic algorithms to see if you can prove it is not prime and then tries a series of special purpose primality tests which are not efficient but are good enough for ”small” numbers, say ones with fewer than 1000 digits.

Cracking RSA

- How hard is it to crack the RSA encryption scheme?
- Of course if you can factor pq then you can decipher messages so the question becomes how hard is that or put another way, what do you have to avoid in order not to let the enemy have too easy a factoring problem?
- First of all, let's make it clear that if you knew pq and $\phi(pq)$ then you would know p and q. This matters because if you know $\phi(pq)$ then you know how to invert the encoding exponent which means you can decipher anything.

Cracking RSA, cont'd

- Also a problem arises if the message is too short: Suppose that (n, e) is an RSA pair, c is a ciphertext (i.e. of the form m^e for some m) and we can find a numbers x such that $c \equiv x^e \bmod n$. Then $m \equiv x$.
- How can this be turned into an attack? We could search all x less than some number looking for the type of match on the previous line.
- We could also do the same running through pairs of numbers x and y less than some number. If m was really the product of two small numbers then we would find it essentially by brute force. Depending on how much compute power you have, you could do this for products of many "small" numbers.
- One often avoids this type of problem by making sure that whatever message is sent, you fill it out will dummy entries in order to defeat this short ciphertext attack.