Factoring - the ancients

- The original method for factoring n was just to try all primes up to $\sqrt{n}$. This worked well before the computer era.
- One factoring trick that has survived from before modern times was known to Fermat; it is based on the fact that if $n = pq$ and $p > q$ then

$$n = \frac{(p+q)^2}{4} - \frac{(p-q)^2}{4}.$$

- If you want to factor n, consider the sequence $n+1, n+4, \ldots, n+k^2, \ldots$ and look for a square. If you find one then you can factor n.
- If $n = pq$ then this will succeed in at most $(p-q)/2$ steps.
- The take-away for RSA then is not to make your primes be too similar in size.

Factoring - Pollard: the ρ algorithm

- Suppose that n has a prime factor p. If we pick numbers $0 < b_1, b_2, b_3, \ldots < n$ then eventually we will have for some $i < j$, $b_i \equiv b_j \mod p$ and we will get that p divides $\gcd(b_i - b_j, n)$.
- In this way, if n is not prime, we can find a prime factor. If p is not too big, it isn't hard to find it at least probabilistically.
- This is the birthday problem: what is the probability that k numbers, chosen less than n are not congruent mod p?

$$(1 - 1/p)(1 - 2/p) \cdots (1 - (k-1)/p) \approx e^{-k^2/2p}.$$

- If $k \approx 4\sqrt{p}$ then this probability is $< .0004$.

Factoring - Pollard, cont'd

- This assumes that our choices of the sequence of b_i's is drawn from a uniform distribution on the integers from 1 to n.
- In practice, this is hard to achieve in a computationally simple way.
- What is done is the following: start with some integer x_0 less than n. This can be randomly chosen but is often just set to 2.
- Choose some "generic" polynomial f and compute $x_{i+1} \equiv f(x_i) \mod n$ recursively.
- In practice, a sufficiently generic polynomial is $x^2 + 1$. The distribution of values modulo n is pseudo-random and is good enough for what we are doing.

Factoring - Pollard, example

- $x_0 = 2$ and $n = 1079$.
- $x_1 = x_0^2 + 1 \mod 1079$, $x_2 = x_1^2 + 1 \mod 1079$, etc.
- So the sequence is $2, 5, 26, 677, 833 \dots$

Δ	2	5	26	677
2				
5	3			
26	24	21		
677	675	672	651	
833	831	828	807	156

- $\gcd(1079, 156) = 13$ and $1079 = 13 \times 83$.

- We are trying to factor n. Choose some integer a such that $1 < a < n-1$ and a bound B. Do the following computation:
- $b_1 \equiv a \mod n$, $b_2 \equiv b_1^2 \mod n$, ..., $b_j \equiv b_{j-1}^j \mod n$, ... $b_B \equiv b_{B-1}^B \mod n$.
- Notice that $b_i \equiv a^{i!} \mod n$ for all i.
- Now if p divides n and $p-1$ divides $B!$ then by Fermat's little theorem $a^{p-1} \equiv 1 \mod p$ and hence $b_B \equiv 1 \mod p$.
- This means that p divides $b_B - 1$ and so the gcd of $b_B - 1$ and n is a factor of n.

Factoring - Pollard: the $p-1$ algorithm, cont'd

- Since in practice you don't know what p is, you try to set a bound B so that there is a high chance that $p-1$ divides $B!$. This will happen if $p-1$ has small factors.
- At each stage, you compute $b_i \equiv b_{i-1}^i \mod n$ and then determine $\gcd(b_i - 1, n)$ and see if it is not 1.
- Example: $n = 295927$, $a = 2$ and set the bound at 10 (no more than 10 steps). In this case, $n = 541 \times 547$ and $540 = 2^2 3^3 5$ so in fact 9! will work. Try it yourself!
- To thwart this attack, you need to guarantee to when you pick p, $p-1$ has at least one large prime factor. One way to do this is to only look for primes of the form $kp+1$ where k is allowed to vary and p is some fixed large prime.

Test 1 results

- Average: 16.1
- $\geq$ 20: 26
- 17 - 19: 22
- 14 - 16: 18
- $\leq$ 13: 20