Lecture 13 Pollard's $p-1$ method. §3.5.

We are still studying RSA.

Last time: to create private keys need to generate large primes.

Today: to break RSA, try to factor $N=pq$.

(Evidence suggests that breaking RSA is easier than factoring but it is still unclear.)

Algorithms to factor $N=pq$

1. Try dividing prime by prime
2. Pollard's $p-1$ method
3. Difference of squares method.

① Has time complexity $\mathcal{O}(\sqrt{N}) = \mathcal{O}(2^{k/2})$ is exponential.

② Efficient if $p-1$ or $q-1$ factors into small primes (e.g. if $p-1$ or $q-1$ is smooth).

So just as we want to avoid smooth $p-1$ in Diffie-Hellman b/c of Pohlig-Hellman, want to avoid smooth primes in RSA b/c of Pollard's $p-1$ method.

Idea If we can find L s.t. $(p-1) \mid L$ & $(q-1) \nmid L$, then $a^L \equiv 1 \bmod p$ $\forall a \in (\mathbb{Z}/N)^\times$ but most likely $a^L \not\equiv 1 \bmod q$.

So $a^L - 1$ is divisible by p & not by q.

So $\gcd(a^L - 1, N) = p$. & we factor N.

Q: How to find such an L?

If $(p-1)$ has small prime divisors (if it is smooth)

then $(p-1) \mid n!$ for some small-ish n.

So try $L = n!$ as a candidate.

Algorithm

Set $a = 2$ or some other elt of $(\mathbb{Z}/N)^\times$.

Compute $\gcd(a^{n!} - 1, N)$ for $n = 1, 2, \ldots$

until gcd $\neq 1$.

Note $a^{n!}$ is itself unreasonably large but we only need $a^{n!} \bmod N = (a^{n-1})^n \bmod N$.

Hopefully when gcd $\neq 1$ it is one of the prime factors of N.

Ex $N = 611$, $a = 2$.

$n = 1$: $2^{1!} = 2$ $\quad \gcd(2-1, 611) = 1$

$n = 2$: $2^{2!} = 4$ $\quad \gcd(4-1, 611) = 1$

$n = 3$: $2^{3!} = 64$ $\quad \gcd(64-1, 611) = 1$

$n = 4$: $2^{4!} = (64)^4 = 378$ $\quad \gcd(377, 611)$

$= \gcd(377, 234)$

$= \gcd(234, 143)$

$= \gcd(143, 91)$

$= \gcd(91, 52)$

$= \gcd(52, 39)$

$= \gcd(39, 13) = 13.$

So 13 must be a factor of 611! $611 = 13 \cdot 47$.

13 was a bad choice to use as part of an RSA public key bc $13-1$ is smooth

47 is much better bc $46 = 2 \cdot 23$ is not smooth.

Time complexity

$a^{n!}$ is computed in $O(\log_2 n!) = O(n \log_2(n))$ time. If $n \approx \log N$ then time complexity is $O(\log N \cdot \log\log N)$ – polynomial time.

So need $p-1$, $q-1$ to each have largest prime factor $\gg \log N$.

Next time Difference of squares method.