**Primality testing**

Recall $\mathbb{N} = \{0, 1, 2, 3, \ldots\}$ and $\mathbb{N}^+ = \{1, 2, 3, \ldots\}$.

**Definition**

$p \in \mathbb{N}^+$ is a prime $\iff$ the only numbers dividing $p$ are 1 and $p$.

**E.g.**: 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, \ldots

**Theorem (M. Agrawal, N. Kayal, and N. Saxena, 2002)**

There is a deterministic, $O(n^{12})$-time algorithm for primality testing, where $n$ = the number of bits of the number. (Later improved to $O(n^6)$-time.)

**HOWEVER**

In practice, even the improved AKS algorithm is slower than probabilistic methods.

**Towards a practical primality test, Part 1**

**Theorem (Fermat’s Little Lemma)**

Suppose $p$ is prime. Then $a^{p-1} \equiv 1 \pmod{p}$ for each $a \in \{1, \ldots, p-1\}$

- For the proof see the text.
- To run some computational experiments, see [fermat](http://www.cis.syr.edu/~royer/cis675/Code/rsa.hs)

Let $\mathbb{Z}_N^* = \{ a \mid 1 \leq a < N \ & \ gcd(a, N) = 1 \}$.

**Corollary (Fermat’s Test)**

If $a \in \mathbb{Z}_N^*$ and $a^{N-1} \neq 1 \pmod{N}$, then $N$ is not a prime.

**Lemma**

If one $a \in \mathbb{Z}_N^*$ fails Fermat’s Test (i.e., $a^{N-1} \neq 1 \pmod{N}$), then at least 50% of elements of $\mathbb{Z}_N^*$ also fail Fermat’s Test.

**Proof on next page.**

**Towards a practical primality test, Part 2**

Let $\mathbb{Z}_N^* = \{ a \mid 1 \leq a < N \ & \ gcd(a, N) = 1 \}$.

**Lemma**

If one $a \in \mathbb{Z}_N^*$ fails Fermat’s Test (i.e., $a^{N-1} \neq 1 \pmod{N}$), then at least 50% of elements of $\mathbb{Z}_N^*$ also fail Fermat’s Test.

**Proof.**

Let $\text{Pass} = \{ a \in \mathbb{Z}_N^* \mid a^{N-1} \equiv 1 \pmod{N} \}$.

Let $\text{Fail} = \{ b \in \mathbb{Z}_N^* \mid b^{N-1} \neq 1 \pmod{N} \}$ and $b_0 \in \text{Fail}$.

**Claim 1:** If $a \in \text{Pass}$, then $b_0 \cdot a \in \text{Fail}$.

**Claim 2:** Suppose $a_1, a_2 \in \text{Pass}$. Then $b_0 \cdot a_1 \equiv b_0 \cdot a_2 \pmod{N} \iff a_1 = a_2$.

**Claim 3:** $\text{Card}(\text{Pass}) \leq \text{Card}(\text{Fail})$. \qed
Towards a practical primality test, Part 3

An attempt at a probabilistic primality test

```plaintext
function primality2(N, k)
    // Input: N ∈ N⁺  Output: Yes/No  Runtime: O(k · (log₂ N)³) time.
    for i = 1, . . . , k do
        Pick aᵢ ∈ {1, . . . , N − 1} at random.
        If gcd(aᵢ, N) ≠ 1 or aᵢ^{N−1} ≠ 1 (mod N) then return No.
    end-for
    return Yes.  // a₁, . . . , aₖ all passed the Fermat test.
end-function
```

- If primality2(N, k) = No, then N is not prime. . . , No false negatives.
- If N is prime or a Carmichael Number, then primality2(N, k) = Yes.
- Recall: If one a ∈ ℤₙ* fails, at least 1/2 of these a’s fail.
- Suppose N is not prime or a C.N. Then: Prob[aᵢ passes the test] ≤ 1/2 and Prob[a₁, . . . , aₖ all pass] ≤ 1/2ᵏ.
- If primality2(N, k) = Yes, then Prob[N is prime or a C.N.] ≥ 1 − 1/2ᵏ.
  \(1/2^{100} ≈ 0.000000000000000000000000000008\)

Towards a practical primality test, Part 4

- The Rabin-Miller Test is a patch of the Fermat test so that:
  - RabinMiller(N, k) = No  \(⇒\) N is not prime
  - RabinMiller(N, k) = Yes  \(⇒\) Prob[N is prime] ≥ 1 − \(1/4\)ᵏ

Generating Random Primes

Theorem (Lagrange’s Prime Number Theorem)

\[\lim_{N \to \infty} \frac{\pi(N)}{N \ln N} = 1.\]

```
function findPrime(n, k)  // Find a random n-bit prime, maybe
    loop until exit
        Pick a random n-bit number N
        if Rabin-Miller(N, k) then return N
    end-loop
end-function
```

- Each loop iteration has a \(\frac{1}{n}\) chance of finding a prime (& halting).
- On average, the loop halts after \(O(N)\) iterations. (Exercise 1.34)
- So what are we going to do with these random primes?
- E-commerce!

ON to RSA

Symmetric Cryptosystems

A and B have a shared secret — the value of the key
\[A = \text{amazon.com} \quad B = \text{book buyer}\]
\[A \quad p \xrightarrow{key} \quad c \quad \xrightarrow{key} \quad p \quad B\]

Asymmetric Cryptosystems

A and B have no shared private info —
but the want to communicate securely.

Public key Cryptosystems

Break the key into two parts: \(k_{pub}\) and \(k_{prv}\).
Each party has her/his key pair
PKC: the setup

Alice
- $E_A$: Alice’s public encryption key
- $D_A$: Alice’s private decryption key

Bob
- $E_B$: Bob’s public encryption key
- $D_B$: Bob’s private decryption key

Encryption of a message to user $U$
\[ \text{encrypt}(E_U, p) = c \quad (\text{Everyone knows } E_U) \]

Decryption of a message to user $U$
\[ \text{decrypt}(D_U, c) = p \quad (\text{Only user } U \text{ knows } D_U) \]

PKC: The exchange phase

Alice wants to send $m$ to Bob

Alice
- Looks up $E_B$.
- Computes $c = \text{encrypt}(E_B, m)$.
- Sends $c$ over an open channel.

Bob
- Receives $c$.
- Computes $m = \text{decrypt}(D_B, c)$.

Eves
- Captures $c$.
- Now what?

We want to make it really hard for Eves to recover $m$.

RSA: Background Facts

Suppose $p$ and $q$ are two (distinct) primes and $N = p \cdot q$.

Definition: $\phi(n) = \text{Card}\{m \in \mathbb{Z}_n^* : \gcd(m, n) = 1\}$.
For a prime $r$, $\phi(r) = r - 1$. For $N$ as above, $\phi(n) = (p - 1)(q - 1)$.

Theorem (Euler)
For each $a$, $M > 0$, $a^{\phi(M)} \equiv 1 \pmod{M}$. (An extension of F’s Little Lemma.)

Property
Suppose $p, q, N$ are as above and $e \cdot d \equiv 1 \pmod{\phi(N)}$. Then:
- $x \mapsto x^d \pmod{N}$ is a bijection on $\{0, 1, \ldots, N - 1\}$.
- $(x^e)^d \equiv x \pmod{N}$.

Proof of 2 for the case of gcd$(x, N) = 1$.
\[ e \cdot d \equiv 1 \pmod{\phi(N)} \iff e \cdot d = 1 + k \cdot \phi(N) \text{ for some } k. \]
So $(x^e)^d = x^{e \cdot d} = x^{1 + k \cdot \phi(N)} = x \cdot (x^{\phi(N)})^k \equiv x \cdot (1)^k \equiv x \pmod{N}$. ■

The RSA Algorithm: Setup

Alice:
- Picks two large primes $p$ & $q$.
- Computes $n = p \cdot q$ and $\phi(n) = (p - 1) \cdot (q - 1)$.
- Picks $e \overset{\text{ran}}{\in} \{1, \ldots, \phi(n) - 1\}$ with gcd$(e, \phi(n)) = 1$.
- Computes $d \in \{1, \ldots, \phi(n) - 1\}$ with $d \cdot e \equiv 1 \pmod{\phi(n)}$. (How?)
- Publishes $e$ and $n$.
  She keeps $d, p, q,$ and $\phi(n)$ secret.

Bob:
Bob does the same thing.
The RSA Algorithm: The protocol

Bob wants to send Alice a message.

**Bob:**
- Converts a message to a number $m$. (*Assume $0 \leq m < n$.*)
- Computes $c = m^e \mod n$.
- Sends $c$ to Alice.

**Alice:**
- Receives $c$.
- Computes $m' = c^d \mod n$.
- Converts $m'$ to a text message. // $m = m'$ by Property 2.

**Eves:**
- Knows $N$, $e$, and $c$. But how to find any of $m$, $d$, $p$ or $q$?

**Fact:** If factoring $N$ is hard, so is Eves’ problems.