Public-Key Crypto

Basics

Paul Garrett

garrett@math.umn.edu

University of Minnesota

Terminology

A **symmetric** or **private-key** cipher is one in which knowledge of the *encryption* key is explicitly or implicitly equivalent to knowing the *decryption* key.

A **asymmetric** or **public-key** cipher is one in which the *encryption* key is effectively public knowledge, without giving any useful information about the *decryption* key.

Until 30 years ago all ciphers were private-key.

The very possibility of public-key crypto did not exist until the secret work of some British CESG-at-GCHQ people Ellis-Cocks-Williamson in the 1960's, and public-domain work of Merkle, Diffie-Hellman, and Rivest-Shamir-Adleman in the 1970's.

Examples of symmetric/private-key ciphers

Cryptograms (substitution ciphers) [*broken: letter frequencies, small words*]

Anagrams (permutation ciphers) [*broken: double anagramming*]

Vigenère [*broken: Kasiski attack, Friedman attack*]

Enigma, Purple [*broken: key distribution problems, too small keyspace*]

DES [*broken: too small keyspace*]

3DES [*slow*]

Blowfish [*in use*], Arcfour [*in use*], TEA, IDEA

Serpent, Twofish, RC6, MARS [*AES finalists*]

AES (Rijndael)

Examples of asymmetric/public-key ciphers

RSA (Rivest-Shamir-Adlemen)

ElGamal

Elliptic curve cipher
(∼ abstracted ElGamal)

Knapsack ciphers [*discredited*]

Coding-theory ciphers [*out of fashion...*]
NTRU

Arithmetica (word-problem ciphers)

Basic public-key examples and background

RSA cipher

Diffie-Hellman key exchange

classical number-theoretic algorithms:
- Euclidean algorithm
- fast exponentiation

factorization algorithms:
- futility of trial division
- Pollard's rho
- quadratic sieve
- elliptic curve sieve
- number field sieve

primality-testing algorithms:
- Fermat
- Miller-Rabin

primality certificates

further protocols

Feasible and infeasible computations

The only cipher provably impervious to *every* hostile computation is the **One-Time Pad** (sometimes called *Vernam cipher*).

Otherwise, in both private-key and public-key ciphers the *design goal* is that authorized encryptor and decryptor should have **feasible** computations, while unauthorized decryptors have **infeasible** computations.

Feasibility here is meant in the most practical possible sense.

Traditional complexity theory is not necessarily relevant.

The **P**=**NP**? issue may *not* be relevant.

Ever-increasing **CPU speed** *is* relevant.

Parallelizability *is* relevant.

Quantum algorithms *are* relevant

(potentially).

Archetypical trap-door

It is easy to *multiply* large integers
(from $\sim 10^{100}$ to $\sim 10^{1000}$ and larger)

Thus, it is easy to **verify** that $n = p \cdot q$ for
given large primes p, q and n their product.

But it *appears to be difficult* to **factor**
$n \sim 10^{200}$ into primes.

There is no *proof* that factorization is
difficult.

There are very few lower-bound results for the complexity of large-integer computations. There are exceptions, e.g., some recent work of Shparlinski.

Others have taken provable security as the primary criterion for cipher development, e.g., Goldreich. The practical impact of this approach is not yet clear.

Mandatory review of RSA

One-time preparation: *Alice* chooses two large random primes p, q, from 10^{100} to 10^{600} depending on the desired security. She computes the **RSA modulus** $n = p \cdot q$. She chooses decryption exponent e (often $e = 3$), and computes the multiplicative inverse d of e mod $(p-1)(q-1)$. She publishes n, e (on her web page?) and keeps d secret. Primes p and q are thrown away.

Encryption: Bob wishes to encrypt a *plaintext* message x and send it to Alice on an insecure channel. Suppose $1 < x < n$ for simplicity. Bob computes and transmits $y = x^e \bmod n$.

Decryption: When Alice receives the ciphertext y, she computes $y^d \bmod n$, which *is* the plaintext x.

Why is this ok?

Why is it feasible for Alice to find two primes $\sim 10^{200}$ or so?

Why is it feasible for Alice to compute $e^{-1} \bmod (p-1)(q-1)$?

Why is it feasible for Bob to compute $x^e \bmod n$?

Why is it feasible for Alice to compute $y^d \bmod n$?

Why is $y^d \bmod n = x$?

Why is it *not* feasible for Eve (the eavesdropper) to compute d from n and e?

Why is it *not* feasible for Eve to compute x from $x^e \bmod n$?

How do we get a good supply of *random numbers*?

Minor qualifications about RSA

Want p and q congruent to 3 mod 4.

In fact, maybe want p and q to be **strong primes**, namely so that $p-1$ and $q-1$ are not exclusively composed of small prime factors.

Want to be sure that e is relatively prime to $(p-1)(q-1)$: if we want $e = 3$ or some other pre-specified number, must tweak p and q. Otherwise, tweak e.

Very unlikely that $\gcd(x, n) > 1$, so ignore this.

Need **good-quality randomization** for choice of p and q. Else potential for *catastrophic failure*. (Related recent examples in software implementations of various security protocols.)

Diffie-Hellman Key Exchange

Alice and Bob have never met, and can only communicate across an insecure channel on which Eve is eavesdropping.

Eve has considerably greater computational power than Alice and Bob, and hears everything they say to each other.

Yet Alice and Bob can establish a **shared secret** which Eve cannot also acquire (assuming the difficulty of computing discrete logs).

The shared secret is then typically used as a *key* for a symmetric/private-key cipher to encrypt a subsequent conversation.

Alice and Bob agree on a large random prime p ($\sim 10^{130}$ or larger) and a random base g in the range $1 < g < p$. Alice secretly chooses a random a in the range $1 < a < p$ and computes $A = g^a \bmod p$. Similarly, Bob secretly chooses a random b in the range $1 < b < p$ and computes $B = g^b \bmod p$. Alice sends A over the channel, and Bob sends B over the channel.

So Alice knows p, g, a, A, B, Bob knows p, g, A, b, B, and Eve knows p, g, A, B.

Alice computes

$$K_A = B^a \bmod p$$

and Bob computes

$$K_B = A^b \bmod p$$

Since

$$K_A = K_B \bmod p$$

Alice and Bob now have a shared secret which it is *infeasible* for Eve to obtain.

Ingredient:
Fast exponentiation algorithm

To compute $b^n \bmod n$, with $n \sim 10^{100}$ or larger, do **not** multiply 10^{100} times.

Rather, note that **repeated squaring** reduces the number of operations:

$$b^{69} = b^{2^6+2^2+2^0} = (((((b^2)^2)^2)^2)^2)^2 \cdot (b^2)^2 \cdot b$$

To compute $x^e \bmod n$

initialize $(X, E, Y) = (x, e, 1)$
while $E > 0$
    if E is even
        replace X by $X^2 \bmod n$
        replace E by $E/2$
    elsif E is odd
        replace Y by $X \cdot Y \bmod n$
        replace E by $E - 1$

The final value of Y is $x^e \bmod n$.

Ingredient: Euclidean Algorithm

To compute *gcd*'s, and to compute $e^{-1} \bmod m$, use the familiar Euclidean algorithm. To compute $\gcd(x, y)$ takes at most $2 \log_2 y$ steps, if $x \geq y$.

To compute $\gcd(x, y)$:

Initialize $X = x$, $Y = y$, $R = X \bmod Y$
while $R > 0$
replace X by Y
replace Y by R
replace R by $X \bmod Y$
When $R = 0$, $Y = \gcd(x, y)$

This gives the familiar pattern: for example to compute $\gcd(1477, 721)$:

$1477 - 2 \cdot 721 = 35$
$721 - 20 \cdot 35 = 21$
$35 - 1 \cdot 21 = 14$
$21 - 1 \cdot 14 = 7$
$14 - 2 \cdot 7 = 0$

And 7 is the *gcd*.

Multiplicative inverses via Euclid

To compute $e^{-1} \bmod x$ with $\gcd(e, x) = 1$, minimizing memory use, rewrite each of the steps in the previous as

$$\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} 1 & -q \\ 0 & 1 \end{pmatrix} \begin{pmatrix} X \\ Y \end{pmatrix} = \begin{pmatrix} \text{new } X \\ \text{new } Y \end{pmatrix}$$

where $R = X - qY$ with $|R| < Y$.

Thus, we obtain an integral matrix $\begin{pmatrix} a & b \\ c & d \end{pmatrix}$ with determinant ± 1 such that

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix} \begin{pmatrix} x \\ e \end{pmatrix} = \begin{pmatrix} \gcd(x, e) \\ 0 \end{pmatrix}$$

When $\gcd(x, e) = 1$, we have

$$ax + be = 1$$

and thus

$$b = e^{-1} \bmod x$$

Ingredient: Euler's Theorem

Let $\varphi(n)$ be Euler's *totient function*, which counts the integers ℓ in the range $1 \leq \ell \leq n$ which are relatively prime to n.

Theorem: For $\gcd(x, m)$=1, $x^{\varphi(n)}$=1 mod n.

(This is an immediate corollary of Lagrange's theorem, applied to the group $\mathbb{Z}/n^{\times}$.)

This proves that **RSA decryption works**, using $\varphi(pq) = (p-1)(q-1)$: with $y = x^e \bmod n$, letting $ed = 1 + M \cdot \varphi(n)$, all equalities modulo n,

$$y^d = (x^e)^d = x^{1+M\cdot\varphi(n)}$$

$$= x \cdot (x^{\varphi(n)})^M = x \cdot 1^M = x \bmod n$$

Ingredient:
Infeasibility of factoring n

Nothing like a proof exists to support the claim that factoring is hard, but there is much practical evidence.

Several decades of new insights into factoring have yielded very clever factorization algorithms, which are *sub-exponential*, but still *super-polynomial*.

If large quantum computers ever become feasible, Peter Shor's quantum factoring algorithm will factor large numbers very nicely, and RSA will be broken. Some of the more-sophisticated public-key systems (e.g., lattice-based ciphers such as NTRU) are not known to have fast quantum algorithms to break them. (Apparently Shor's discrete log algorithm may also apply to more abstract discrete log computations.)

With quantum computers, Grover's $\sqrt{n}$-

time quantum search of n unordered things
would require increased key size for nearly
all ciphers.

Ingredient:
Infeasibility of taking e^{th} roots $\bmod n$

Based on pragmatic experience, it appears to be roughly as difficult to compute roots modulo n as to factor n.

Computing roots via **discrete logarithms** appears to have the same degree of difficulty, despite existence of several clever algorithms to compute discrete logarithms.

If large quantum computers ever become feasible, Peter Shor's quantum discrete logarithm algorithm will compute discrete logs nicely, and the Diffie-Hellman key exchange will be broken, as will the classical ElGamal cipher will be broken.

We will focus on the issue of factorization rather than discrete logarithm computation.

Square-root oracles yield fast factorization

As evidence that square-root taking is as hard as factoring: *if we have a magic box (***oracle***) that takes square roots modulo $n = pq$, we can factor n*:

Suppose for simplicity that $n = p \cdot q$ with p and q distinct primes. Pick *random* x mod n and give the oracle $x^2 \bmod n$. The oracle returns y such that $y^2 = x^2 \bmod n$.

There are 4 square roots of $x^2 \bmod n$, namely the 4 solutions z of

$$z = \pm x \bmod p \qquad z = \pm x \bmod q$$

Since x was random, *regardless* of the oracle's choices of square roots, the probability is $1/2$ that

$$1 < \gcd(x - y, n) < n$$

Repeat as necessary...

Unauthorized computation of RSA decryption exponent?

For an eavesdropper to compute the RSA decryption exponent

$$d = e^{-1} \bmod (p-1)(q-1)$$

it would suffice (since the Euclid algorithm computes inverses quickly) to know the quantity $m = (p-1)(q-1)$.

But observe that knowledge of the RSA modulus $n = pq$ and of $m = (p-1)(q-1)$ would amount to knowing the factors p, q, since the roots of the polynomial equation

$$X^2 + (m - n + 2)X + n = 0$$

are p, q.

Presumably factoring is hard, so this is evidence (?) that an eavesdropper will not find a trick whereby to obtain $(p-1)(q-1)$*, from which to obtain the decryption exponent.*

Ingredient: big primes?

To acquire **200-digit prime numbers**, trial division would not succeed in the lifetime of the universe using all the computational power of the internet.

Trial division confirms that a number is prime by **failing to factor it**.

It turns out that **primality testing is much easier than factoring**.

Factoring big numbers is hard, despite striking (and wacky) modern factorization techniques much better than trial division.

Even more surprising are fast modern **probabilistic primality tests**.

(And, one can *construct* large primes with accompanying **certificates of primality** indicating how to reprove their primality upon demand.)

Failure of trial division:

Trial division attempts to divide a given number N by integers from 2 up through $\sqrt{N}$. Either we find a proper factor of N, or N is prime. (If N has a proper factor ℓ larger than $\sqrt{N}$, then $N/\ell \leq \sqrt{N}$.) The extreme case takes roughly $\sqrt{N}$ steps, or at least $\sqrt{N}/\ln N$.

If $N \sim 10^{200}$ is prime, or if it is the product of two primes each $\sim 10^{100}$, then it will take about 10^{100} trial divisions to discover this. Even if we're clever, it will take more than 10^{98} trial divisions.

If we could do 10^{12} trials per second, and if there were a 10^{12} hosts on the internet, with $< 10^8$ seconds per year, a massively parallel trial division would take ...

$$10^{66} \text{ years}$$

Examples of trial division

What are the practical limitations of trial division? On a 2.5 Gigahertz machine, code in C++ using GMP

1002904102901 has factor 1001401
('instantaneous')

100001220001957 has factor 10000019
(3 seconds)

10000013000000861 has factor 100000007
(27 seconds)

1000000110000000721 has factor 1000000007
(4 minutes)

Nowhere near 10^{200} *...*

Facts about primes

The number $\pi(N)$ of primes less than N is

$$\pi(N) \sim \frac{N}{\log N}$$

This is the *Prime Number Theorem*
(Hadamard and de la Vallée Poussin, 1896).

Riemann observed (1858-9) that *if* all
the complex zeros of the **zeta function**
$\zeta(s) = \sum n^{-s}$ lay on the line $\mathrm{Re}(s) = \frac{1}{2}$
then (as refined...)

$$\pi(N) = \int_2^N \frac{dt}{\log t} + O(\sqrt{N}\,\log N)$$

The conjecture on the location of the zeros
is the **Riemann Hypothesis**.

No result approaching this is known: there
is *no* known zero-free region $\mathrm{Re}(s) \geq \sigma$ for
$\sigma < 1$.

The Prime Number Theorem uses the non-
vanishing of $\zeta(s)$ on $\mathrm{Re}(s) = 1$.

Special Primes

As of January 2000, the largest prime known was the 38^{th} Mersenne prime

$$2^{6972593} - 1$$

Theorem *(Lucas-Lehmer)* Define $L_o = 4$, $L_n = L_{n-1}^2 - 2$. Let p be an odd prime. The Mersenne number $2^p - 1$ is **prime** if and only if

$$L_{p-2} = 0 \bmod 2^p - 1$$

Theorem (*Proth*) The Fermat number $F_n = 2^{2^n} + 1$ is **prime** if and only if

$$3^{(F_n-1)/2} = -1 \bmod F_n$$

Hunting for primes

Nevertheless, when developing expectations for hunting for primes, we pretend that primes are distributed as evenly as possible.

Note: it is *not true* that primes are distributed evenly, even under the Riemann Hypothesis.

But if primes *were* evenly distributed, then near x primes would be about $\ln x$ apart.

Thus, in hunting for primes near x expect to examine $\frac{1}{2}\ln x$ candidates:

For $x \sim 10^{20}$ we have $\frac{1}{2}\ln x \sim 23$

For $x \sim 10^{100}$ we have $\frac{1}{2}\ln x \sim 115$

For $x \sim 10^{500}$ we have $\frac{1}{2}\ln x \sim 575$

The Birthday Paradox [sic]

For $n+1$ things chosen (with replacement) from N the probability that they're **all different** is

$$p = (1 - \frac{1}{N})(1 - \frac{2}{N}) \cdots (1 - \frac{n}{N})$$

Then from $\ln(1-x) > -x$ for small x one has

$$\ln p > -\sum_{\ell=1}^{n} \frac{\ell}{N} \sim -\frac{\frac{1}{2}n^2}{N}$$

Thus, to be sure that $p \geq \frac{1}{2}$ it suffices to take n such that

$$n > \sqrt{N} \cdot \sqrt{2 \ln 2} \sim 1.1774 \cdot \sqrt{N}$$

Thus, with 23 people in a room the probability is greater than $\frac{1}{2}$ that two will have the same birthday.

Pollard's rho method (circa 1976)

We'll try to beat the $\sqrt{N}$ steps trial division needs to factor N.

First try: Suppose that $N = p \cdot M$ with p prime and $p < \sqrt{N}$. If we choose somewhat more than $\sqrt{p}$ integers x_i at random, then the probability is $> \frac{1}{2}$ that for some $i \neq j$ we'll have

$$x_i = x_j \bmod p$$

The probability is roughly $\frac{1}{\sqrt{N/p}} \sim 0$ that $x_i = x_j \bmod N$, so most likely for **some** pair

$$\gcd(x_i - x_j, N) = \text{proper factor of } N$$

But we might have to compare

$$\sqrt{p} \cdot \sqrt{p} = p \sim \sqrt{N}$$

pairs, no better than trial division.

(In any case, we compute *gcd*'s quickly by the **Euclidean algorithm**.)

Second try at Pollard's rho

Since we would have had trouble making a large number of truly random choices anyway, let's stipulate that we choose the x_i's in a more structure way, in a sort of **random walk** in $\mathbb{Z}/N$. Let $f : \mathbb{Z}/N \to \mathbb{Z}/N$ be a deterministic **'random' function**, fix x_o, and define

$$x_{i+1} = f(x_i)$$

Since f is deterministic

$$f(x_i) = x_j \quad \Longrightarrow$$

$$f(x_{i+1}) = f(f(x_i)) = f(x_j) = x_{j+1}$$

So if the walk enters a **cycle** it stays there.
We use **Floyd's cycle-detection trick**:

Floyd's cycle-detection trick:

Fix x_o, define $y_o = x_o$, and define

$$x_{i+1} = f(x_i) \qquad y_{i+1} = f(f(y_i))$$

so the y_i's take the same walk but twice as fast.

Once the cycle is entered, the y's walk one unit faster than the x's, so in fewer additional steps than the cycle length, $x_j = y_j \bmod p$.

In summary: the initial walk plus cycle takes $\sqrt{p} \leq N^{1/4}$ steps, and another $\sqrt{p}$ for the y's to catch the x's modulo p, so

$$2\sqrt{p} \leq 2N^{1/4} \quad \textbf{steps}$$

to find the factor p of N.

Examples of Pollard's rho factorization

Take $x_o = 2$ and $f(x) = x^2 + 2 \bmod N$ (*this is random...?*). In less than 10 seconds total,

2661 steps to find factor

10000103 of 100001220001957

14073 steps to find factor

100000007 of 10000013000000861

9630 steps to find factor

1000000103 of 10000001100000000721

(Even larger...) 129665 steps for factor

10000000019 of 100000001220000001957

162944 steps for factor

100000000103 of 1000000001060000000309

Yet larger...

89074 steps for

$$1000000000039 \text{ of } 100000000016000000004719$$

12 seconds, 584003 steps for

$$10000000000037 \text{ of } 1000000000016600000004773$$

2 minutes, 5751662 steps for

$$100000000000031 \text{ of } 10000000000016400000000004123$$

But still this is slowing down, and we're nowhere near factoring a 200-digit number...

Modern factorization methods

Since the 1970's, better methods have been found (but not polynomial-time):

quadratic sieve: the most elementary of modern factorization methods, and still very good by comparison to other methods. Descended from Dixon's algorithm.

elliptic curve sieve: to factor n, this replaces the group $\mathbb{Z}/n^\times$ with an elliptic curve E defined over $\mathbb{Z}/n$. In effect, the difference between $\mathbb{Z}/n^\times$ and $\mathbb{Z}/n - \{0\}$ is what indicates that n is composite, and an analogous discrepancy in the case of elliptic curves can be similarly exploited.

number field sieve: Descended from several sources, including Adleman, Pomerance, and Rumely (1983), which made novel use of exponential sums (hence, of irrational algebraic numbers)

Bargain-basement Primality Test: Fermat pseudoprimes

Fermat's Little Theorem asserts that for p prime, $b^p = b \bmod p$.

Proven by induction on b, using

$$(b+1)^p = b^p + \binom{p}{1} b^{p-1} + \ldots + \binom{p}{p-1} b + 1$$

$$= b^p + 1 \bmod p$$

The binomial coefficients are *integers*, and on the other hand, they are divisible by p, since

$$\binom{p}{i} = \frac{p!}{i!\,(p-i)!}$$

and the denominator has no factor of p. (*Unique Factorization...*)

Thus, if n is an integer and $b^n \neq b \bmod n$ for some b, then n is **composite**.

The *converse* is false, but not *very* false...

The only *non-prime* $n < 5000$ with
$2^n = 2 \bmod n$ are 341 561 645 1105 1387
1729 1905 2047 2465 2701 2821 3277 4033
4369 4371 4681

Requiring also $3^n = 3 \bmod n$ leaves 561
1105 1729 2465 2701 2821

Requiring also $5^n = 5 \bmod n$ leaves 561
1105 1729 2465 2821

Compared with 669 primes under 5000, this is a *false positive* failure rate of less than 1%.

n is a **Fermat pseudoprime base** b if
$b^n = b \bmod n$.

Terminology

Usage is not consistent.

My usage is that a number that has passed a primality test (Fermat, Miller-Rabin, etc.) is a **pseudoprime**.

Sometimes a *pseudoprime* is meant to be a *non-prime* which has nevertheless passed a primality test such as Fermat. But for large numbers which have passed pseudoprimality tests we may never know *for sure* whether or not they're prime or composite ...

Another usage is to call a number that has passed a test a **probable prime**.

But this is dangerously close to **provable prime**, which is sometimes used to describe primes with accompanying certificates of their primality.

There are only 172 non-prime Fermat pseudoprimes base 2 under 500,000 versus 41,538 primes, a false positive rate of less than 0.41%

There are only 49 non-prime Fermat pseudoprimes base 2 and 3 under 500,000, a false positive rate of less than 0.118%

There are only 32 non-prime Fermat pseudoprimes base 2, 3, 5 under 500,000

There are still 32 non-prime Fermat pseudoprimes base 2, 3, 5, 7, 11, 13, 17 under 500,000

561 1105 1729 2465 2821 6601 8911 10585
15841 29341 41041 46657 52633 62745 63973
75361 101101 115921 126217 162401 172081
188461 252601 278545 294409 314821
334153 340561 399001 410041 449065
488881

Adding more such requirements does not shrink these lists further.

n is a **Carmichael number** if it is a *non-prime* Fermat pseudoprime to *every* base b.

In 1994 Alford, Granville, and Pomerance showed that there are infinitely-many Carmichael numbers.

And it appears that among *large* numbers Carmichael numbers become more common.

Nevertheless, the Fermat test is a very fast way to test for *compositeness*, and is so easy and cheap that it is still the best first approximation to primality.

It is **cheap** because $b^n \bmod n$ can be computed in $\sim \log n$ steps, not n...

Better primality test: Miller-Rabin (1978)

If $n = r \cdot s$ is composite (with $\gcd(r, s) = 1$) then by Sun-Ze's theorem there are at least 4 solutions to

$$x^2 = 1 \bmod n$$

namely the 4 choices of sign in

$$x = \pm 1 \bmod r \quad x = \pm 1 \bmod s$$

Thus, if we find $b \neq \pm 1 \bmod n$ such that $b^2 = 1 \bmod n$, n is definitely *not* composite.

Roughly, the **Miller-Rabin test** looks for such extra square roots of 1 modulo n (details below).

[0.0.1] Theorem: (Miller-Rabin) For composite n, at least $3/4$ of b in the range $1 < b < n$ will detect the compositeness (via the Miller-Rabin test)

Pseudo-corollary If n passes the Miller-Rabin test with k random bases b, then (*exercise: explain the fallacy*)

$$\text{probability}(n \text{ is prime}) \geq 1 - \left(\frac{1}{4}\right)^k$$

Miller-Rabin test base b:
factor $n - 1 = 2^s \cdot m$ with m odd
replace b by $b^m \bmod n$
    if $b = \pm 1 \bmod n$ **stop:** n is 3/4 prime
    else continue
set $r = 0$
while $r < s$
    replace b by $b^2 \bmod n$
    if $b = -1 \bmod n$ **stop:** n is 3/4 prime
    elsif $b = +1 \bmod n$ **stop:** n is composite
    else continue
    replace r by $r + 1$
if we fall out of the loop, n is composite.

If n passes this test it is a
strong pseudoprime base b.

Failure rate of Miller-Rabin?

The fraction of b's which detect compositeness is apparently much greater than 3/4. For $n = 21311$ the detection rate is 0.9976. For 64777 the detection rate is 0.99972. For 1112927 the detection rate is 0.9999973

For $n < 50,000$ there are only 9 non-prime strong pseudoprimes base 2, namely 2047 3277 4033 4681 8321 15841 29341 42799 49141

For $n < 500,000$ there are only 33 non-prime strong pseudoprimes base 2.

For $n < 500,000$ there are *no* non-prime strong pseudoprimes base 2 and 3

For $100,000,000 < n < 101,000,000$ there are 3 strong pseudoprimes base 2 whose compositeness is detected base 3, namely 100463443 100618933 100943201

Some big strong pseudoprimes
Primality testing Fermat base 2, Miller-Rabin base 2, 3, 5, to find next prime after...

('instantaneous')
First prime after 10^{21} is $10^{21} + 117$

('instantaneous')
First prime after 10^{50} is $10^{50} + 151$

('hint of time taken')
First prime after 10^{100} is $10^{100} + 267$

(3 seconds)
First prime after 10^{200} is $10^{200} + 357$

(8 seconds)
First prime after 10^{300} is $10^{300} + 331$

(97 seconds)
First prime after 10^{1000} is $10^{1000} + 453$

Primality Certificates

With origins in work of Eduard Lucas in 1876 and 1891, a very simple form of the **Pocklington-Lehmer theorem** asserts that N is prime if we have

a factorization $N - 1 = p \cdot U$
where p is prime
where $p > \sqrt{N}$
b with $b^{N-1} = 1 \bmod N$
but $\gcd(b^U - 1, N) = 1$

The factorization $N - 1 = p \cdot U$ and the b is the simplest instance of a **certificate of primality** for N.

This requires recursive certification of the prime p.

(The Lucas-Lehmer and Proth criteria are cousins of this idea.)

Lemma *(Fermat, Euler)* For a positive integer N, let b be such that $b^{N-1}=1 \bmod N$ but $\gcd(b^{(N-1)/p}-1, N)=1$. Then a prime divisor q of N satisfies $q=1 \bmod p$

Proof of lemma: As $b \cdot b^{N-2} = 1 \bmod N$ it must be that b is prime to N, so b is prime to q. Let t be the order of b in $\mathbb{Z}/q^{\times}$. By Fermat's Little Theorem $b^{q-1} = 1 \bmod q$, so $t|q-1$. But the *gcd* condition implies that

$$b^{(N-1)/p} \neq 1 \bmod q$$

Thus, t does not divide $(N-1)/p$. Yet, $t|N-1$. Thus, $p|t$. From $t|q-1$ and $p|t$ we get $p|q-1$, or $q = 1 \bmod p$. ///

Proof of theorem

(Note that if N is prime then $\mathbb{Z}/N$ has a primitive root b which fulfills the condition of the theorem.)

If the conditions of the theorem are met, then all divisors of N are 1 modulo p. If N were not prime, it would have a prime divisor q in the range $1 < q \leq \sqrt{N}$. But $q = 1 \bmod p$ and $p > \sqrt{N}$ make this impossible. Thus, N is prime. ///

Example

By trial division, $p = 1000003$ is prime.

The first strong pseudoprime above $1000 \cdot p$ of the form $p \cdot U + 1$ is

$$N = 1032003097 = 1032 \cdot p + 1$$

By luck, with $b = 2$

$$2^{N-1} = 1 \bmod N$$

while

$$\gcd(2^{(N-1)/p} - 1, N) = \gcd(2^{1032} - 1, N) = 1$$

Therefore, N is *certified* prime.

Continued Example

Let p be the certified prime 1032003097.

The first strong pseudoprime above $10^9 \cdot p$ of the form $p \cdot U + 1$ is
$N = 10320032476724521 63$ which is

$$N = p \cdot (10^9 + 146) + 1$$

By luck, with $b = 2$

$$2^{N-1} = 1 \bmod N$$

while

$$\gcd(2^{(N-1)/p} - 1, N) = 1$$

Therefore, N is *certified* prime.

Continued

Let p be the certified prime
1032003247672452163

The first strong pseudoprime N above
$10^{17} \cdot p$ of the form $p \cdot U + 1$ is

$$p \cdot (10^{17} + 24) + 1$$

$$= 1032003247672452410680779441388519 13$$

By luck, with $b = 2$

$$2^{N-1} = 1 \bmod N$$

while

$$\gcd(2^{(N-1)/p} - 1, N) = 1$$

Therefore, N is *certified* prime.

Continued

Let p be the certified prime
1032003247672452410680779441388519 13

The first strong pseudoprime N above
$10^{34} \cdot p$ of the form $p \cdot U + 1$ is

$$p \cdot (10^{34} + 224) + 1$$

$$\begin{aligned} = \quad & 1032003247672452410680779441388 \\ & 5422468727478629339992494594 8 \\ & 7102828513 \end{aligned}$$

By luck, with $b = 2$

$$2^{N-1} = 1 \bmod N$$

while

$$\gcd(2^{(N-1)/p} - 1, N) = 1$$

Therefore, N is *certified* prime.

Continued

Let p be the certified prime

$$\begin{aligned}&10320032476724524106807794413885422\\&46872747862933999249459487102828513\end{aligned}$$

The first strong pseudoprime N above $10^{60} \cdot p$ of the form $p \cdot U + 1$ is (computing for about 5 seconds)

$$\begin{aligned}p \cdot (10^{60} + 1362) + 1 &= 10320032476724524106807794413\\&88542246872747862933999249460\\&89269125184288018334722159917\\&11945402406825893161069777638\\&21434052434707\end{aligned}$$

By luck, $b = 2$ works again and N is *certified* prime.

Continued

Let p be the certified prime

$$10320032476724524106807794413$$
$$88542246872747862933999249460$$
$$89269125184288018334722159917$$
$$11945402406825893161069777638$$
$$21434052434707$$

The first strong pseudoprime N above $10^{120} \cdot p$ of the form $p \cdot U + 1$ is (computing a few seconds)

$$p \cdot (10^{120} + 796) + 1 =$$

$$10320032476724524106807794413885422$$
$$46872747862933999249460892691251$$
$$84288018334722159917119454024068258$$
$$93161069777638222555270198542721189$$
$$01900435345279628510707298895463402$$
$$57087058223646693262594438839294027$$
$$08540315833410956211543000018615057$$
$$38026773$$

$b = 2$ works again and N is *certified* prime.

Fast deterministic test for primality

In 2002, Agarwal, Kayal, and Saxena announced a fast (i.e., polynomial time) **deterministic** algorithm for primality testing.

Their algorithm has been checked by a number of experts, including Pomerance.

Still, their algorithm is much slower than the probabilistic Miller-Rabin test.

And there has been recent progress in fast deterministic construction of *random certifiable* primes by Peter Smith, improving Maurer's probabilistic method, and approaching the speed of Miller-Rabin.

Basic Reading List

History, Context

David Kahn *The Codebreakers*

Bruce Schneier *Secrets and Lies*

Technical

Bruce Schneier *Applied Cryptography*

Menezes, van Oorshot, Vanstone
Handbook of Applied Cryptography

D. Stinson
Cryptography: Theory and Practice

Bach, Shallit *Algorithmic Number Theory*

Introductory

N. Koblitz *A Course in Number Theory and Cryptography*

A. Salomaa *Public-Key Crypto*

P. Garrett *Making, Breaking Codes: an Introduction to Cryptology*