Cryptanalysis

Lecture 6: Introduction to Public Key Systems

John Manferdelli
jmanfer@microsoft.com
JohnManferdelli@hotmail.com

Public Key (Asymmetric) Cryptosystems

- An asymmetric cipher is a pair of key dependant maps, (E(PK,-),D(pK,-)), based on related keys (PK, pK).
- D(pK,(E(PK,x))=x, for all x.
- PK is called the public key. pK is called the private key.
- Given PK it is infeasible to compute pK and infeasible to compute x given y=E(PK,x).

Idea from Diffie, Hellman, Ellis, Cocks, Williamson. Diffie and Hellman, "New Directions in Cryptography“, IEEE Trans on IT 11/1976. CESG work in 1/70-74.

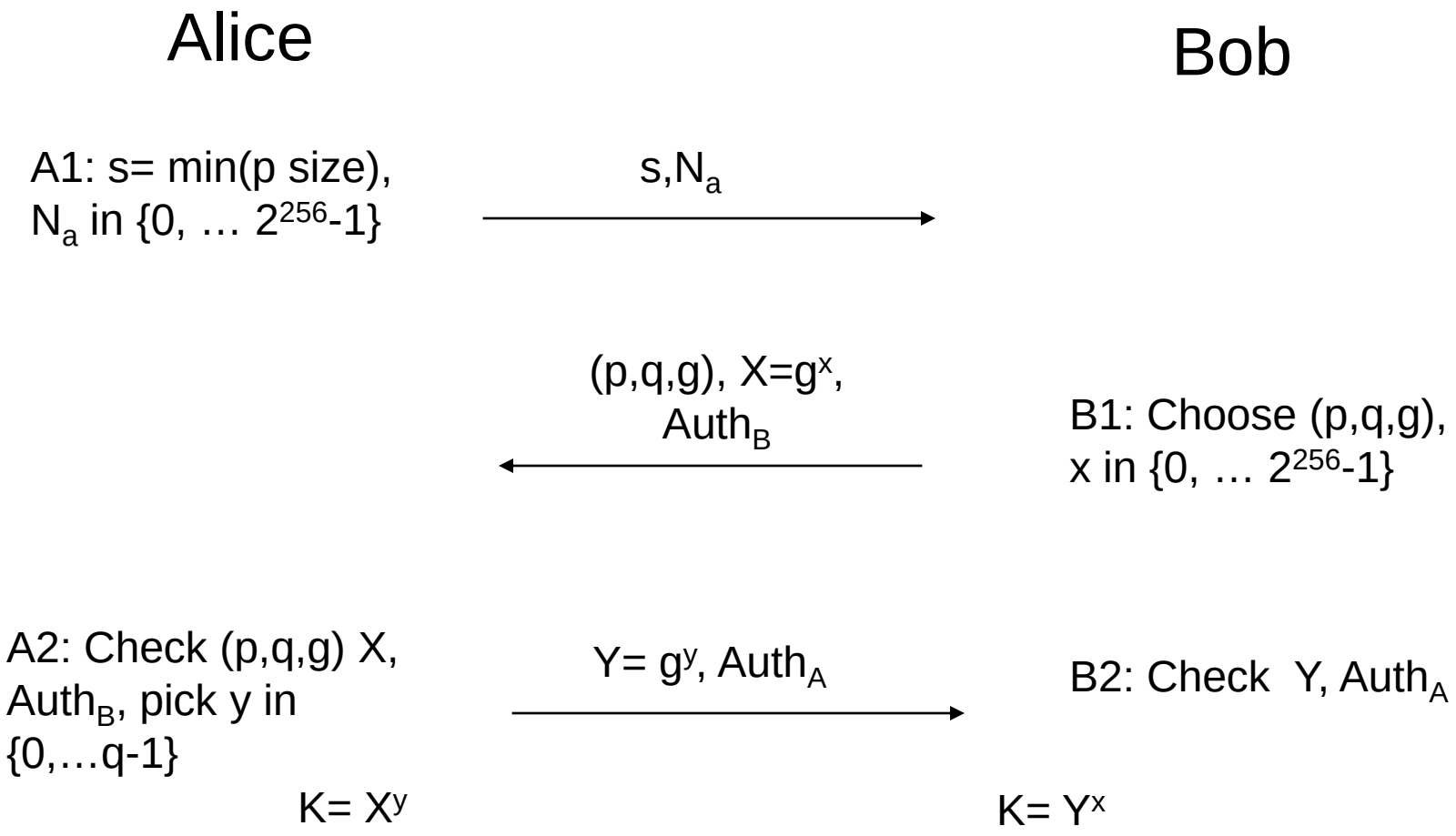
Uses of Public-Key Ciphers

- Symmetric Key Distribution
- Key Exchange and other protocols
- Digital Signatures
- Sealing Symmetric Keys
- Authentication
- Proving Knowledge without disclosing secrets (used in anonymous authentication)
- Symmetric Key systems cannot do any of these. However, symmetric key systems are *much* faster than Public Key systems.

Symmetric Key Distribution

- Imagine you are the head of security for Microsoft and insist that all Microsoft financial communications transmitted over the Internet be encrypted for your finance machines. You buy “black boxes” for every Internet egress point, each with a known Public Key (the private key is generated on the black box and is known only to that hardware).
- Every day, just before 0^h Zulu, you generate a new symmetric key K_d, encrypt it and transmit $E(PK_i, K_d)$ to each black box, i, (Hopefully, using a mechanism that insures that it comes from you or what happens?)
- What’s good about this? Keys are never touched by humans.
- What would you do if you were worried that some black boxes could be compromised (i.e.- private keys determined)?

Diffie Hellman Key Exchange



Digital Signatures

- I want to send you messages you can rely from time to time, like: M="I, John Manferdelli, promise on November 1, 2008, that (1) I will give everyone in CSE 599r an A, (2) I will eat my vegetables, (3) I won't watch Apple ads."
- How can I prove (electronically) they come from me?
- I generate a public/private key pair PK_{JLM}, pK_{JLM}.
- One day in class I personally give you PK_{JLM} on a piece of paper.
- When I send a message like M I also transmit: D(pK_{JLM}, SHA-256(M)).
- When you get M, you calculate SHA-256(M) and compare it to E(PK_{JLM}, SHA-256(M)), if it matches, you can tell it's from me.

Sealing Symmetric Keys

- I want to send you a confidential document, M (like an email). I know your public key PK_{you} (maybe you told it to me, maybe it's in a directory, maybe someone I trust gave it to me and vouched for it).
- I generate a new symmetric key, K, at random.
- I encrypt M with CBC-AES using K and transmit to you:
  1. IV
  2. $CBC\text{-}AES_K(IV,M)$
  3. $E(PK_{you}, K)$
  4. I may also sign the message so you can be sure it came from me
- This is essentially how S/MIME mail works.

Authentication

- I am on an physically secure line (no one can eavesdrop or modify messages between me and the other end point) so I'm not worried about confidentiality.
- I want to make sure you, my lawyer, is on the other end and I know your public key PK_{you}.
- Before I say anything I'd regret reading in the New York Times, I generate a (big) random number, N and append the date and time, calling this entire message, M. I transmit M and ask you to apply $D(pK_{you}, M)$. If $E(PK_{you}, M)=M$, I can be sure it's my attorney; otherwise, I take the fifth.

Existing Public-Key Ciphers

- Public Key systems are based on “computationally hard” “trap door problems” (Not all NP complete problems are suitable).
  - Factoring (n= ab, what is a, what is b?)
  - Discrete Log ($x= y^a \pmod n$, what is a?)
  - Elliptic curve discrete log. (Q=nP, what is n?)
- Rivest Shamir Andelman (1978)
  - Based on factoring
- El Gamal (1984)
  - Based on discrete log
- Elliptic Curve (1985, Miller-Koblitz)
  - Based on elliptic curve discrete log (over finite fields).

Some Number Theory

- Fundamental Theorem of Arithmetic
- Prime Number Theorem
- Euclidean Algorithm for GCD
- Solving congruences
- Chinese Remainder Theorem
- Continued Fractions
- Integer arithmetic mod n
- Fermat's Theorem
- Euler's Theorem
- Finding square root mod n
- Quadratic Reciprocity: Legendre and Jacobi symbols.
- Lattices

Fundamental Theorem of Arithmetic

- Let n be any positive integer, n can be written as a product of primes in an essentially unique way (except for units and the order of the primes).
- It may be hard to actually carry out this factorization.

Distribution of Primes

- Euclid: There are infinitely many primes
- Prime Number Theorem: The number of primes, $\pi(n)$, less than or equal to n is asymptotically equal to n/ln(n).
  - Spookily accurate even for pretty small n.
  - First proof using complex analysis sketched by Riemann, finished by Hadamard and de la Vallee-Poussin. “Elementary” proof by Erdos and Selberg
- Chebyshev’s Theorem: For x>200, $c_1(x/\ln(x)<\pi(x)<c_2(x/\ln(x))$. $c_1=2/3$, $c_2= 1.7$
  - Pretty easy to prove.
- Bertrand’s Postulate: For all n>2 there is a prime between n and 2n

Prime Distribution Example

n	$\pi(n)$	n/ln(n)	$\pi(n)$/ (n/ln(n))
10	4	4.34	.9217
50	14	12.78	1.10
100	25	21.71	1.15
500	95	80.46	1.17
1000	168	144.76	1.16
10^6	78498	72382	1.08
10^9	50847478	48254949	1.05

Solving Congruences mod p

- Solve ax=b (mod p)
  - Procedure: If (a,p) ≠ 1, there is no solution if b ≠ □(mod p) and everything is a solution if b=0(mod p). This leaves us with (a,p)=1. We find u,v such that au+pv=1. (ub) is the solution.
  - This is very fast even if a, b and p are enormous integers.

- Note: See the Short Math reference notes.

- We can also solve recurrences of the form ax=b (mod p^e)

Solving Congruence Example

- Solve $5x=2 \pmod{37}$
- $(15)\ 5 + (-2)\ 37 = 1$ so the solution is $(2 \times 15) = 30 \pmod{37}$
- $5 \times 30 = 150 = 4 \times 37 + 2$
- If only all problems were this easy

Chinese Remainder Theorem

- If $x= a_1 \pmod{m_1}$ and $x= a_2 \pmod{m_2}$ and $(m_1 , m_2)= 1$, then the simultaneous equations have a unique solution mod $m_1 m_2$.

- Proof: $k_1 m_1 + k_2 m_2 =1$. Set $a= a_2 k_1 m_1 + a_1 k_2 m_2$. This is a solution. Also a practical computational method.

CRT Example

- N= 1517= 37 x 41, solve $5\ x^2 = 2 \pmod{1517}$.

- First solve $5\ x^2 = 2 \pmod{37}$ and $5\ x^2 = 2 \pmod{41}$.
  - (15) 5 + (-2) 37 = 1 and (-8) 5 + (1) 41 =1 so:
  - $x^2 = 2 \times 15 = 30 \pmod{37}$ and $x^2 = 2 \times (-8) = 25 \pmod{41}$.
  - $20^2 = 30 \pmod{37}$ and $5^2 = 25 \pmod{41}$.

- Use CRT
  - (10) 37 + (-9) 41= 1
  - (5) (10) (37) + (20) (-9) (41)= 538 =y (mod 1517)
  - $5\ (538)^2 = 2 \pmod{1517}$.

Continued Fractions

- Continued Fraction: $x_0 = x$, $a_i = \lfloor x_i \rfloor$. $x_{i+1} = (x_i - a_i)^{-1}$.
- $p_n/q_n = a_0 + 1/(a_1 + 1 / (a_2 + \ldots))$.
- $p_{-2} = 0$, $p_{-1} = 1$, $q_{-2} = 1$, $q_{-1} = 0$.
- $p_{n+1} = a_{n+1}\, p_n + p_{n-1}$.
- $q_{n+1} = a_{n+1}\, q_n + q_{n-1}$.

- Theorem: If $|x-(r/s)| < 1/(2s^2)$, $r,s \in \mathbb{Z}$, then $r/s = p_i/q_i$ for some i.

Universal Exponent Theorem

- Suppose n is composite, say, n=pq. Given r>0, $a^r = 1 \pmod n$, for all a: (a,n)=1, we can factor n.

- Method: Let $r=2^k m$; m, odd. Put $b_0 = a^m \pmod n$ and $b_{i+1} = b_i^2 \pmod n$.

  1. If $b_0 = 1$, pick another a.
  2. If $b_{i+1} = 1$ and $b_i = -1$, pick another a.
  3. If $b_{i+1} = 1$ and $b_i \neq \pm 1$, $d=(b_i-1,n)$ is a non-trivial factor of n.

Universal Exponent Example

- Let n= 1517. Note that $a^{1440}=1 \pmod{n}$.
- $1440= 2^5(45)$, m= 45.

- $b_0= 2^{45}=401 \pmod{1517}$
- $b_1= 401^2 =1 \pmod{1517}$. Try again.

- $b_0= 3^{45}=776 \pmod{1517}$
- $b_1= 776^2=1444 \pmod{1517}$
- $b_2= 1444^2 =778 \pmod{1517}$.
- $b_3= 778^2 =1 \pmod{1517}$.
- (778-1, 1517)= (777, 1517)=37.

$\phi(n)$

- Definition: $\phi(n)= |\{a: (a,n)=1, 0<a<n\}|$.

$n>2 \rightarrow \phi(n)$ is even
n is prime iff $\phi(n)= n-1$
$\phi(p^e)= (p-1)p^{e-1}$
If $(m_1, m_2)= 1$, $\phi(m_1 m_2)= \phi(m_1)\phi(m_2)$
$\sum_{d|n} \phi(d)= n$
If $n= p_1^{e[1]}p_2e^{[2]}\ldots p_k^{e[k]}$, then $\phi(n)= \prod (1-1/p_i)$
Average value of $\phi(n)$ is $6n/(\pi^2)$

f(n) is multiplicative if $(n,m)=1 \rightarrow f(nm)=f(n)f(m)$
If $n= p_1^{e[1]}p_2e^{[2]}\ldots p_k^{e[k]}$, $\mu(n)= 0$ if $e[i]>1$ for any I, otherwise $\mu(n)= (-1)^k$
If f is multiplicative, so is $F(n)= \sum_{d|n} f(d)$, $f(n)= \sum_{d|n} \mu(d)F(n/d)$

□(n) Example

□(1)=1

□(5)=4

□(25)=20

□(125)=40

Law of Quadratic Reciprocity

- If p and q are primes, define (a/p) = 1 if there is an x: $x^2=a \pmod p$, 0 of p|a, and -1 is there is no such x.

- $(a/p)= a^{(p-1)/2} \pmod p$
- $((ab)/p)= (a/p)\ (b/p)$

- Gauss: $(p/q)(q/p)= (-1)^{[(p-1)/2\ (q-1)/2]}$.

- This allows us to solve quadratic equations in a prime field.

Quadratic Reciprocity Example

	7	11	13	17	29	31
7		-1	-1	-1	1	1
11	1		-1	-1	-1	-1
13	-1	-1		1	1	-1
17	-1	-1	1		-1	-1
29	1	-1	1	-1		-1
31	-1	1	-1	-1	-1	

- $(7/11)(11/7)=(-1)^{5 \times 3}=-1$
- $(7/13)(13/7)=(-1)^{6 \times 3}=1$
- $(7/17)(17/7)=(-1)^{8 \times 3}=1$
- $(11/31)(31/11)=(-1)^{15 \times 5}=-1$

- Entry in row i, column j is $p[i]^{(p[j]-1)/2}$

Factoring and exponents

- Suppose n is the product of two (possibly unknown) primes, p and q.
  1. If p and q are known, we can calculate $\phi(n)$.
  2. If $\phi(n)$ and n are known, we can factor n.

- Proof:
  - If p and q are known, $\phi(n)= (p-1)(q-1)$.
  - If n and $\phi(n)$ are known, set $f(x)= x^2 - Ax+1$ where $A=n-\phi(n)+n+1$. p and q are the roots of $f(x)=0$.

- Note: If $(e, \phi(n))=1$, we can calculate d: $ed=1 \pmod{\phi(n)}$ if we know p and q, this theorem tells us if we know a universal exponent, we can calculate d.

Large Integer Computation

- Almost all public key algorithms are based on “hard” number theory problems over enormous (e.g.- 2048 bit) integers.
- We need to know how to do arithmetic on computers with huge numbers
  - Addition/subtraction
  - Multiplication
  - Modulus
  - Modular inverses
  - Exponentiation
  - Testing Primality
  - Factoring

Algorithm Timings

- Adding two m-bit numbers takes O(m) time.
- Multiplying two m-bit numbers takes <$O(m^2)$.
- Multiplying a 2m-bit number and reducing modulo and m-bit number takes $O(m^2)$.
- Computing (a, b) for a, b< n takes $O(\ln^2(n))$ time (i.e.- fast). This is Euclid's Algorithm and it started Knuth, Euclid and everyone else off on computational complexity. If n has m bits this is $O(m^2)$.
- Testing an number n for primality takes $O(n^{c\lg(\lg(n))})=O(2^{cm\lg(m)})$.
- Best known factoring: $O(n^{c(\lg(n)^{(1/3)}(\lg(\lg(n))^{(2/3)})})=O(2^{cm(m^{(1/3)}(\lg(m)^{(2/3)})})$. [a lot longer].

The Multiplicative Group (mod n)

- G_n= {a: (a,n)= 1, 0<a<n} is the multiplicative group mod n
- $|G_n| = \varphi(n)$ so (a,n)=1 → $a^{\varphi(n)} = 1 \pmod n$
- a is called a primitive root if $ord_n(a) = \varphi(n)$
- If a is a primitive root, $a^b = 1 \pmod n$ → $b | \varphi(n)$
- $ord_n(a^u) = ord_n(a)/(u, ord_n(a))$. If m has a primitive root, there are exactly $\varphi(\varphi((m))$ such primitive roots.
- Theorem: If $n=p^j$, p an odd prime and b is a primitive root mod n then n is not b-pseudoprime
- If n is prime, n has a primitive root.
- n has a primitive root iff n = 2, 4, p^k, $2p^k$ where p is an odd prime.

Finding generators

- For a cyclic group, G of order find a generator, *g*

$$n = p_1^{e_1} \cdots p_k^{e_k}$$

- ```
  while ()
  a: choose a random g∈G
     for i = 1 to k
        b = g^(n/pi)
        if (b = 1) goto a:
  return g
  ```

- G has ϕ(n) generators. Using the lower bound for ϕ(n), the probability that g in line 2 is a generator is at least 1/(6 ln ln *n*)
  ```

Representing Large Integers

- Numbers are represented in base 2^{ws} where ws is the number of bits in the “standard” unsigned integer (e.g. – 32 on IA32, 64 on AMD-64)
- Each number has three components:
  - Sign
  - Size in 2^{ws} words
  - 2^{ws} words where $n= i[ws-1]2^{ws(size-1)} + \ldots + i[1]2^{ws} + i[0]$
  - Assembly is often used in inner loops to take advantage of special arithmetic instructions like “add with carry”

Classical Algorithms Speed

- For two numbers of size s_1 and s_2 (in bits)
  - Addition/Subtraction: $O(s_1)+ O(s_2)$ time and $\max(s_1, s_2)+1$ space
  - Multiplication/Squaring: $O(s_1) \times O(s_2)$ time and space (you can save roughly half the multiplies on squaring)
  - Division: $O(s_1) \times O(s_2)$ time and space
    - Uses heuristic for estimating iterative single digit divisor: less than 1 high after normalization
  - Extended GCD: $O(s_1) \times O(s_2)$
  - Modular versions use same time (plus time for one division by modulus) but smaller space
  - Modular Exponentiation ($a^e \pmod n$): $O((\text{size } e)(\text{size } n)^2)$ using repeated squaring
  - Solve simultaneous linear congruence's (using CRT): $O(m^2) \times$ time to solve 1 where m = number of prime power factors of n

Karasuba Multiplication

- $(a2^k+b)\ (c2^k+d)= ac2^{2k}+(ad+bc)2^k+bd$
  - 4 multiplies
  - Asymptotically n^2
- To save 1 multiply compute
  - t= (a+b)(c+d)= ac+ad+bc+bd
  - ac
  - bd
  - t-ac-bd= ad+bc
  - 3 multiplies, 2 adds
  - Asymptotically $n^{lg(3)}$, lg(3) is about 1.58

Integer Squaring

- Reduced number of multiplies because of symmetry
  - $a= 2^n a_1 + a_0$, $b= 2^n b_1 + b_0$
  - $ab= 2^{2n} a_1 b_1 + 2^n (a_1 b_0 + b_1 a_0)+ b_0 a_0$
    - 4 multiplies
  - $a^2= 2^{2n} a_1^2 + 2^{n+1} a_1 a_0 + a_0^2$
    - 3 multiplies
- Cost: If a is t words long, a^2 takes (t+1)t/2 single precision multiplies

Integer Division Algorithm

- $x= (x_n\ x_{n-1} \ldots x_0)_b$, $y= (y_n\ y_{n-1} \ldots y_0)_b$
- $x/y=q= (q_{n-t}\ q_{n-1} \ldots q_0)_b$, $x \bmod y= r = (r_n\ r_{n-1} \ldots r_0)_b$

- Key Step: Estimate Quotient
  - If $y_t \geq [b/2]$, the estimate
  - $q_{i-t-1}= (x_i b+x_{i-1})/y_t$ is at most 2 greater than the correct value
  - If $q_{i-t-1}= (x_i b^2+x_{i-1}b+x_{i-2})/(y_t b+y_{t-1})$ is at most 1 greater than the correct value

Integer Division

1. Normalize: while($x \geq y b^{n-t}$) q_{n-t}++; $x -= y b^{n-t}$;
2. For(i=n, downto t+1)
   - 2.1 if($x_i = y_t$) $q_{i-t-1} = b-1$
     else $q_{i-t-1} = [x_i b + x_{i-1}/y_t]$
   - 2.2 while($q_{i-t-1}(y_t b + y_{t-1}) > (x_i b^2 + x_{i-1} b + x_{i-2})$) q_{i-t-1}--
   - 2.3 $x -= q_{i-t-1}\, y b^{i-t-1}$
   - 2.4 if ($x > 0$) $x += y b^{i-t-1}$; q_{i-t-1}++;
3. r= x
4. return(q,r)

Cost: (n-t)(t+3) multiplies, (n-t) divisions.

Extended Binary GCD

Input: $x= (x_n\ x_{n-1} \ldots x_0)_b$, $y= (y_n\ y_{n-1} \ldots y_0)_b$. Output: a, b, v: ax+by=v= gcd(x,y).

```
1.  g=1
2.  while(x&1==y&1==0) x/= 2, y/= 2, g*=2
3.  u=x, v=y, A=1, B=0, C=0, D=1
4.  while (u&1==0)
        u/= 2
            if(A=B=0 (mod 2))  A/=2, B/=2
        else A= (A+y)/2, B= (B-x)/2
5.  while (v&1==0)
        v/= 2
            if(C=D=0 (mod 2))  C/=2, D/=2
        else C= (C+y)/2, D= (D-x)/2
6.  if(u>=v) u-=v, A-=C, B-=D
      else     v-= u, C-=A, D-=B
7.    if (u==0) a= C, b= D, return(a,b,gv)
      else goto 4
```

Cost: 2 ([lg(x)]+[lg(y)]+2) iterations

Montgomery Multiplication

- Motivation: Modular reduction is expensive (a divide operation). Can we replace the divide with some cheap operation (like shifting?)
- Let A, B, and M be *n*-block integers represented in base *x* with $0 \leq M < x^n$.
- Let $R = x^n$. gcd(R,M) = 1.
- The *Montgomery Product* of A and B modulo M is the integer $ABR^{-1} \bmod M$.
- Let $M' = -M^{-1} \bmod R$ and $S = ABM' \bmod R$.
- Fact: $(AB+SM)/R \equiv ABR^{-1} \pmod{M}$.

Montgomery Multiplication and Timing

- (r, n)= 1, r= ab (mod n), $a^{\#}$=ar (mod n), rr'-nn'=1, all t words long.

```
MontPro(a#, b#)
    t= a# b# ,m= rn' (mod n), u= (mn+t)/r
    if(n>u) u-= n;
    return(u)
MontMult(a,b,n)
    Compute n', a#, b#
    x#= MontPro(a#, b#)
    return(MontPro(x#,1))
```

Cost: Reduction takes 2t(t+1) multiplies, no divisions. Multiply takes 4t(t+1). vs 2t(t+1) for classical.

Exponentiation and Timing

- Right to left squaring and multiplication
- Left to right squaring and multiplication
- Left to right k-ary

- Square and multiply exponentiation (SME) timing, if bitlen(e)=t+1 and wt(e) is the Hamming wt, SME takes t squarings and wt(e) multiplies.

Montgomery Exponentiation and Timing

$x= (x_l\ x_{l-1} \ldots x_0)_b$, $e= (e_t\ e_{t-1} \ldots e_0)_b$, $m = (m_{l-1}\ m_{l-2} \ldots m_0)_b$,
$R= b^l$, $m'= -m^{-1} \pmod b$

MontExp(x,e,m)

1. $x^{\#}$= MontMult(x, R^2, m), A= R (mod m)
2. for(i= t downto 0)
   - 2.1 A= MontMult(A,A)
   - 2.2 if (e_i==1) A= MontMult(A, $x^{\#}$)
3. return(MontMult(A,1))

Cost: Total: 3l(l+1)(t+1). [For Classical: 2l(l+1) plus l divisions.]

Step	1	2	3
# MontMult	1	3/2 t	1
# SP Mult	2l(l+1)	3tl(l+1)	l(l+1)

Montgomery Example

- Suppose N = 79, a = 61 and b = 5
- $R = 10^2 = 100$. RR'-NN'=1, R'=64, N'=81.
  - $a' = 61 \cdot 100 = 17 \pmod{79}$
  - $b' = 5 \cdot 100 = 26 \pmod{79}$
  - abR (mod 79)= 61 x 5 x 100 (mod 79)= 6
  - X= a'b'=442= abR^2 (mod N)
  - m= (X (mod R))(N' (mod R))= 42 x 81 =2 (mod R)
  - x= (X+mN)/R= (442+2x79)/100 = 6

Example from Mark Stamp

Exponentiation Optimizations

- Arbitrary g, e
- Fixed g
  - RSA
- Fixed e
  - DH
  - El Gamal
- Useful in El Gamal verify
  - Example: $\beta^{\phi(m)}(\gamma^{-a})^r$

RSA Public-Key Cryptosystem

Alice (Private Keyholder)

- Select two large random primes p & q.
- Publish the product n=pq.
- Use knowledge of p & q to compute Y.

Anyone (Public Key Holder)

- To send message Y to Alice, compute $Z=Y^X \bmod n$.
- Send Z and X to Alice.

Rivest, Shamir and Adleman, “On Digital Signatures and Public Key Cryptosystems.” CACM, 2/78.

RSA Details

- Encryption: $E(Y) = Y^e \bmod n$.
- Decryption: $D(Y) = Y^d \bmod n$.
  - $D(E(Y)) = (Y^e \bmod n)^d \pmod n = Y^{ed} \pmod n = Y$
- Speedup: Compute mod p and mod q then assemble using CRT
- Remember (p,q)= 1 → there are p', q': p'p+q'q=1
- Saves roughly factor of 4 in time

RSA Example

- p=691, q=797, n=pq=550727. $\phi(n)$= 690 x 796= 2^3x3x5x23x199.
- Need (e, $\phi(n)$)=1, pick e=7.
- 1= 7 x 78463 + (-1) $\phi(n)$, so d= 78463.

- 78463= 2^{16}+ 2^{13}+ 2^{12}+ 2^9+ 2^6+ 2^5+ 2^4+ 2^3+ 2^2+ 2^1+ 2^0 = 65536+8192+4096+512+64+32+16+8+4+2+1. Use this in the successive squaring calculation.

- Public Key: <n=550727, e=7>
- Private Key: <p=691, q=797, d=78463>.

- Encrypt 10. 10^7 (mod n)= 86914.
- Decrypt: $(86914)^{78463}$ (mod n)=10.
- Successive squares: 86914, 271864, 268188, 407871, 97024, 79965, 460755, 375388,444736, 362735, 289747, 500129, 378508,532103, 446093, 371923, 66612.

RSA Signatures

An additional property

$D(E(Y)) = Y^{ed} \bmod n = Y$

$E(D(Y)) = Y^{de} \bmod n = Y$

Only Alice (knowing the factorization of n) knows D.
Hence only Alice can compute $D(Y) = Y^{d} \bmod n$.

This D(Y) serves as Alice's signature on Y.

Generating Primes

- Probabilistic testing for primality is faster than factoring.
- T(n, p, t) is a test, depending on a parameter, t, which results in a “yes/no” answer to the question: “Is n prime?”
  - If “yes”, the answer is true with probability p.
  - If “no,” n is definitely not prime.
- Fermat Test. If (n,t)=1,
  - T(n, .5, t):= “yes” if $t^{(n-1)} = 1 \pmod{n}$.
- Note for most tests trial divide by all primes p □B, first.

Pseudoprimes

- Recall Fermat's Little Theorem: If n is a prime $a^{(n-1)}= 1 \pmod n$ for all a with (a, n)= 1.
- n is a b-pseudoprime iff $b^{(n-1)}=1 \pmod n$ and n is *not* prime.
  - There are 3 2-psuedo primes <1000: 341, 561, 645.
  - Theorem: There are infinitely many 2-pseudoprimes.

    Proof: $d|n \rightarrow 2^d-1|2^n-1$. Suppose n is a 2-pseudoprime, so is $m=2^n-1$.

    m is obviously composite. Since n is a 2 pseudoprime $n| k=2^n-2$ so $2^n-1|2^k-1$.
- n is a Carmichael number if n is a b-pseudoprime for every b: (b,n)=1. 561 is a Carmichael number.
- If n is a Carmichael number then $n=p_1p_2\ldots p_r$ and $p_i-1|n-1$. r>2.
- Alford, Granville, Pomerance: There are infinitely many Carmichael numbers

Liars, Witnesses and Certificates

- n is prime iff $\phi(n)=n-1$. So if we find a b such that $ord_n(b)= n-1$. b is a witness to n's primality.
- To show $ord_n(b)= n-1$, we can factor $n-1= \prod p_i^{e[i]}$ and check that for each $k= n-1/p_i$, $b^k \neq 1 \pmod n$.
- If $b^{n-1} \neq 1 \pmod n$ b is said to be a "witness" to n's compositeness.
- If n is a b-pseudoprime b is said to liar for the primality of n.
- Given $n-1=2^t m$, m odd and b. A b-sequence is $b_1= b^m$, $b_{i+1}= b_i^2$, i= 1,2,..t,. If a b sequence does not end in 1 or if a terminal 1 is not preceded by a -1, n is composite. Again b is a "witness" as to the compositeness of n.
- If n is composite and a b sequence acts like one for a prime, n is called a b-strong pseudoprime.
- b is a strong liar for compositeness if n passes the strong pseudoprime test with b

Primality Testing

- Deterministic test: n is prime if m does not divide n for all $m < \sqrt{n}$.
  - Check m= 2 and m odd
  - Sieve of Eratosthenes
  - Keep a list of primes
  - Still to slow
- Probabalistic
  - Fermat
  - Solovay-Strassen
  - Miller-Rabin: Try bases b= 2, 3, … p_k , if n is a b-sequence "passes" the primality condition, conclude n is prime.
  - If the extended Riemann Hypothesis is true the Miller-Rabin test is dispositive as to the primality of n if we try all bases up to $2(\ln(n))^2$.

Testing Primality - Miller Rabin

- MR(n, .25, t), n≥3, n, odd. Set n-1= $2^s r$, r, odd. (t≥3, in practice)
- Takes ~ O(lg(n)^3).

```
for(i=1; i≤t) {
        Choose a, 1<a<n-1.  2 is a good choice first time.
        Compute y=a^r (mod n)
        If y≠1 and y≠(n-1) {
           j=1
           while(j ≤(s-1), y≠n-1)
              y= y^2 (mod n)
              if (y=1)
                 return("no")
              j= j+1
            }
          if(y≠ n-1)
            return("no")
        return("yes")
        }
```

Probability of Success for M-R

- Theorem (Strong liars are scarce): If n is composite and odd then at most (n-1)/4 residue classes can be strong liars.
  - Case 1: $n= p^e$. Let g be a primitive root of n. $ord_n(g)=(p-1)p^{e-1}$. If n is a g^a-pseudoprime, $(p-1)p^{e-1}|a(p^e-1) \rightarrow p^{e-1}|a$. So n is a g^a-pseudoprime iff $p^{e-1}|a$ and there are p-1 such a's. $(p-1)/p^e \geq 1/4$.
  - Case 2: $n= p_1^{e[1]}p_2^{e[2]}\ldots p_r^{e[r]}$. $n-1=2^st$, t odd. Define k to be smallest such that if $e=2^k$, $b^e= -1 \pmod n \rightarrow b^e= -1 \pmod{p_i}$, all i . So $2^{k+1} | p_i-1$, all i so $2^{k+1}|(n-1) \rightarrow k<s$. Set $m=2^kt$, $b^m=(-1)^t= -1$, and $L=\{a: |a^m|=1, 1 \leq a<n\}$. We will show L contains all the strong liars and $|L| \leq (n-1)/4$.

Proof Continued

- L contains all the strong liars and |L|□(n-1)/4.
  - If a is a strong liar, and v= 2^jt, j=0 and a^v= 1 or -1 or a^v= -1 for j□k, thus a □ L and a □ L → ab □ L.
  - For a□L, put S(a)={x: 1□x<n, and x=a or ab mod $p_i^{e[i]}$, all i}. There are 2^r elements in S(a) only 2 of which are in L ({a, ab}) and each appears in at most 2 of the sets. Thus there are at least |L|(2^r-2)/2 integers that are not strong liars. If r□3, were done. If r=2 there is at least one non-strong liar in S(a) for every one that is. If x is in the union of the S(a)'s, n is an x-psuedoprime but is a is not a Carmichael number, at most half the positive integers less than a are liars: if x is a liar and (a,y)=1, xy is not a liar. So if x_1 and x_2 are both liars yx_1 and yx_2 are not. All that is left to show is that n is not a Carmichael number is r=2 but that is true.

Primality Testing Example

- From Trappe and Washington. n=561.
- $n-1=560=2^4 \times 5 \times 7$. Pick a=2.
  - $b_0 = 2^{35} = 263 \pmod{n}$
  - $b_1 = b_0^2 = 166 \pmod{n}$
  - $b_2 = b_1^2 = 67 \pmod{n}$
  - $b_3 = b_2^2 = 1 \pmod{n}$
- 561 is composite. In fact, $(b_2-1, 561)=33$.

Strong Primes

p is a “strong prime” if

1. p-1 has a large prime factor, r.
2. p+1 has a large prime factor, s.
3. r-1 has a large prime factor, t.

Other criteria (X9.31)

- If e is odd (e,p-1) =1=(e, q-1)
- (p-1, q-1) should be “small”
- p/q should not be near the ratio of two small integers
- p-q has a large prime factor
- Add Frobenius test
- Add a Lucas test

Gordan's Algorithm

Gordan's algorithm

1. Generate 2 primes, s,t of roughly same length.
2. Pick i_0. Find first prime in sequence, (2it+1), $i=i_0$, $i_0+1,\ldots$; denote this prime as r= (2it+1).
3. Compute, $p_0= 2(s^{(r-2)} \pmod r)s-1$.
4. Select j_0. Find first prime in sequence, (p_0+2jrs), $j=j_0,j_0+1,\ldots$; denote this prime as $p= (p_0+2jrs)$.
5. return(p)

Attacks

- Elementary
  - Common Modulus: $K_1 = (e_1, d_1, pq)$, $K_2 = (e_2, d_2, pq)$
- Low Public Exponent
  - Wiener: Let N=pq, q<p<2q, d<1/3 $n^{1/4}$, given <N,e> and ed=1 (mod $\phi(n)$), we can find d efficiently.
    - Uses continued fractions
  - Coppersmith's Theorem: Let N be an integer and f a monic polynomial over Z, $X = N^{1/d-\beta}$ for some $\beta >= 0$. Given <N, f>, we can efficiently find all integers $|x_0| < X$ satisfying $f(x_0) = 0 \pmod N$. Running time is dominated by LLL on lattice with dimension $O(\min(1/\beta, \lg(N))$.

Attacks, continued

- Related Messages and low exponents
  - Coppersmith's theorem can be used to strengthen Franklin-Reiter Related Message attack if e=3 and pad is <1/9 message length.
- Timing/Glitching
- Bleichenbacher's Attack on PKCS 1
- Factoring
  - Pollard rho
  - p-1
  - Quadratic Sieve
  - Number Field Sieve

- Reference: Boneh, Twenty years of attacks on RSA. Notices AMS.

Common Modulus Attack

- (e1, e2)=1.
- $c_1 = m^{e1} \pmod n$
- $c_2 = m^{e2} \pmod n$
- d1 e1 + d2 e2 = 1
- $(c_1)^{d1} (c_2)^{d2} = m$, oops!

Small exponent attack on RSA

- If $q<p<2q$, $n=pq$, $1 \leq d, e < \phi(n)$. If $d< 1/3\ n^{1/4}$, d can be calculated quickly.

- Proof: $q<\sqrt{n}$, $n-\phi(n)<3\sqrt{n}$. $ed= 1+\phi(n)k$.
- So, $\phi(n)k<ed<\phi(n)1/3\ n^{1/4}$. $kn-ed= k(n-\phi(n))-1$.
- $0< k/d – e/n <1/(3d^2)$. By continued fractions result, the successive approximations A/B with k=A, d=B and C=(ed-1)/k allows us to compute $\phi(n)=C$. Now use the previous result.

Short plaintext

- $c = m^e \pmod{n}$, m, unknown (but small).
- Make two lists: $cx^{-e} \pmod{n}$ and y^e with x,y “small.”
- When they match:
  - $cx^{-e} = y^e \pmod{n}$ and $c = (xy)^e \pmod{n}$.

Glitching Attack

- n=pq. <e,d> are the encryption and decryption exponents. Attack is on private key which is used for signing, say, a hash. Let p'p + q'q=1.
  - Suppose signer uses the CRT, $m_1 = m \pmod p$ and $m_2 = m \pmod q$. The correct solution is $m_1^d = a_1 \pmod p$ and $m_2^d = a_2 \pmod q$ and the CRT gives $y = a_2\, p'p + a_1 q'q$.
- Suppose the computation is done on a w-bit (e.g.-32) machine which miscomputes a x b for two specific w-bit values a, b.
  - We want m around $\sqrt{n}$ satisfying $p<m<q$ involving a and b; for example, $m = c_k\, 2^{wk} + c_{k-1}\, 2^{w(k-1)} + \ldots + a\, 2^w + b$.
  - We submit m for signing. Because of the error, the signer will (mis)compute $y = m^d \pmod n$ in a way we can take advantage of.
  - In normal squaring, m_1^2 will be computed correctly (mod p) but m_2^2 will be computed incorrectly (mod q). We get $m_1^d = a_1 \pmod p$ [correct] and $m_2^d = a_2' \neq a_2 \pmod q$ [wrong!]. $y \neq y' = a_2'\, p'p + a_1 q'q$.
  - Resulting y' will be correct (mod p) but wrong (mod q).
  - Now $(y'^e - m, n) = q$. Oops.

Glitching Attack Example

- p=37, q=41. n=pq=1517. $\phi(n) = 36 \times 40 = 2^5 \times 3^2 \times 5 = 1440$.
- Note as before that 10(37)+(-1)41=1.c= $\sqrt{1517} \sim 38$. We pick m= 39.
- Now imagine an RSA scheme with e=7 and the foregoing parameters.
  - 3 (1440) +(-617) 7=1, so d= -617=823 (mod 1440).
  - m_1= m (mod 37)=2, m_2= m (mod 41)= 39.
  - d_1= d (mod 36)= 31, d_2= d (mod 40)= 23.
  - 2^{31}=22 (mod 37), 39^{23}= 33 (mod 41).
  - By the CRT, $y=m^d$ (mod n)= (10)(37)(33)+(-9)(41)22= 1058. We confirm 1058^7= 39 (mod n).
- Now suppose w=3, 39= 4 x 8 + 7 and suppose the error in the computer is that it thinks 4 x 7 = 26.
  - Computing m_2^2 (mod 41) we get 13 instead of the correct answer, 4.
  - Using the usual exponentiation procedure, we would compute 39^{23} (mod 41) =12 (wrong!) and y'= (10)(37)(12)+(-9)(41)22 =873. 873^7 (mod n)=1334.
  - (1334-39,1517)= (1295, 1517)=37. Bingo!

Repeated Squaring

```
// Compute y = x^d (mod N)
// where, in binary, d = (d_0,d_1,d_2,…,d_n) with d_0 = 1
s = x
for i = 1 to n
     s = s^2 (mod N)
     if d_i == 1 then
          s = s · x (mod N)
     end if
next i
return s
```

Timing Attack (Kocher)

- Attack on repeated squaring
  - Does not work if CRT or Montgomery used
  - In most applications, CRT and Montgomery multiplication are used
- This attack originally designed for smartcards
- Can be generalized (differential power analysis)
- Recover private key bits one (or a few) at a time
  - Private key: $d = d_0,d_1,\ldots,d_n$ with $d_0 = 1$
  - Recover bits in order, $d_1,d_2,d_3,\ldots$

Example from Mark Stamp

Kocher's Attack

- Suppose bits $d_0, d_1, \ldots, d_{k-1}$, are known
- We want to determine bit d_k
- Randomly select C_j for j=0,1,…,m-1, obtain timings $T(C_j)$ for $C_j^d \pmod N$
- For each C_j emulate steps i=1,2,…,k-1 of repeated squaring
- At step k, emulate $d_k = 0$ and $d_k = 1$
- *Variance* of timing difference will be smaller for correct choice of d_k

Example from Mark Stamp

Preventing Timing Attack

- RSA Blinding
- To decrypt C, generate random r

  $Y = r^e C \pmod N$
- Decrypt Y then multiply by $r^{-1} \pmod N$:

  $r^{-1}Yd = r^{-1}(r^eC)^d = r^{-1}rC^d = C^d \pmod N$
- Since r is random, timing information is hidden

Factoring

- Security of RSA algorithm depends on (presumed) difficulty of factoring
  - Given n = pq, find p or q and RSA is broken
- Factoring like “exhaustive search” for RSA

- What are best factoring methods?
- How does RSA “key size” compare to symmetric cipher key size?

Factoring Methods: Motivation

- Trial division
  - Obvious method but not practical
- Get element order
  - $x^{2k} = 1 \pmod n$
  - $(x^k-1)(x^k+1) = 0 \pmod n$.
  - See below for exploiting this
  - How do we find k?
- Find $x^2 = y^2 \pmod n$, $x \neq \pm y$, calculate $(x+y,n)$, $(x-y,n)$
  - Theorem: If x, y are chosen at random subject to $x^2 = y^2 \pmod n$ then $P(x \neq \pm y) = ½$.
  - Next question: How do we find such x,y.

Trial Division

- Given n, trial divide n by 2,3,4,5,6,7,…,$\lfloor\sqrt{(n)}\rfloor$
- Expected work is about $\sqrt{n}/2$
- Trying only prime numbers reduces search $\pi(n) \approx n/\ln(n)$ is number of primes up to n.

Pollard p-1

- Suppose p|n and p-1 has small factors. Pick a>1 and B. We're hoping B!=(p-1)k.
- Set $b_1 = a$, $b^{j+1} = b_j^j \pmod n$.
- Put $b = b_B \pmod n = a^{B!} \pmod n$
- Now look at (b-1, n)=d. If 1<d<n, we have a factor; if not, universal exponent theorem might work.
- Lenstra's Elliptic Curve Factoring Method is an extension of this idea.

Kraitchik

- We want to factor n = pq.
- Find x,y such that $n = x^2 - y^2$

- How do we find such x, y?
- Ad hoc:
  - n= 193541963777
  - $439935^2 = 2^8 \times 7^2 \times 67 \pmod{n}$
  - $1069^2 \times 7^2 \times 67 = 449490^2 \pmod{n}$
  - $(439934 \times 1609)^2 = (2^4 \times 44940)^2$

Factoring – Pollard

- $f(x) = x^2 + 1 \pmod{n}$.
- $x_{i+1} = f(x_i) \pmod{n}$.
- Look at $(x_i - x_j, n)$.
  - Actually, use Floyd's trick and look at $(x_m - x_{2m}, n)$.
- Loop expected after about 2n iterations.
  - Actually, after $\sqrt{(\pi n/2)}$ steps).
- Unfortunately, this is exponential in lg(n)

Pollard factoring Example

- We use our old favorite n=1517.
- f(952)= 952^2+1 (mod 1517)= 656
- f(360)= 360^2+1 (mod 1517)= 656
- 952^2- 360^2= (952-360)(952+360).
- 952-360=592
- (592, 1517)= 37.

- Question: Where does the name factoring come from?

Factor Bases

- Pick a set of primes: **B**= {-1, 2,3,5,7,…, p} (the “bases”). Numbers which completely factor are called B-smooth.
- $a_i = \sqrt{((\lfloor\sqrt{n}\rfloor+i)^2})–n$
- Find a_i so that it completely factors over $p \in$ **B**, these numbers are called **B**-smooth
- Example:
  - $a_1^2 = p_1\ p_2$, $a_2^2 = p_2\ p_3$, $a_3^2 = p_1\ p_3$
  - $(a_1a_2a_3)^2=(p_1p_2p_3)^2$
  - Compute $((a_1a_2a_3-p_1p_2p_3), n)$.

Linear Algebra

- Let **B**={p: p<B} and |B|= k.
- If we have r>k “smooth” numbers
  - $x_i^2 = \prod_{j \le k} p_j^{e[j]}$. …… E_i
  - We can find a_j = 1,0: $\prod_{j \le k} e[i]=0 \pmod 2$ --- Gaussian elimination!
  - So $\prod_l x_i^2 = \prod_j p_j^{2d[j]}$. …… E_i
  - This gives us the relations we want

Factor Basis Example

- n=3837523. B={2,3,5,11,13,19}.
- $9398^2 = 5^5 \times 19 \pmod{n}$
- $19095^2 = 2^2 \times 5 \times 11 \times 13 \times 19 \pmod{n}$
- $1964^2 = 3^2 \times 13^3 \pmod{n}$
- $17078^2 = 2^6 \times 3^2 \times 11 \pmod{n}$
- $(9398 \times 19095 \times 1964 \times 17078)^2 - (2^4 \times 3^2 \times 5^3 \times 11 \times 13^2 \times 19x)^2 = 0 \pmod{n}$
- $2230387^2 = 2586705^2 \pmod{n}$
- (223038-2586705, 3837523)= 1093

Sieving

- To factor n, set m= $\lfloor\sqrt{n}\rfloor$. Pick a **B**.
- $f(x)= (x+m)^2-n$
- For small x, we are likely to have small f(x) and hopefully factors over **B**.
- Collect r>k of these as follows (sieving):
  1. If $x^2=n \pmod p$, n is a QR mod p
  2. Write down the f(m+i), $-C \leq i \leq C$ (The sieving interval)
  3. Use the regularly spaced solutions to $x^2=n \pmod p$, to reduce each f(m+i)
  4. Do this for each p.
- Use these to get B or more factorizations and (by solving B or more linear systems)
- There's a > ½ probability that the resulting relation will find a factor.

Quadratic Sieve

- To analyze QS, we need to finds a good interval and estimate sieving and solving times

# of decimal digits	50	60	70	80	90	100	110	120
# factor bas x 1000	3	4	7	15	30	51	120	245
# sieving interval x 10^6	.2	2	5	6	8	14	16	26

Sieve Example

- n= 7429, m=86. B={-1,2,3,5,7}

s	-3	-2	-1	0	1	2	3
$(s+m)^2-n$	-540	-373	-204	-33	140	315	492
p=2	-135		-51		35		123
p=3	-5		-17	-11		35	41
p=5	-1				7	7	
p=7					1	1	

Quadratic Sieve Analysis

- Define $L_n[u,v]= \exp(v(\lg(n))^u(\lg(\lg(n)^{(1-u)}$.
- Let $\Psi(x,B)$ =|{y: y$\leq$x and y is B-smooth}|.
- **Theorem [deBruijn, 1966]:** Let α>0, then for x$\geq$10, w $\geq$ln(x)$^{(1-\alpha)}$, $\Psi(x,x^{(1/w)}) = xw^{(-w+f(x,w))}$, where f(x,w)/w $\rightarrow$0 as w$\rightarrow \infty$.
- Corrollary: If a, u, v >0, then $\Psi(n^a, L_n[u,v])= n^a L_n[1-u,-(a/v)(1-u)+o(1)]$ as n$\rightarrow \infty$.
- For QS generate numbers f(s)~ $\sqrt{n}$. Set a= ½ in Corollary. Probability of finding one that is $L_n[u,v]$-smooth one is $L_n[1-u,-1/(2v)(1-u)+o(1)]$ so we must try $L_n[1-u, 1/(2v)(1-u)+o(1)]$ to find one.
- Size of factor base is ~ $L_n[u,v]$.
- Choose u= 1/2. $L_n[1/2, x]\ L_n[1/2, y]= L_n[1/2, x+y]$.
- Size of sieving interval is $L_n[1/2, v]\ L_n[1/2, 1/(4v)]= L_n[1/2, v+1/(4v)]$
- Sieving time is $L_n[1/2, v+1/(4v)]$, solving sparse equations is $L_n[1/2, 2v+o(1)]$. Total time is minimized when v=1/2 and is $L_n[1/2, 1+o(1)]$.

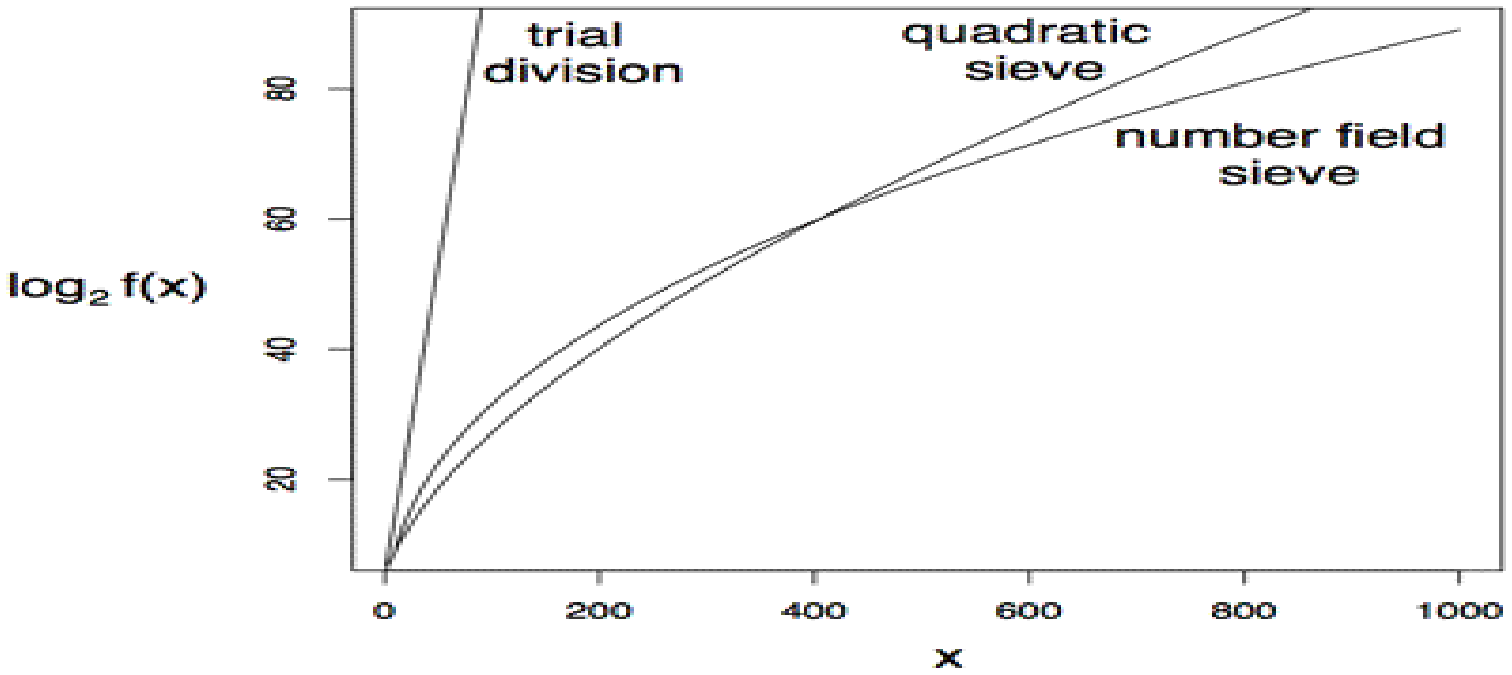
Three more algorithms

- Multiple Polynomial Quadratic Sieve: Use many polynomials (shorter sieve intervals)

- Number Field Sieve: Extends QFS by allowing elements to be algebraic integers in algebraic number field.

- Elliptic Curve Factoring Method: Does arithmetic over elliptic curve mod n. Q=k x P. Operations project mod p if p|n. If Q is the identity (0:1:0) mod p, third coordinate, z, is 0 mod p. Then (z,n)=p. Now check to see if the difference of two points (for different k) have last coordinates: $(z_1-z_2,n)=p$.

Factoring Algorithms



Example from Mark Stamp

Work Factors

Method	f(x)
Trial Division	$n/\lg(n)$
Quadratic Sieve	$(n \lg(n))^{1/2}$
Number Field Sieve	$1.9223\ n^{1/3} \lg(n)^{2/3}$

- Quadratic Sieve: $L_n[1/2, 1+o(1)]$
- ECM: $L_p[1/2, -\sqrt{}(1/2)]$ where p is smallest prime dividing n.
- Fastest in 1998: $L_n[1/2, 1+o(1)]$
- NFS (Pollard again): $L_n[1/3, (64/9)^{(1/3)}]$.
- QS best for N up to 390 bit 117 digits), then NFS.

RSA Caution: Homomorphism

- Commutivity
  - Given plain/cipher pairs (p_i, c_i), i= 1, 2,…, n, one can produce product pairs like ($p_1p_5p_2$, $c_1c_5c_2$) of corresponding plain/cipher pairs.
  - Solution: padding

Practical Factoring Results

- On August 22, 1999, the 155 digit (512 bit) RSA Challenge Number was factored with the General Number Field Sieve.

- Sieving took 35.7 CPU-years in total on...

160	175-400 MHz SGI and Sun workstations
8	250 MHz SGI Origin 2000 processors
120	300-450 MHz Pentium II PCs
4	500 MHz Digital/Compaq boxes

- Total CPU-effort : 8000 MIPS years over 3.7 months.

RSA Summary

- RSA is a great algorithm.
- Just don’t do anything stupid.
  - Reasonable exponents
  - Good padding
  - Good prime generation

End