Integer Factoring

Cryptography

Integer Factoring

1. Pollard's $P-1$ Method
2. Difference of Two Squares
   1. Quadratic Sieve
   2. Number Field Sieve

Factoring Algorithms

- The two best factoring algorithms known are
  - Number Field Sieve (NFS) : A general factoring method
  - Elliptic Curve Method (ECM) : Factoring for special purpose
- Factoring methods are usually divided into
  - Dark Age Methods
    - Trial Division, $P-1$ method, $P+1$ method, Pollard ρ-method
  - Modern Methods
    - CFRAC, Quadratic Sieve (QS), NFS, ECM
- We shall look at the $P-1$ method and explain the idea behind the Modern Methods

Trial Division

- Suppose N is the number to be factored
- Trial Division:

```
for (p = 1; p < sqrt(N); p++)
  { while ( (N%p) == 0 )
      { print p;
        N = N / p;
      }
  }
print N;
```

- The trial division takes time at worst $O(\sqrt{N})$
- The input size to the algorithm is of size $\log_2 N$, hence the complexity is *exponential*
- Although slow, this is the method of choice for numbers $N < 10^{12}$

Integer Factoring

1. **Pollard's $P-1$ Method**
2. Difference of Two Squares
   1. Quadratic Sieve
   2. Number Field Sieve

Smooth Numbers

- Let B be a number
  - N is **B-smooth** if every prime factor p of N is less than B
  - Example: $N = 2^{78} \cdot 3^{89} \cdot 11^3$ is 12-smooth
    - Sometimes we say that the number is just *smooth* if the bound B is small compared to N
    - Smooth numbers are basically easy to factor using Trial Division, hence almost all good factoring algorithms make use of smooth numbers
  - A number is **B-power smooth** if every prime power dividing N is less than B
  - Example: $N = 2^5 \cdot 3^3$ is 33-power smooth

Pollard's $P-1$ Method

- John Pollard has invented almost all the modern factoring algorithms
  - $P-1$ Method
  - Rho-Method
  - Number Field Sieve
- Suppose the number we wish to factor is $N = pq$
- Suppose we know that $p-1$ is B-power smooth (by some pure guess) but that $q-1$ is not B-power smooth
  - Now $p-1$ divides $B!$, but probably $q-1$ does not divide $B!$

Pollard's $P-1$ Method

- Pollard's $P-1$ Method
  - Compute $a = 2^{B!} \pmod N$
  - Imagine taking this modulo p and modulo q
  - Then $a \equiv 1 \pmod p$
    - Since $(p-1) \mid B!$ and $2^{p-1} \equiv 1 \pmod p$ by Fermat's Little Theorem
  - But it is *unlikely* that $a \equiv 1 \pmod q$
  - Hence p will divide $a-1$, but q will not divide $a-1$
  - We recover p from $p = \gcd(a-1, N)$

Pollard's $P-1$ Method

- Pseudo code:

```
a = 2
for (j=2; j<=B; j++)
    { a=a^j mod N; }
d = gcd(a-1,N);
if (d!=1 and d!=N) then
    Output d is a factor of N
else
    Output No Result
```

- Example
  - $N = 15770708441$
  - Put $B = 180$
  - Obtain $a = 2^{B!} \pmod N$
    $= 1162022425$
  - Then $d = \gcd(a-1, N)$
    $= 135979 \mid N$
  - It works because
  - $N = 135979 \times 115979$
  - $135979 - 1$
    $= 2 \times 3 \times 131 \times 173$

Complexity

- Complexity
  - We can show that the complexity of Pollard's $P-1$ method is
    $O(B (\log B)(\log N)^2 + (\log N)^3)$
  - So if $B = O((\log N)^k)$, then this is a polynomial time factoring algorithm
    - Only works for special numbers though

Safe Primes

- Safe Primes
  - Due to the $P-1$ method, RSA primes are recommended to be chosen of the form
    $p-1 = 2p_1$ and $q-1 = 2q_1$
    where p_1 and q_1 are both primes
  - p and q are then called **safe primes**
  - This is not really needed these days
    - The probability that a 512 bit prime p is such that $p-1$ is B-power smooth for a small value of B is very small
    - Hence choosing random 512 bit primes would nullify the $P-1$ method

Integer Factoring

1. Pollard's $P-1$ Method
2. **Difference of Two Squares**
   1. **Quadratic Sieve**
   2. **Number Field Sieve**

Difference of Two Squares

- A basic trick in factoring algorithms is to produce two numbers x and y such that $x^2 = y^2 \pmod N$
  - Therefore $x^2 - y^2 = (x-y)(x+y) = 0 \pmod N$
  - We compute $d = \gcd(x-y, N)$
- If $N = pq$ then we can have four possible cases
  - p divides $x-y$ and q divides $x+y$ $\Rightarrow d = p$
  - p divides $x+y$ and q divides $x-y$ $\Rightarrow d = q$
  - p and q both divide $x-y$ but not $x+y$ $\Rightarrow d = N$
  - p and q both divide $x+y$ but not $x-y$ $\Rightarrow d = 1$
- These all happen with equal probability, so with probability 1/2 we obtain a factor of N

Modern Factoring Methods

- Q: How to find x and y such that $x^2 = y^2 \pmod N$?
- This Fermat's idea inspired modern factoring methods
  - Continued Fraction
  - Quadratic Sieve
  - Number Field Sieve
    - Special Number Field Sieve (SNFS)
      - Number of nice algebraic form: $r^e \pm s$ for small r and s
      - Record: $2^{1039} - 1$ (1039th Mersenne number)
    - General Number Field Sieve (GNFS)
      - No known nice algebraic form
      - Record: RSA-768

Complexity of Algorithms

- Binary length of input $= \lfloor \log_2 N \rfloor + 1 \propto \log N$
- $L(s) = O(e^{c(\log N)^s (\log\log N)^{1-s}})$
  - $s = 1$: exponential time
  - $0 < s < 1$: sub-exponential time
  - $s = 0$: polynomial time
- Complexity of integer factoring algorithms
  - Trial Division : $L(1)$ $O(e^{(1/2)\log N})$
  - Quadratic Sieve : $L(1/2)$ $O(e^{(1+o(1))(\log N)^{1/2}(\log\log N)^{1/2}})$
  - GNFS : $L(1/3)$ $O(e^{((64/9)^{1/3}+o(1))(\log N)^{1/3}(\log\log N)^{2/3}})$
  - SNFS : $L(1/3)$ $O(e^{((32/9)^{1/3}+o(1))(\log N)^{1/3}(\log\log N)^{2/3}})$

RSA Challenge (inactive since 2007)

# decimals	date or year	algorithm	effort (MIPS years)
39	Sep 13, 1970	CF	
50	1983	CF	
55–71	1983–1984	QS	
45–81	1986	QS	
78–90	1987–1988	QS	
87–92	1988	QS	
93–102	1989	QS	
107–116	1990	QS	275 for C116
RSA-100	Apr 1991	QS	7
RSA-110	Apr 1992	QS	75
RSA-120	Jun 1993	QS	835
RSA-129	Apr 1994	QS	5000
RSA-130	Apr 1996	NFS	1000
RSA-140	Feb 1999	NFS	2000
RSA-155	Aug 1999	NFS	8400

Challenge Number	Prize ($US)	Status	Submission Date
RSA-576	$10,000	Factored	December 3, 2003
RSA-640	$20,000	Factored	November 2, 2005
RSA-704	$30,000	Not Factored	
RSA-768	$50,000	Factored	December 12, 2009
RSA-896	$75,000	Not Factored	
RSA-1024	$100,000	Not Factored	
RSA-1536	$150,000	Not Factored	
RSA-2048	$200,000	Not Factored	

Franke, Kleinjung, etc

Factorization of RSA-768

- http://eprint.iacr.org/2010/006
- RSA-768: (232 digits)

123018668453011775513049495838496272077285356959533479219732
245215172640050726365751874520219978646938995647494277406384
592519255732630345373154826850791702612214291346167042921431
160222124047927473779408066535141959745985690214341 3

- Factorization: (Both factors have 384 bits and 116 digits)

334780716989568987860441698482126908177047949837137685689124
313889828837938780022876147116525317430877378144679994 89
×
367460436667995904282446337996279526322791581643430876426760
322838157396665112792333734171433968102700927987363089 17

Smooth Numbers

- A ***factor base*** $F = \{p_1, p_2, \ldots, p_m\}$ consists of primes
- A number is ***smooth*** over F if all of its prime factors belong to F
- Try to find smooth numbers $r_i = p_1^{e_1} p_2^{e_2} p_3^{e_3} \ldots p_m^{e_m}$ and record e_i in the vectors $v_i = (e_1, e_2, e_3, \ldots\ldots, e_m)$
- With enough smooth numbers, we can find a_i (= 0 or 1) by linear algebra such that

$$\sum a_i v_i \equiv (0,0,\ldots,0) \pmod 2 \Rightarrow \prod r_i^{a_i} \text{ becomes a square}$$

Smooth Numbers

- **Example**
- $F = \{2, 3, 5, 7\}$ is a factor base
- Some smooth numbers over F : $r_1 = 105 = 3\times5\times7$, $r_2 = 140 = 2^2\times5\times7$, $r_3 = 392 = 2^3\times7^2$, $r_4 = 588 = 2^2\times3\times7^2$
- Corresponding exponent vectors: $v_1 = (0, 1, 1, 1)$, $v_2 = (2, 0, 1, 1)$, $v_3 = (3, 0, 0, 2)$, $v_4 = (2, 1, 0, 2)$
- $1\,v_1 + 1\,v_2 + 0\,v_3 + 1\,v_4 = (4, 2, 2, 4) \equiv (0, 0, 0, 0) \pmod 2$
- We get a square: $r_1 r_2 r_4 = 2^4 \times 3^2 \times 5^2 \times 7^4$
- Recall: The goal is to find $x^2 \equiv y^2 \pmod N$
- The question then becomes finding many smooth numbers over a factor base F

Quadratic Sieve

- To factor N, define $g(x) = x^2 - N$
- x is an integer between $\sqrt{N}$ and $\sqrt{2N}$
- note that $g : \boldsymbol{Z} \rightarrow \boldsymbol{Z}_N$ preserves multiplications
- We want to find enough smooth $g(x_i)$
- $p \mid g(x_i)$ implies $p \mid g(x_i + p)$
- If $p \mid (x_i^2 - N)$, then $0 \equiv x_i^2 - N \equiv x_i^2 - N + 2px_i + p^2 \equiv (x_i + p)^2 - N \pmod p$
- "Sieve" $g(x_i)$ with every prime p in factor base

Quadratic Sieve

- Record $g(x_i)$ in an array G for each x_i
- If $p \mid g(x_i)$, $G[x_i] := G[x_i] + \lfloor \log p \rfloor$
  - integer addition for speed
- After sieving, check the smoothness of $g(x_i)$ whose $G[x_i] \geq$ chosen threshold

	2	3	5	7	4	9
$g(m+71)$	√					
$g(m+72)$		√	√			√
$g(m+73)$	√				√	
$g(m+74)$				√		
$g(m+75)$	√	√				
$g(m+76)$						
$g(m+77)$	√		√		√	
$g(m+78)$		√				
$g(m+79)$	√					
$g(m+80)$						
$g(m+81)$	√	√		√	√	√
$g(m+82)$			√			
$g(m+83)$	√					
$g(m+84)$		√				
$g(m+85)$	√				√	

Number Field

- Choose irreducible $f(x) = c_d x^d + c_{d-1} x^{d-1} + \ldots + c_1 x + c_0$ and $g(x) = e_1 x + e_0$ such that $f(m) \equiv g(m) \equiv 0 \pmod N$ for some integer m
- Let α be a complex root of $f(x)$
  - $\boldsymbol{Q}(\alpha)$ is a finite field extension of $\boldsymbol{Q}$
  - $\boldsymbol{Q}(\alpha)$ is the ***number field*** associated to α
- Define a ring homomorphism φ: $\boldsymbol{Z}[\alpha] \rightarrow \boldsymbol{Z}_N$ by $\varphi(\alpha) \equiv m \pmod N$
  - φ is a homomorphism since $f(\alpha) = 0$ and $f(m) \equiv 0 \pmod N$
  - $\varphi(a - b\alpha) \equiv a - bm \pmod N$

Squares in Both Sides

- Suppose we find a set S consisting of relatively prime pairs (a, b) satisfying both:

$$\prod_{(a,b)\in S}(a - b\alpha) = \beta^2 \quad \text{for some } \beta \in Z[\alpha]$$

(called the **algebraic side**)

$$\prod_{(a,b)\in S}(a - bm) = y^2 \quad \text{for some } y \in Z$$

(called the **rational side**)

- Let $x = \varphi(\beta)$, then we have $x^2 = \varphi(\beta)^2 = \varphi(\beta^2)$
  $= \varphi(\prod (a - b\alpha)) \equiv \prod (a - bm) = y^2 \pmod N$
- Hence N is factored with probability ≥ ½

Major Steps of GNFS

1. Polynomial Selection
   - Find 'good' polynomials to speed up sieving
2. Sieving
   - Find sufficiently many relations to produce a matrix
3. Matrix Reduction
   - Find linear dependencies mod 2 among the rows of the matrix
4. Square Root
   - Calculate a square root in the number field for each dependency, until the factorization is found