The Computational Challenges of public-key cryptography

James Davenport
Dept. {Computer,Mathematical} Science{,s}
University of Bath
J.H.Davenport@bath.ac.uk

Public Key Cryptography

Two main methods:

RSA based on factoring large numbers (*not* large primes);

Diffie-Hellman based on discrete logarithms.

As code-breaking computations, require similar resources and techniques; though the underlying number theory is different.

Focus on Diffie-Hellman

- older (in the public arena)
- easier (in terms of the number theory involved)
- you all use it (`ssh`, `https`).

Diffie-Hellman Key exchange
A and B wish to have a private conversation

1. A and B (and the rest of the world) agree p, a prime, and x.
2. A thinks of a, and sends $x^a \pmod p$ to B.
3. B thinks of b, and sends $x^b \pmod p$ to A.
4. A and B compute $(x^a)^b = (x^b)^a \pmod p$.

This is then a *shared private key*.

Discrete Logarithms (Indices) [3]

If $y = x^a \pmod p$, then say a is $\log_x y \pmod p$. (In general, we drop the $_x$, p).

What does Eve do?

1. Observe $y = x^a$ and $z = x^b$;
2. Compute $a = \log_x y$;
3. Compute $(x^b)^a = z^a \pmod p$.

How to compute Discrete Logarithms?

Sequential search $O(p)$,

Improved search $O(\sqrt{p})$ [4],

Index Calculus $O(e^{\sqrt{\log p \log\log p}})$

(known as $L_{\frac{1}{2}}(p)$).

Index Calculus: An example mod 17

$\log_3(6) = 15$, $\log_3(10) = 3$, $\log_3(15) = 6$.

$$\begin{pmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix} \begin{pmatrix} \log_3 2 \\ \log_3 3 \\ \log_3 5 \end{pmatrix} = \begin{pmatrix} 15 \\ 3 \\ 6 \end{pmatrix}$$

$$\begin{pmatrix} \log_3 2 \\ \log_3 3 \\ \log_3 5 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix}^{-1} \begin{pmatrix} 15 \\ 3 \\ 6 \end{pmatrix} = \begin{pmatrix} 14 \\ 1 \\ 5 \end{pmatrix}$$

Modulo 16, and this matrix is singular (det=2).

So, in practice

1. How do we find the equations for log(∏primes)?
2. How do we solve for log(prime)?
3. Given these, how do we find a log?

If a number is a product of primes less than B, we say it is B-smooth.

Finding the equations

- Pick a and hope $x^a \pmod p$ is B-smooth.
- Pick A, write $x^a \equiv \frac{c}{d} \pmod p$, and hope *both* c and d are B-smooth.
- Allow one larger prime ($< B^2$): two equations with the same large prime give one without it.
- Allow two larger primes: need to chase cycles in graph of large primes.

"Pathologically parallel"

- It doesn't matter much if we miss a few equations.
- It doesn't matter much if we double-count.
- We can do it by e-mail [2, "Keeping track of the contributors was the hardest part"].

Solving the equations

The matrix is

- Very large (literally millions of rows)
- Very sparse (< 10 nonzeros/row)
- Overdetermined (too many rows)
- underdetermined (some columns zero)
- To be solved modulo $p - 1$

So what can we do? We know

Good Lanczos (and Wiedemann) work over finite fields.

Bad They generally need n iterations to produce the answer (no concept of convergence here!).

Very Bad Time is therefore $O(n^2 e)$, where e=#nonzeros/row.

The way forward [1, many]

A two-phase approach.

1. Structured Gaussian Elimination (SGE) while $n^2 e$ is decreasing in the "light" part of the matrix.
2. Lanczos in the resulting system

   (followed by back-substitution).

Parallelising this stage

- Parallelise the SGE: hard to think of an 'optimal' (i.e. mirroring the sequential) was of doing it.
- Parallel sparse Lanczos:

  You tell me!

Finding $\log(y)$
(No equivalent in factoring)

1. Take a until yx^a is B-smooth, then read off $\log(yx^a) = a + \log(y)$.
2. Take a until $yx^a \equiv c/d$ with c, d B-smooth, then read off $\log(yx^a) = a + \log(y)$.
3. As above, but allowing large primes whose logs we found.
4. Also allow large primes whose log we didn't determine initially, but now can.

Factoring $(10^{211} - 1)/9$

10.9 CPU years on 125 workstations produce 56394064 relations.

Reduced to a matrix with 4820249 rows and 4895741 columns with 234162626 1's, i.e., an average of 48.6 1's per row.

The Block Lanczos program took 121 hours on the Cray C90 in order to find 64 dependencies.

The square root program, finally, needed 15.5 hours on one CPU of CWI's SGI Origin 2000, and three dependencies to find the two prime factors:

References

[1] A.J. Holt and J.H. Davenport. Resolving Large Prime(s) Variants for Discrete Logarithm Computation. In P.G. Farrell, editor, *Proceedings 9th IMA Conf. Coding and Cryptography*, pages 207–222, 2003.

[2] A.K. Lenstra and M.S. Manasse. Factoring by Electronic Mail. In J.-J. Quisquater and J. Vandewalle, editors, *Proceedings EUROCRYPT '89 Springer Lecture Notes in Computer Science vol. 434*, pages 355–371, 1990.

[3] A.M. Odlyzko. Discrete Logarithms: the Past and the Future. *To appear in Designs*, 1999.

[4] J.M. Pollard. Monte Carlo methods for index computations mod p. *Math. Comp.*, 32:918–924, 1978.