# Discrete Logarithms using the Index Calculus Method

Aaron Neil Bradford

January 27, 2006

# The Discrete Logarithm Problem

- Given $a$, $b \in Z_n$ find an integer $s$ such that

$$b = a^s \pmod{n}$$

# Maple's mlog()

- Suppose that

$$n = \prod p_i^{e_i}$$

Then Maple will solve a number of instances of the DLP modulo each $p_i$ and combine them to solve the problem modulo $n$. This is the Pohlig-Hellman algorithm.

# Maple's mlog() cont.

- To solve each instance of the DPL modulo $p_i$, Maple uses Shanks' Baby-Step, Giant-Step algorithm.
- Requires $O(p_i^{1/2})$ time and space. If $p_i$ is large this can take a lot of resources.
- Other basic methods for solving the DLP can be substituted for Shanks' method in the Pohlig-Hellman algorithm.

# Modifying Maple’s _mlogprime()

- When $p_i$ is large (about 32 bits or more) we would like to use instead the Index Calculus method to solve the DLP.
- We construct a number of linear congruences modulo $p_i – 1$ and use these to find the exponent $s$.

# Modifying Maple's _mlogprime() cont.

- Notice that if

$$a^c = r / t \pmod{p_i}$$

for some value $c$ where $r$ and $t$ are

$$r = \prod q_i^{r_i}$$
$$t = \prod q_i^{t_i}$$

and the $q_i$ are "small" primes, then

$$c = \sum r_i \log_a q_i - \sum t_i \log_a q_i \pmod{p_i - 1}$$

# Optimising this method

- The limiting factor in this method is calculating *r* and *t*, and deciding whether or not each is divisible by only “small” primes (if *r* and *t* are such, they are called *smooth*).
- To calculate *r* and *t* we use a slight modification of the Extended Euclidean algorithm.
  - We can get multiple (*r*, *t*) pairs from each run

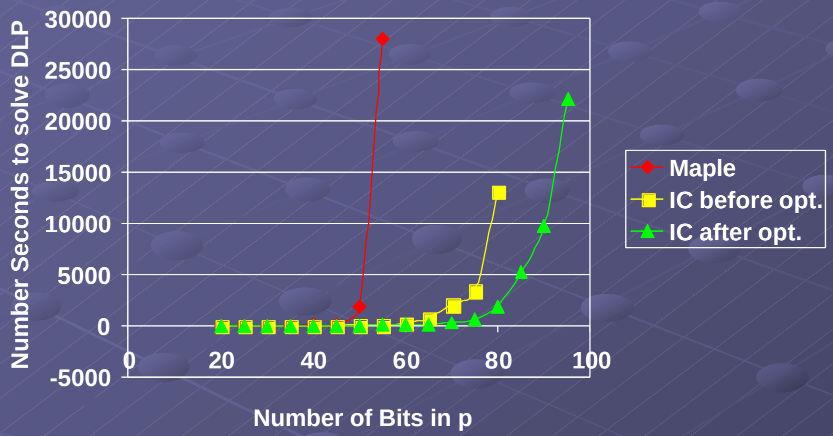
# Optimising this method cont.

- The strategies that we employed to optimise the smoothness check on $r$ and $t$ were
  - If $r$ is not smooth then we need not check $t$.
  - It is likely that $r$ (or $t$) is divisible by 2, 3, …, 29, so we trial divide these immediately.
  - It is unlikely that $r$ (or $t$) is divisible by 31, 37, …, so we check the gcd of the product of these against $r$ before trial dividing these primes.

# Optimising this method cont.

- 29 is the $10^{th}$ prime; we also tested ending the automatically divided primes with the $6^{th}$, $8^{th}$, $12^{th}$ and $15^{th}$ primes, but 10 worked best.
- We bundled the rest of the “small” primes in groups of 20 for computing the gcd of the product of each group with $r$.
- We tested groups of 10, 20, 30, 40 and 50, as well as groups of increasing size (i.e. first group 20, second group 30, etc.), but fixing the group size at 20 worked best.

# Timings




| Bits in p | Maple | IC before opt. | IC after opt. |
|---|---|---|---|
| 20 | 0.009 | 0.418 | 0.170 |
| 25 | 0.038 | 0.364 | 0.314 |
| 30 | 0.324 | 1.027 | 0.587 |
| 35 | 2.744 | 1.812 | 1.632 |
| 40 | 21.539 | 3.856 | 2.970 |
| 45 | 164.565 | 9.556 | 4.572 |
| 50 | 1815.272 | 26.516 | 8.603 |
| 55 | 27978.411* | 79.474 | 18.345 |
| 60 | | 221.718 | 45.756 |
| 65 | | 708.589 | 114.607 |
| 70 | | 2058.332 | 303.185 |
| 75 | | 3449.033 | 606.303 |
| 80 | | 13103.594 | 1875.977 |
| 85 | | | 5278.757 |
| 90 | | | 9690.946 |
| 95 | | | 22124.051 |