Fast constant-time gcd computation and modular inversion

Daniel J. Bernstein

Bo-Yin Yang

Paper coming soon.
Implementations coming soon from larger group.

Normally compute $1/x$ in $\mathbf{F}_p$ as x^{p-2}.

$n^{3+o(1)}$ bit ops	using schoolbook multiplication
$n^{2.58\ldots+o(1)}$ bit ops	using Karatsuba multiplication
$n^{2+o(1)}$ bit ops	using FFT-based multiplication

Normally compute $1/x$ in $\mathbf{F}_p$ as x^{p-2}.

$n^{3+o(1)}$ bit ops	using schoolbook multiplication
$n^{2.58\ldots+o(1)}$ bit ops	using Karatsuba multiplication
$n^{2+o(1)}$ bit ops	using FFT-based multiplication

Why not use extensions of Euclid's algorithm?

$n^{2+o(1)}$ bit ops	using schoolbook multiplication
$n^{1.58\ldots+o(1)}$ bit ops	using Karatsuba multiplication
$n^{1+o(1)}$ bit ops	using FFT-based multiplication

Normally compute $1/x$ in $\mathbf{F}_p$ as x^{p-2}.

$n^{3+o(1)}$ bit ops	using schoolbook multiplication
$n^{2.58\ldots+o(1)}$ bit ops	using Karatsuba multiplication
$n^{2+o(1)}$ bit ops	using FFT-based multiplication

Why not use extensions of Euclid's algorithm?

$n^{2+o(1)}$ bit ops	using schoolbook multiplication
$n^{1.58\ldots+o(1)}$ bit ops	using Karatsuba multiplication
$n^{1+o(1)}$ bit ops	using FFT-based multiplication

Usual answer: Need constant-time algorithm.

Normally compute $1/x$ in $\mathbf{F}_p$ as x^{p-2}.

$n^{3+o(1)}$ bit ops	using schoolbook multiplication
$n^{2.58\ldots+o(1)}$ bit ops	using Karatsuba multiplication
$n^{2+o(1)}$ bit ops	using FFT-based multiplication

Why not use extensions of Euclid's algorithm?

$n^{2+o(1)}$ bit ops	using schoolbook multiplication
$n^{1.58\ldots+o(1)}$ bit ops	using Karatsuba multiplication
$n^{1+o(1)}$ bit ops	using FFT-based multiplication

Usual answer: Need constant-time algorithm.

Our algorithm is constant-time; $n^{1+o(1)}$ bit ops; simpler than previous variable-time algorithms. No division subroutine between recursive calls.