High-speed cryptography, part 4: fast multiplication and its applications

Daniel J. Bernstein
University of Illinois at Chicago &
Technische Universiteit Eindhoven

Survey paper:
cr.yp.to/papers.html#multapps

Integer-factorization bottleneck:
Given sequence of numbers,
find nonempty subsequence
with square product.
e.g. given $6, 7, 8, 10, 15$,
discover $6 \cdot 10 \cdot 15 = 30^2$.

Discrete-log bottleneck:
Given sequence of numbers,
find 1 as nontrivial
product of powers.
e.g. given $6, 7, 8, 10, 15$,
discover $6^3 7^0 8^{-2} 10^3 15^{-3} = 1$.

More generally: find kth power.

Two very common
cryptographic bottlenecks:
Multiply large polynomials;
multiply large integers.

Two very common
cryptographic bottlenecks:
Multiply large polynomials;
multiply large integers.

All of these computations
can be performed in
essentially *linear* time.

Two very common
cryptographic bottlenecks:
Multiply large polynomials;
multiply large integers.

All of these computations
can be performed in
essentially *linear* time.

Do real applications
reach large enough sizes
to benefit from these techniques?
In cryptanalysis, definitely.
In cryptography, sometimes:
Gaudry–Schost Kummer surface;
McBits; many more examples.

The fast Fourier transform

Use $(c_0, c_1, \ldots, c_{n-1}) \in \mathbf{C}^n$
to represent $f = \sum_j c_j x^j \in \mathbf{C}[x]$.

Summary of representation size:
"f has n coeffs". Warning:
f does not determine n.

$f = f_0(x^2) + x f_1(x^2)$ where
$(c_0, c_2, \ldots) \in \mathbf{C}^{\lceil n/2 \rceil}$,
$(c_1, c_3, \ldots) \in \mathbf{C}^{\lfloor n/2 \rfloor}$
represent f_0, f_1 respectively.

$\mathbf{C}[x]$-morphism $y \mapsto x^2$
from $\mathbf{C}[x][y]$ to $\mathbf{C}[x]$
maps $f_0(y) + x f_1(y)$ to f.

Quickly evaluate $f(\alpha), f(-\alpha)$
by evaluating $f_0(\alpha^2)$; $f_1(\alpha^2)$;

$$f(\alpha) = f_0(\alpha^2) + \alpha f_1(\alpha^2);$$
$$f(-\alpha) = f_0(\alpha^2) - \alpha f_1(\alpha^2).$$

Evaluate $f(\alpha)$ for, e.g.,
all $\alpha \in \mathbf{C}$ with $\alpha^{1024} = 1$
by evaluating $f_0(\beta)$, $f_1(\beta)$
for all $\beta \in \mathbf{C}$ with $\beta^{512} = 1$;
plus 1024 adds, 512 mults.

Apply this recursively $\Rightarrow$
$n \lg n$ adds, $(n/2) \lg n$ mults
to evaluate n-coeff f
for all $\alpha \in \mathbf{C}$ with $\alpha^n = 1$
if n is a power of 2.

Another view of the FFT

If $f \in \mathbf{C}[x]$ and
$f \bmod x^4 - 1 =$
$c_0 + c_1 x + c_2 x^2 + c_3 x^3$ then
$f \bmod x^2 - 1 =$
$(c_0 + c_2) + (c_1 + c_3)x$,
$f \bmod x^2 + 1 =$
$(c_0 - c_2) + (c_1 - c_3)x$.

$\mathbf{C}[x]$-morphism $\mathbf{C}[x]/(x^4 - 1) \hookrightarrow\!\!\!\!\rightarrow$
$\mathbf{C}[x]/(x^2 - 1) \oplus \mathbf{C}[x]/(x^2 + 1)$
maps $c_0 + c_1 x + c_2 x^2 + c_3 x^3$ to
$((c_0 + c_2) + (c_1 + c_3)x,$
$(c_0 - c_2) + (c_1 - c_3)x)$.

If $f \in \mathbf{C}[x]$ and

$f \bmod x^{2n} - \alpha^2 =$

$c_0 + c_1 x + \cdots + c_{2n-1} x^{2n-1}$ then

$f \bmod x^n - \alpha =$

$$(c_0 + \alpha c_n) + (c_1 + \alpha c_{n+1})x + (c_2 + \alpha c_{n+2})x^2 + \cdots,$$

$f \bmod x^n + \alpha =$

$$(c_0 - \alpha c_n) + (c_1 - \alpha c_{n+1})x + (c_2 - \alpha c_{n+2})x^2 + \cdots.$$

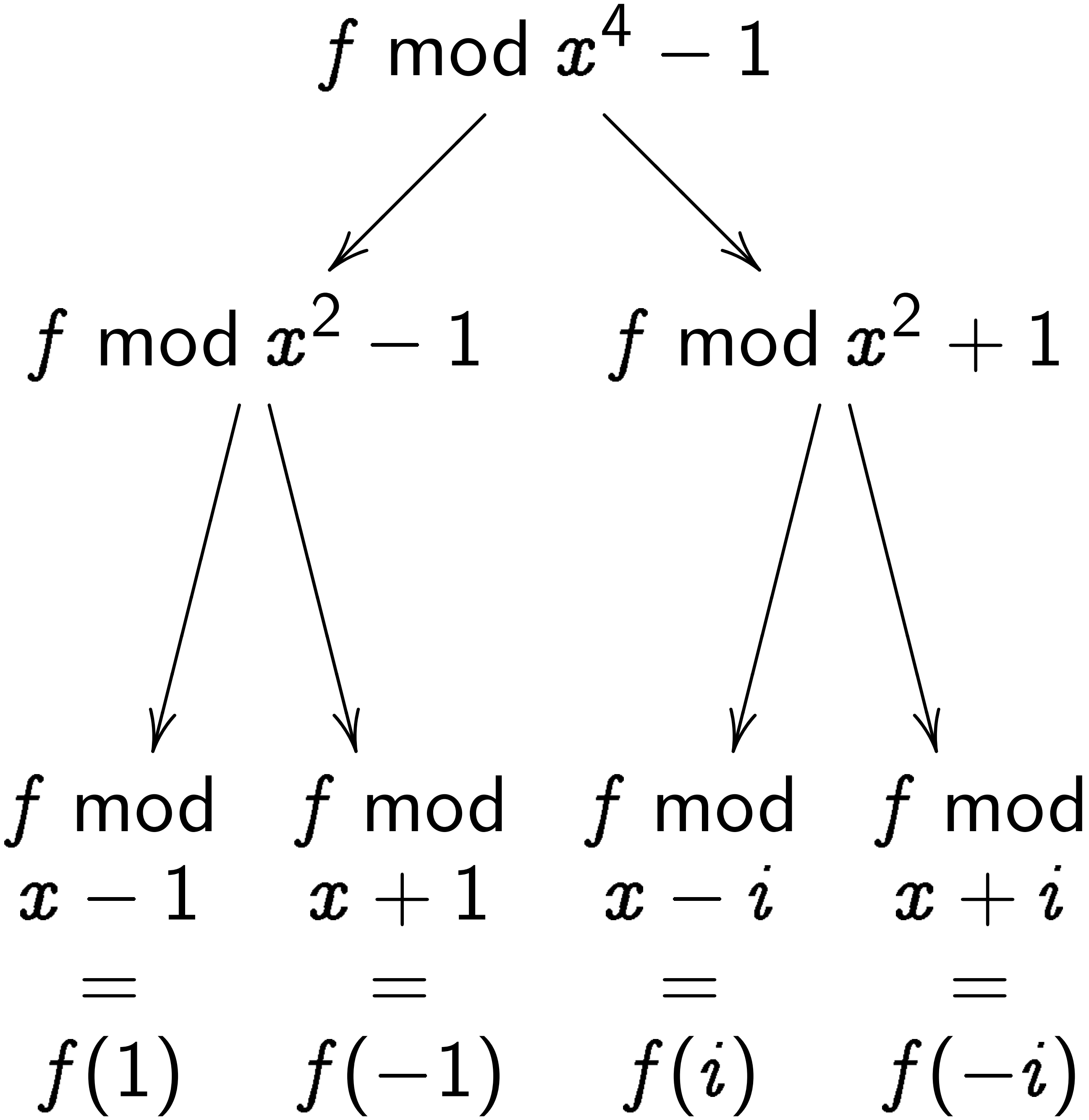
Given $c_0, c_1, \ldots, c_{2n-1} \in \mathbf{C}$,

use n mults, $2n$ adds to compute

$c_0 + \alpha c_n, c_1 + \alpha c_{n+1}, \ldots,$

$c_0 - \alpha c_n, c_1 - \alpha c_{n+1}, \ldots.$

Apply this recursively:



(basic FFT idea: 1866 Gauss;
this view: 1972 Fiduccia)

1966 Sande, 1966 Stockham:
Can very quickly multiply
in $\mathbf{C}[x]/(x^n - 1)$ or $\mathbf{C}[x]$ or $\mathbf{R}[x]$
by mapping $\mathbf{C}[x]/(x^n - 1)$ to $\mathbf{C}^n$.

Given $f, g \in \mathbf{C}[x]/(x^n - 1)$:
compute fg as $T^{-1}(T(f)T(g))$
using $T : \mathbf{C}[x]/(x^n - 1) \hookrightarrow\!\!\!\!\rightarrow \mathbf{C}^n$.
Compute T quickly by the FFT.

Given $f, g \in \mathbf{C}[x]$, $\deg fg < n$:
compute fg from
its image in $\mathbf{C}[x]/(x^n - 1)$.

1966 Sande, 1966 Stockham:
Can very quickly multiply in $\mathbf{C}[x]/(x^n - 1)$ or $\mathbf{C}[x]$ or $\mathbf{R}[x]$ by mapping $\mathbf{C}[x]/(x^n - 1)$ to $\mathbf{C}^n$.

Given $f, g \in \mathbf{C}[x]/(x^n - 1)$: compute fg as $T^{-1}(T(f)T(g))$ using $T : \mathbf{C}[x]/(x^n - 1) \hookrightarrow\!\!\!\!\rightarrow \mathbf{C}^n$. Compute T quickly by the FFT.

Given $f, g \in \mathbf{C}[x]$, $\deg fg < n$: compute fg from its image in $\mathbf{C}[x]/(x^n - 1)$.

Later authors: Replace $\mathbf{C}$ with, e.g., $R = \mathbf{Z}/(3 \cdot 2^{41} + 1)$; 23 has order 2^{41} in R^*.

Multiplication and division

Given $r, s \in \mathbf{Z}$, can compute rs
in time $\leq b(\lg b)^{1+o(1)}$
where b is number of input bits.

(1971 Pollard; independently
1971 Nicholson; independently
1971 Schönhage Strassen)

Also time $\leq b(\lg b)^{1+o(1)}$
where b is number of input bits:
Given $r, s \in \mathbf{Z}$ with $s \neq 0$,
compute $\lfloor r/s \rfloor$ and $r \bmod s$.

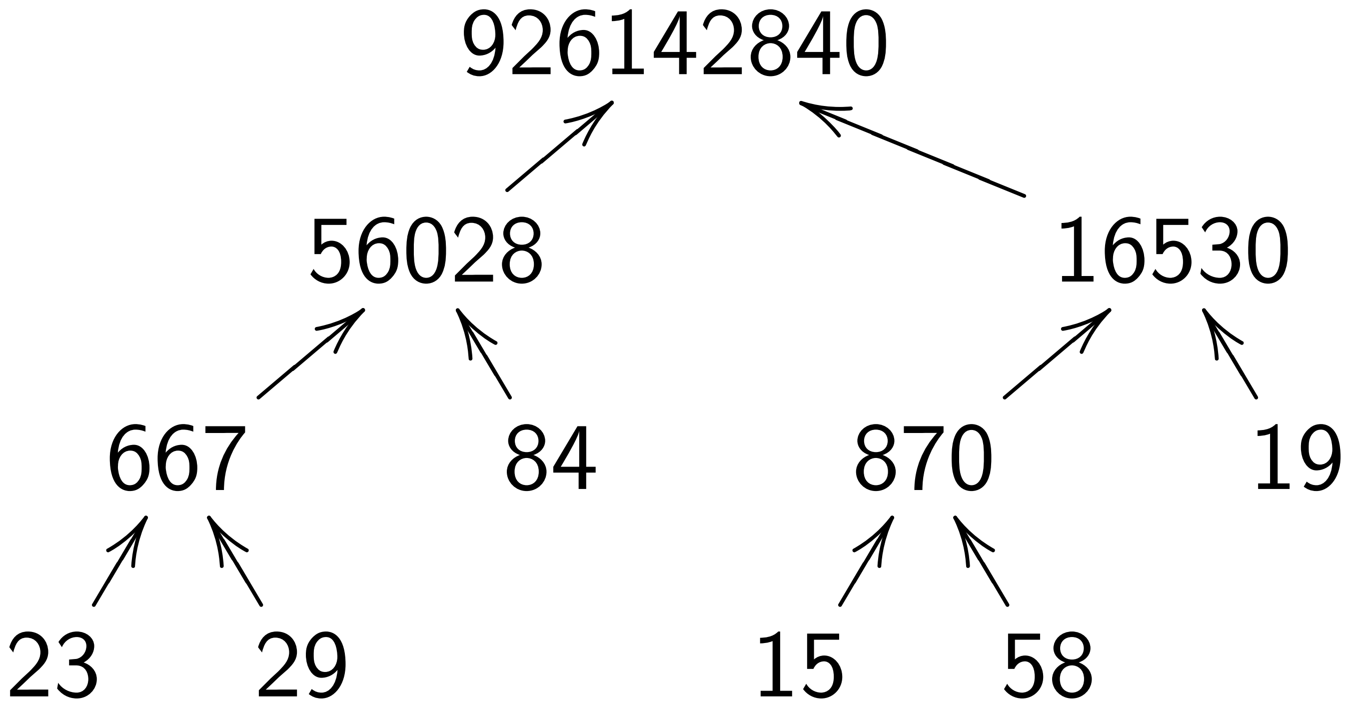
(reduction to product:
1966 Cook)

Product trees

Time $\leq b(\lg b)^{2+o(1)}$
where b is number of input bits:
Given $x_1, x_2, \ldots, x_n \in \mathbf{Z}$,
compute $x_1 x_2 \cdots x_n$.

Actually compute
product tree of $x_1, x_2, \ldots, x_n$.
Root is $x_1 x_2 \cdots x_n$.
Has left subtree if $n \geq 2$:
product tree of $x_1, \ldots, x_{\lceil n/2 \rceil}$.
Also right subtree if $n \geq 2$:
product tree of $x_{\lceil n/2 \rceil + 1}, \ldots, x_n$.

e.g. tree for 23, 29, 84, 15, 58, 19:



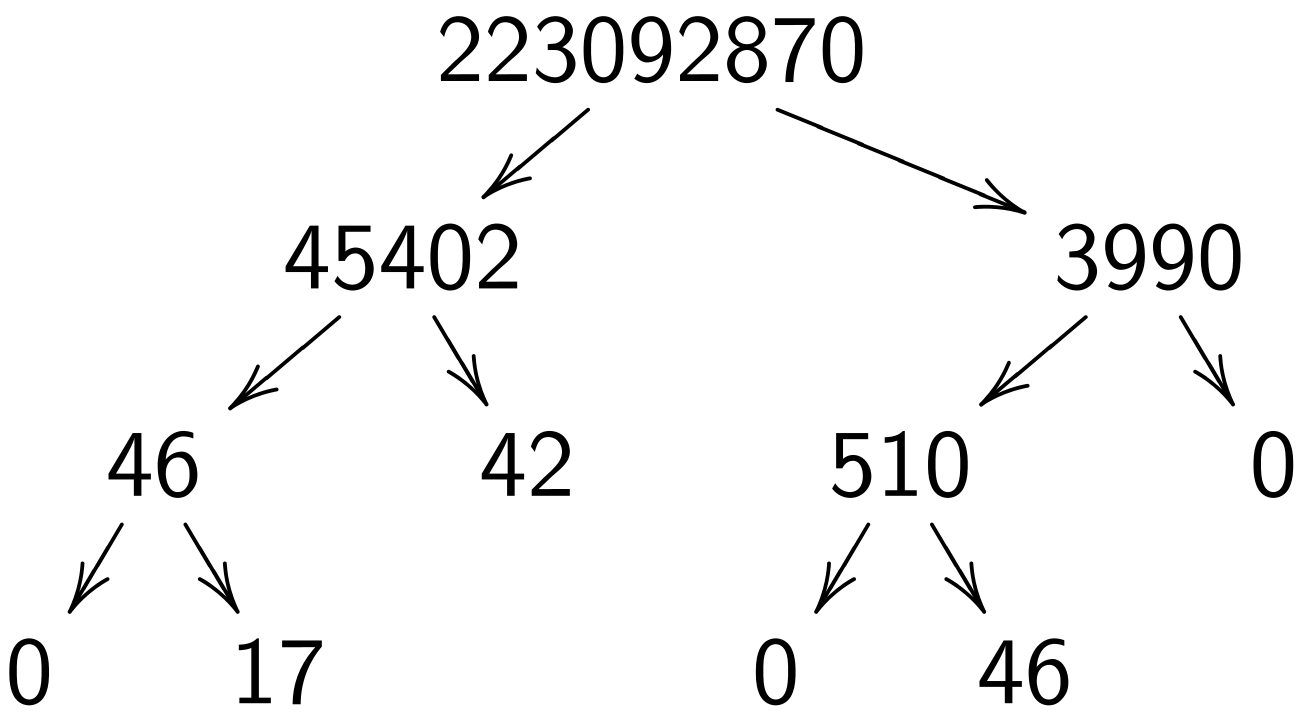
Tree has $\le (\lg b)^{1+o(1)}$ levels.
Each level has $\le b(\lg b)^{0+o(1)}$ bits.

Obtain each level
in time $\le b(\lg b)^{1+o(1)}$
by multiplying lower-level pairs.

<u>Remainder trees</u>

Remainder tree
of $r, x_1, x_2, \ldots, x_n$ has
one node $r \bmod t$ for each node t
in product tree of $x_1, x_2, \ldots, x_n$.

e.g. remainder tree of
$223092870, 23, 29, 84, 15, 58, 19$:



Time $\leq b(\lg b)^{2+o(1)}$:
Given $r \in \mathbf{Z}$ and nonzero $x_1, \ldots, x_n \in \mathbf{Z}$, compute remainder tree of $r, x_1, \ldots, x_n$.

In particular, compute $r \bmod x_1, \ldots, r \bmod x_n$.

In particular, see which of $x_1, \ldots, x_n$ divide r.

(1972 Moenck Borodin, for "single precision" x_i's, whatever exactly that means)

Small primes, union

Time $\leq b(\lg b)^{2+o(1)}$:
Given $x_1, x_2, \ldots, x_n \in \mathbf{Z}$ and finite set $Q \subseteq \mathbf{Z} - \{0\}$, compute $\{p \in Q : x_1 x_2 \cdots x_n \bmod p = 0\}$.

In particular, when p is prime, see whether p divides any of $x_1, x_2, \ldots, x_n$.

Algorithm:

1. Use a product tree to compute $r = x_1 x_2 \cdots x_n$.
2. Use a remainder tree to see which $p \in Q$ divide r.

Small primes, separately

Time $\leq b(\lg b)^{3+o(1)}$:
Given $x_1, x_2, \ldots, x_n \in \mathbf{Z}$ and finite set Q of primes, compute $\{p \in Q : x_1 \bmod p = 0\}$, $\ldots$, $\{p \in Q : x_n \bmod p = 0\}$.
(2000 Bernstein)

Algorithm for $n \geq 1$:

1. Replace Q with $\{p \in Q : x_1 \cdots x_n \bmod p = 0\}$.
2. If $n = 1$, print Q and stop.
3. Recurse on $x_1, \ldots, x_{\lceil n/2 \rceil}, Q$.
4. Recurse on $x_{\lceil n/2 \rceil + 1}, \ldots, x_n, Q$.

Factor $2543, 6766, 8967, 7598$
over $\{\underline{2}, \underline{3}, 5, \underline{7}, 11, 13, \underline{17}\}$

$2543, 6766$
over
$\underline{2}, 3, 7, \underline{17}$

$8967, 7598$
over
$\underline{2}, \underline{3}, \underline{7}, 17$

2543
over
$2, 17$

6766
over
$\underline{2}, \underline{17}$

8967
over
$2, \underline{3}, \underline{7}$

7598
over
$\underline{2}, 3, 7$

Each level has $\leq b(\lg b)^{0+o(1)}$ bits.

Exponents of a small prime

Time $\le b(\lg b)^{2+o(1)}$:
Given nonzero $p, x \in \mathbf{Z}$,
find $e, p^e, x/p^e$ with maximal e.

Algorithm:

1. If $x \bmod p \ne 0$:
   Print $0, 1, x$ and stop.
2. Find $f, (p^2)^f, r = (x/p)/(p^2)^f$
   with maximal f.
3. If $r \bmod p = 0$: Print
   $2f + 2, (p^2)^f p^2, r/p$ and stop.
4. Print $2f + 1, (p^2)^f p, r$.

Exponents of small primes

Time $\le b(\lg b)^{3+o(1)}$:
Given finite set Q of primes and nonzero $x \in \mathbf{Z}$, find maximal $e, \prod_{p\in Q} p^{e(p)}, x/\prod_{p\in Q} p^{e(p)}$.

Algorithm:

1. Replace Q with $\{p \in Q : x \bmod p = 0\}$.
2. Find maximal f, s, r with $s = \prod (p^2)^{f(p^2)}$, $r = (x/\prod p)/s$.
3. Find $T = \{p \in Q : r \bmod p = 0\}$.
4. Output $e, s\prod_{p\in T} p, r/\prod_{p\in T} p$ where $e(p) = 2f(p^2) + [p \in T]$.

Smooth parts, old approach

Time $\le b(\lg b)^{3+o(1)}$:
Given nonzero $x_1, x_2, \ldots, x_n \in \mathbf{Z}$
and finite set Q of primes,
compute Q-smooth part of x_1,
Q-smooth part of x_2, ...,
Q-smooth part of x_n.

Q-smooth means product
of powers of elements of Q.

Q-smooth part means
largest Q-smooth divisor.
In particular, see which of
$x_1, x_2, \ldots, x_n$ are smooth.

Algorithm:

1. Find $Q_1 = \{p : x_1 \bmod p = 0\}$,
   $\ldots, Q_n = \{p : x_n \bmod p = 0\}$.
2. For each i separately:
   Find maximal e, s, r with
   $s = \prod_{p \in Q_i} p^{e(p)}$, $r = x_i/s$.
   Print s.

e.g. factor $2543, 6766, 8967, 7598$
over $\{2, 3, 5, 7, 11, 13, 17\}$:
2543 over $\{\}$, smooth part 1;
6766 over $\{2, 17\}$, smooth part 34;
8967 over $\{3, 7\}$, smooth part 147;
7598 over $\{2\}$, smooth part 2.

Smooth multiplicative dependencies

Recall cryptanalytic bottleneck:
find kth power nontrivially as
product of powers of
$x_1, x_2, \ldots, x_n$.

Choose y; imagine $y = 2^{40}$.
Define Q as set of primes $\le y$.
See which of $x_1, x_2, \ldots, x_n$
are y-smooth, i.e., Q-smooth.
Know their factorizations.
Do linear algebra over $\mathbf{Z}/k$
on the exponent vectors.

Smooth parts, new approach

Given nonzero $x_1, x_2, \ldots, x_n \in \mathbf{Z}$ and finite set Q of primes:
Time typically $\le b(\lg b)^{2+o(1)}$ to obtain smooth parts of x's.
(2004 Franke Kleinjung Morain Wirth, in ECPP context)

Algorithm:
Compute $r = \prod_{p \in Q} p$.
Compute $r \bmod x_1, \ldots, r \bmod x_n$.
For each i separately:
Replace x_i by $x_i / \gcd\{x_i, r \bmod x_i\}$ repeatedly until gcd is 1.

Slight variant (2004 Bernstein):
Time always $\le b(\lg b)^{2+o(1)}$.

Compute smooth part of x_i as
$\gcd\{x_i, (r \bmod x_i)^{2^k} \bmod x_i\}$
where $k = \lceil \lg \lg x_i \rceil$.

Subroutine: Computing gcd
takes time $\le b(\lg b)^{2+o(1)}$.
(1971 Schönhage;
core idea: 1938 Lehmer;
$b(\lg b)^{5+o(1)}$: 1971 Knuth)

Or, to see if x_i is smooth,
see if $(r \bmod x_i)^{2^k} \bmod x_i = 0$.

Minor problem: New algorithm finds the smooth numbers but doesn't factor them.

Minor problem: New algorithm
finds the smooth numbers
but doesn't factor them.

Solution:
Feed the smooth numbers
to the old algorithm.
Very few smooth numbers,
so this is very fast.

Bottom line for cryptanalysis:
time per input number to
find and factor smooth numbers
has dropped by $(\lg b)^{1+o(1)}$.

Is smooth the right question?

After finding smooth numbers, do first step of linear algebra: Throw away primes that appear only once; throw away numbers with those primes; repeat until stable.

Don't want *all* smooth numbers. Want smooth numbers only if they are built from primes that divide the *other* numbers.

An alternate approach

Given nonzero $x_1, x_2, \ldots, x_n \in \mathbf{Z}$:
Compute $r = x_1 x_2 \cdots x_n$.
Compute $(r/x_1) \bmod x_1$, ...,
$(r/x_n) \bmod x_n$.
For each i separately: see if
$((r/x_i) \bmod x_i)^{2^k} \bmod x_i = 0$
where $k = \lceil \lg \lg x_i \rceil$.

Finds x_i iff all primes in x_i
are divisors of other x's.
Time $\le b(\lg b)^{2+o(1)}$.

(2004 Bernstein)

Compute $(r/x_1) \bmod x_1$, . . . ,
$(r/x_n) \bmod x_n$ by computing
$r \bmod x_1^2, \ldots, r \bmod x_n^2$.
(1972 Moenck Borodin)

Compute $(r/x_1) \bmod x_1$, . . . , $(r/x_n) \bmod x_n$ by computing $r \bmod x_1^2, \ldots, r \bmod x_n^2$. (1972 Moenck Borodin)

Problem: Recognizing the interesting x's is not enough; also need their factorizations.

Compute $(r/x_1) \bmod x_1, \ldots,$
$(r/x_n) \bmod x_n$ by computing
$r \bmod x_1^2, \ldots, r \bmod x_n^2$.
(1972 Moenck Borodin)

Problem: Recognizing the
interesting x's is not enough;
also need their factorizations.

Solution:
Again, very few of them.
Have ample time to
use rho method (1974 Pollard)
or use ECM (1987 Lenstra)
or factor into coprimes.

Factoring into coprimes

Time $\le b(\lg b)^{O(1)}$:
Given positive $x_1, x_2, \ldots, x_n$,
find coprime set Q
and complete factorization
of each x_i over Q.

(announced 1995 Bernstein;
journal version: 2005)

Immediately gives $b(\lg b)^{O(1)}$
for the other factoring problems.
Subsequent research: lg speedups,
constant-factor speedups, etc.

Typical application:
detecting multiplicative relations.

Does $91^{1952681} 119^{1513335} 221^{634643}$ equal $1547^{1708632} 6898073^{439346}$?

Each side has logarithm ≈ 19466590.674872.

Typical application:
detecting multiplicative relations.

Does $91^{1952681} 119^{1513335} 221^{634643}$ equal $1547^{1708632} 6898073^{439346}$?

Each side has logarithm ≈ 19466590.674872.

More generally:
What is kernel of $(a, b, c, d, e) \mapsto 91^a 119^b 221^c 1547^{-d} 6898073^{-e}$?

Kernel lets us find relations,
not just verify relations.

Factor into coprimes:

$91 = 7 \cdot 13$; $119 = 7 \cdot 17$;

$221 = 13 \cdot 17$; $1547 = 7 \cdot 13 \cdot 17$;

$6898073 = 7^4 \cdot 13^2 \cdot 17$.

$(a, b, c, d, e) \mapsto$
$91^a 119^b 221^c 1547^{-d} 6898073^{-e} =$
$7^{a+b-d-4e} 13^{a+c-d-2e} 17^{b+c-d-e}$.

Kernel is generated by
$(1, 1, 1, 2, 0)$ and $(3, 2, 0, 1, 1)$.

Factor into coprimes:

$91 = 7 \cdot 13$; $119 = 7 \cdot 17$;

$221 = 13 \cdot 17$; $1547 = 7 \cdot 13 \cdot 17$;

$6898073 = 7^4 \cdot 13^2 \cdot 17$.

$(a, b, c, d, e) \mapsto$
$91^a 119^b 221^c 1547^{-d} 6898073^{-e} =$
$7^{a+b-d-4e} 13^{a+c-d-2e} 17^{b+c-d-e}$.

Kernel is generated by
$(1, 1, 1, 2, 0)$ and $(3, 2, 0, 1, 1)$.

Factoring into coprimes
remains fast for larger numbers.
Factoring into primes does not.

Can apply same algorithms
in more generality: e.g.,
replace integers with polynomials.

Typical application:
Take a squarefree $g \in (\mathbf{Z}/2)[x]$.
What are g's irreducible divisors?

One answer: Find basis $h_1, h_2, \ldots$
for $\{h \in (\mathbf{Z}/2)[x] : (gh)' = h^2\}$
as a vector space over $\mathbf{Z}/2$.
Factor $g, h_1, h_2, \ldots$ into coprimes.
This list of coprimes contains
all irreducible divisors of g.

(1993 Niederreiter, 1994 Göttfert)

More examples, applications of factoring into coprimes: see 1890 Stieltjes; 1974 Collins; 1985 Kaltofen; 1985 Della Dora DiCrescenzo Duval; 1986 Bach Miller Shallit; 1986 von zur Gathen; 1986 Lüneburg; 1989 Pohst Zassenhaus; 1990 Teitelbaum; 1990 Smedley; 1993 Bach Driscoll Shallit; 1994 Ge; 1994 Buchmann Lenstra; 1996 Bernstein; 1997 Silverman; 1998 Cohen Diaz y Diaz Olivier; 1998 Storjohann; . . .

`cr.yp.to/coprimes.html`

Exercise: Given 2^{23} RSA keys, how would you check for primes shared among those keys?

2012 Heninger–Durumeric–Wustrow–Halderman, best-paper award at USENIX Security Symposium; 2012 Lenstra–Hughes–Augier–Bos–Kleinjung–Wachter, independent “Ron was wrong, Whit is right” paper, Crypto:

RSA keys on the Internet use such bad randomness that this does find factors!