High-speed Diffie-Hellman, part 1

D. J. Bernstein
University of Illinois at Chicago
Can quickly compute
$4^n \mod 2^{262} - 5081$
given $n \in \{0, 1, 2, \ldots, 2^{256} - 1\}$.

Similarly, can quickly compute
$4^{mn} \mod 2^{262} - 5081$ given $n$
and $4^m \mod 2^{262} - 5081$.

“Discrete-logarithm problem”:
given $4^n \mod 2^{262} - 5081$, find $n$.
Is this easy to solve?
Diffie-Hellman secret-sharing system using $p = 2^{262} - 5081$:

Alice’s secret key $m$

↓

Alice’s public key $4^m \mod p$

↓

{Alice, Bob}’s shared secret $4^{mn} \mod p$

Bob’s secret key $n$

↓

Bob’s public key $4^n \mod p$

↓

{Bob, Alice}’s shared secret $4^{mn} \mod p$

Can attacker find $4^{mn} \mod p$?
Bad news: DLP can be solved at surprising speed! Attacker can find $m$ and $n$ by “index calculus.”

To protect against this attack, replace $2^{262} - 5081$ with a much larger prime. Much slower arithmetic.

Alternative: Elliptic-curve cryptography. Replace \( \{1, 2, \ldots, 2^{262} - 5082\} \) with a comparable-size “safe elliptic-curve group.” Somewhat slower arithmetic.
An elliptic curve over $\mathbb{R}$

Consider all pairs of real numbers $x, y$
such that $y^2 - 5xy = x^3 - 7$.

The “points on the elliptic curve $y^2 - 5xy = x^3 - 7$ over $\mathbb{R}$”
are those pairs and one additional point, $\infty$.

i.e. The set of points is

\[
\{(x, y) \in \mathbb{R} \times \mathbb{R} : y^2 - 5xy = x^3 - 7\} \cup \{\infty\}.
\]

($\mathbb{R}$ is the set of real numbers.)
Graph of this set of points:

Don’t forget $\infty$.
Visualize $\infty$ as top of $y$ axis.
There is a standard definition of $0$, $-$, $+$ on this set of points.

Magical fact: The set of points is a “commutative group”; i.e., these operations $0$, $-$, $+$ satisfy every identity satisfied by $\mathbb{Z}$.

e.g. All $P, Q, R \in \mathbb{Z}$ satisfy

$$(P + Q) + R = P + (Q + R),$$

so all curve points $P, Q, R$
satisfy $$(P + Q) + R = P + (Q + R).$$

($\mathbb{Z}$ is the set of integers.)
Visualizing the group law

\[ 0 = \infty; \ -\infty = \infty. \]

Distinct curve points \( P, Q \) on a vertical line have \( -P = Q; \)
\[ P + Q = 0 = \infty. \]

A curve point \( R \)
with a vertical tangent line has \( -R = R; \)
\[ R + R = 0 = \infty. \]
Here $-P = Q$, $-Q = P$, $-R = R$:
Distinct curve points $P, Q, R$ on a line have $P + Q = -R$; $P + Q + R = 0 = \infty$.

Distinct curve points $P, R$ on a line tangent at $P$ have $P + P = -R$; $P + P + R = 0 = \infty$.

A non-vertical line with only one curve point $P$ has $P + P = -P$; $P + P + P = 0$. 
Here $P + Q = -R$: 

\[ \begin{align*} 
- \end{align*} \]
Here $P + P = -R$: 

\[ \begin{align*} 
\end{align*} \]
Curve addition formulas

Easily find formulas for \( + \) by finding formulas for lines and for curve-line intersections.

\[ x \neq x': \quad (x, y) + (x', y') = (x'', y'') \]

where \( \lambda = (y' - y)/(x' - x) \),
\[ x'' = \lambda^2 - 5\lambda - x - x', \]
\[ y'' = 5x'' - (y + \lambda(x'' - x)). \]

\[ 2y \neq 5x: \quad (x, y) + (x, y) = (x'', y'') \]

where \( \lambda = (5y + 3x^2)/(2y - 5x) \),
\[ x'' = \lambda^2 - 5\lambda - 2x, \]
\[ y'' = 5x'' - (y + \lambda(x'' - x)). \]

\( (x, y) + (x, 5x - y) = \infty. \)
An elliptic curve over $\mathbb{Z}/13$

Consider the prime field $\mathbb{Z}/13 = \{0, 1, 2, \ldots, 12\}$ with $-, +, \cdot$ defined mod 13.

The “set of points on the elliptic curve $y^2 - 5xy = x^3 - 7$ over $\mathbb{Z}/13$” is

$\{(x, y) \in \mathbb{Z}/13 \times \mathbb{Z}/13 : y^2 - 5xy = x^3 - 7\} \cup \{\infty\}$. 
Graph of this set of points:

As before, don’t forget $\infty$. 
The set of curve points is a commutative group with standard definition of 0, −, +.

Can visualize 0, −, + as before. Replace lines over \( \mathbb{R} \) by lines over \( \mathbb{Z}/13 \).

Warning: tangent is defined by derivatives; hard to visualize.

Can define 0, −, + using same formulas as before.
Example of line over $\mathbb{Z}/13$:

Formula for this line: $y = 7x + 9$. 
\[ P + Q = -R: \]
An elliptic curve over $\mathbb{F}_{16}$

Consider the non-prime field
$$(\mathbb{Z}/2)[t]/(t^4 - t - 1) = \{ 0t^3 + 0t^2 + 0t^1 + 0t^0, 0t^3 + 0t^2 + 0t^1 + 1t^0, 0t^3 + 0t^2 + 1t^1 + 0t^0, 0t^3 + 0t^2 + 1t^1 + 1t^0, 0t^3 + 1t^2 + 0t^1 + 0t^0, \ldots, 1t^3 + 1t^2 + 1t^1 + 1t^0 \}$$
of size $2^4 = 16$. 
Graph of the “set of points on the elliptic curve $y^2 - 5xy = x^3 - 7$ over $(\mathbb{Z}/2)[t]/(t^4 - t - 1)$”: 

![Graph showing points on an elliptic curve over a finite field.](image-url)
Line \( y = tx + 1 \):
\[ P + Q = -R: \]
More elliptic curves

Can use any field $k$.

Can use any nonsingular curve
\[ y^2 + a_1 xy + a_3 y = x^3 + a_2 x^2 + a_4 x + a_6. \]

"Nonsingular": no $(x, y) \in k \times k$ simultaneously satisfies
\[ y^2 + a_1 xy + a_3 y = x^3 + a_2 x^2 + a_4 x + a_6 \text{ and } 2y + a_1 x + a_3 = 0 \text{ and } a_1 y = 3x^2 + 2a_2 x + a_4. \]

Easy to check nonsingularity.
Almost all curves are nonsingular when $k$ is large.
e.g. $y^2 = x^3 - 30x$:
\[(x, y) \in k \times k : \]

\[
y^2 + a_1 xy + a_3 y = \]

\[
x^3 + a_2 x^2 + a_4 x + a_6 \} \cup \{ \infty \}
\]

is a commutative group with standard definition of \(0, -, +\).

Points on line add to \(0\) with appropriate multiplicity.

Group is usually called “\(E(k)\)” where \(E\) is “the elliptic curve

\[
y^2 + a_1 xy + a_3 y = \]

\[
x^3 + a_2 x^2 + a_4 x + a_6.\]’’

Fairly easy to write down explicit formulas for \(0, -, +\) as before.
Using explicit formulas can quickly compute $n$th multiples in $E(k)$ given $n \in \{0, 1, 2, \ldots, 2^{256} - 1\}$ and given $E, k$ with $\#k \approx 2^{256}$.

(How quickly? We’ll study this later.)


Can find curves where ECDLP seems extremely difficult: $\approx 2^{128}$ operations.
See “Handbook of elliptic and hyperelliptic curve cryptography” for much more information.

Two examples of elliptic curves useful for cryptography:

“NIST P-256”: \( E(\mathbb{Z}/p) \) where \( p \) is the prime \( 2^{256} - 2^{224} + 2^{192} + 2^{96} - 1 \) and \( E \) is the elliptic curve \( y^2 = x^3 - 3x + (a \text{ particular constant}) \).

“Curve25519”: \( E(\mathbb{Z}/p) \) where \( p \) is the prime \( 2^{255} - 19 \) and \( E \) is the elliptic curve \( y^2 = x^3 + 486662x^2 + x \).
Fast arithmetic

1. Someone specifies $k$. How quickly can we perform arithmetic in $k$?

2. Someone specifies $k$ and $E$. How quickly can we compute $n$th multiples in $E(k)$?

3. How quickly can we compute $n$th multiples in $E(k)$ if we choose $k$ and $E$?
Some examples of finite fields:

\[ \mathbb{Z}/(2^{255} - 19) . \]
\[ (\mathbb{Z}/(2^{61} - 1))[t]/(t^5 - 3) . \]
\[ (\mathbb{Z}/223))[t]/(t^{37} - 2) . \]
\[ (\mathbb{Z}/2)[t]/(t^{283} - t^{12} - t^7 - t^5 - 1) . \]

How quickly can we add, subtract, multiply in these fields?

Answer will depend on platform: AMD Athlon, Sun UltraSPARC IV, Intel 8051, Xilinx Spartan-3, etc. Warning: different platforms often favor different fields!
Fast integer arithmetic

How to multiply big integers?

Child’s answer: Use polynomial with coefficients in \( \{0, 1, \ldots, 9\} \) to represent integer in radix 10.

With this representation, multiply integers in two steps:
1. Multiply polynomials.
2. “Carry” extra digits.

Polynomial multiplication involves *small* integers.
Have split one big multiplication into many small operations.
Example of representation:

\[ 839 = 8 \cdot 10^2 + 3 \cdot 10^1 + 9 \cdot 10^0 = \text{value (at } t = 10\text{) of polynomial} \]

\[ 8t^2 + 3t^1 + 9t^0. \]

 Squaring: \((8t^2 + 3t^1 + 9t^0)^2 = 64t^4 + 48t^3 + 153t^2 + 54t^1 + 81t^0.\) Carrying:

\[
\begin{align*}
64t^4 &+ 48t^3 + 153t^2 + 54t^1 + 81t^0; \\
64t^4 &+ 48t^3 + 153t^2 + 62t^1 + 1t^0; \\
64t^4 &+ 48t^3 + 159t^2 + 2t^1 + 1t^0; \\
64t^4 &+ 63t^3 + 9t^2 + 2t^1 + 1t^0; \\
70t^4 &+ 3t^3 + 9t^2 + 2t^1 + 1t^0; \\
7t^5 &+ 0t^4 + 3t^3 + 9t^2 + 2t^1 + 1t^0. \\
\end{align*}
\]

In other words, \(839^2 = 703921.\)
What operations were used here?

- Multiply: 8 and 9 result in 72.
- Multiply: 3 and 9 result in 27, which is then added to 72 to get 153.
- Add: 153 and 6 result in 159.
- Add: 159 and 15 result in 174.
- Divide by 10: 174 divided by 10 leaves a remainder of 15.
- Mod 10: 159 mod 10 is 9.
Scaled variation:

\[ 839 = 800 + 30 + 9 = \]

value (at \( t = 1 \)) of polynomial \( 800t^2 + 30t^1 + 9t^0 \).

Squaring: \((800t^2 + 30t^1 + 9t^0)^2 = 640000t^4 + 48000t^3 + 15300t^2 + 540t^1 + 81t^0\).

Carrying:

\[ 640000t^4 + 48000t^3 + 15300t^2 + 540t^1 + 81t^0; \]
\[ 640000t^4 + 48000t^3 + 15300t^2 + 620t^1 + 1t^0; \]
\[ 700000t^5 + 0t^4 + 3000t^3 + 900t^2 + 20t^1 + 1t^0. \]
What operations were used here?

800 \rightarrow 7200 \\
30 \rightarrow 900 \\
9 \text{ multiply} \rightarrow 7200 \\
\quad \rightarrow \quad \text{add} \quad \rightarrow \quad 15300 \\
\quad \quad \rightarrow \quad \text{add} \quad \rightarrow \quad 15900 \\
\quad \quad \quad \rightarrow \quad \text{subtract} \quad \rightarrow \quad 15000 \\
\quad \quad \quad \quad \rightarrow \quad \text{mod 1000} \quad \rightarrow \quad 900
Speedup: double inside squaring

Squaring \( \cdots + f_2 t^2 + f_1 t^1 + f_0 t^0 \) produces coefficients such as
\( f_4 f_0 + f_3 f_1 + f_2 f_2 + f_1 f_3 + f_0 f_4 \).

Compute more efficiently as
\( 2f_4 f_0 + 2f_3 f_1 + f_2 f_2 \).

Or, slightly faster,
\( 2(f_4 f_0 + f_3 f_1) + f_2 f_2 \).

Or, slightly faster,
\( (2f_4) f_0 + (2f_3) f_1 + f_2 f_2 \) after precomputing \( 2f_1, 2f_2, \ldots \).

Have eliminated \( \approx 1/2 \) of the work if there are many coefficients.
Speedup: allow negative coeffs

Recall 159 $\leftrightarrow$ 15, 9.
Scaled: 15900 $\leftrightarrow$ 15000, 900.

Alternative: 159 $\leftrightarrow$ 16, $-1$.
Scaled: 15900 $\leftrightarrow$ 16000, $-100$.

Use digits \{'-5, -4, \ldots, 4, 5\}'
instead of \{0, 1, \ldots, 9\}.
Several small advantages:
easily handle negative integers;
easily handle subtraction;
reduce products a bit.
Speedup: delay carries

Computing (e.g.) big $ab + c^2$: multiply $a, b$ polynomials, carry, square $c$ poly, carry, add, carry.

e.g. $a = 314, b = 271, c = 839$:  
$$(3t^2 + 1t^1 + 4t^0)(2t^2 + 7t^1 + 1t^0) = 6t^4 + 23t^3 + 18t^2 + 29t^1 + 4t^0;$$
carry: $8t^4 + 5t^3 + 0t^2 + 9t^1 + 4t^0$.

As before $(8t^2 + 3t^1 + 9t^0)^2 = 64t^4 + 48t^3 + 153t^2 + 54t^1 + 81t^0; 7t^5 + 0t^4 + 3t^3 + 9t^2 + 2t^1 + 1t^0$. 

$+$: $7t^5 + 8t^4 + 8t^3 + 9t^2 + 11t^1 + 5t^0; 7t^5 + 8t^4 + 9t^3 + 0t^2 + 1t^1 + 5t^0$. 
Faster: multiply $a, b$ polynomials, square $c$ polynomial, add, carry.

$$(6t^4 + 23t^3 + 18t^2 + 29t^1 + 4t^0) + (64t^4 + 48t^3 + 153t^2 + 54t^1 + 81t^0) = 70t^4 + 71t^3 + 171t^2 + 83t^1 + 85t^0;$$

$$7t^5 + 8t^4 + 9t^3 + 0t^2 + 1t^1 + 5t^0.$$

Eliminate intermediate carries.
Outweighs cost of handling slightly larger coefficients.

Important to carry between multiplications (and squarings) to reduce coefficient size; but carries are usually a bad idea for additions, subtractions, etc.
Speedup: polynomial Karatsuba

Computing product of polys $f, g$ with (e.g.) $\deg f < 20, \deg g < 20$: 400 coefficient mults, 361 coefficient adds.

Faster: Write $f$ as $F_0 + F_1 t^{10}$ with $\deg F_0 < 10, \deg F_1 < 10$. Similarly write $g$ as $G_0 + G_1 t^{10}$.

Then $fg = (F_0 + F_1)(G_0 + G_1)t^{10} + (F_0G_0 - F_1G_1 t^{10})(1 - t^{10})$. 
20 adds for $F_0 + F_1$, $G_0 + G_1$.  
300 mults for three products $F_0G_0$, $F_1G_1$, $(F_0 + F_1)(G_0 + G_1)$.  
243 adds for those products.  
9 adds for $F_0G_0 - F_1G_1 t^{10}$ 
with subs counted as adds 
and with delayed negations.  
19 adds for $\cdots (1 - t^{10})$.  
19 adds to finish.  

Total 300 mults, 310 adds.  
Larger coefficients, slight expense; 
still saves time.  

Can apply idea recursively 
as poly degree grows.
Many other algebraic speedups in polynomial multiplication: Toom, FFT, etc.

Increasingly important as polynomial degree grows. $O(n \lg n \lg \lg n)$ coeff operations to compute $n$-coeff product.

Useful for sizes of $n$ that occur in cryptography? Maybe; active research area.
Using CPU’s integer instructions

Replace radix 10 with, e.g., $2^{24}$. Power of 2 simplifies carries.

Adapt radix to platform.

e.g. Every 2 cycles, Athlon 64 can compute a 128-bit product of two 64-bit integers. (5-cycle latency; parallelize!)
Also low cost for 128-bit add.

Reasonable to use radix $2^{60}$. Sum of many products of digits fits comfortably below $2^{128}$.
Be careful: analyze largest sum.
e.g. In 4 cycles, Intel 8051 can compute a 16-bit product of two 8-bit integers. Could use radix $2^6$. Could use radix $2^8$, with 24-bit sums.

e.g. Every 2 cycles, Pentium 4 F3 can compute a 64-bit product of two 32-bit integers. (11-cycle latency; yikes!) Reasonable to use radix $2^{28}$.

Warning: Multiply instructions are very slow on some CPUs. e.g. Pentium 4 F2: 10 cycles!
Using floating-point instructions

Big CPUs have separate floating-point instructions, aimed at numerical simulation but useful for cryptography.

In my experience, floating-point instructions support faster multiplication (often much, much faster) than integer instructions, except on the Athlon 64. Other advantages: portability; easily scaled coefficients.
e.g. Every 2 cycles, Pentium III can compute a 64-bit product of two floating-point numbers, and an independent 64-bit sum.

e.g. Every cycle, Athlon can compute a 64-bit product and an independent 64-bit sum.

e.g. Every cycle, UltraSPARC III can compute a 53-bit product and an independent 53-bit sum. Reasonable to use radix $2^{24}$.

e.g. Pentium 4 can do the same using SSE2 instructions.
How to do carries in floating-point registers? (No CPU carry instruction: not useful for simulations.)

Exploit floating-point rounding: add big constant, subtract same constant.

e.g. Given $\alpha$ with $|\alpha| \leq 2^{75}$: compute 53-bit floating-point sum of $\alpha$ and constant $3 \cdot 2^{75}$, obtaining a multiple of $2^{24}$; subtract $3 \cdot 2^{75}$ from result, obtaining multiple of $2^{24}$ nearest $\alpha$; subtract from $\alpha$. 
Reducing modulo a prime

Fix a prime \( p \).
The prime field \( \mathbb{Z}/p \)
is the set \( \{0, 1, 2, \ldots, p - 1\} \)with \( - \) defined as \( - \mod p \),\( + \) defined as \( + \mod p \),\( \cdot \) defined as \( \cdot \mod p \).

e.g. \( p = 1000003 \):
\begin{align*}
1000000 + 50 & = 47 \text{ in } \mathbb{Z}/p; \\
-1 & = 1000002 \text{ in } \mathbb{Z}/p; \\
117505 \cdot 23131 & = 1 \text{ in } \mathbb{Z}/p.
\end{align*}
How to multiply in \( \mathbb{Z}/p \)?

Can use definition:
\[
fg \mod p = fg - p \left\lfloor \frac{fg}{p} \right\rfloor.
\]

Can multiply \( fg \) by a precomputed \( 1/p \) approximation; easily adjust to obtain \( \left\lfloor \frac{fg}{p} \right\rfloor \).

Slight speedup: “2-adic inverse”; “Montgomery reduction.”

We can do better: normally \( p \) is chosen with a special form (or dividing a special form; see “redundant representations”) to make \( fg \mod p \) much faster.
e.g. In $\mathbb{Z}/1000003$:

$314159265358 =
314159 \cdot 1000000 + 265358 =
314159(-3) + 265358 =
-942477 + 265358 =
-677119.$

Easily adjust to range
$\{0, 1, \ldots, p - 1\}$
by adding/subtracting a few $p$’s. (Beware timing attacks!)

Speedup: Delay the adjustment; extra $p$’s won’t damage subsequent field operations.
Can delay carries until after multiplication by 3.

e.g. To square 314159 in $\mathbb{Z}/1000003$: Square poly $3t^5 + 1t^4 + 4t^3 + 1t^2 + 5t^1 + 9t^0$, obtaining $9t^{10} + 6t^9 + 25t^8 + 14t^7 + 48t^6 + 72t^5 + 59t^4 + 82t^3 + 43t^2 + 90t^1 + 81t^0$.

Reduce: replace $(c_i)t^{6+i}$ by $(-3c_i)t^i$, obtaining $72t^5 + 32t^4 + 64t^3 - 32t^2 + 48t^1 - 63t^0$.

Carry: $8t^6 - 4t^5 - 2t^4 + 1t^3 + 2t^2 + 2t^1 - 3t^0$. 
To minimize poly degree, mix reduction and carrying, carrying the top sooner.

e.g. Start from square $9t^{10} + 6t^9 + 25t^8 + 14t^7 + 48t^6 + 72t^5 + 59t^4 + 82t^3 + 43t^2 + 90t^1 + 81t^0$.

Reduce $t^{10} \rightarrow t^4$ and carry $t^4 \rightarrow t^5 \rightarrow t^6$: $6t^9 + 25t^8 + 14t^7 + 56t^6 - 5t^5 + 2t^4 + 82t^3 + 43t^2 + 90t^1 + 81t^0$.

Finish reduction: $-5t^5 + 2t^4 + 64t^3 - 32t^2 + 48t^1 - 87t^0$. Carry $t^0 \rightarrow t^1 \rightarrow t^2 \rightarrow t^3 \rightarrow t^4 \rightarrow t^5$: $-4t^5 - 2t^4 + t^3 + 2t^2 - t^1 + 3t^0$. 
Speedup: non-integer radix

Consider \( \mathbb{Z}/(2^{61} - 1) \).

Five coeffs in radix 2^{13}?
\[ f_4 t^4 + f_3 t^3 + f_2 t^2 + f_1 t^1 + f_0 t^0. \]
Most coeffs could be 2^{12}.

Square \( \cdots + 2(f_4 f_1 + f_3 f_2)t^5 + \cdots \).
Coeff of \( t^5 \) could be \( > 2^{25} \).

Reduce: \( 2^{65} = 2^4 \) in \( \mathbb{Z}/(2^{61} - 1) \);
\( \cdots + (2^5(f_4 f_1 + f_3 f_2) + f_0^2)t^0 \).
Coeff could be \( > 2^{29} \).
Very little room for additions, delayed carries, etc. on 32-bit platforms.
Scaled: Evaluate at $t = 1$.

$f_4$ is multiple of $2^{52}$;
$f_3$ is multiple of $2^{39}$;
$f_2$ is multiple of $2^{26}$;
$f_1$ is multiple of $2^{13}$;
$f_0$ is multiple of $2^0$. Reduce:

$$\cdots + (2^{-60}(f_4 f_1 + f_3 f_2) + f_0^2)t^0.$$ 

Better: Non-integer radix $2^{12.2}$.

$f_4$ is multiple of $2^{49}$;
$f_3$ is multiple of $2^{37}$;
$f_2$ is multiple of $2^{25}$;
$f_1$ is multiple of $2^{13}$;
$f_0$ is multiple of $2^0$.

Saves a few bits in coeffs.
More finite fields

Fix a prime $p$. Fix a poly $\varphi$ in one variable $t$ with $\varphi$ irreducible mod $p$.

The finite field $(\mathbb{Z}/p)[t]/\varphi$ is the set of polynomials $f_{\deg \varphi-1}t^{\deg \varphi-1} + \cdots + f_1t^1 + f_0t^0$ with each $f_i \in \mathbb{Z}/p$ and with $-,+,\cdot$ defined modulo $p$ and modulo $\varphi$.

$(\mathbb{Z}/p)[t]/\varphi$ is an “extension” of the prime field $\mathbb{Z}/p$; it has “characteristic” $p$. 
e.g. 223 is prime, and poly $t^6 - 3$ is irreducible mod 223, so $(\mathbb{Z}/223)[t]/(t^6 - 3)$ is a field.

$223^6$ elements of field, namely polynomials $f_5 t^5 + f_4 t^4 + f_3 t^3 + f_2 t^2 + f_1 t + f_0$ with each $f_i \in \{0, 1, \ldots, 222\}$.

After adding, subtracting, multiplying: replace $t^6$ by 3, replace $t^7$ by $3t$, etc.; and reduce coefficients modulo 223.

e.g. $(9t^4 + 1)^2 = 81t^8 + 18t^4 + 1 = 243t^2 + 18t^4 + 1 = 18t^4 + 20t^2 + 1$. 
Have two levels of polynomials when \( p \) is large: element of \((\mathbb{Z}/p)[t]/\varphi\) is poly mod \( \varphi \); each poly coefficient is integer represented as poly in some radix.

e.g. \( f_4 t^4 + f_3 t^3 + f_2 t^2 + f_1 t^1 + f_0 t^0 \)
in \((\mathbb{Z}/(2^{61} - 1))[t]/(t^5 - 3)\)
could have each coefficient \( f_i \) represented as poly of degree \(< 3\) in radix \( 2^{61}/3 \).

When \( p \) is small, especially \( p = 2 \), many speedups beyond this talk: batching coefficients, using fast Frobenius, et al.