Speeding up the computation of group order of genus 2 curves over finite fields

Nicolas Thériault

Departamento de Matemática y Ciencia de la Computación,
Universidad de Santiago de Chile.

A quick announcement

Polynesian Journal of Mathematics

Why count points on hyperelliptic curves

They can be used for cryptographic applications.

- They are interesting for cryptosystems based on the discrete logarithm problem.
- You need a curve whose group order has a large prime factor, ideally the order is prime.
- You may want to select a curve completely at random (no special structure) and see if it can be useful.

Why count points on hyperelliptic curves

They can be used for cryptographic applications.

- They are interesting for cryptosystems based on the discrete logarithm problem.
- You need a curve whose group order has a large prime factor, ideally the order is prime.
- You may want to select a curve completely at random (no special structure) and see if it can be useful.

Because its a difficult to do...

Why count points on hyperelliptic curves

They can be used for cryptographic applications.

- They are interesting for cryptosystems based on the discrete logarithm problem.
- You need a curve whose group order has a large prime factor, ideally the order is prime.
- You may want to select a curve completely at random (no special structure) and see if it can be useful.

Because its a difficult to do...

Because we end up learning more about curves and do some nice math...

Genus 2 hyperelliptic curves

In many "simple" groups, the discrete logarithm is easy to solve.

In finite fields and similar multiplicative structures, the difficulty of the discrete logarithm is similar to the difficulty of factoring integers.

In elliptic and hyperelliptic curves of genus 2, there are no known "non-generic" efficient algorithms (on classical computers) for the discrete logarithm.

For the same security, they require much smaller groups.

For genus 2 curves, the size of the field is half what it would be for an elliptic curve with the same security (the group order is similar).

Group operations require similar times on elliptic and hyperelliptic curves, but hyperelliptic ones require less energy.

Quantum computing

In theory, a quantum computer could solve the discrete logarithm problem in polynomial time (and revolutionize computing, etc).

Quantum computing

In theory, a quantum computer could solve the discrete logarithm problem in polynomial time (and revolutionize computing, etc).

In practice, we have to take things with a grain of salt:

- There are few algorithms that are more efficient on quantum computers, Shor's algorithm for integer factorization and discrete logarithms is one of them.
- Not all quantum computers are fully programable (Shor is not always easily applicable).

Quantum computing

In theory, a quantum computer could solve the discrete logarithm problem in polynomial time (and revolutionize computing, etc).

In practice, we have to take things with a grain of salt:

- There are few algorithms that are more efficient on quantum computers, Shor's algorithm for integer factorization and discrete logarithms is one of them.
- Not all quantum computers are fully programable (Shor is not always easily applicable).

- The record for integer factorization is at 26 bits.
- The record for discrete logarithms in finite fields is at 18 bits.
- The record for discrete logarithm in an elliptic curve is over $\mathbb{F}_7$ (3 bits).

For short and mid-term security applications, using the discrete logarithm problem may still be reasonable, especially if it gives good performance.

Hyperelliptic curves

A non-singular hyperelliptic curve over a finite field $\mathbb{F}_q$ of odd characteristic can be given by an equation of the form

$$y^2 = f(x) = f_{d_f} \prod_{i=1}^{d_f} (x - \theta_i)$$

where the θ_i are distinct elements of $\mathbb{F}_{q^s}$ such that $f(x)$ is in $\mathbb{F}_q[x]$.

If $d_f = 2g + 1$, the curve is in an imaginary model, and we can reduce to $f_{d_f} = 1$.
If $d_f = 2g + 2$, the curve is real (purely real if none of the θ_i are in $\mathbb{F}_q$).

In both cases, g is the genus of the curve.

For each hyperelliptic curve we can define the Jacobian group: the divisor class group.

Point counting, elliptic curves

For an elliptic curve over the finite field $\mathbb{F}_q$, the Hasse-Weil theorem tells us the characteristic polynomial of the Frobenius endomorphism is of the form

$$T^2 + aT + q = 0$$

with $|a| \leq 2\sqrt{q}$, from which we find the group order satisfies

$$q + 1 - 2\sqrt{q} \leq \#E(\mathbb{F}_q) \leq q + 1 + 2\sqrt{q}$$

Schoof's algorithm computes the group order (counts the points) of the elliptic curve through modular arithmetic:

- Computes $a \bmod \ell$ for small ℓ
- Studies the Frobenius in the ℓ-torsion group
- Complexity: $O(\ell^{3+\epsilon} \log q)$ operations in $\mathbb{F}_q$ for $a \bmod \ell$.
- Requires $O(\log q)$ primes ℓ, to determine a, with $\ell \in O(\log q)$.

It is possible to remove a factor of ℓ using further techniques (Schoof-Elkies-Atkin).

Point counting, genus 2 curves

For genus 2 curves, we still talk about "point counting" even though "divisor counting" would be more appropriate.

The Hasse-Weil theorem tells us the characteristic polynomial of the Frobenius satisfies

$$T^4 + aT^3 + bT^2 + aqT + q^2 = 0$$

with $|a| \leq 4\sqrt{q}$ and $|b| \leq 6q$.

We can still work through modular arithmetic:

- Compute a mod ℓ and b mod ℓ for small ℓ
- Study the Frobenius in the ℓ-torsion group
- Complexity (not really correct, simplifying for now): $O(\ell^{6+\epsilon} \log q)$ operations in $\mathbb{F}_q$ to find both a mod ℓ and b mod ℓ.
- Requires $O(\log q)$ primes ℓ to determine b, with $\ell \in O(\log q)$.

Gaudry and Schost, 2012

- They used the genus 2 “division” polynomials of Cantor with various ideas to improve the computation of the ℓ-torsion group.
- They obtain ℓ-torsions for ℓ up to 31.
- They compute modular information up to a given size, and then complete with Pollard’s Kangaroo algorithm.
- For very small ℓ, they obtain divisors of order ℓ^k and work with “liftings” from mod ℓ^{k-1} up to mod ℓ^k.
- Requires to work in extension fields (the extension degree increases with k).
- In general each increase of k by 1 takes ℓ^2 more times the work.
- For $\ell = 2$, they use the Kummer surface representation of the curve (pseudo-group) to do the work (requires that all the Weierstrass points are $\mathbb{F}_q$-rational).
- They used systems of non-linear equations for $\ell = 3$ and 5.
- In total, it took close to 1.000.000 CPU hours to find a “good” curve’.

Some distributions

Assume we have a curve over $\mathbb{F}_p$ with p of 127 or 128 bits

- On average, 1 out of every ≈ 175 curve has prime order.
  - If you discard curves with small prime factors (for example $\ell = 2, 3, 5$, and 7), 1 out of every ≈ 47 curve has prime order.
  - This selection is easy to do and quite fast (< 5 minutes per curve, with an average time around $1 - 2$ minutes).
- If you ask that the quadratic twist of the curve also has prime order, then 1 out of every ≈ 25500 curve is “good”.
  - Of the curves without a small prime factor, 1 out of every ≈ 6800 has prime order and its quadratic twist too.
  - If we extend the initial verification to the quadratic twist as well, 1 out of every ≈ 2500 curve is “good”.
  - In practice, requiring the quadratic twist to have prime order is not necessary: avoid a specialized attack (when the validity of the point is not checked) that has been generalized since, and both versions can be avoided by verifying the divisor satisfy the curve equation (should always be done)

Point counting, powers of ℓ

To obtain divisors of order ℓ^k, we begin with divisors of order ℓ^{k-1} and compute a pre-image of the multiplication by ℓ, a "ℓ-section".

In general there are ℓ^{2g} pre-images of the multiplication by ℓ.

With one pre-image and the ℓ-torsions, we know all pre-images.

Up to now, it was a hard problem to obtain one (or more) pre-image for $\ell > 2$.

In general solving techniques tend to produce some false solutions.

bisections (with J. Miret and J. Pujolàs, 2015)

- Associates the existence of bisections to quadratic characters of $u(x)$ evaluating at the roots of $f(x)$.
- Computes 4 independent square roots and solves a 4×4 linear system (always invertible).
- Basically has the same complexity as doing the bisections on the Kummer surface (used by Gaudry and Schost), but works directly on the curve model.
- Allows to manage cases where not all the 2-torsions are $\mathbb{F}_q$-rational.
- Includes an algorithm for real curves.
- Gave us the idea to look at point counting.

Trisections in hyperelliptic curves (with E. Riquelme, 2018)

- We obtain an algorithm that allows to trisect without producing false solutions (parasitic factors are cleaned out).
- Given in a symbolic form, with fast evaluation.
- Requires computing the roots of a polynomial of degree 81.
- The dominant cost in the trisection comes from factorization.

Trisections in elliptic curves (with J. Pujolàs, 2022)

- We associate trisections with the existence (and computation) of two independent cube roots and a few polynomial operations.
- Wee define trisectors: the cube roots obtained above (if they exist).
- For each trisector, there are 3 possible choices, which correspond to the selection of a cube root of unity.
- Each choice of a pair of trisectors produces exactly one trisection.

ℓ-sectors (with J. Miret and J. Pujolàs)

- Generalizes the previous result using Weil reciprocity.
- Defines $2g$-tuples which are all ℓ-powers if and only if the divisor admits a ℓ-section.
- The polynomials defining the principal divisor associated to ℓ-torsions are evaluated at the divisor that we want to ℓ-sect.
- Produces a character associated to the existence of ℓ-sectors.
- Includes/explains the cases of bisections and trisections.

ℓ-sections in genus 2 (with D. Cabarcas, J. Miret and J. Pujolàs)

- ℓ-sectors allow to produce a non-linear system of equations (in 4 variables) with a unique solution.
- We "add" extra variables: (ugly) rational functions of the first 4 variables.
- We use Weil reciprocity to extend the system (obtain more equations).
- The new equations come from other ℓ-torsion divisors, but the associated ℓ-sectors are fixed by the first 4 (the generators).
- The system is linearized (new variables are produced for each product of variables). This solving technique is a restriction of the Gröbner basis technique.
- Gives a system of close to $4\ell^2$ variables that can be solved with linear algebra techniques ($5\ell^2$ before removing known dependencies).
- Complexity: 4 factorizations of degree ℓ polynomials and $O(\ell^6)$ to solve the system (all in an extension field)

ℓ-sections in point counting

- The ℓ-torsions are defined in an extension of degree $d \leq \ell^4 - 1$.
- In general, going from an extension of degree $\tilde{d}$ to an extension of degree $\tilde{d} \cdot \ell$ increases the depth of each generator of the ℓ-Sylow subgroup by 1 (you can ℓ-sect once more).
- There is one exception: going from degree d to an extension of degree $d \cdot \ell$ can increases the depth more: at least once, but sometimes (no always) more.
- With fast arithmetic, the cost of computations (solving the system), increases by a factor of ℓ.
- For the factorization cost, wait a few slides... (also by a factor of ℓ).
- This makes it very interesting to look at ℓ^k-torsion for smaller values of ℓ rather than increase to the next prime ℓ.

Computing the ℓ-Sylow (with J. Pujolàs and E. Riquelme, 2023)

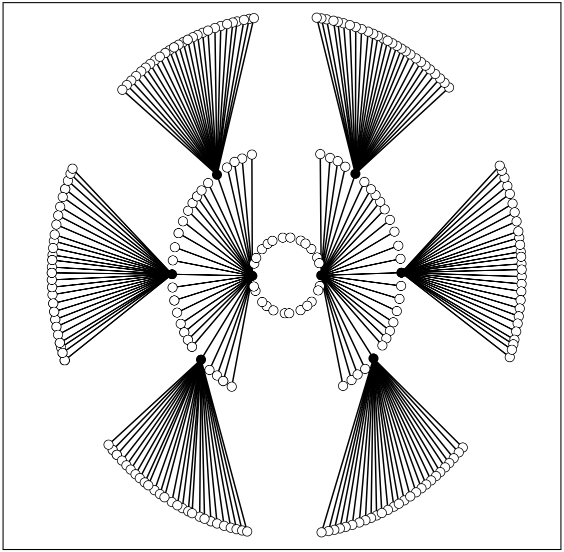
Instead of working with all divisors fo oredr ℓ^k, we want to use only a few or even only one.

In general, choosing one at random can produce information loss.

An efficient algorithm was developed to find a basis of the ℓ-Sylow subgroup over $\mathbb{F}_{q^k}$ ($\mathbb{F}_{q^k}$-rational divisors of order ℓ^j).

- The basis gives all the information that could be obtained for the combination of all the elements of the ℓ-Sylow.
- Allows to find divisors of higher order in the same field extension.
- En general the "deeper" divisor gives the most information that can be extracted from the ℓ-Sylow.

Computing the ℓ-Sylow (with J. Pujolàs and E. Riquelme, 2023)



Some statistics

We considered $10,000$ randomly selected curves, and extracted those with the full 3-torsion group (3488 such curves).

We computed the generators of the 3-Sylow, and indicated by $1 \leq n_1 \leq n_2 \leq n_3 \leq n_4$ the depth of each generator.

Looking at the greatest difference in depths, we find:

$n_4 - n_1$	number of curves	percentage
0	3743	57.5%
1	1718	26.4%
2	590	9.1%
3	244	3.7%
4	149	2.3%
5	48	0.7%
6 to 8	20	0.3%

Note that except for $n_4 - n_1 = 0$ or 1, the probability that a divisor in the 3-Sylow has maximal depth is below 10% (depends on n_1, n_2, n_3 and n_4).

However, not all generators work well for point counting...

Some statistics

In a second run, we looked at another set of $10,000$ randomly selected curves, 3640 of them with full torsion.

For each generator we check the order $\phi^2(D) + qD$ (which is multiplied by the first unknown coefficient).

Let n_j be the maximal "useful" depth:

$n_j - n_1$	number of curves	percentage
0	3876	60.9%
1	2054	32.3%
2	395	6.2%
3	31	0.5%
4	4	0.1%

Divisor of maximal useful depth give more information on the coefficients of the characteristic polynomial, but at basically the same cost (since they are in the same extension). On average, we get an increase in the modulo by a factor of close to 71%.

The probability of picking a divisor of maximal useful depth at random is quite low.

Some statistics

for the 5-sylow, we used 1,400 randomly selected curves, and extracted 298 with the full 5-torsion group. For the greatest difference in depths, we find:

$n_4 - n_1$	number of curves	percentage
0	646	58.6%
1	330	29.9%
2	93	8.4%
3	28	2.5%
4	5	0.5%

And for the greatest useful depth (290 curves with full torsion), we find:

$n_j - n_1$	number of curves	percentage
0	813	73.2%
1	268	24.1%
2	29	2.6%

On average, we get an increase in the modulo by a factor of close to 86%.

Polynomial factorization, context

In the computation of the group order of a hyperelliptic curve of genus 2, a (very significant) part of the algorithm requieres to compute ℓ-th roots in a very large extension of the base field.

For example, computing square roots in $\mathbb{F}_{p^{196608}} = \mathbb{F}_{p^{3 \cdot 2^{16}}}$

or cube roots in $\mathbb{F}_{p^{19683}} = \mathbb{F}_{p^{3^9}}$

or fifth roots in $\mathbb{F}_{p^{93750}} = \mathbb{F}_{p^{6 \cdot 5^6}}$

etc.

Timings

Factorization algorithms over finite fields have improved over the years, but mostly concentrated on larger degree n and smaller fields (of q elements)...

- Cantor and Zassenhaus, 1981: $O(n^{2+\epsilon}\log_2 q)$
- von zur Gathen and Shoup, 1992: $O(n^{2+\epsilon}) + O(n^{1+\epsilon}\log_2 q)$
- Kaltofen and Shoup 1998: $O(n^{5/2+\epsilon}) + O(n^{1+\epsilon}\log_2 q)$
- Kaltofen and Shoup 1997: $O(n^{2+\epsilon}) + O(n^{1+\epsilon}(\log_2 q)^{0.69})$
- Kedlaya and Umans, 2008: $O(n^{3/2+\epsilon}) + O(n^{1+\epsilon}\log_2 q)$, si $\log q < n$

We are looking at a very special case, with degree $n = \ell$ (fixed and small) over a field of q^k elements, q fixed but large and k increasing (quickly).

The dominant term becomes $O(\ell^{1+\epsilon} k \log_2 q)$ is dominant, and has not changed (much) since von zur Gathen and Shoup.

This term comes from computing $x^{q^k} \bmod f(x)$ or $x^{(q^k-1)/2} \bmod f(x)$.

1. p-powers (with R. Avanzi)

To compute the polynomial exponentiation in $\mathbb{F}_{q^k}[x]/f(x)$, we take advantage of the q-th power Frobenius and its powers.

This is similar to von zur Gathen and Shoup iterated Frobenius algorithm, but at a subfield level.

If it can be done with a very efficient Frobenius operation in $\mathbb{F}_{q^k}$, we can reduce the factorization cost to

$$O(n^{1+\epsilon} \log k) + O(n^{1+\epsilon} \log q)$$

multiplications in $\mathbb{F}_{q^k}$.

As a result, going up to an extra degree ℓ extension to compute the next power-of-ℓ torsion should cost (close to) an extra factor of ℓ for the factorization (linear growth).

2. Roots of unity (with J. von zur Gathen)

Equal degree factorization is based on the factorization

$$b(x)^{q^d} - b(x) = b(x) \cdot \left(b(x)^{(q^d-1)/2} - 1 \right) \cdot \left(b(x)^{(q^d-1)/2} + 1 \right) \ .$$

This splits the polynomial into (roughly) two halves, and if we repeat this $O(\log n)$ times we can separate all the factors. This is because -1 is a primitive square root of unity.

If there is a primitive m-th root of unity, we can obtain an m-way split from just one exponentiation (and a few more *gcd*s).

If we have "nice enough" primitive roots of unity, we can reduce the average number of polynomial exponentiations required to isolate a factor of $f(x)$.

In general this could speed-up equal-degree factorization by a factor of $\log \log q$ (but that version is too general here).

For ℓ-sections, we work in extensions with a basis of the full ℓ-torsion group, so ℓ divides $q^d - 1$...

We get the average number of exponentiations very close to 1 (best possible case).

3. Tower fields (in preparation)

The algorithm to compute group orders requires that we work in increasing extensions over a base field (lifting), and the factorization algorithm has to compute the Frobenius of various field elements (polynomial coefficients).

In general the field towers are of the form $\mathbb{F}_{p^{n \cdot \ell^k}}$.

Example of field tower: $\mathbb{F}_{p^{10 \cdot 3^k}}$ with k increasing from 0 to 9.

We can choose the field towers to simplify both operations.

How to choose the field towers

We assume there is a primitive ℓ-root of unity in $\mathbb{F}_q$ (ℓ^2 if $\ell = 2$)

First we look for a value of β such that $z^\ell - \beta$ is irreducible in $\mathbb{F}_q[z]$.

Next we look for $\alpha = m\beta$ such that $z^n + z + \alpha$ is irreducible in $\mathbb{F}_q[z]$.

In general, $f_k(z) = z^{n\cdot\ell^k} + z^{\ell^k} + \alpha$ will then be irreducible for every k.

sketch of the proof: If $z^{n\cdot\ell^k} + z^{\ell^k} + m\beta$ is irreducible, then finding a factor of $z^{n\cdot\ell^{k+1}} + z^{\ell^{k+1}} + m\beta$ requires that we find a ℓ-th root of $m\beta$, but we made sure β didn't have an ℓ-th root.

Lifting to the next extension is then very easy (introducing some 0s in the representation).

Applying the Frobenius is also cheaper than for extensions defined by polynomials of the form $z^{n\cdot\ell^k} + z + \alpha$.

Bringing everything together

Improvements for ℓ-sections:

- Replace the non-linear system of equation by a linear one
- Direct construction of the system
- Factor 4 polynomials of degree ℓ instead of one polynomial of degree ℓ^4
- Specialized factorization algorithms (much faster)

With these improvements we can:

- Compute 2^{19}-torsions in less time than Gaudry-Schost needed for 2^{16}-torsions (closer to our 2^{20}-torsions)
- Compute 3^{10}-torsions in less time than Gaudry-Schost needed for 3^{6}-torsions (closer to our 3^{11}-torsions)
- Use 5^k- and 7^k-torsions in all curves
- Use 11^2-torsions in some curves (in very rare cases we could use 11^3-torsions)
- Avoid some computations of ℓ-torsions

Bringing everything together

Record by Gaudry and Schost:

- Needed ≈ 1.000 hours for each complete order computation, with powers of 2, 3, and 5, and ℓ-torsions up to $\ell = 31$.
- Looked for group orders of the form $16 \cdot \textit{prime}$ (due to the form of the 2-torsions).

New record:

- For each curve and each ℓ^k, the algorithm optimizes the choices of k (based on the original extension for the ℓ-torsions).
- We can do complete group order computations in 85 to 100 hours, using powers of 2, 3, 5, 7, and (sometimes) 11, and ℓ-torsions up to $\ell = 13$.
- We can look for curves with prime order.

Bringing everything together

The curve

$$C \ : \ y^2 = x^5 + x^3 + 4x + 1$$

over the field $\mathbb{F}_p$ with

$$p = 2^{127} - 138727$$

has prime order

$$\begin{array}{r} 28948022309329048854709139445701324423 \\ 2 \\ 3269126657454658361913763657057919213 7 \end{array} ,$$

with characteristic polynomial

$$\begin{aligned} T^4 + aT^3 + bT^2 + apT + p^2 &= 0 \\ a &= 6956615573005025511 \\ b &= 81462019801547732244226593786 14446157 \end{aligned} .$$

Based on divisors of orders 2^{20}, 3^{10}, 5^6, 7^4, 11, and 13.

For most of the curves we have 2^{19} or 2^{20}, 3^{10} or 3^{11}, 5^5 or 5^6, 7^3 or 7^4, 11 or 11^2, and 13.

Can we say anything about asymptotic?

Yes!

Can we say anything about asymptotic?

- Working with ℓ-torsions:
  - Computing the ℓ-torsion (Gaudry-Schost): $O(\ell^{6+\epsilon})$, approximated by $\approx 0.0008\ell^6$ sec.
  - Applying the Frobenius: $O(\ell^4 \log q)$.
  - Determining the coefficients: $O(\ell^6)$ for a full search, down to $O(\ell^5)$ with Baby Steps - Giant Steps (ℓ^2 values).
- Working with ℓ^k-torsions:
  - First extension: $d = O(\ell^4)$ (pessimistic, most curves have d between ℓ^2 and ℓ^3).
  - For now, forget about generators of the ℓ-Sylow (pessimistic).
  - Extension degree for ℓ^k-tosions: $O(\ell^{k+3})$
  - Factorization: $O(\ell^{k+4} \log q)$, approximated by $\approx 0.0003\ell^{k+4} \log q$ sec.
  - Solving the system: $O(\ell^{k+9})$, approximated by $\approx 0.0001\ell^{k+9}$ sec.
  - Frobenius computations: $O(\ell^{k+3} \log q)$.
  - Determining the coefficients: $O(\ell^{k+4})$ with Baby Steps - Giant Steps.
- Pollard Kangaroo:
  - a completely known (modulus greater than the interval length), missing factor of N on the modulus of b.
  - Kangaroo: $O(N^{0.5+\epsilon})$, approximated by $\approx \sqrt{N}/20{,}000$ sec.

The timing estimates (in seconds) are based on implementations (Gaudry-Schost and ours).

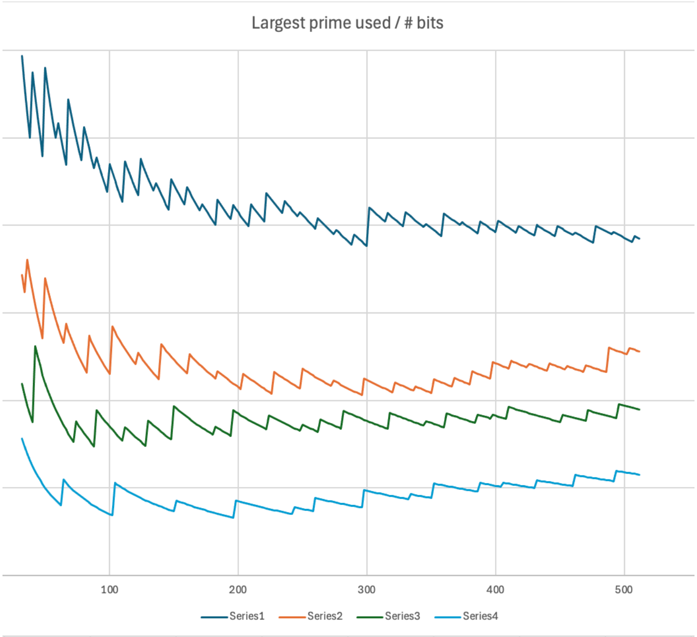
Can we say anything about asymptotic?

For each field size from 16 bits to 256 bits (groups of 32 bits to 512 bits), we ran four series for which we computed the largest prime ℓ required to compute the complete group order and the estimated total time used:

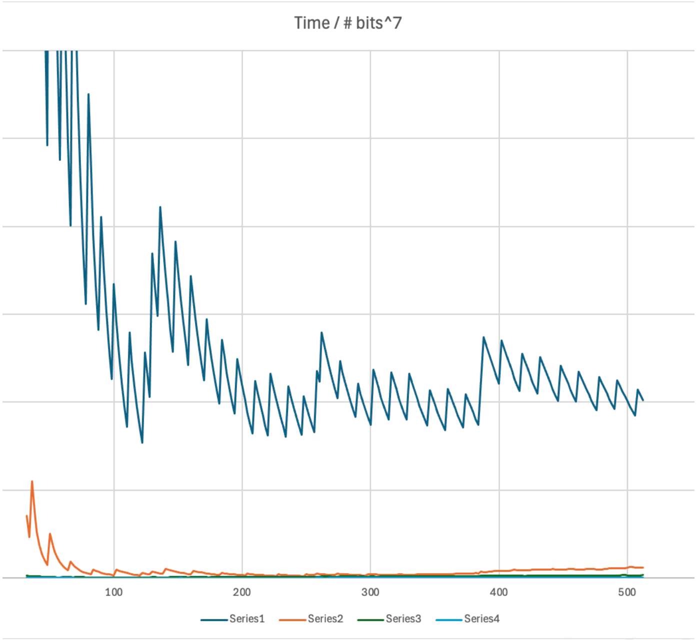
- Series 1 and 3: using only modular information to compute the coefficients
- Series 2 and 4: using Pollard's Kangaroo to complete the computation of b

- Series 1 and 2: using only ℓ-torsions
- Series 3 and 4: combining with ℓ^k-torsions

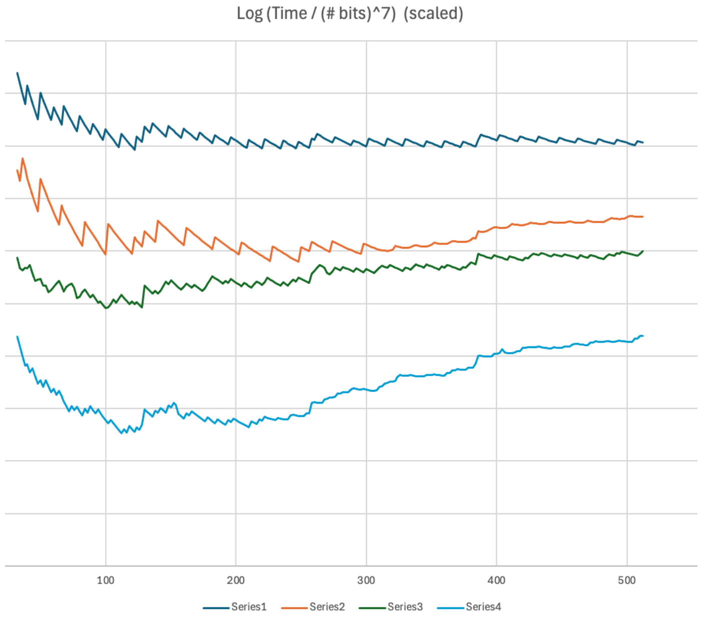
Largest prime used compared to log q



Total time compared to $(\log q)^7$



Total time compared to $(\log q)^7$, logarithmic scale



Can we say anything about asymptotic?

- In all cases, the complexity is $O(\log q)^{7+\epsilon})$.
- The constant hidden in the O-notation changes dramatically.

- Using Pollard's Kangaroo to complete the computation of b can reduce the total time by a factor between 3,000 and 20,000 at cryptographic sizes.
- That saving does not keep up at larger sizes (the exponential growth of the Kangaroos limits the reduction of the modulo we can have).
- It's unlikely that we can get real asymptotic improvements from this idea.

- Using ℓ^k-torsions reduces the total time by a factor between 20,000 and 160,000 at cryptographic sizes.
- That saving seems to hold at larger sizes, and asymptotic arguments may be possible...

- $20{,}000 \approx 140^2$, so each savings is comparable to reducing by two powers of $\log q$, and four powers of $\log q$ when combining both ideas (but that's really a change in the constant, not the asymptotic form).

- For groups of up to 2^{256} elements, combining the two approaches saves a factor close to $0.5 \cdot 10^9$.