The State of HEC Implementation (The Large Prime Field case)

Roberto Maria Avanzi

COSY – Ruhr University of Bochum
IEM – University of Duisburg–Essen

Research partially supported by the EU through the projects AREHCC and ECRYPT

The State of HEC Implementation (The Large Prime Field case) and one More Thing

Roberto Maria Avanzi

COSY – Ruhr University of Bochum
IEM – University of Duisburg–Essen

Research partially supported by the EU through the projects AREHCC and ECRYPT

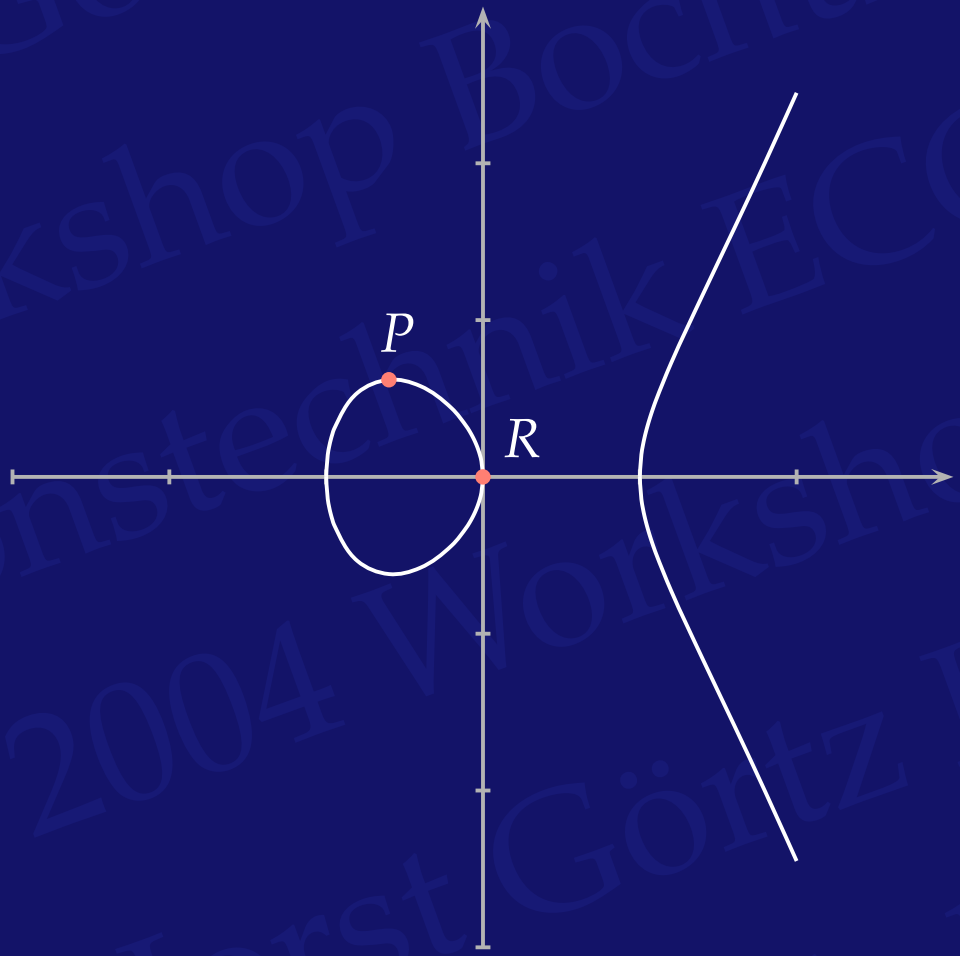
Outline

- ECC, HEC and their arithmetic.
- Why field arithmetic is crucial.
- Setting goals.
- The arithmetic library `nuMONGO`.
- Lazy and incomplete reduction in the formulae.
- Experiments, performance, results for ECC/HEC.
- Trace Zero Varieties ...
- ... and their performance.
- Conclusions and considerations.

ECC, HEC and their arithmetic

Elliptic curve group law in $E(\mathbb{R})$

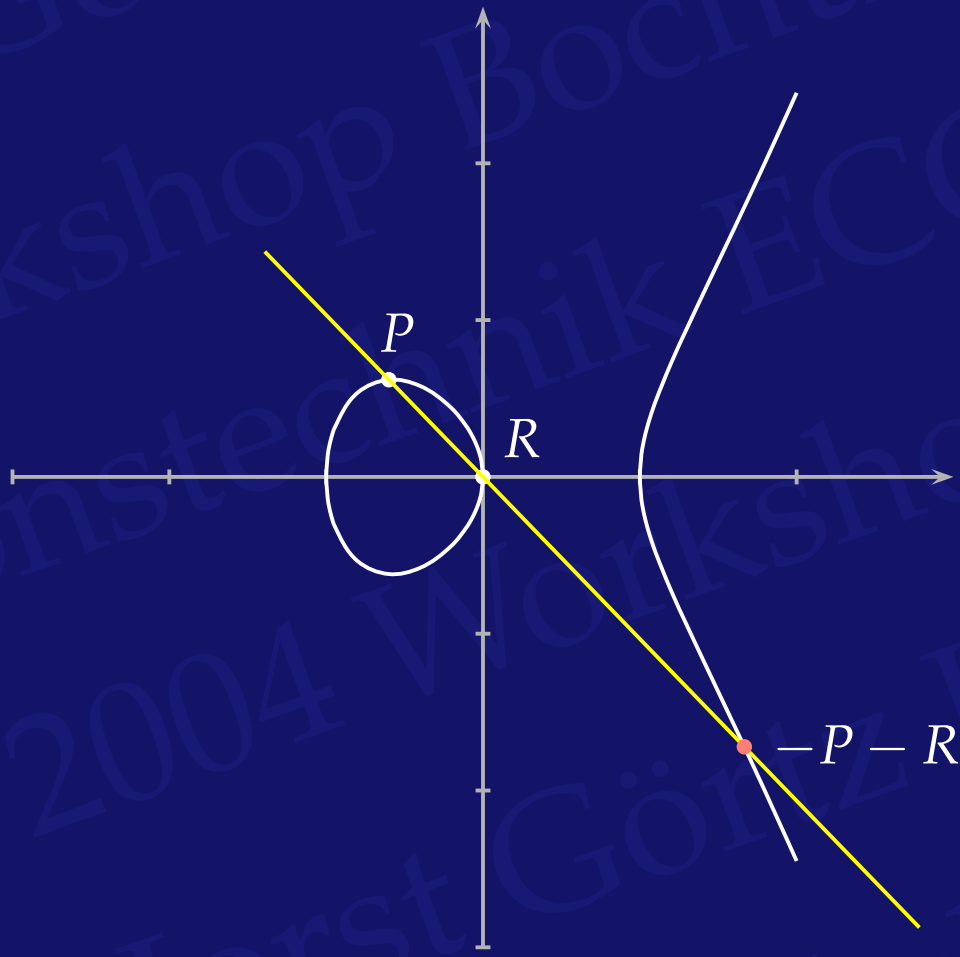
$$y^2 = x^3 - x$$



Courtesy of Ciet, Lange, Sica and Quisquater

Elliptic curve group law in $E(\mathbb{R})$

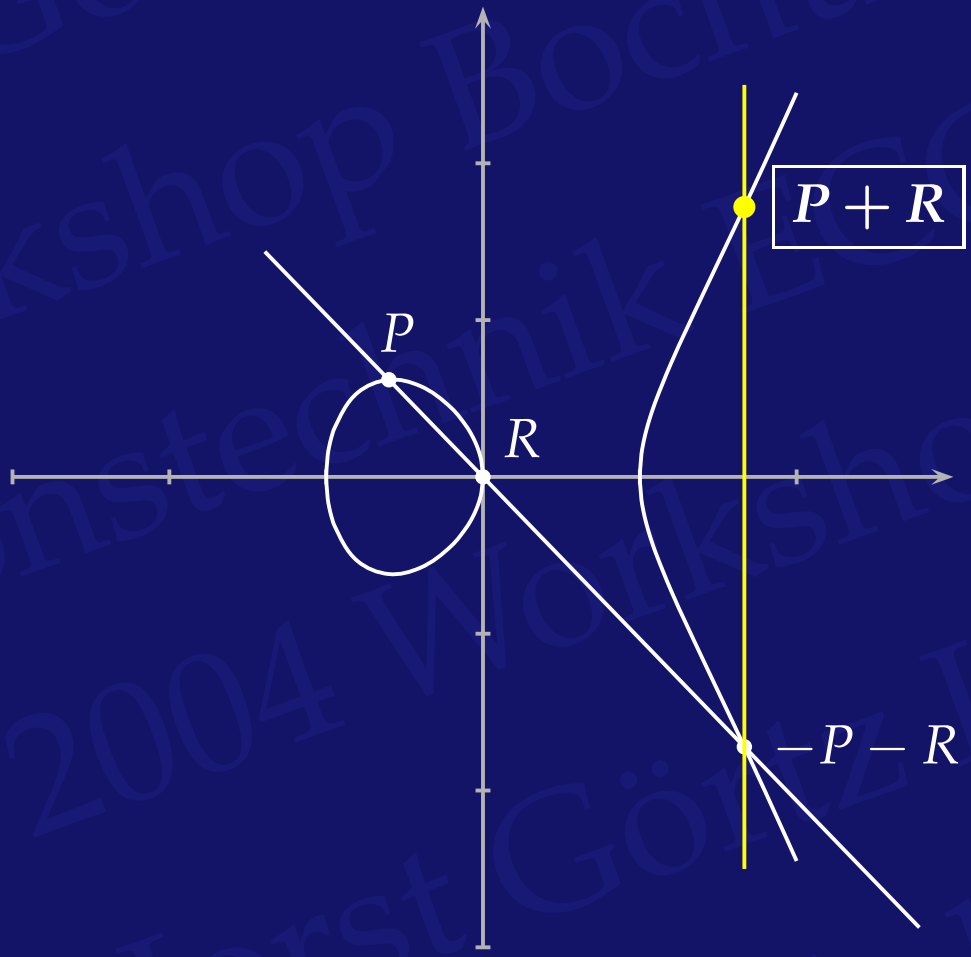
$$y^2 = x^3 - x$$



Courtesy of Ciet, Lange, Sica and Quisquater

Elliptic curve group law in $E(\mathbb{R})$

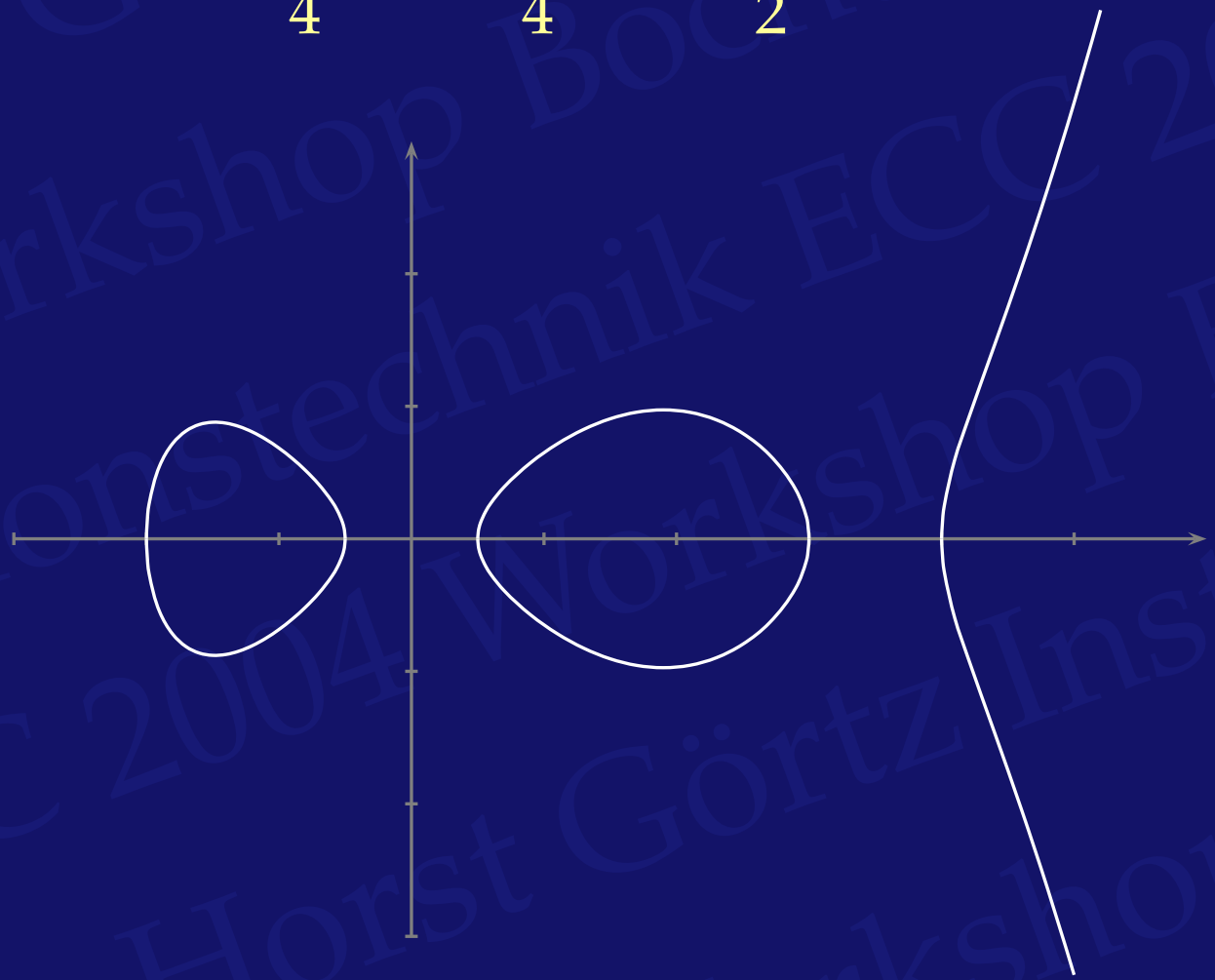
$$y^2 = x^3 - x$$



Courtesy of Ciet, Lange, Sica and Quisquater

Hyperelliptic curves: genus 2

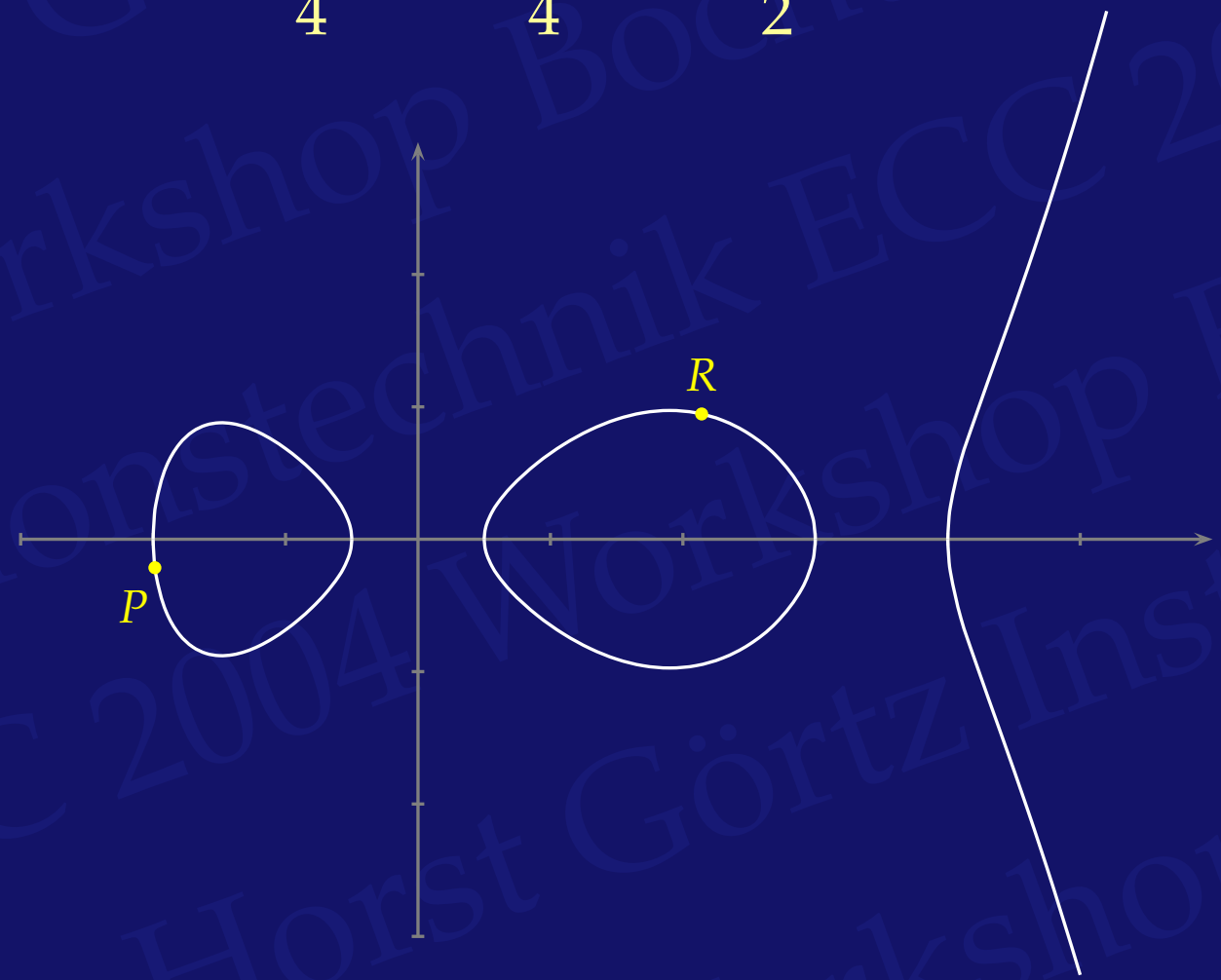
$$y^2 = x^5 - 5\,x^4 - \frac{9}{4}\,x^3 + \frac{101}{4}\,x^2 + \frac{1}{2}\,x - 6$$



Courtesy of Tanja Lange and Nicolas Thériault

Hyperelliptic curves: genus 2

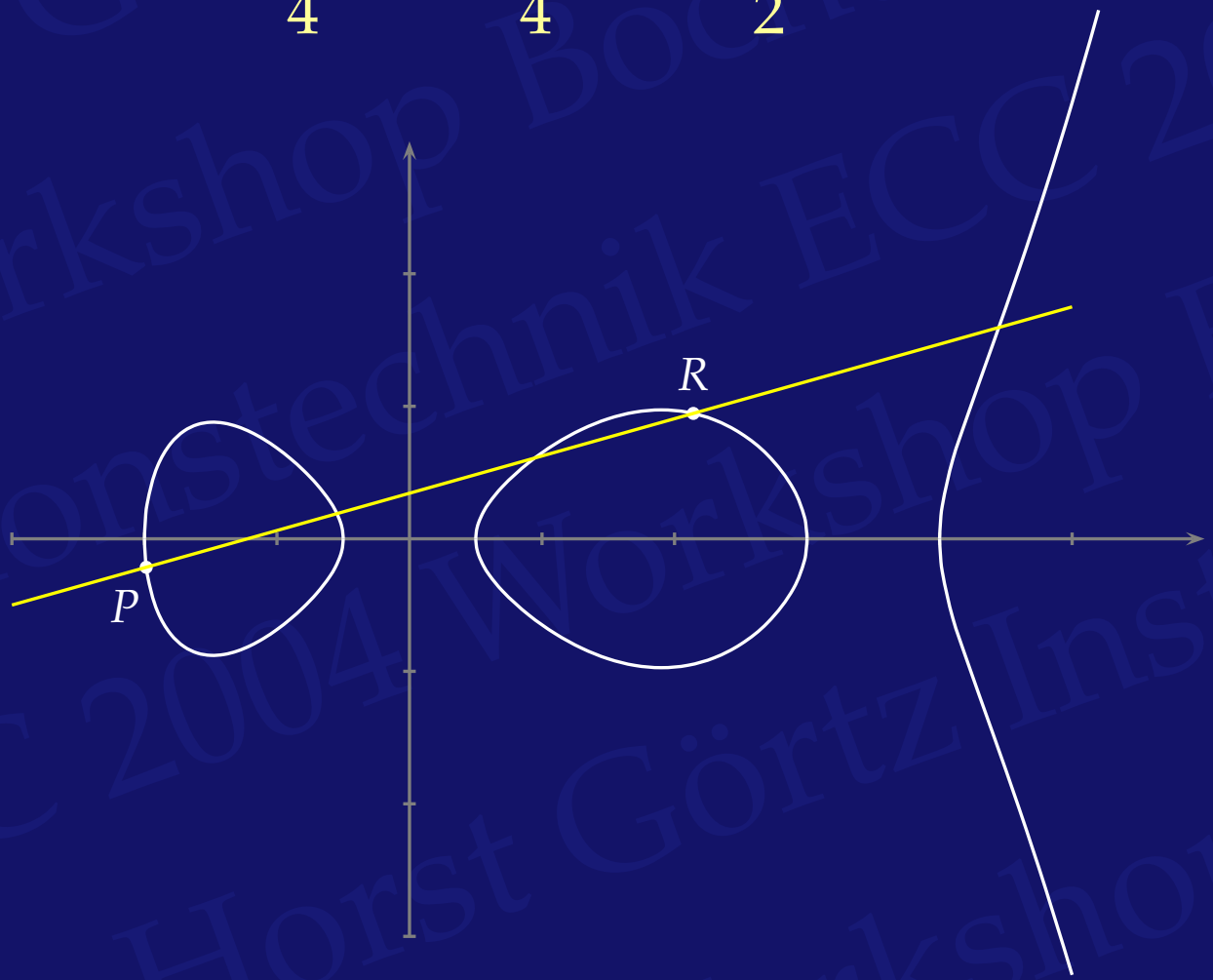
$$y^2 = x^5 - 5\,x^4 - \frac{9}{4}\,x^3 + \frac{101}{4}\,x^2 + \frac{1}{2}\,x - 6$$



Courtesy of Tanja Lange and Nicolas Thériault

Hyperelliptic curves: genus 2

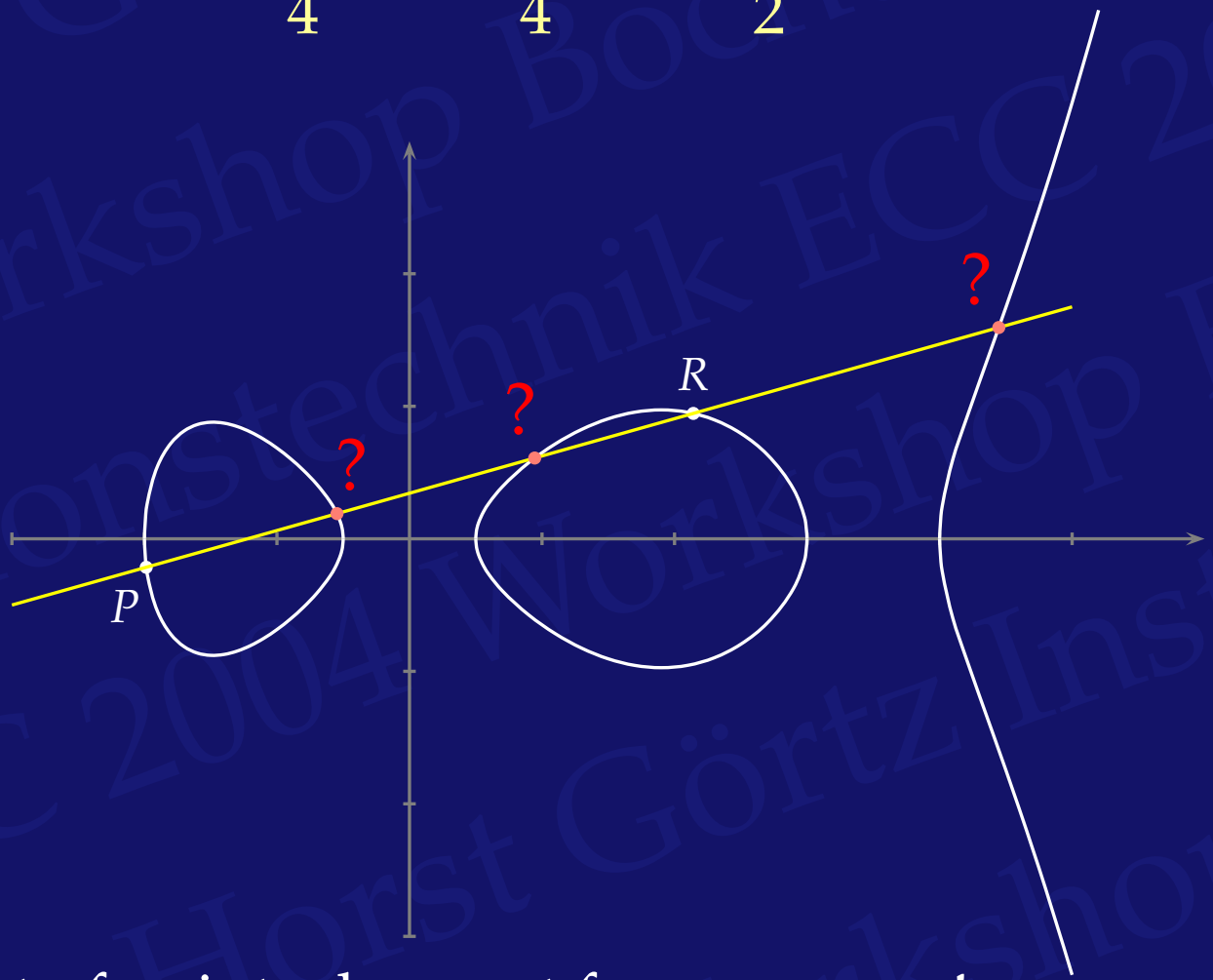
$$y^2 = x^5 - 5\,x^4 - \frac{9}{4}\,x^3 + \frac{101}{4}\,x^2 + \frac{1}{2}\,x - 6$$



Courtesy of Tanja Lange and Nicolas Thériault

Hyperelliptic curves: genus 2

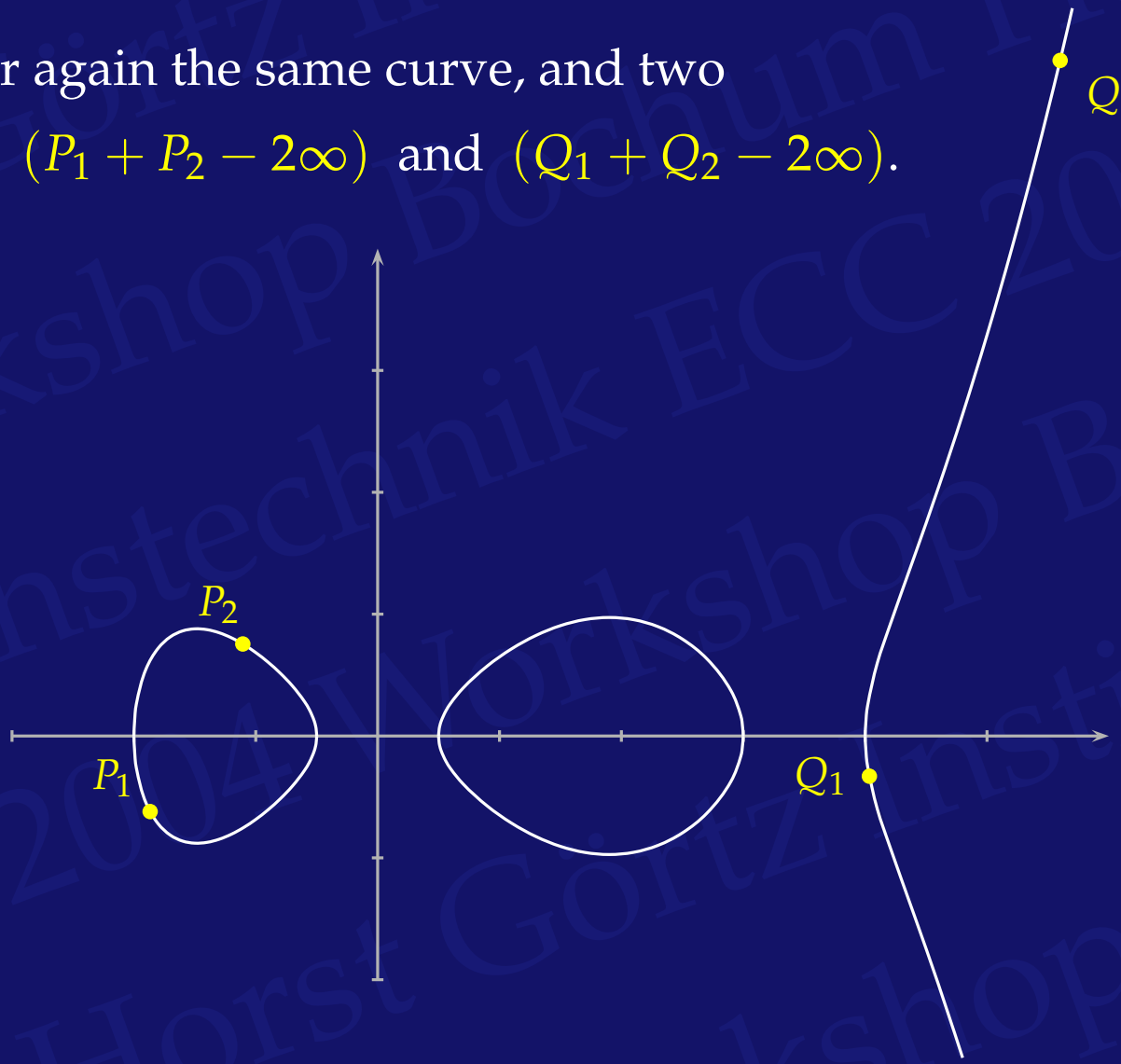
$$y^2 = x^5 - 5\,x^4 - \frac{9}{4}\,x^3 + \frac{101}{4}\,x^2 + \frac{1}{2}\,x - 6$$



The set of points does not form a group!

Addition in the Jacobian (genus 2)

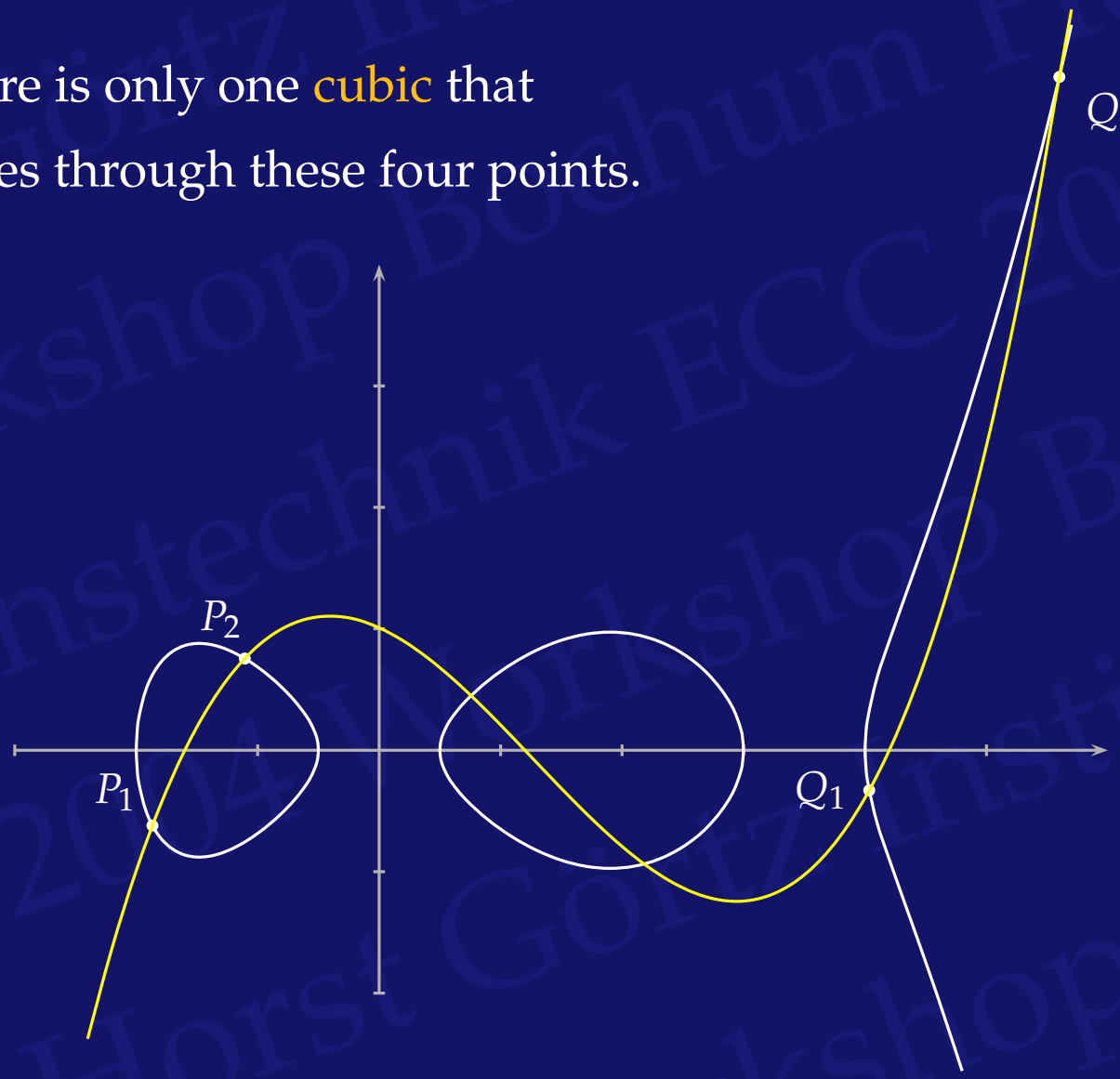
Consider again the same curve, and two divisors $(P_1 + P_2 - 2\infty)$ and $(Q_1 + Q_2 - 2\infty)$.



Courtesy of Tanja Lange and Nicolas Thériault

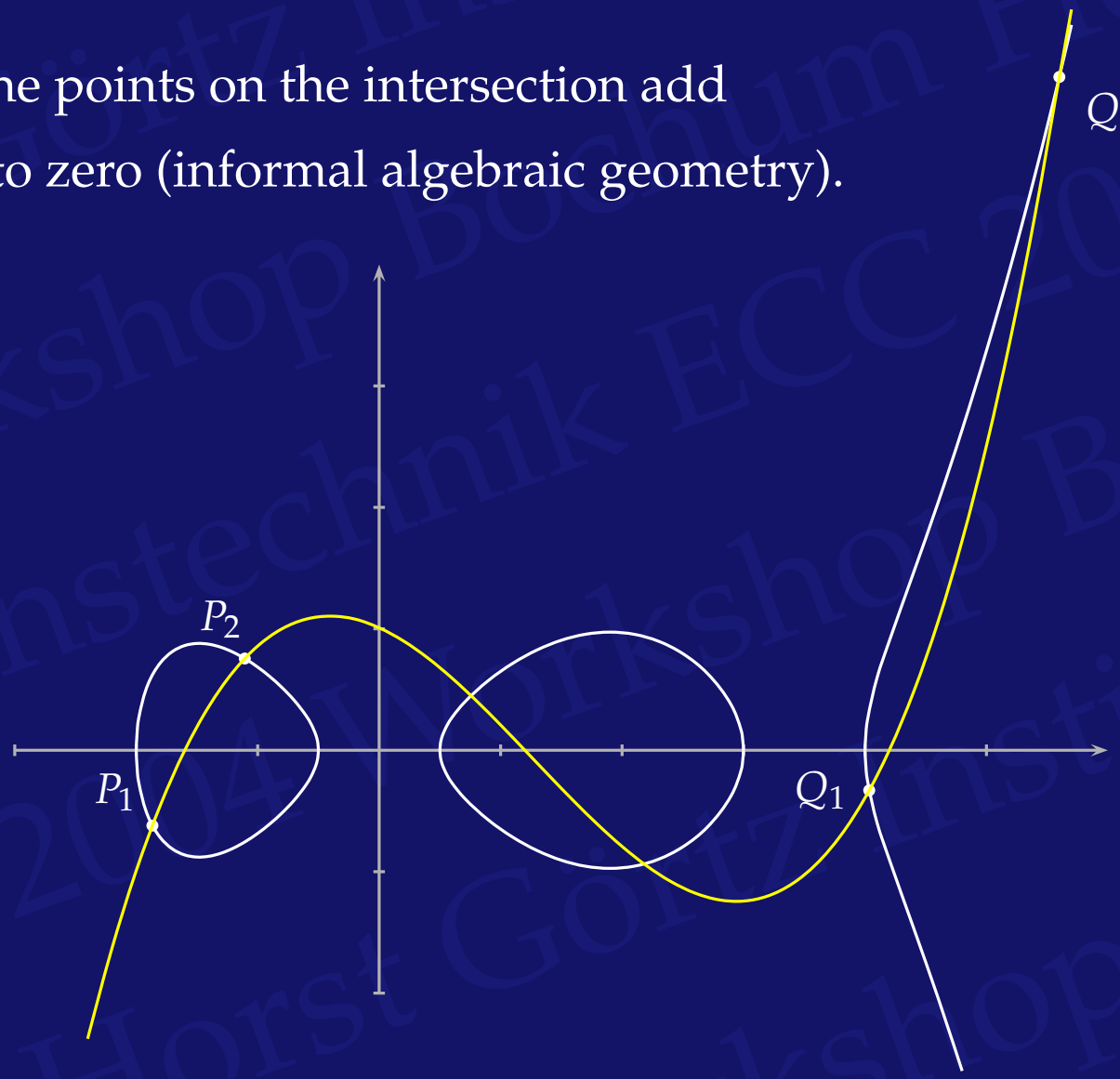
Addition in the Jacobian (genus 2)

There is only one cubic that goes through these four points.



Addition in the Jacobian (genus 2)

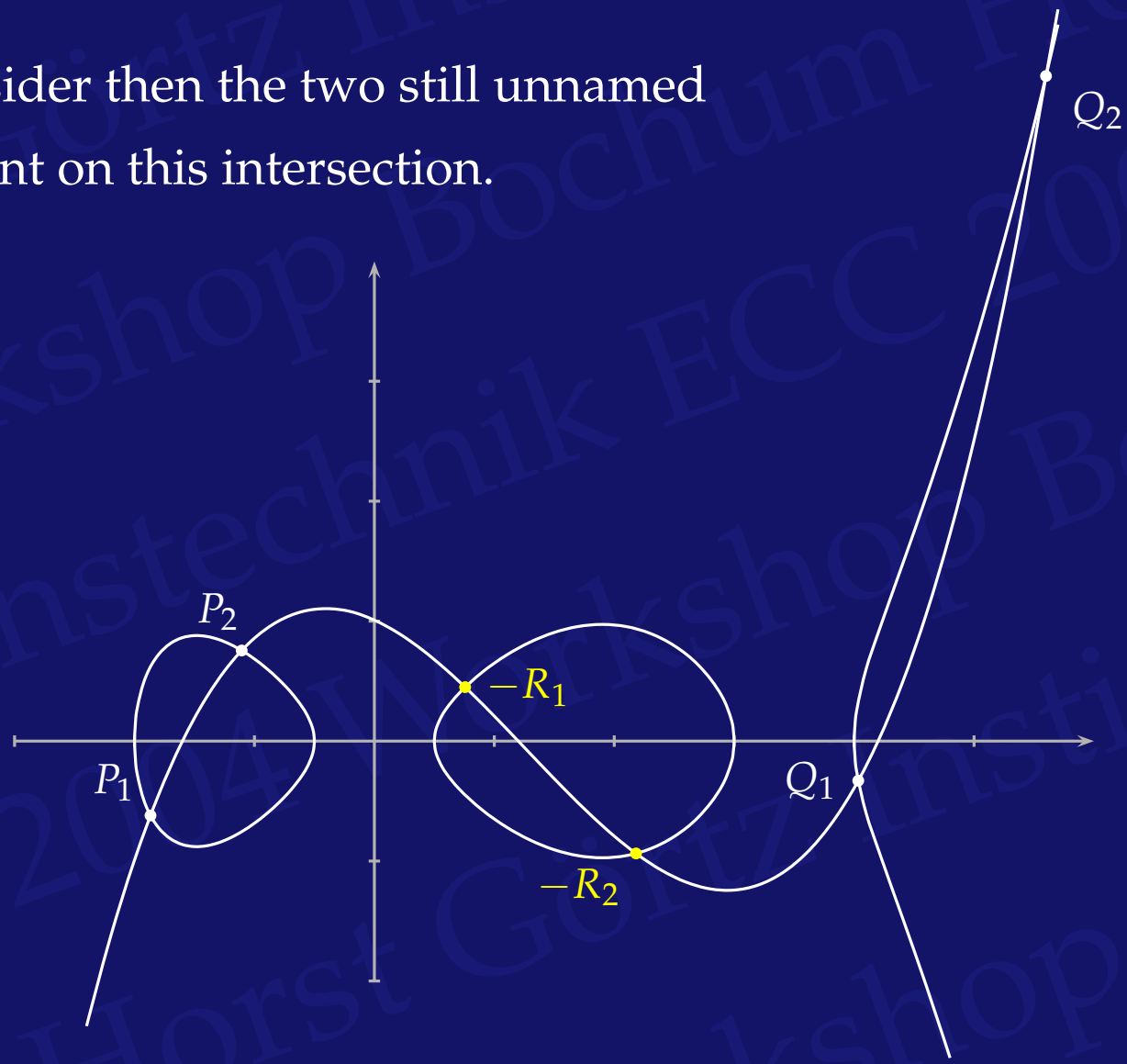
All the points on the intersection add up to zero (informal algebraic geometry).



Courtesy of Tanja Lange and Nicolas Thériault

Addition in the Jacobian (genus 2)

Consider then the two still unnamed point on this intersection.

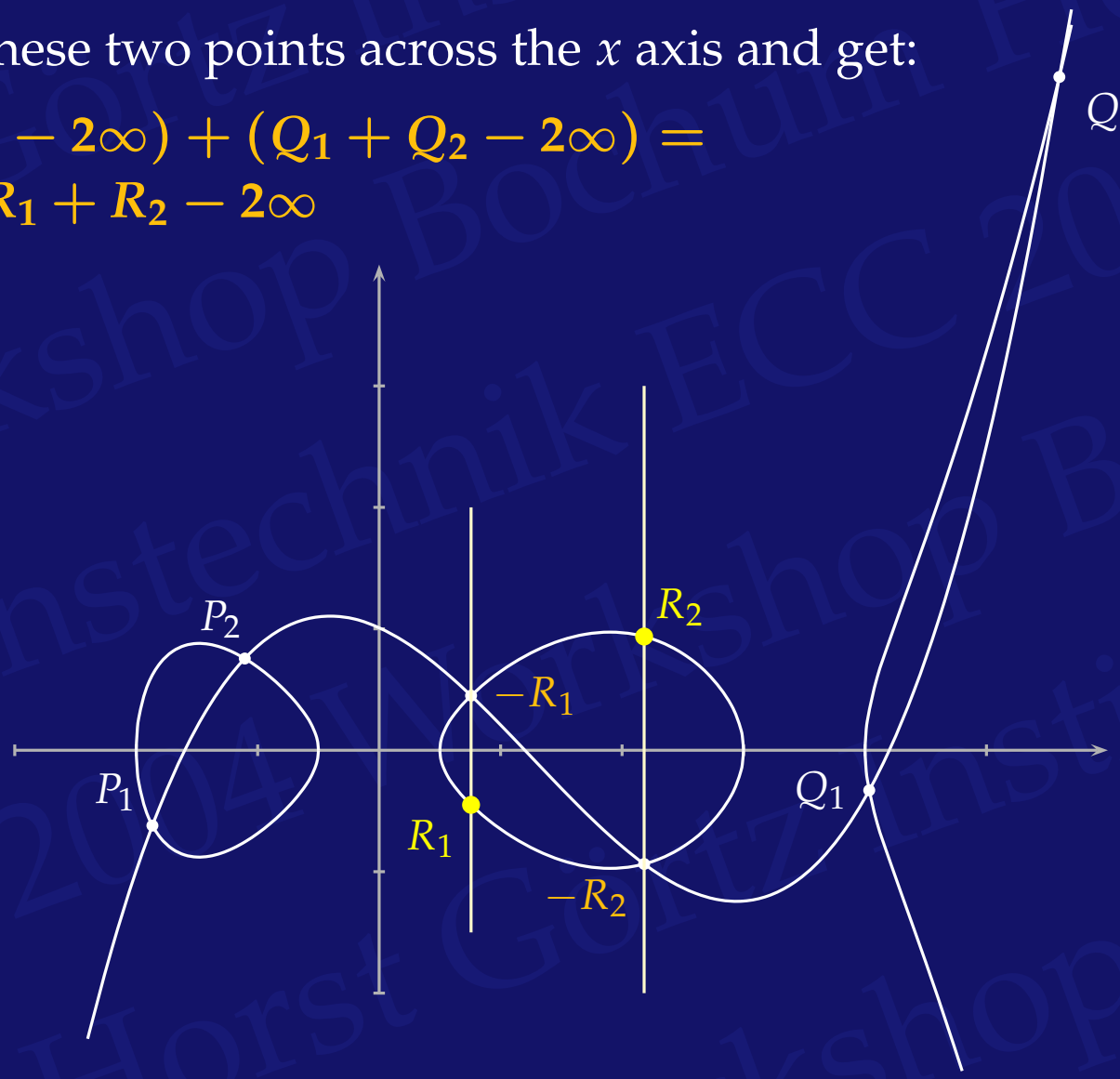


Courtesy of Tanja Lange and Nicolas Thériault

Addition in the Jacobian (genus 2)

We reflect these two points across the x axis and get:

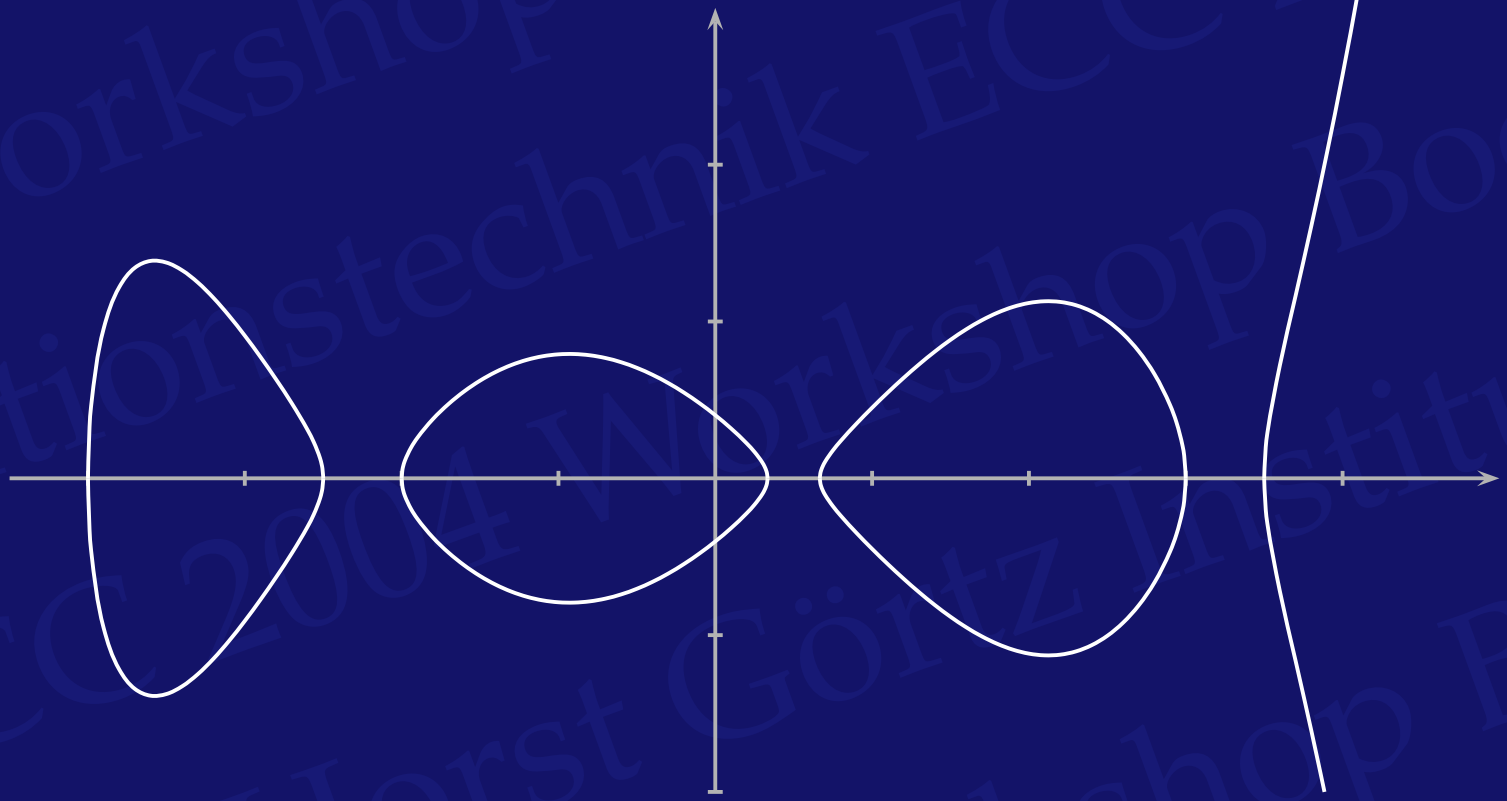
$$(P_1 + P_2 - 2\infty) + (Q_1 + Q_2 - 2\infty) = \\ = R_1 + R_2 - 2\infty$$



Addition in the Jacobian (genus 3)

A genus 3 curve:

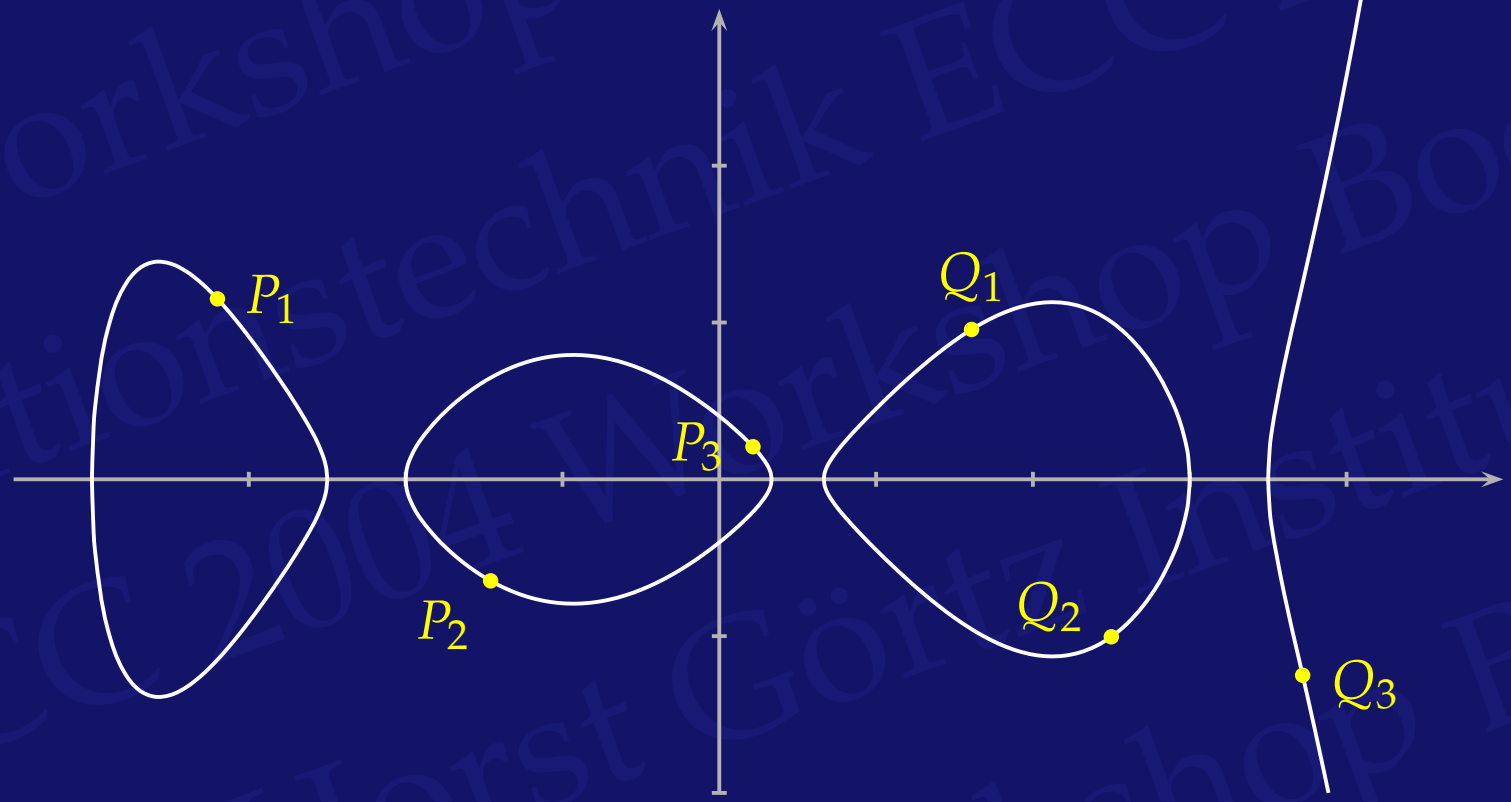
$$y^2 = x^7 + \tfrac{1}{2}x^6 - \tfrac{847}{144}x^5 - \tfrac{325}{144}x^4 + \\ + \tfrac{1763}{192}x^3 + \tfrac{403}{144}x^2 - \tfrac{1667}{576}x + \tfrac{35}{96}$$



Courtesy of Nicolas Thériault

Addition in the Jacobian (genus 3)

We want to add the divisors
$D_1 = P_1 + P_2 + P_3 - 3\infty$ and
$D_2 = Q_1 + Q_2 + Q_3 - 3\infty$

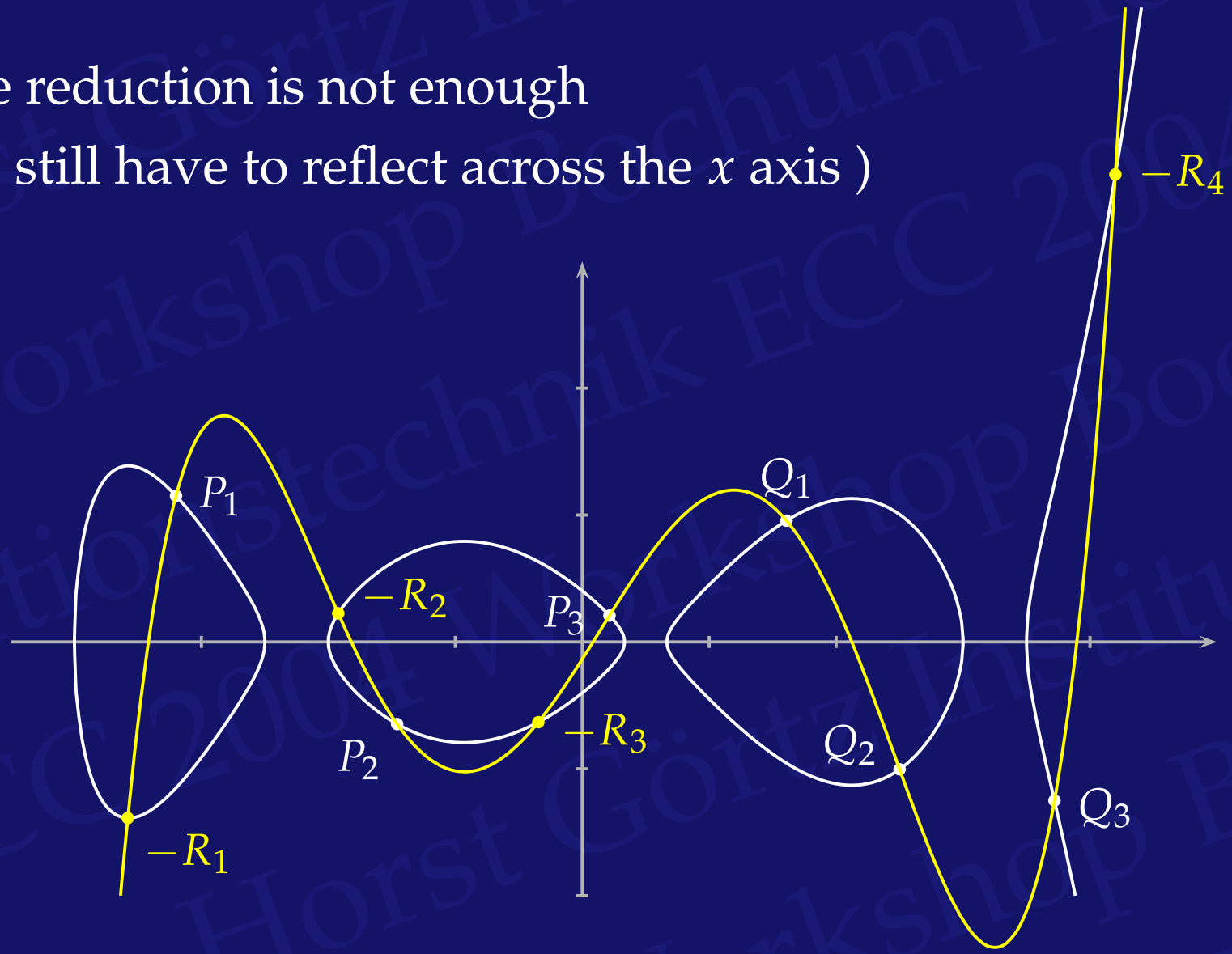


Courtesy of Nicolas Thériault

Addition in the Jacobian (genus 3)

One reduction is not enough

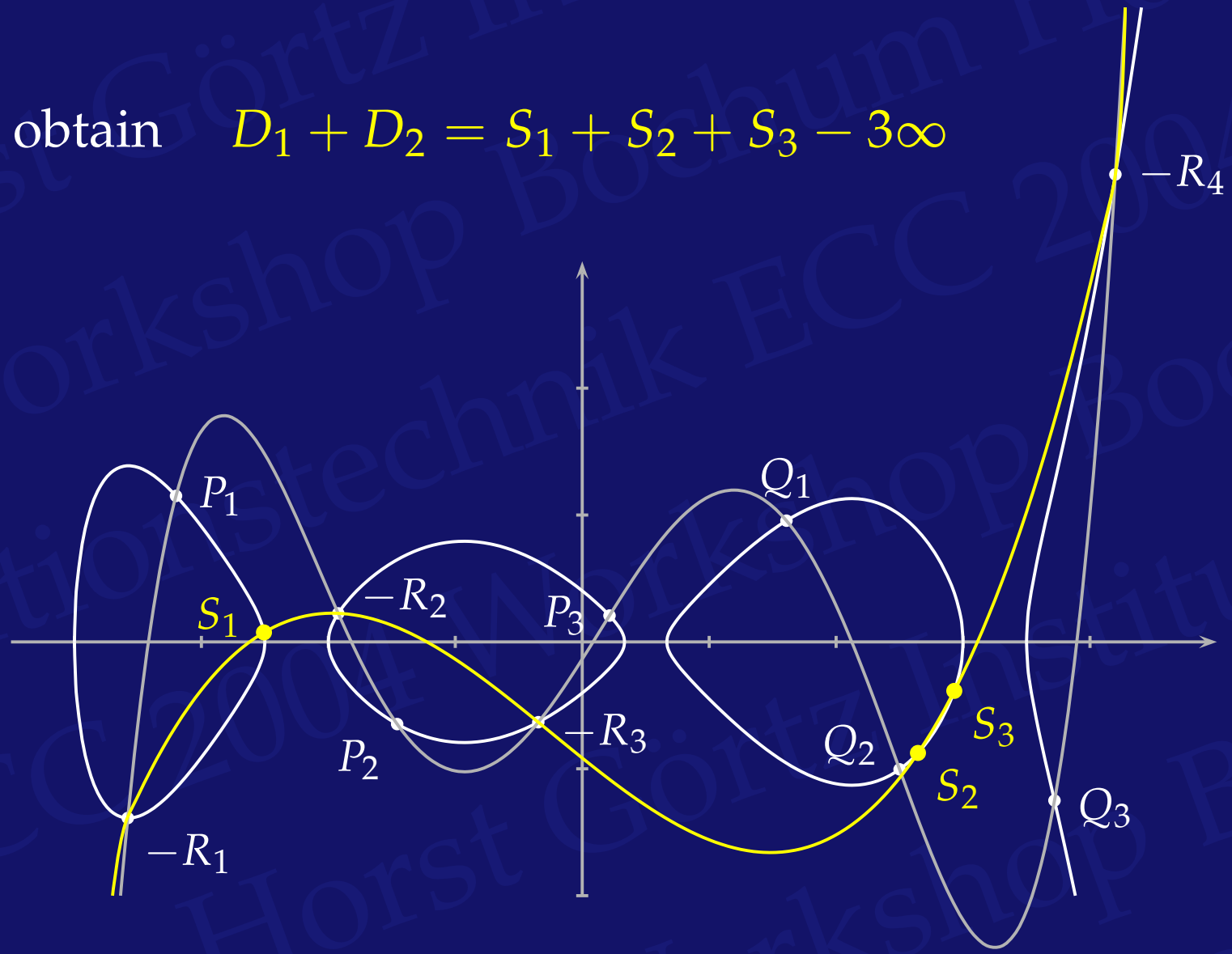
(we still have to reflect across the x axis)



Courtesy of Nicolas Thériault

Addition in the Jacobian (genus 3)

We obtain $D_1 + D_2 = S_1 + S_2 + S_3 - 3\infty$



Courtesy of Nicolas Thériault

A pattern

- Use sets of (in general) g points on curves of genus g.
- Intersecting original curve with a second one gives a set of points that "adds up to zero" (including the point at infinity with correct multiplicity). Formally, this set is the support of the divisor of a function on the curve.
- Use this second fact to "reduce" the number of points in a formal sum of divisor (i.e. putting all points together).
- This means we work in the set of divisors on the curve ...
- ... modulo the divisors of functions.
- Our objects are divisor classes.

Divisor Classes and Mumford Representation

(A flashback from CHES 2003...)

This defines a group, the Jacobian of $\mathcal{C}$, $\mathrm{Jac}(\mathcal{C})$.

If definition field $\mathbb{F}_q$, then $\#\,\mathrm{Jac}(\mathcal{C}) \approx q^g$.

But working with "point sets" and
intersecting curves is very inefficient.

Better:

- Mumford representation and
- Cantor's algorithm $\Rightarrow$ explicit formulæ.

Divisor Classes and Mumford Representation

(A flashback from CHES 2003...)

Curve $\mathcal{C} : y^2 + h(x)y = f(x)$

Let $D = \sum m_P P - \left(\sum m_P\right)\infty$ have $\sum m_P \leqslant g$.
(this is degree of associated effective divisor)

D represented by unique pair of polynomials
$U(t), V(t) \in \mathbb{F}_q[t]$ with: $g \geqslant \deg_t U > \deg_t V$, U monic.

$$\begin{cases} U(t) = \prod (t - x_P)^{m_P} \\ V(x_P) = y_P \ \text{ for all } P \\ U(t) \ \text{ divides } \ f(t) - V(t)^2 - h(t)V(t) \end{cases}$$

Coordinates of D: the coefficients of U and V .

Cantor's Algorithm

In: $D_1 = [U_1, V_1], D_2 = [U_2, V_2], \mathcal{C} : y^2 + h(x)y = f(x)$

Out: $D = [U, V]$ with $D = D_1 + D_2$

1. $d \leftarrow \gcd(U_1, U_2, V_1 + V_2 + h) =$
   $= s_1 U_1 + s_2 U_2 + s_3(V_1 + V_2 + h)$
2. $U \leftarrow U_1 U_2 / d^2$
   $V \leftarrow \dfrac{s_1 U_1 V_2 + s_2 U_2 V_1 + s_3(V_1 V_2 + f)}{d} \bmod U$
3. DO
4. $U \leftarrow \dfrac{f - V^2 - hV}{U}, \quad V \leftarrow (-h - V) \bmod U$
5. WHILE $(\deg U > g)$
6. Make U monic
7. RETURN $([U, V])$

Why field arithmetic is crucial

- Cantor's algorithm makes use of polynomial arithmetic to add divisors.

To the field arithmetic

- Cantor's algorithm makes use of polynomial arithmetic to add divisors.
- Which in turn is based on field arithmetic.

To the field arithmetic

- Cantor's algorithm makes use of polynomial arithmetic to add divisors.
- Which in turn is based on field arithmetic.
- The explicit formulae (Harley, Lange, Matsuo, Tsuji, Chao et. al., Pelzl, Paar, Wollinger, Sakai, Sakurai, and many others) make the operations, well, explicit.

To the field arithmetic

- Cantor's algorithm makes use of polynomial arithmetic to add divisors.
- Which in turn is based on field arithmetic.
- The explicit formulae (Harley, Lange, Matsuo, Tsuji, Chao et. al., Pelzl, Paar, Wollinger, Sakai, Sakurai, and many others) make the operations, well, explicit.
- Key to performance is then efficient field arithmetic.

To the field arithmetic

- Cantor's algorithm makes use of polynomial arithmetic to add divisors.
- Which in turn is based on field arithmetic.
- The explicit formulae (Harley, Lange, Matsuo, Tsuji, Chao et. al., Pelzl, Paar, Wollinger, Sakai, Sakurai, and many others) make the operations, well, explicit.
- Key to performance is then efficient field arithmetic.
- In fact, standard field arithmetic implementations penalize HEC more than ECC.

  We shall now see why.

The problems

- Per-field operation overheads in software libraries.
  - Fixed function call overhead.
  - Variable-length operands processed in loops:
    - A lot of control code...
    - ...and branch mispredictions, esp. for smaller operands.
  - Memory management costs.

The problems

- Per-field operation overheads in software libraries.
- As genus increases ($1 \to 2 \to 3 \to \ldots$) ...

The problems

- Per-field operation overheads in software libraries.
- As genus increases ($1 \to 2 \to 3 \to \ldots$) ...
  - ... field size decreases

The problems

- Per-field operation overheads in software libraries.
- As genus increases ($1 \to 2 \to 3 \to \ldots$) ...
  - ... field size decreases ($160 \to 80 \to 54 \to \ldots$)

The problems

- Per-field operation overheads in software libraries.
- As genus increases ($1 \to 2 \to 3 \to \ldots$) ...
  - ... field size decreases ($160 \to 80 \to 54 \to \ldots$) ...
  - ... and number of field operations per group operations increases (grows approximately like g^3) ...

The problems

- Per-field operation overheads in software libraries.
- As genus increases ($1 \to 2 \to 3 \to \ldots$) ...
  - ... field size decreases ($160 \to 80 \to 54 \to \ldots$) ...
  - ... and number of field operations per group operations increases (grows approximately like g^3) ...
  - ... whereas the bit operations per field operation do not decrease as quickly ...

The problems

- Per-field operation overheads in software libraries.
- As genus increases ($1 \to 2 \to 3 \to \ldots$) ...
  - ... field size decreases ($160 \to 80 \to 54 \to \ldots$) ...
  - ... and number of field operations per group operations increases (grows approximately like g^3) ...
  - ... whereas the bit operations per field operation do not decrease as quickly ...
  - ... hence the overheads (per field operation) get worse!

The problems

- Per-field operation overheads in software libraries.
- As genus increases ($1 \to 2 \to 3 \to \ldots$) ...
  - ... field size decreases ($160 \to 80 \to 54 \to \ldots$) ...
  - ... and number of field operations per group operations increases (grows approximately like g^3) ...
  - ... whereas the bit operations per field operation do not decrease as quickly ...
  - ... hence the overheads (per field operation) get worse!
- The penalties grow dramatically with the genus.

The problems

- Per-field operation overheads in software libraries.
- As genus increases ($1 \to 2 \to 3 \to \ldots$) ...
  - ... field size decreases $(160 \to 80 \to 54 \to \ldots)$ ...
  - ... and number of field operations per group operations increases (grows approximately like g^3) ...
  - ... whereas the bit operations per field operation do not decrease as quickly ...
  - ... hence the overheads (per field operation) get worse!
- The penalties grow dramatically with the genus.
- They quickly become the dominant part of the computation!

The problems

- Per-field operation overheads in software libraries.
- As genus increases ($1 \to 2 \to 3 \to \ldots$) ...
  - ... field size decreases ($160 \to 80 \to 54 \to \ldots$) ...
  - ... and number of field operations per group operations increases (grows approximately like g^3) ...
  - ... whereas the bit operations per field operation do not decrease as quickly ...
  - ... hence the overheads (per field operation) get worse!
- The penalties grow dramatically with the genus.
- They quickly become the dominant part of the computation!

Fair comparison of ECC with HEC taking the overheads into account missing in literature. Excellent work on software implementations of ECC + comparison of methods (Hankerson et al.) – & many scattered results on HEC.

Setting Goals

Goals of project

- Implementation on personal computers of curves over large prime fields.
  A lot of work has been done on binary fields for HEC (Bochum) – in comparison the prime fields have been just touched.

Goals of project

- Implementation on personal computers of curves over large prime fields.
- Design software library which tries to remove the overheads – for all required operand sizes.
  From 32 bits to 256 bits, to allow implementation of, say, 120 bit genus 3 HEC and 256 bit ECC. Or 512 bit genus 2, or ...

Goals of project

- Implementation on personal computers of curves over large prime fields.
- Design software library which tries to remove the overheads – for all required operand sizes.
- Generic library. No special primes.
  At the moment there is little reserch on HEC with primes of special type.
  $\Rightarrow$ comparison with generic primes.

Goals of project

- Implementation on personal computers of curves over large prime fields.
- Design software library which tries to remove the overheads – for all required operand sizes.
- Generic library. No special primes.
- Find optimal routines for multiplication and inversion for each operand size.
  - Optimize performance for operands of "n words and a half". Works for 48 bits: good for genus 3.

Goals of project

- Implementation on personal computers of curves over large prime fields.
- Design software library which tries to remove the overheads – for all required operand sizes.
- Generic library. No special primes.
- Find optimal routines for multiplication and inversion for each operand size.
- Investigate lazy / incomplete reduction techniques for the various ECC and HEC ($g \leqslant 3$) group formulas.

Goals of project

- Implementation on personal computers of curves over large prime fields.
- Design software library which tries to remove the overheads – for all required operand sizes.
- Generic library. No special primes.
- Find optimal routines for multiplication and inversion for each operand size.
- Investigate lazy / incomplete reduction techniques for the various ECC and HEC ($g \leqslant 3$) group formulas.
- Identify areas for future research to focus for making low genus HEC more competitive .

Goals of project

- Implementation on personal computers of curves over large prime fields.
- Design software library which tries to remove the overheads – for all required operand sizes.
- Generic library. No special primes.
- Find optimal routines for multiplication and inversion for each operand size.
- Investigate lazy / incomplete reduction techniques for the various ECC and HEC ($g \leqslant 3$) group formulas.
- Identify areas for future research.
- Speed is not the first objective - rather, it is a consequence of the design.

The arithmetic library

nuMONGO

nuMONGO design principles

- Use Montgomery's reduction algorithm.
  (Fastest general-purpose.)

nuMONGO design principles

- Use Montgomery's reduction algorithm.
- Separate routines for each operand size.
  - Choose best set of algorithms for each size. Comparison of different multiplication and inversion algorithms.

nuMONGO design principles

- Use Montgomery's reduction algorithm.
- Separate routines for each operand size.
- Inline when it gives performance increase. Try all combinations of inlining at different levels in code.
  - For example, multiplications are inlined up to 160 bits - for larger operands they will be subroutines – otherwise the code for the group operations will explode.

nuMONGO design principles

- Use Montgomery's reduction algorithm.
- Separate routines for each operand size.
- Inline when it gives performance increase. Try all combinations of inlining at different levels in code.
- Use binary extended gcd for almost-inversion, followed by multiplication by a suitable power of two from a table, and Montgomery reduction.

nuMONGO design principles

- Use Montgomery's reduction algorithm.
- Separate routines for each operand size.
- Inline when it gives performance increase. Try all combinations of inlining at different levels in code.
- Use binary extended gcd for almost-inversion, followed by multiplication by a suitable power of two from a table, and Montgomery reduction.
- Lazy and incomplete reduction.

nuMONGO design principles

- Use Montgomery's reduction algorithm.
- Separate routines for each operand size.
- Inline when it gives performance increase. Try all combinations of inlining at different levels in code.
- Use binary extended gcd for almost-inversion, followed by multiplication by a suitable power of two from a table, and Montgomery reduction.
- Lazy and incomplete reduction.
- Written in C++ with no "expensive' features, and inline assembler. Portable version exists.

To multiply "small" large integers:

`Input:` $a = (a_{\ell-1}, \ldots, a_1, a_0)$ and $b = (b_{\ell-1}, \ldots, b_0)$

`Output:` $r = (r_{2\ell-1}, \ldots, r_0) = ab$.

Schoolbook multiplication (operand scanning)

1. $r_0 \leftarrow 0, \ldots, r_{2\ell-1} \leftarrow 0$
2. for i from 0 to $\ell - 1$ do {
3. $\quad c \leftarrow 0$
4. $\quad$ for j from 0 to $\ell - 1$ do {
5. $\quad\quad (c, r_{j+i}) \leftarrow a_i b_j + r_{j+i} + c$ }
6. $\quad r_{i+\ell} \leftarrow c$ }
7. return(r)

To multiply "small" large integers:

Input: $a = (a_{\ell-1}, \ldots, a_1, a_0)$ and $b = (b_{\ell-1}, \ldots, b_0)$

Output: $r = (r_{2\ell-1}, \ldots, r_0) = ab$.

Comba's method (product scanning)

1. $s_0 \leftarrow 0, s_1 \leftarrow 0, s_2 \leftarrow 0$
2. for k from 0 to $2(\ell-1)$ do {
3. for each pair (i, j) such that $i + j = k$ and $0 \leqslant i, j < \ell$, do {
4. $(s_2, s_1, s_0) \leftarrow (s_2, s_1, s_0) + a_i b_j$ }
5. $r_k \leftarrow s_0, s_0 \leftarrow s_1, s_1 \leftarrow s_2, s_2 \leftarrow 0$ }
6. $r_{2\ell-1} \leftarrow s_0$
7. return(r)

These methods are then combined with Karatsuba's multiplication technique for operands which are "large enough".

$$
\begin{aligned}
&(a_1X + a_0)(b_1X + b_0) = \\
&\quad = a_0b_0 \\
&\qquad + \big((a_0 + a_1)(b_0 + b_1) - a_0b_0 - a_1b_1\big)X \\
&\qquad\quad + a_1b_1X^2
\end{aligned}
$$

One multiplication in double(d) precision is reduced to 3 multiplications in single precision (and not 4).

All combinations are tried.

Performance

Timings of basic operations in µsec (1 GHz AMD Athlon PC) and ratios

Lib / Op / Bits		32	64	96	128	160	192	256
nuMONGO	m	0.0079	0.0267	0.054	0.11	0.146	0.198	0.392
	R	0.0298	0.0487	0.097	0.159	0.241	0.319	0.493
	I	0.61	1.987	4.457	7.6	11.2	16.3	28.8
	R/m	3.77	1.824	1.796	1.445	1.651	1.61	1.258
	I/(R+m)	16.19	26.35	29.52	28.25	28.94	31.53	32.55
gmp v. 4.1	m	0.094	0.16	0.206	0.238	0.308	0.354	0.508
	R	0.234	0.423	0.65	0.81	0.986	1.154	1.528
	I	2.53	6.41	9.77	13.3	17.2	21.26	29.6
	R/m	2.489	2.644	3.155	3.403	3.201	3.26	3.008
	I/(R+m)	7.713	10.99	11.41	12.69	13.29	14.1	14.54

Consistent with Hankerson's et al. results (they have cheaper inversion for special primes and thus higher I/m ratios).

Lazy and incomplete reduction in the formulae

Incomplete reduction

Problem: compute $\sum a_i b_i \bmod p$
with p, a_i and b_i all $\leqslant w$ bits long.

Problem: compute $\sum a_i b_i \bmod p$
with p, a_i and b_i all $\leqslant w$ bits long.

Fact: Reduction algorithms efficient only if p is restricted (Bailey–Paar) or if the input is less than $p \cdot 2^w$.

Problem: compute $\sum a_i b_i \bmod p$ with p, a_i and b_i all $\leqslant w$ bits long.

Fact: Reduction algorithms efficient only if p is restricted (Bailey–Paar) or if the input is less than $p \cdot 2^w$.

Idea #1: add all the $a_i b_i$, and reduce only at end (Lim–Hwang).

Incomplete reduction

Problem: compute $\sum a_i b_i \bmod p$ with p, a_i and b_i all $\leqslant w$ bits long.

Fact: Reduction algorithms efficient only if p is restricted (Bailey–Paar) or if the input is less than $p \cdot 2^w$.

Idea #1: add all the $a_i b_i$, and reduce only at end (Lim–Hwang).

Problem: must have p small to ensure $\sum a_i b_i < p \cdot 2^w$.

Incomplete reduction

Problem: compute $\sum a_i b_i \bmod p$ with p, a_i and b_i all $\leqslant w$ bits long.

Fact: Reduction algorithms efficient only if p is restricted (Bailey–Paar) or if the input is less than $p \cdot 2^w$.

Idea #1: add all the $a_i b_i$, and reduce only at end (Lim–Hwang).

Problem: must have p small to ensure $\sum a_i b_i < p \cdot 2^w$.

Idea #2: Fully reduce when overflow occurs.

Incomplete reduction

Problem: compute $\sum a_i b_i \bmod p$ with p, a_i and b_i all $\leqslant w$ bits long.

Fact: Reduction algorithms efficient only if p is restricted (Bailey–Paar) or if the input is less than $p \cdot 2^w$.

Idea #1: add all the $a_i b_i$, and reduce only at end (Lim–Hwang).

Problem: must have p small to ensure $\sum a_i b_i < p \cdot 2^w$.

Idea #2: Fully reduce when overflow occurs.

Problem: Reduction still too frequent if p close to 2^w. Montgomery's representation does not allow adding reduced and unreduced elements.

Incomplete reduction

Solution: replace result with one $\equiv \bmod p$.
Keep most significant digits always $< p$.
Compatible with all reduction methods.
From A.-Mihăilescu's SAC 2003 paper on PAFF.

Incomplete reduction

Solution: replace result with one $\equiv \bmod p$.
Keep most significant digits always $< p$.
Compatible with all reduction methods.
From A.-Mihăilescu's SAC 2003 paper on PAFF.

- **Algorithm:** *Incomplete reduction*
  Compute: x with $x \equiv \sum_1^t a_i b_i \bmod p2^w$ and $0 \leqslant x \leqslant p2^w$
  Notation: $x = (x_{\text{hi}}, x_{\text{lo}})_{2^w}$.
  1. Initialise $x \leftarrow a_0 b_0$
  2. for $i = 1$ to t do
  3. $\quad x \leftarrow x + a_i b_i$
  4. $\quad$ if overflow or $x_{\text{hi}} \geqslant p$ then $x_{\text{hi}} \leftarrow x_{\text{hi}} - p$
- No restrictions on prime p apart from length!

Implementing the formulae: I

In the explicit formulae for ECC and HEC for $g = 2$ and 3 there are expressions of the type $\sum a_i b_i$.

Implementing the formulae: I

In the explicit formulae for ECC and HEC for $g = 2$ and 3 there are expressions of the type $\sum a_i b_i$.

HEC arithmetic based on polynomial arithmetic
$\Rightarrow$ *a lot* of expressions like $\sum a_i b_i$.
On the other hand, in ECC they occur *"by chance"*.

Implementing the formulae: I

In the explicit formulae for ECC and HEC for $g = 2$ and 3 there are expressions of the type $\sum a_i b_i$.

HEC arithmetic based on polynomial arithmetic
$\Rightarrow$ *a lot* of expressions like $\sum a_i b_i$.
On the other hand, in ECC they occur *"by chance"*.

5 coordinate systems for ECC: $\mathcal{A}, \mathcal{P}, \mathcal{J}, \mathcal{J}^c, \mathcal{J}^m$.
3 coordinate systems for genus 2 HEC: $\mathcal{A}, \mathcal{P}, \mathcal{N}$.
1 coordinate system for genus 3 HEC: $\mathcal{A}$.

Implementing the formulae: I

In the explicit formulae for ECC and HEC for $g = 2$ and 3 there are expressions of the type $\sum a_i b_i$.

HEC arithmetic based on polynomial arithmetic
$\Rightarrow$ *a lot* of expressions like $\sum a_i b_i$.
On the other hand, in ECC they occur *"by chance"*.

5 coordinate systems for ECC: $\mathcal{A}, \mathcal{P}, \mathcal{J}, \mathcal{J}^c, \mathcal{J}^m$.
3 coordinate systems for genus 2 HEC: $\mathcal{A}, \mathcal{P}, \mathcal{N}$.
1 coordinate system for genus 3 HEC: $\mathcal{A}$.

There are operations in one coordinate system as well as operations in mixed coordinate systems.

Implementing the formulae: II

Costs of Group Operations on EC and HEC

	Doubling	
	operation	costs
EC	$2\mathcal{A} = \mathcal{A}$	I, 2m, 2s, 4R
	$2\mathcal{P} = \mathcal{P}$	7m, 5s, 10R
	$2\mathcal{J} = \mathcal{J}$	4m, 6s, 8R
	$2\mathcal{J}^c = \mathcal{J}^c$	5m, 6s, 9R
	$2\mathcal{J}^m = \mathcal{J}^m$	4m, 4s, 8R
g=2	$2\mathcal{A} = \mathcal{A}$	I, 22m, 5s, 22R
	$2\mathcal{P} = \mathcal{P}$	38m, 6s, 38R
	$2\mathcal{N} = \mathcal{N}$	34m, 7s, 37R
g=3	$2\mathcal{A} = \mathcal{A}$	I, 71(m/s), 57R

Original formulae based on Cohen-Miyaji-Ono, Lange, Paar-Pelzl-Wollinger and others.

Implementing the formulae: II

Costs of Group Operations on EC and HEC

	Addition			
	operation	costs	operation	costs
EC	$\mathcal{A}+\mathcal{A}=\mathcal{A}$	I, 2m, 1s, 3R		
	$\mathcal{P}+\mathcal{P}=\mathcal{P}$	12m, 2s, 13R	$\mathcal{A}+\mathcal{P}=\mathcal{P}$	9m, 2s, 10R
	$\mathcal{J}+\mathcal{J}=\mathcal{J}$	12m, 4s, 16R	$\mathcal{A}+\mathcal{J}=\mathcal{J}$	8m, 3s, 11R
	$\mathcal{J}^c+\mathcal{J}^c=\mathcal{J}^c$	11m, 3s, 14R	$\mathcal{A}+\mathcal{J}^c=\mathcal{J}^c$	8m, 3s, 11R
	$\mathcal{J}^m+\mathcal{J}^m=\mathcal{J}^m$	13m, 6s, 19R	$\mathcal{A}+\mathcal{J}^m=\mathcal{J}^m$	9m, 5s, 14R
g=2	$\mathcal{A}+\mathcal{A}=\mathcal{A}$	I, 22m, 3s, 18R		
	$\mathcal{P}+\mathcal{P}=\mathcal{P}$	45m, 5s, 42R	$\mathcal{A}+\mathcal{P}=\mathcal{P}$	40m, 3s, 33R
	$\mathcal{N}+\mathcal{N}=\mathcal{N}$	47m, 7s, 50R	$\mathcal{A}+\mathcal{N}=\mathcal{N}$	36m, 5s, 37R
g=3	$\mathcal{A}+\mathcal{A}=\mathcal{A}$	I, 76(m/s), 55R		

Original formulae based on Cohen-Miyaji-Ono, Lange, Paar-Pelzl-Wollinger and others.

Experiments, performance, results for ECC/HEC

Which experiments

- Group sizes of $n = 128$ to 256 bits in small steps , plus some larger examples .

Which experiments

- Group sizes of $n = 128$ to 256 bits in small steps .
- For $g = 1, 2, 3$ curves over fields of about n/g bits , constructed of prime or almost prime order (by point counting or CM - using for the latter homebrew implementations) .

Which experiments

- Group sizes of $n = 128$ to 256 bits in small steps .
- For $g = 1, 2, 3$ curves over fields of about n/g bits .
- All coordinate systems.
  For mixed systems, kept precomputations in $\mathcal{A}$.

Which experiments

- Group sizes of $n = 128$ to 256 bits in small steps .
- For $g = 1, 2, 3$ curves over fields of about n/g bits .
- All coordinate systems.
- Scalar multiplication based on
  - Binary representation.
  - NAF.
  - w-NAF (see my SAC 2004 paper!)

Which experiments

- Group sizes of $n = 128$ to 256 bits in small steps .
- For $g = 1, 2, 3$ curves over fields of about n/g bits .
- All coordinate systems.
- Scalar multiplication based on Binary representation, NAF & w-NAF.
- Compared with implementation based on gmp and some results in literature. Run on commodity PCs.

Which experiments

- Group sizes of $n = 128$ to 256 bits in small steps .
- For $g = 1, 2, 3$ curves over fields of about n/g bits .
- All coordinate systems.
- Scalar multiplication based on Binary representation, NAF & w-NAF.
- Compared with implementation based on gmp and some results in literature. Run on commodity PCs.
- Relative portability always kept in mind.
  - ARM port being tested and benchmarked now.
  - PowerPC port being developed (NEW!).

Results and Comparisons: I

Comparison of running times, in msec (1 GHz AMD Athlon PC)

curve	coord.	scalar mult.	Bitlength of group order (approximate)					
			128	160	192	256	320	512
ec	$\mathcal{A}$	binary	1.671	3.074	5.385	12.619		
		wNAF	1.363	2.489	4.335	10.099		
	$\mathcal{J}^m$	binary	0.607	1.076	1.782	3.945		
		wNAF	0.474	0.838	1.395	3.048		
g=2	$\mathcal{A}$	binary	0.888	1.899	2.546	5.514	10.409	39.673
		wNAF	0.73	1.558	2.053	4.464	8.343	31.246
	$\mathcal{N}$	binary	0.844	1.564	2.038	4.413	8.265	29.11
		wNAF	0.675	1.262	1.623	3.575	6.53	22.73
g=3	$\mathcal{A}$	binary	1.896	2.992	3.597	6.001	12.66	42.907
		wNAF	1.424	2.077	2.584	4.86	9.92	34.117

Results and Comparisons: II

Timings with gmp			
ec	160	192	256
$\mathcal{A}$	5.468	8.305	15.354
$\mathcal{P}$	4.306	5.845	9.16
$\mathcal{J}$	3.775	5.4	8.878
$\mathcal{J}^c$	4.029	5.75	9.67
$\mathcal{J}^m$	3.75	5.182	9.075

nuMONGO no lazy/inc red			
ec	160	192	256
$\mathcal{A}$	3.074	5.385	12.619
$\mathcal{P}$	1.227	2.041	4.541
$\mathcal{J}$	1.109	1.829	4.069
$\mathcal{J}^c$	1.176	1.939	4.292
$\mathcal{J}^m$	1.076	1.782	3.945

Results and Comparisons: II

Timings with gmp			
ec	160	192	256
$\mathcal{A}$	5.468	8.305	15.354
$\mathcal{P}$	4.306	5.845	9.16
$\mathcal{J}$	3.775	5.4	8.878
$\mathcal{J}^c$	4.029	5.75	9.67
$\mathcal{J}^m$	3.75	5.182	9.075

Good nuMONGO			
ec	160	192	256
$\mathcal{A}$	3.074	5.385	12.619
$\mathcal{P}$	1.152	1.879	4.243
$\mathcal{J}$	1.05	1.702	3.876
$\mathcal{J}^c$	1.109	1.812	3.995
$\mathcal{J}^m$	1.076	1.782	3.945

Results and Comparisons: III

Timings with `gmp`

`hec`		160	192	256	320	512
g=2	$\mathcal{A}$	9.292	12.082	18.873	29.5	72.09
	$\mathcal{P}$	12.15	14.961	23.442	32.212	81.586
	$\mathcal{N}$	11.349	13.278	20.4	28.93	74.389
g=3	$\mathcal{A}$	19.799	22.452	40.39	59.691	129.541

`nuMONGO` no lazy/inc red

`hec`		160	192	256	320	512
g=2	$\mathcal{A}$	2.234	2.708	5.788	11.112	41.691
	$\mathcal{P}$	2.02	2.352	4.894	9.415	33.23
	$\mathcal{N}$	1.831	2.113	4.494	8.731	30.653
g=3	$\mathcal{A}$	4.469	5.184	6.52	13.54	47.372

Results and Comparisons: III

Timings with gmp

hec		160	192	256	320	512
g=2	$\mathcal{A}$	9.292	12.082	18.873	29.5	72.09
	$\mathcal{P}$	12.15	14.961	23.442	32.212	81.586
	$\mathcal{N}$	11.349	13.278	20.4	28.93	74.389
g=3	$\mathcal{A}$	19.799	22.452	40.39	59.691	129.541

Good nuMONGO

hec		160	192	256	320	512
g=2	$\mathcal{A}$	1.899	2.546	5.514	10.409	39.673
	$\mathcal{P}$	1.641	2.102	4.712	8.653	30.564
	$\mathcal{N}$	1.564	2.038	4.413	8.265	29.11
g=3	$\mathcal{A}$	2.992	3.597	6.001	12.66	42.907

Trace Zero Varieties

(Joint work with Tanja Lange)

(fruits of work mentioned at ECC 2003)

(seeds planted by Gerhard Frey at ECC 1998)

Let $\mathbb{F}_q$, $q = p^n$, $p \geqslant 5$, and let

$$\mathcal{C} \; : \; y^2 = f(x), \qquad f \in \mathbb{F}_p[x], \quad \deg f = 2g+1, \quad f \text{ monic},$$

equation of a non-singular hyperelliptic curve of genus g.

Let $\mathbb{F}_q$, $q = p^n$, $p \geqslant 5$, and let

$$\mathcal{C} \; : \; y^2 = f(x), \qquad f \in \mathbb{F}_p[x], \quad \deg f = 2g+1, \quad f \text{ monic},$$

equation of a non-singular hyperelliptic curve of genus g.

As before, consider the divisor class group of $\mathcal{C}$ over $\mathbb{F}_p$, $\mathrm{Cl}(\mathcal{C}/\mathbb{F}_p)$, but also $\mathrm{Cl}(\mathcal{C}/\mathbb{F}_q)$.

Let $\mathbb{F}_q$, $q = p^n$, $p \geqslant 5$, and let

$$\mathcal{C} \; : \; y^2 = f(x), \qquad f \in \mathbb{F}_p[x], \quad \deg f = 2g+1, \quad f \text{ monic},$$

equation of a non-singular hyperelliptic curve of genus g.

As before, consider the divisor class group of $\mathcal{C}$ over $\mathbb{F}_p$, $\mathrm{Cl}(\mathcal{C}/\mathbb{F}_p)$, but also $\mathrm{Cl}(\mathcal{C}/\mathbb{F}_q)$.

The Frobenius automorphism of the field extension, $\sigma : x \mapsto x^p$, induces an endomorphism of the divisor class group. A class (in $\mathrm{Cl}(\mathcal{C}/\mathbb{F}_q)$) represented by $[U, V]$ is mapped to $[\sigma U, \sigma V]$ *(i.e. it operates on the coefficients)*.

Let $\mathbb{F}_q$, $q = p^n$, $p \geqslant 5$, and let

$$\mathcal{C} : y^2 = f(x), \qquad f \in \mathbb{F}_p[x], \quad \deg f = 2g+1, \quad f \text{ monic},$$

equation of a non-singular hyperelliptic curve of genus g.

As before, consider the divisor class group of $\mathcal{C}$ over $\mathbb{F}_p$, $\mathrm{Cl}(\mathcal{C}/\mathbb{F}_p)$, but also $\mathrm{Cl}(\mathcal{C}/\mathbb{F}_q)$.

The Frobenius automorphism of the field extension, $\sigma : x \mapsto x^p$, induces an endomorphism of the divisor class group. A class (in $\mathrm{Cl}(\mathcal{C}/\mathbb{F}_q)$) represented by $[U, V]$ is mapped to $[\sigma U, \sigma V]$ *(i.e. it operates on the coefficients)*.

The elements of $\mathrm{Cl}(\mathcal{C}/\mathbb{F}_p)$ are the fixed points.

It satisfies a characteristic polynomial

$$P(T) = T^{2g} + a_1 T^{2g-1} + \cdots + a_g T^g + \cdots + a_1 q^{g-1} T + q^g,$$

where $a_i \in \mathbb{Z}$. Let $\tau_1, ..., \tau_{2g}$ be the roots.

It satisfies a characteristic polynomial

$$P(T) = T^{2g} + a_1 T^{2g-1} + \cdots + a_g T^g + \cdots + a_1 q^{g-1} T + q^g,$$

where $a_i \in \mathbb{Z}$. Let $\tau_1, ..., \tau_{2g}$ be the roots.

Group order over any extension field

$$|\mathrm{Cl}(\mathcal{C}/\mathbb{F}_{q^n})| = \prod_{i=1}^{2g} (1 - \tau_i^n).$$

Easy if the τ_i are known. If $n = 1$, it is $P(1)$.

It satisfies a characteristic polynomial

$$P(T) = T^{2g} + a_1 T^{2g-1} + \cdots + a_g T^g + \cdots + a_1 q^{g-1} T + q^g,$$

where $a_i \in \mathbb{Z}$. Let $\tau_1, ..., \tau_{2g}$ be the roots.

Group order over any extension field

$$|\mathrm{Cl}(\mathcal{C}/\mathbb{F}_{q^n})| = \prod_{i=1}^{2g} (1 - \tau_i^n).$$

Easy if the τ_i are known. If $n = 1$, it is $P(1)$.

In general computing group orders efficiently is a difficult problem. It has been solved in many useful instances. For HEC over prime fields first practical results are recent.

Let $\mathbb{F}_q$, $q = p^n$, $p \geqslant 5$, and let

$$\mathcal{C} \; : \; y^2 = f(x), \qquad f \in \mathbb{F}_p[x], \quad \deg f = 2g+1, \quad f \text{ monic},$$

equation of a non-singular hyperelliptic curve of genus g.

Let $\mathbb{F}_q$, $q = p^n$, $p \geqslant 5$, and let

$$\mathcal{C} \; : \; y^2 = f(x), \qquad f \in \mathbb{F}_p[x], \quad \deg f = 2g+1, \quad f \text{ monic},$$

equation of a non-singular hyperelliptic curve of genus g.

As before, consider the divisor class group of $\mathcal{C}$ over $\mathbb{F}_p$, $\mathrm{Cl}(\mathcal{C}/\mathbb{F}_p)$, but also $\mathrm{Cl}(\mathcal{C}/\mathbb{F}_q)$.
$\mathrm{Cl}(\mathcal{C}/\mathbb{F}_p)$ is a subgroup of $\mathrm{Cl}(\mathcal{C}/\mathbb{F}_q)$, hence $\mathrm{Cl}(\mathcal{C}/\mathbb{F}_q)$ is not good in itself, because order not almost prime.

Let $\mathbb{F}_q$, $q = p^n$, $p \geqslant 5$, and let

$$\mathcal{C} \; : \; y^2 = f(x), \qquad f \in \mathbb{F}_p[x], \quad \deg f = 2g+1, \quad f \text{ monic},$$

equation of a non-singular hyperelliptic curve of genus g.

As before, consider the divisor class group of $\mathcal{C}$ over $\mathbb{F}_p$, $\mathrm{Cl}(\mathcal{C}/\mathbb{F}_p)$, but also $\mathrm{Cl}(\mathcal{C}/\mathbb{F}_q)$.
$\mathrm{Cl}(\mathcal{C}/\mathbb{F}_p)$ is a subgroup of $\mathrm{Cl}(\mathcal{C}/\mathbb{F}_q)$, hence $\mathrm{Cl}(\mathcal{C}/\mathbb{F}_q)$ is not good in itself, because order not almost prime.

Need a nice way of working in quotient group $\mathrm{Cl}(\mathcal{C}/\mathbb{F}_q)/\mathrm{Cl}(\mathcal{C}/\mathbb{F}_p)$ (or something close).

Let $\mathbb{F}_q$, $q = p^n$, $p \geqslant 5$, and let

$$\mathcal{C} \; : \; y^2 = f(x), \qquad f \in \mathbb{F}_p[x], \quad \deg f = 2g+1, \quad f \text{ monic},$$

equation of a non-singular hyperelliptic curve of genus g.

Let $\mathbb{F}_q$, $q = p^n$, $p \geqslant 5$, and let

$$\mathcal{C} \; : \; y^2 = f(x), \qquad f \in \mathbb{F}_p[x], \quad \deg f = 2g+1, \quad f \text{ monic},$$

equation of a non-singular hyperelliptic curve of genus g.

Consider $\mathrm{Cl}(\mathcal{C}/\mathbb{F}_q)$ and restrict the computations to the elements $D \in \mathrm{Cl}(\mathcal{C}/\mathbb{F}_{p^n})$ s.t. $\sum_{i=0}^{n-1} \sigma^i(D) = 0$.

Let $\mathbb{F}_q$, $q = p^n$, $p \geqslant 5$, and let

$$\mathcal{C} \; : \; y^2 = f(x), \qquad f \in \mathbb{F}_p[x], \quad \deg f = 2g+1, \quad f \text{ monic},$$

equation of a non-singular hyperelliptic curve of genus g.

Consider $\mathrm{Cl}(\mathcal{C}/\mathbb{F}_q)$ and restrict the computations to the elements $D \in \mathrm{Cl}(\mathcal{C}/\mathbb{F}_{p^n})$ s.t. $\sum_{i=0}^{n-1} \sigma^i(D) = 0$.

Denote this group by G.
Let G_0 be a subgroup of "large" prime order ℓ of G.

Let $\mathbb{F}_q$, $q = p^n$, $p \geqslant 5$, and let

$$\mathcal{C} \; : \; y^2 = f(x), \qquad f \in \mathbb{F}_p[x], \quad \deg f = 2g+1, \quad f \text{ monic},$$

equation of a non-singular hyperelliptic curve of genus g.

Consider $\mathrm{Cl}(\mathcal{C}/\mathbb{F}_q)$ and restrict the computations to the elements $D \in \mathrm{Cl}(\mathcal{C}/\mathbb{F}_{p^n})$ s.t. $\sum_{i=0}^{n-1} \sigma^i(D) = 0$.

Denote this group by G.
Let G_0 be a subgroup of "large" prime order ℓ of G.

G is the trace zero (sub)variety.

Let $\mathbb{F}_q$, $q = p^n$, $p \geqslant 5$, and let

$$\mathcal{C} \; : \; y^2 = f(x), \qquad f \in \mathbb{F}_p[x], \quad \deg f = 2g+1, \quad f \text{ monic},$$

equation of a non-singular hyperelliptic curve of genus g.

Consider $\mathrm{Cl}(\mathcal{C}/\mathbb{F}_q)$ and restrict the computations to the elements $D \in \mathrm{Cl}(\mathcal{C}/\mathbb{F}_{p^n})$ s.t. $\sum_{i=0}^{n-1} \sigma^i(D) = 0$.

Denote this group by G.
Let G_0 be a subgroup of "large" prime order ℓ of G.

G is the trace zero (sub)variety.

$\sigma G_0 = G_0 \;\Rightarrow\;$ exists integer s s.t. $sg = \sigma g$ for all $g \in G_0$.

Groups considered

For security and practical reasons we limit ourselves to $g = 1$ with $n = 3$ and 5, and $g = 2$ with $n = 3$.

$$|G| = p^2 - p(1 + a_1) + a_1^2 - a_1 + 1 \qquad \text{for} \quad g = 1, n = 3 \ ,$$

$$|G| = p^4 - (a_1 + 1)p^3 + (a_1 + 1)^2 p^2 + (5a_1 - (a_1 + 1)^3)p - \\ - (5a_1(a_1^2 + a_1 + 1) - (a_1 + 1)^4) \qquad \text{for} \quad g = 1, n = 5 \ ,$$

$$|G| = p^4 - a_1 p^3 + (a_1^2 + 2a_1 - a_2 - 1)p^2 + (-a_1^2 - a_1 a_2 + 2a_1)p + \\ + a_1^2 + a_2^2 - a_1 a_2 - a_1 - a_2 + 1 \qquad \text{for} \quad g = 2, n = 3 \ .$$

Need to count points on *small* curve over $\mathbb{F}_p$.
For $g = 1$ Schoof's algorithm extremely fast.
For $g = 2$ Gaudry's algorithm very efficient.
s is also easy to compute in the considered cases.

Example: $n = 3$.
$\mathbb{F}_{p^3} = \mathbb{F}_p[\xi]$, where ξ is a root of $z^3 - 2$ irreducible.
Elements of $\mathbb{F}_q$ represented by polynomials.
Multiplication/Squaring Karatsuba-like

$$
\begin{aligned}
(a_0 + a_1\xi + a_2\xi^2)(b_0 + b_1\xi + b_2\xi^2) = \\
&= a_0b_0 + \big((a_0 + a_1)(b_0 + b_1) - a_0b_0 - a_1b_1\big)\xi \\
&\quad + \big((a_0 + a_2)(b_0 + b_2) - a_0b_0 - a_2b_2 + a_1b_1\big)\xi^2 \\
&\quad + \big((a_1 + a_2)(b_1 + b_2) - a_1b_1 - a_2b_2\big)\xi^3 + (a_2b_2)\xi^4 \ ,
\end{aligned}
$$

delaying modular reductions!

To compute the inverse of $a \in \mathbb{F}_{p^3}$ we use linear algebra.

Let $c = c_0 + c_1\xi + c_2\xi^2$ ($c_0, c_1, c_2 \in \mathbb{F}_p$) be a^{-1} where $a = a_0 + a_1\xi + a_2\xi^2 \in \mathbb{F}_{p^3}$.

$ac = 1$ can be written as:

$$\begin{pmatrix} a_0 & a_2\alpha & a_1\alpha \\ a_1 & a_0 & a_2\alpha \\ a_2 & a_1 & a_0 \end{pmatrix} \begin{pmatrix} c_0 \\ c_1 \\ c_2 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} .$$

Now invert the matrix explicitly. We need only one inversion (that of the determinant).
This makes inversion rather fast. In fact, affine coordinates are the fastest ones!

We mentioned the problem of computing s: Let's use it!

Theorem. For the three cases which we consider, there exists an efficient technique for expressing a scalar m in the form $m \equiv \sum_{i=0}^{n-1} m_i s^i \bmod \ell$ where $m_i = O(p^g)$. We have:

1. If $g = 1, n = 3$, then $|m_i| < 4p$ for $p \geqslant 79$.
2. If $g = 1, n = 5$, then $|m_0| \leqslant 2p$, $|m_1| \leqslant p/2$, $|m_2| \leqslant p$, and $|m_3| \leqslant 3p/2$ for $p \geqslant 13613$.
3. If $g = 2, n = 3$, then $|m_0| < 2p^2$, $|m_1| < p^2$ for $p \geqslant 4043549$.

(efficient = at most about as costly as a group operation).

Then we compute $\sum m_i \sigma^i(g) = \sum m_i s^i g$ in place of mg.

Using multi-exponentiation (well, multi-scalar multiplication) techniques, we can save a lot of time.

Here only binary and NAF used.

Group / Rep		Bits	128	160	192	256
Trace Zero Varieties	$g = 1, n = 3$	binary	0.655	1.337	1.837	4.17
		NAF	0.615	1.24	1.695	3.87
	$g = 1, n = 5$	binary	0.521	1.04	1.54	2.36
		NAF	0.495	1.02	1.48	2.24
	$g = 2, n = 3$	binary	1.18	2.42	3.08	4.58
		NAF	1.08	2.26	2.80	4.22
ECC & HEC	ECC	binary	0.584	1.05	1.702	3.876
		NAF	0.517	0.907	1.499	3.325
	HEC ($g = 2$)	binary	0.839	1.564	2.038	4.413
		NAF	0.755	1.391	1.778	4.002

Now, a big breath...

Conclusions and Considerations

- Always use specialized library (we already knew...).
- Lazy & incomp. red. bring speed-up from 3% to 10%.
- Fastest implementation of generic HEC up to date.
- HEC/prime fields still slower than ECC, but gap narrowed, sometimes down to about 15%.
- Modified Jacobian ECC coords seem redundant.

Conclusions and Considerations

- Always use specialized library (we already knew...).
- Lazy & incomp. red. bring speed-up from 3% to 10%.
- Fastest implementation of generic HEC up to date.
- HEC/prime fields still slower than ECC, but gap narrowed, sometimes down to about 15%.
- Modified Jacobian ECC coords seem redundant.

- Should try to find formulae for reducing amount of modular reductions, not only multiplications.
- Necessary study for special prime fields.
- HEC seems every day even more competitive.
- Trace Zero Variety very interesting for small keys.
- Extrapolating: for 320–512 bits HEC as fast as/faster than ECC (i.e. Europeans can have their 512 bit curves too ;-)

Thank you for your attention! Any questions?

Thank you for your attention! Any questions?

Made on a Mac