# Trace Zero Varieties for Cryptographic Applications

Roberto Avanzi
From joint work with Emanuele Cesena and Tanja Lange.

HGI and Faculty of Mathematics – Ruhr University Bochum

SPEED-CC, Berlin, October 13th, 2009

# What are Trace Zero Varieties?

Proposed by Gerhard Frey in 1998.

Now in *twelfth* year.

Constructive application of Weil descent.

Ideal subject to be used as an excuse to go to interesting places

- 2007: SAGA (Papeete, Tahiti, French Polynesia)
- 2009: CHiLE (Frutillar, Lakes Region, Chile)
- 2009: Zurich (Google, Lindt & Sprüngli...)
- 2009: Berlin (Ich bin ein Berliner! i.e. *I am a bismarck donut!*)

# What are Trace Zero Varieties?

Proposed by Gerhard Frey in 1998.

Now in *twelfth* year.

Constructive application of Weil descent.

Ideal subject to be used as an excuse to go to interesting places

- 2007: SAGA (Papeete, Tahiti, French Polynesia)
- 2009: CHiLE (Frutillar, Lakes Region, Chile)
- 2009: Zurich (Google, Lindt & Sprüngli...)
- 2009: Berlin (Ich bin ein Berliner! i.e. *I am a bismarck donut!*)

# What are Trace Zero Varieties?

Proposed by Gerhard Frey in 1998.

Now in *twelfth* year.

Constructive application of Weil descent.

Ideal subject to be used as an excuse to go to interesting places

- 2007: SAGA (Papeete, Tahiti, French Polynesia)
- 2009: CHiLE (Frutillar, Lakes Region, Chile)
- 2009: Zurich (Google, Lindt & Sprüngli...)
- 2009: Berlin (Ich bin ein Berliner! i.e. *I am a bismarck donut!*)

# What are Trace Zero Varieties?

Proposed by Gerhard Frey in 1998.

Now in *twelfth* year.

Constructive application of Weil descent.

Ideal subject to be used as an excuse to go to interesting places

- 2007: SAGA (Papeete, Tahiti, French Polynesia)
- 2009: CHiLE (Frutillar, Lakes Region, Chile)
- 2009: Zurich (Google, Lindt & Sprüngli...)
- 2009: Berlin (Ich bin ein Berliner! i.e. *I am a bismarck donut!*)

# What are Trace Zero Varieties?

Proposed by Gerhard Frey in 1998.

Now in *twelfth* year.

Constructive application of Weil descent.

Ideal subject to be used as an excuse to go to interesting places

- 2007: SAGA (Papeete, Tahiti, French Polynesia)
- 2009: CHiLE (Frutillar, Lakes Region, Chile)
- 2009: Zurich (Google, Lindt & Sprüngli...)
- 2009: Berlin (Ich bin ein Berliner! i.e. *I am a bismarck donut!*)

# What are Trace Zero Varieties?

Proposed by Gerhard Frey in 1998.

Now in *twelfth* year.

Constructive application of Weil descent.

Ideal subject to be used as an excuse to go to interesting places

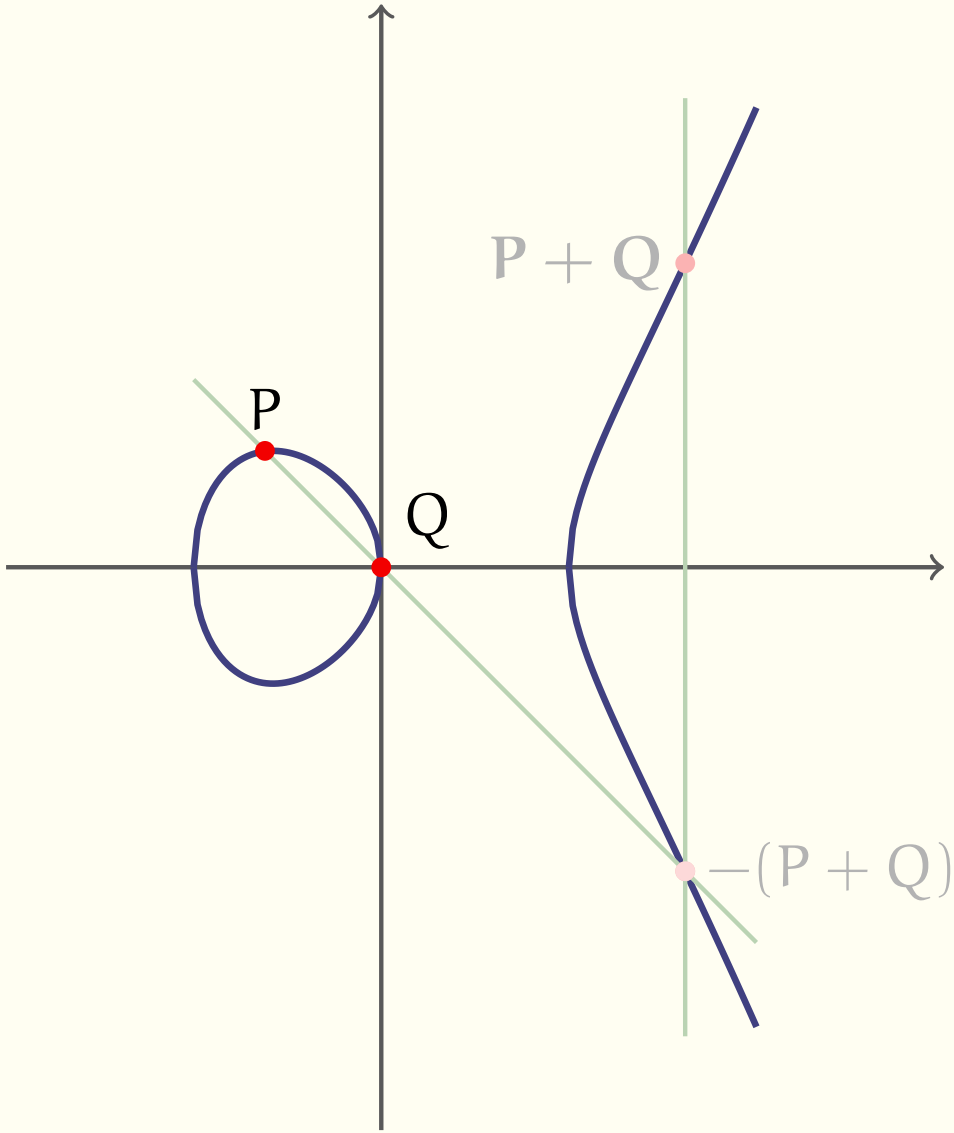
- 2007: SAGA (Papeete, Tahiti, French Polynesia)
- 2009: CHiLE (Frutillar, Lakes Region, Chile)
- 2009: Zurich (Google, Lindt & Sprüngli...)
- 2009: Berlin (Ich bin ein Berliner! i.e. *I am a bismarck donut!*)

# Outline

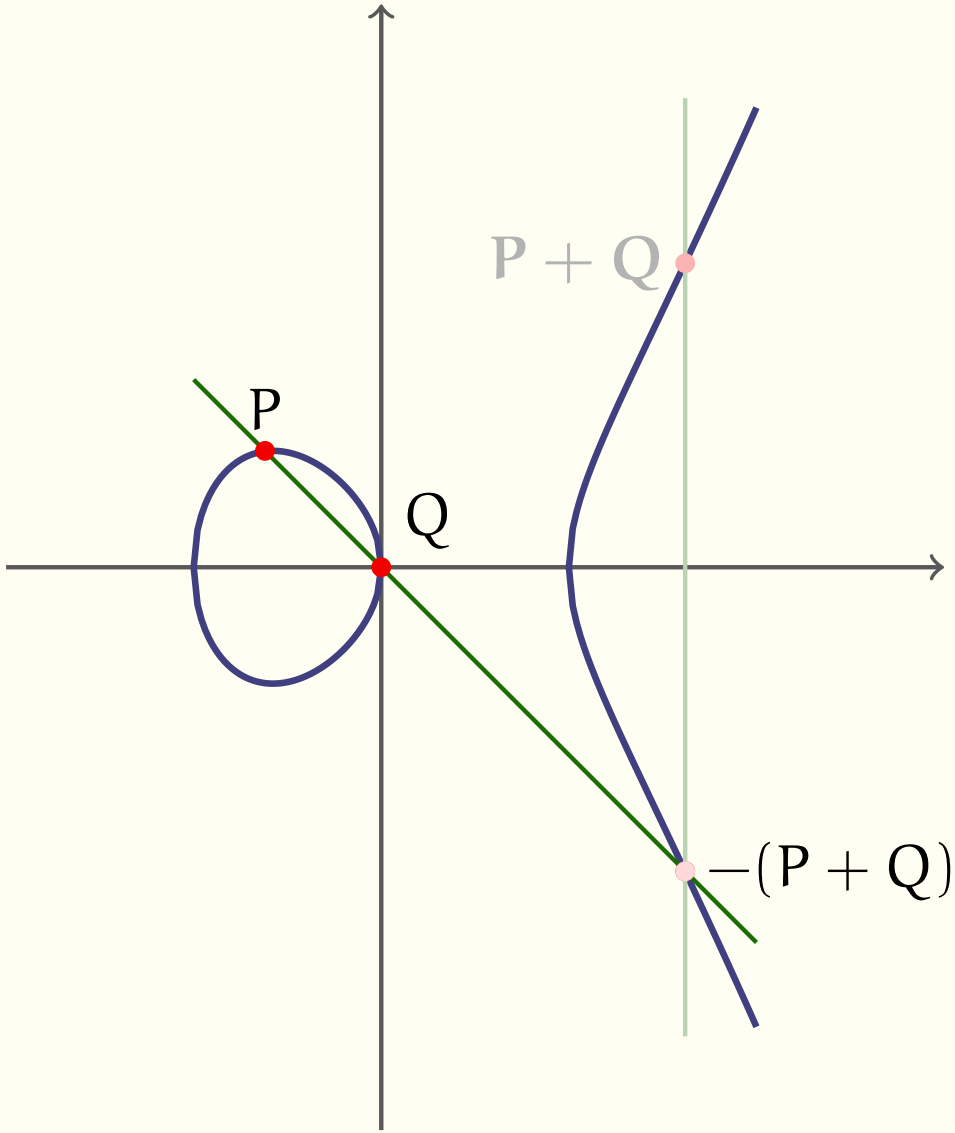
# Outline

# *Why?*

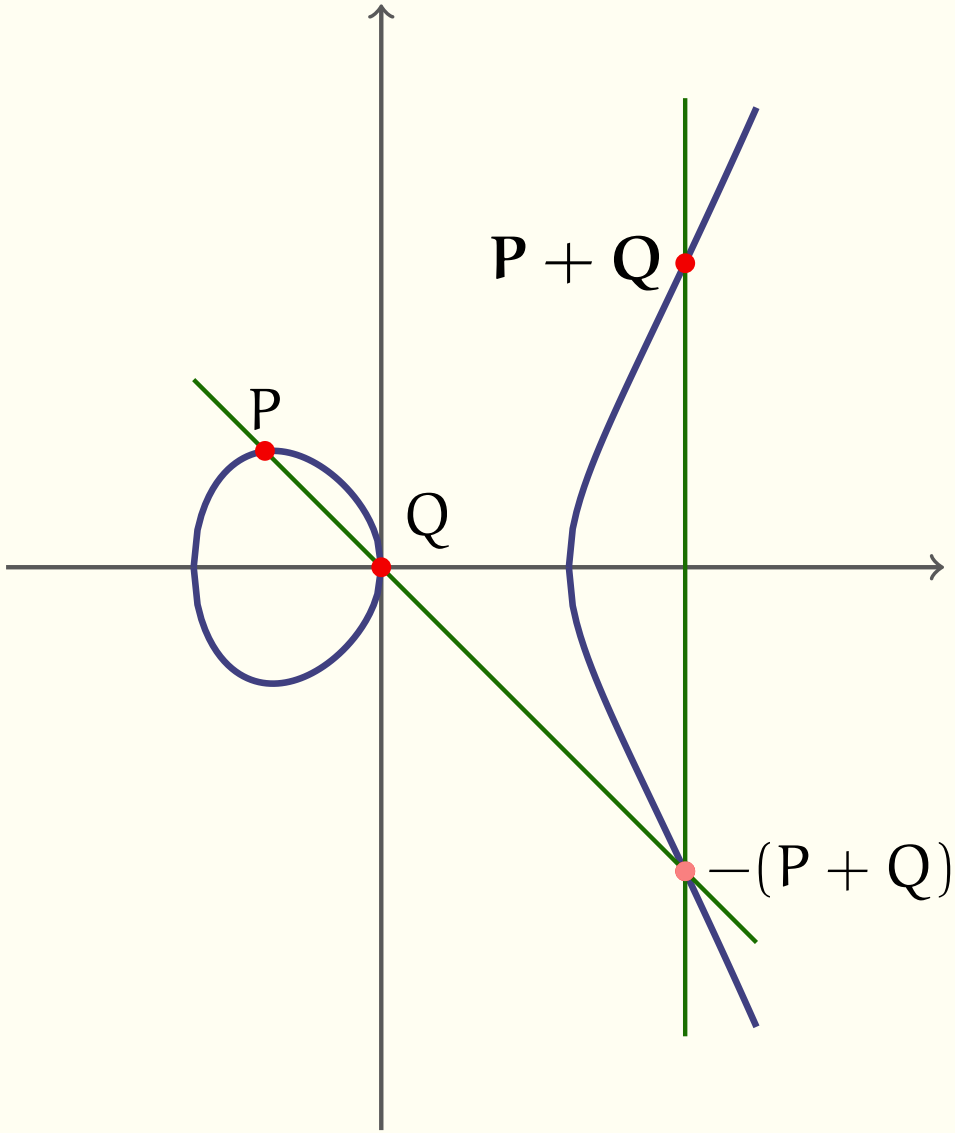
# Elliptic curve group law



# Elliptic curve group law

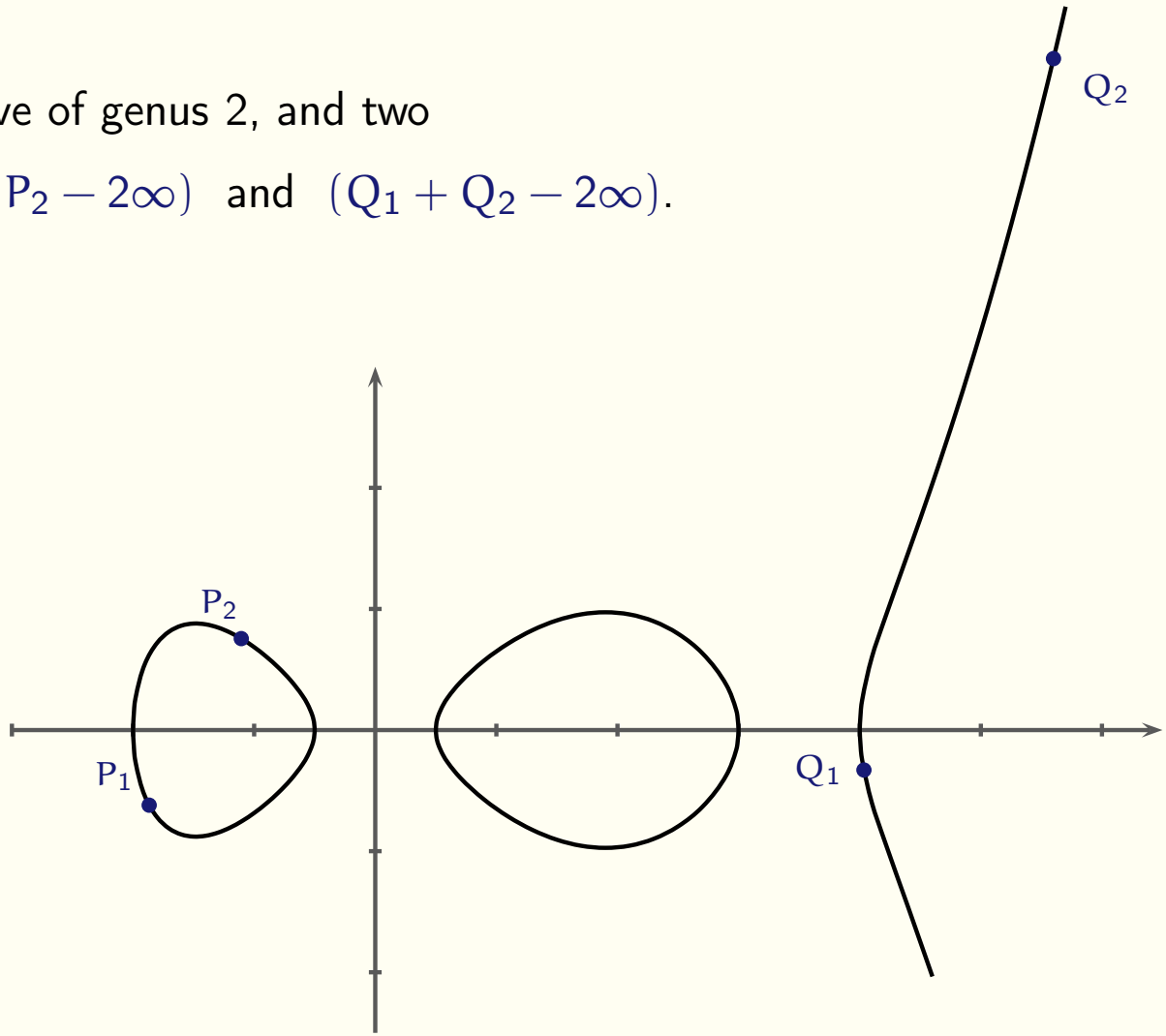


# Elliptic curve group law



# A genus two example

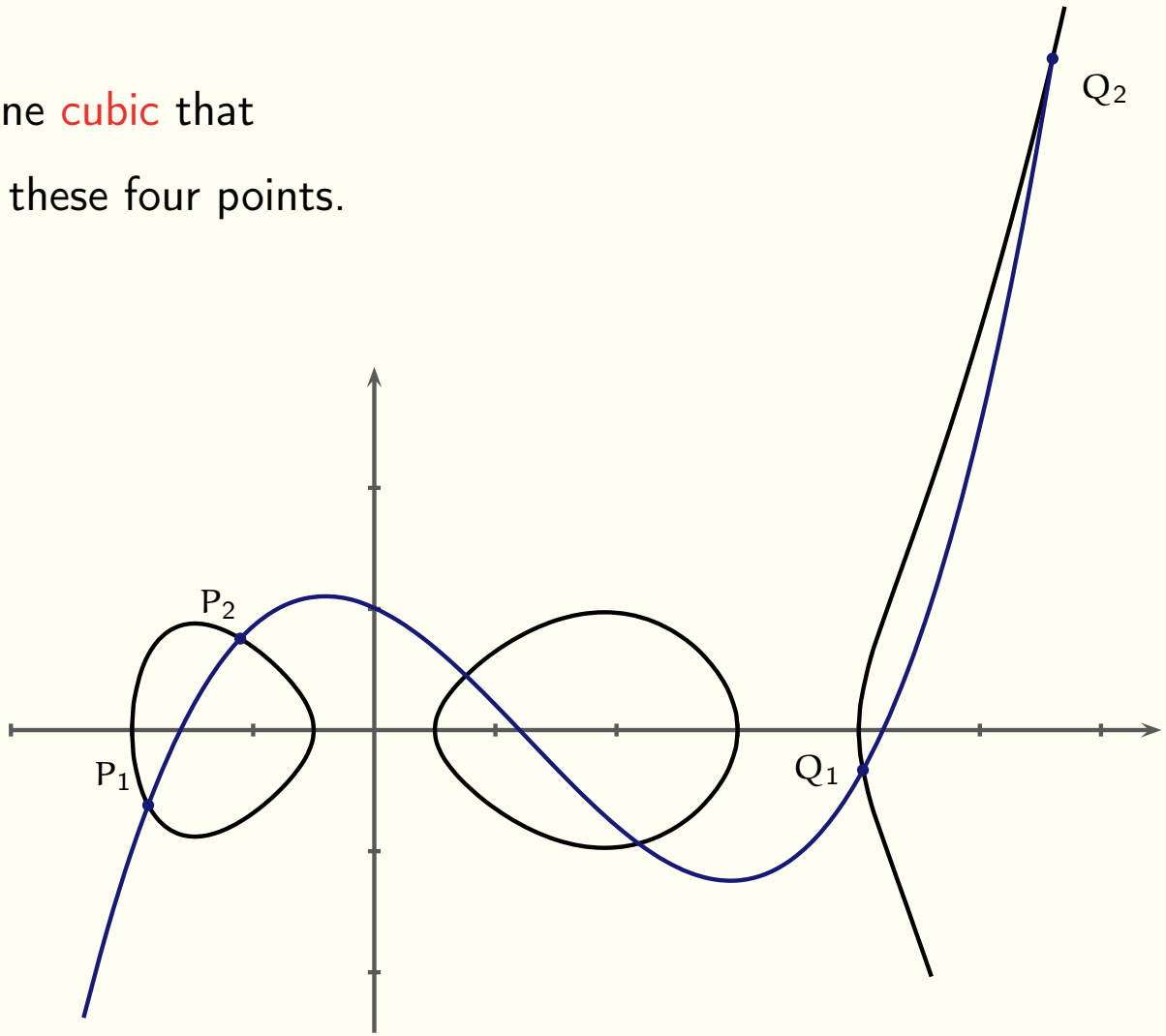
Consider a curve of genus 2, and two divisors $(P_1 + P_2 - 2\infty)$ and $(Q_1 + Q_2 - 2\infty)$.



# A genus two example

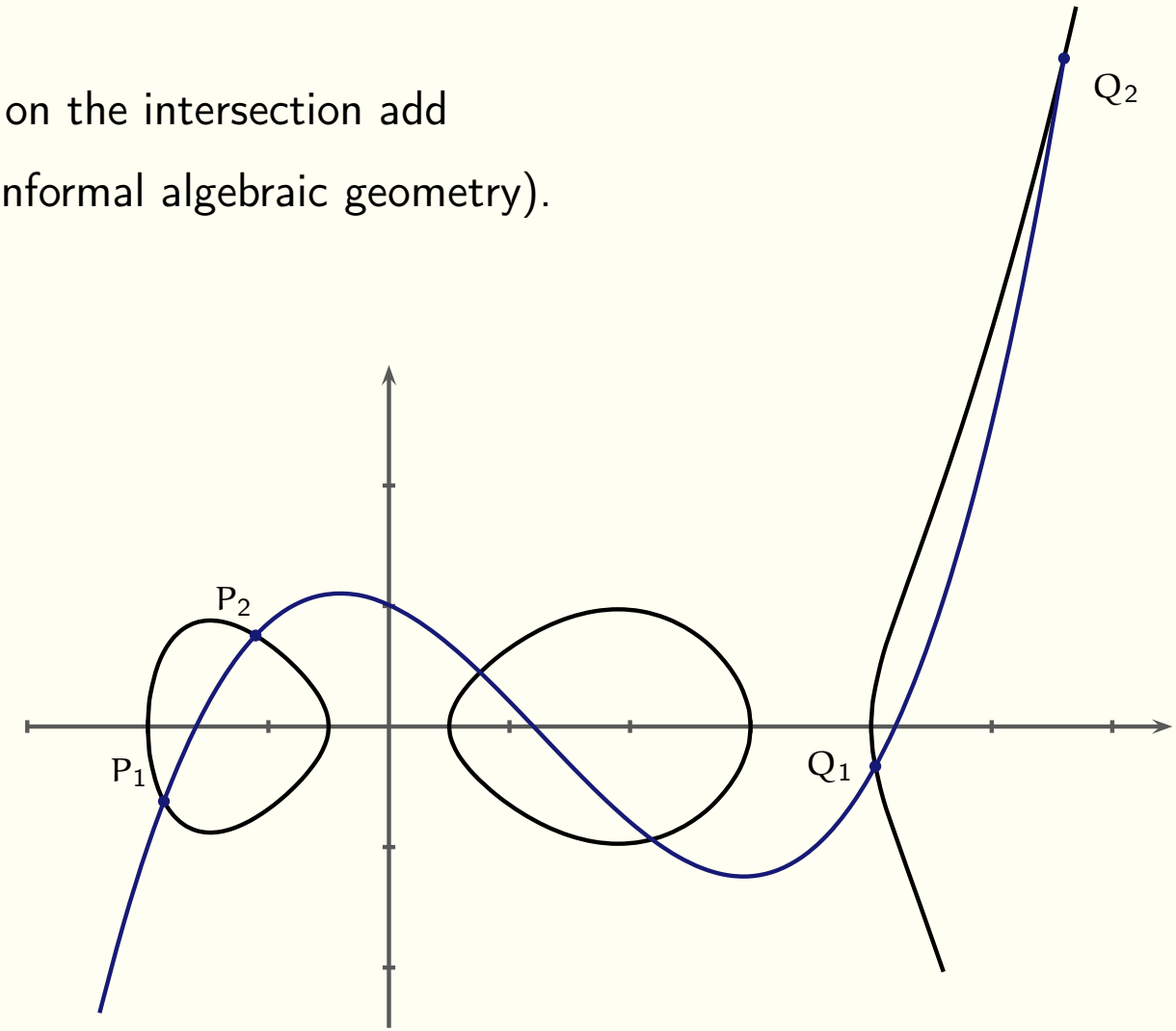
There is only one cubic that

goes through these four points.



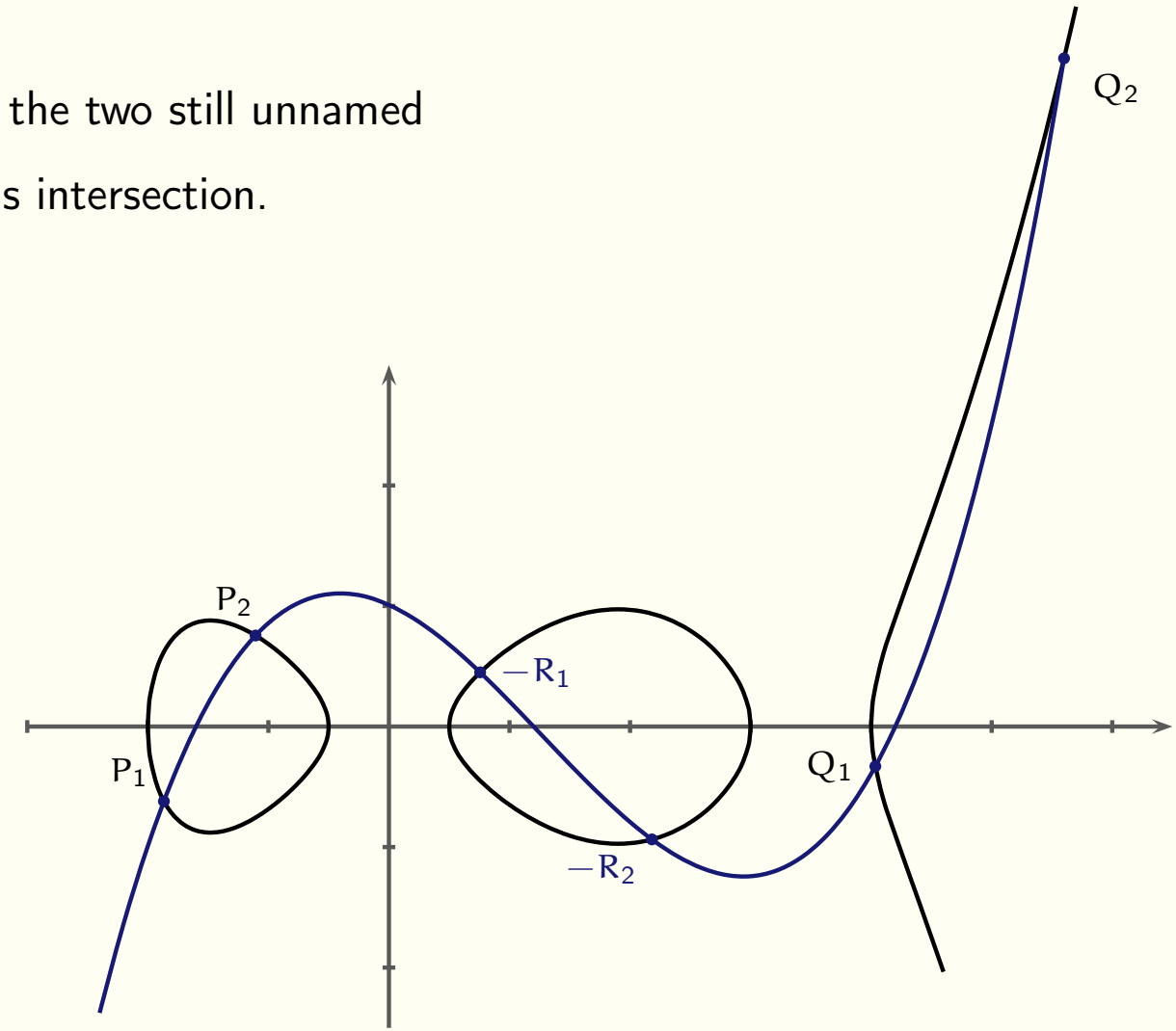
# A genus two example

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



# A genus two example

Consider then the two still unnamed point on this intersection.

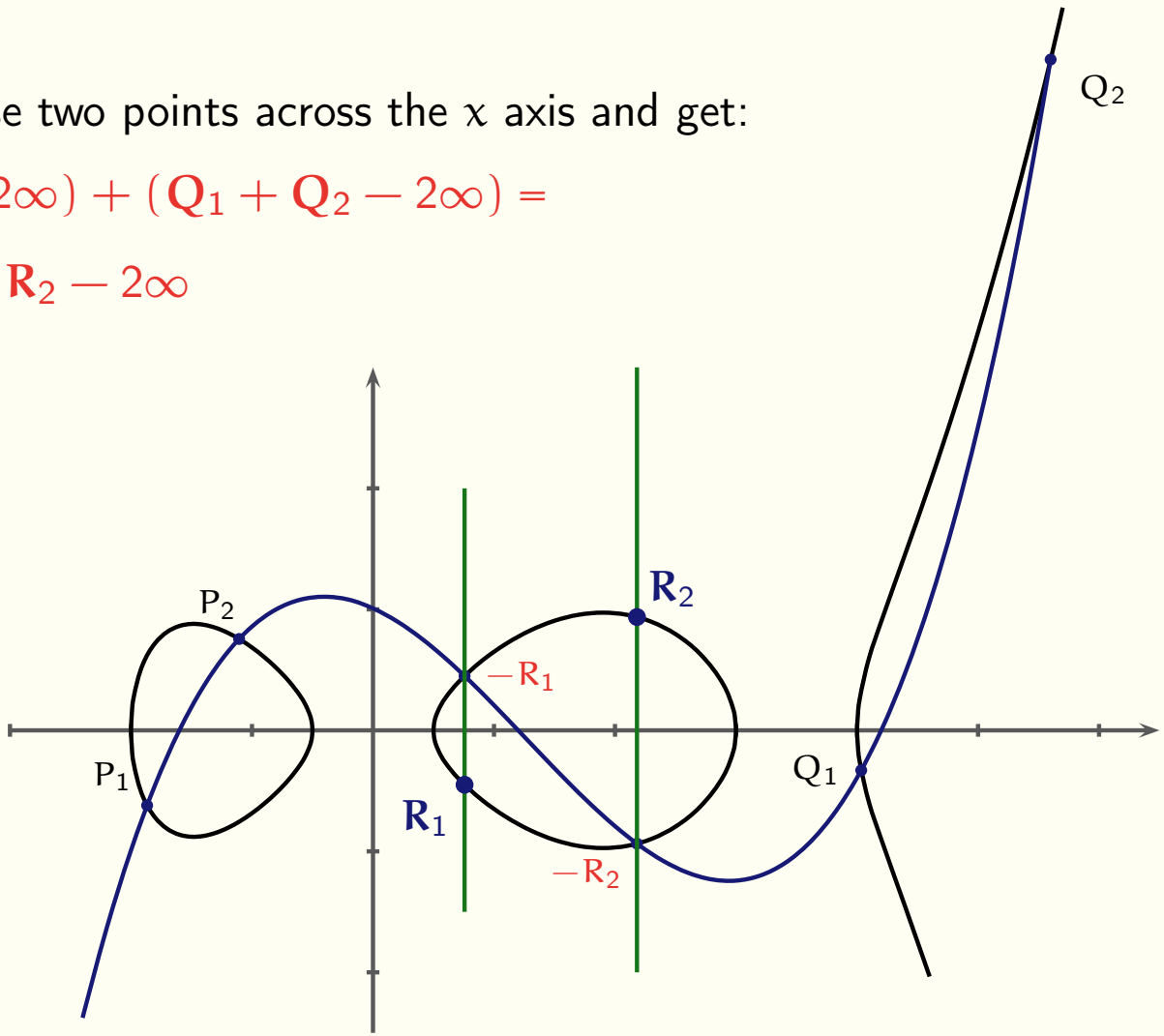


# A genus two example

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

$$(\mathbf{P}_1 + \mathbf{P}_2 - 2\infty) + (\mathbf{Q}_1 + \mathbf{Q}_2 - 2\infty) =$$

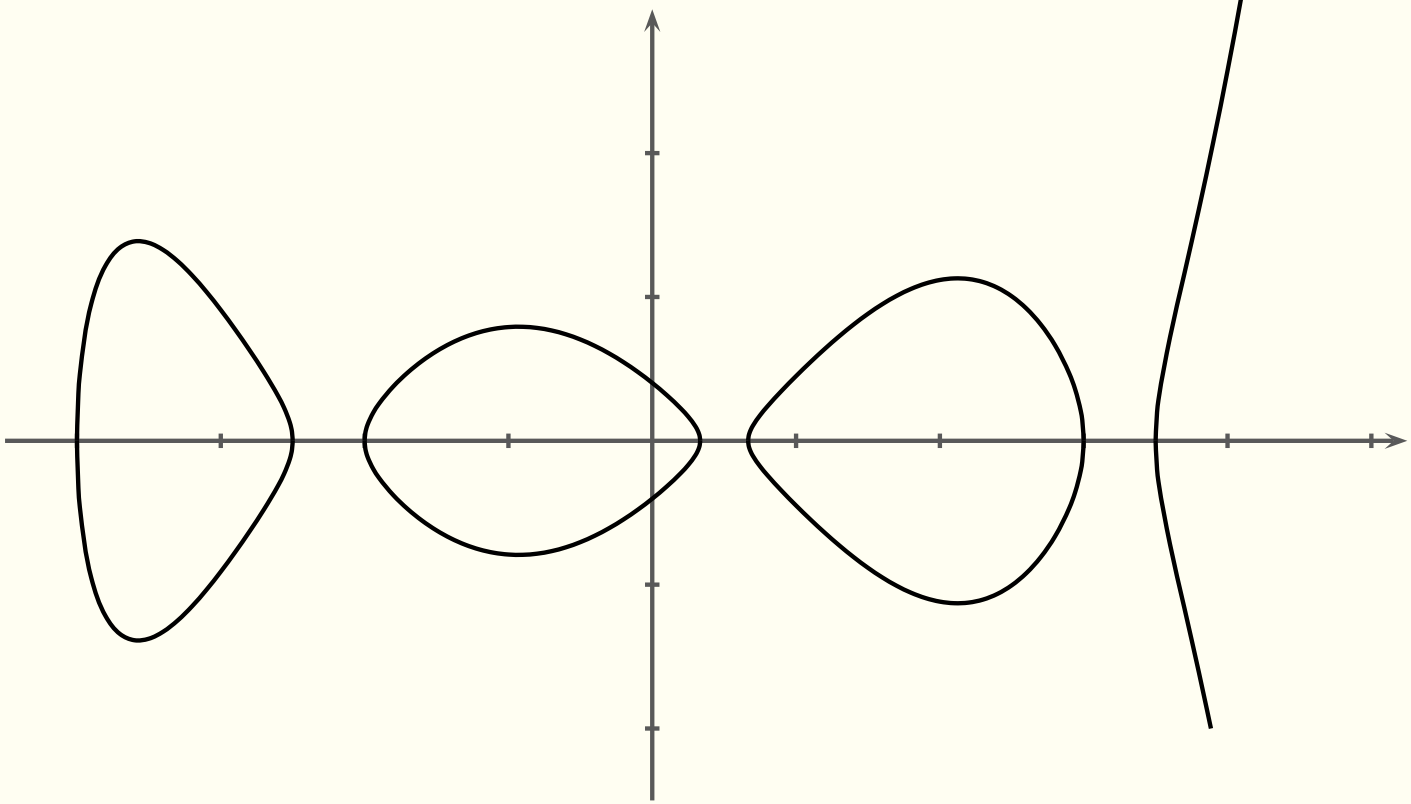
$$= \mathbf{R}_1 + \mathbf{R}_2 - 2\infty$$



# A genus three example

A genus 3 curve:

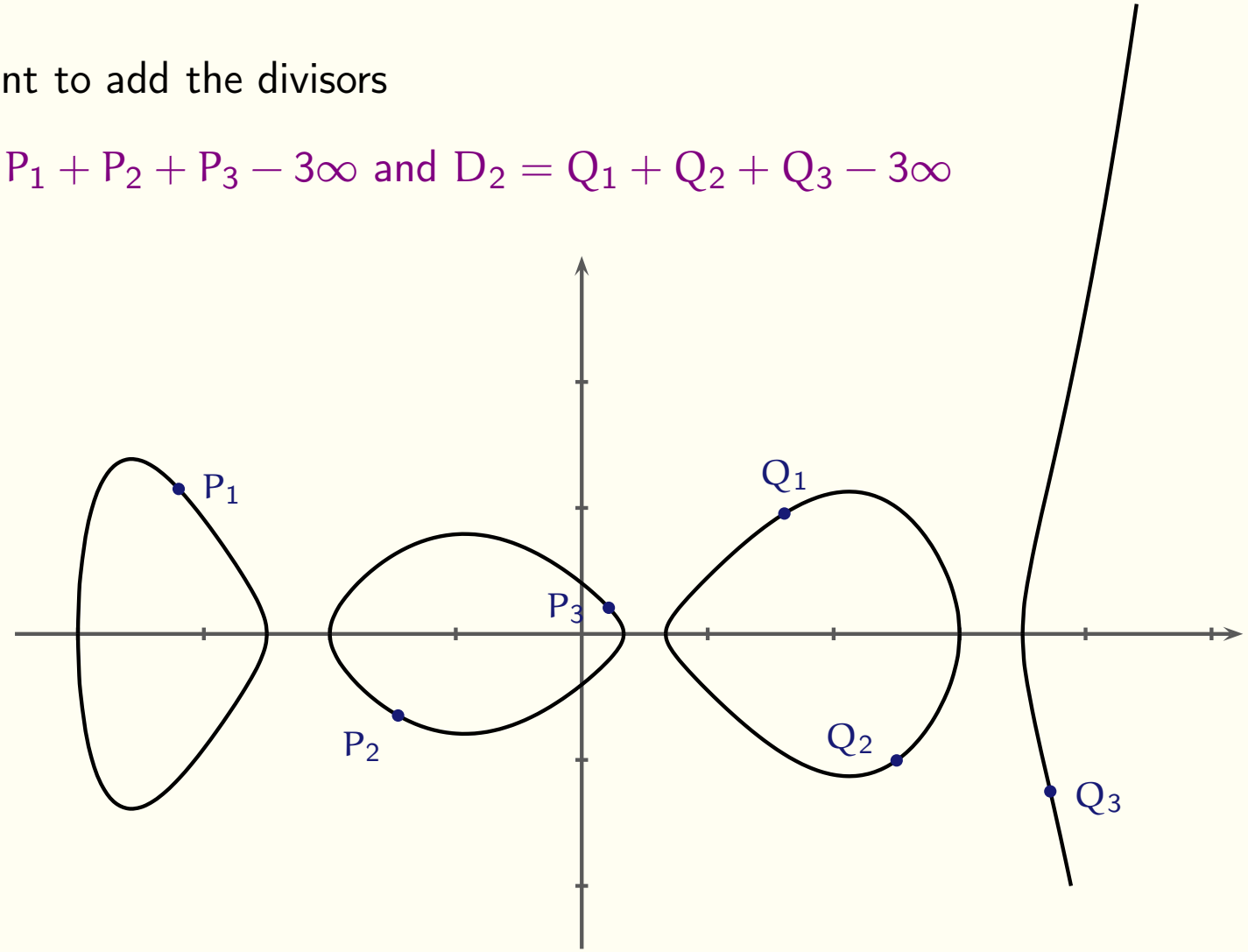
$$y^2 = x^7 + \frac{1}{2}x^6 - \frac{847}{144}x^5 - \frac{325}{144}x^4 + \frac{1763}{192}x^3 + \frac{403}{144}x^2 - \frac{1667}{576}x + \frac{35}{96}$$



# A genus three example

We want to add the divisors

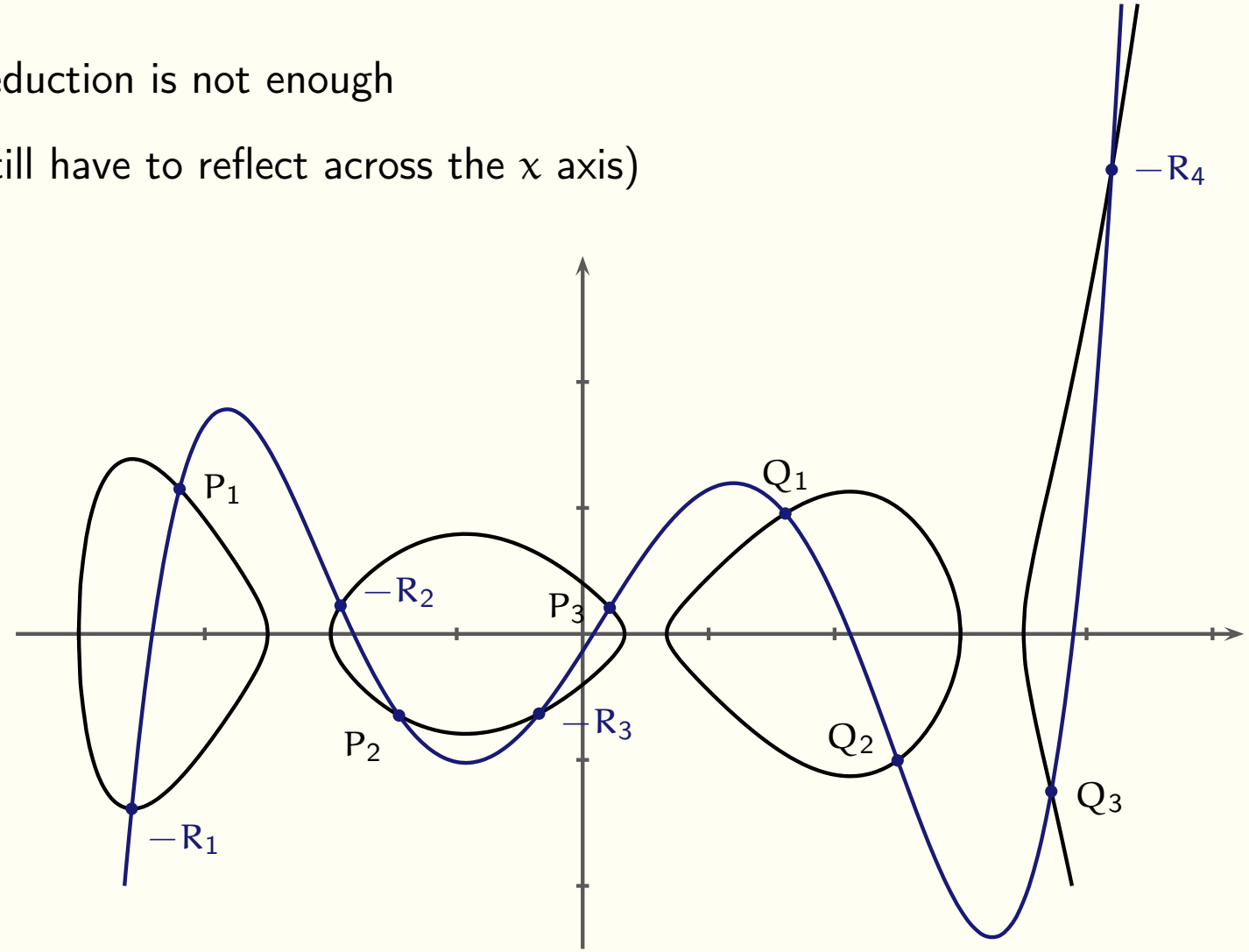
$D_1 = P_1 + P_2 + P_3 - 3\infty$ and $D_2 = Q_1 + Q_2 + Q_3 - 3\infty$



# A genus three example

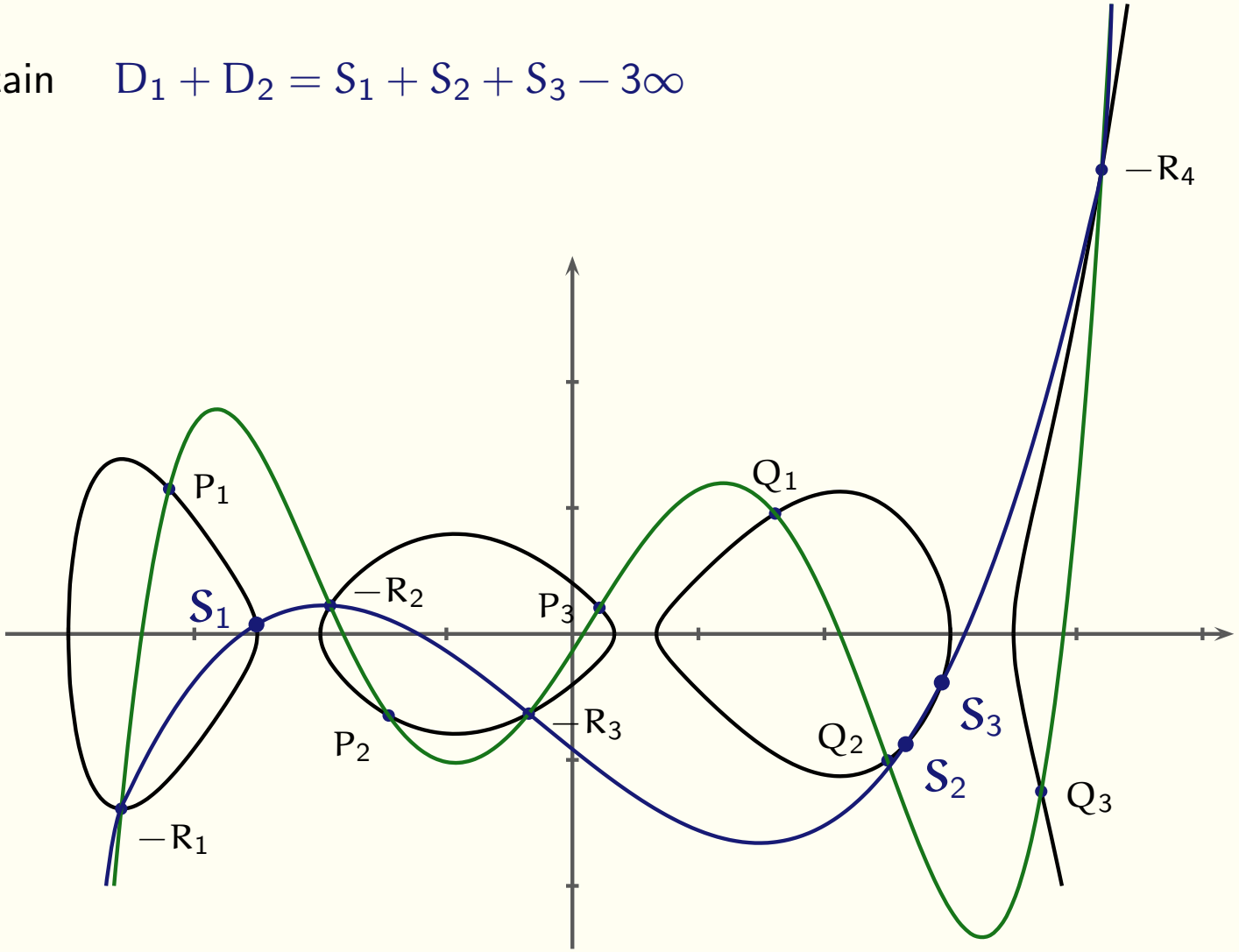
One reduction is not enough

(we still have to reflect across the x axis)



# A genus three example

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



# Complexity, complexity...

The complexity of curve arithmetic increases with genus – difficult to find efficient formulas!

Can we find simpler alternatives?

Maybe recycling genus one (and two) arithmetic for higher-dimensional varieties?

It turns out to be feasible, but *field* implementation complexity will increase.

Bonus: Some nice endomorphisms for free!
Fast endomorphisms speed up arithmetic.

Who wins?

# Complexity, complexity...

The complexity of curve arithmetic increases with genus – difficult to find efficient formulas!

Can we find simpler alternatives?

Maybe recycling genus one (and two) arithmetic for higher-dimensional varieties?

It turns out to be feasible, but *field* implementation complexity will increase.

Bonus: Some nice endomorphisms for free!
Fast endomorphisms speed up arithmetic.

Who wins?

# Complexity, complexity...

The complexity of curve arithmetic increases with genus – difficult to find efficient formulas!

Can we find simpler alternatives?

Maybe recycling genus one (and two) arithmetic for higher-dimensional varieties?

It turns out to be feasible, but *field* implementation complexity will increase.

Bonus: Some nice endomorphisms for free!
Fast endomorphisms speed up arithmetic.

Who wins?

# Complexity, complexity...

The complexity of curve arithmetic increases with genus – difficult to find efficient formulas!

Can we find simpler alternatives?

Maybe recycling genus one (and two) arithmetic for higher-dimensional varieties?

It turns out to be feasible, but *field* implementation complexity will increase.

Bonus: Some nice endomorphisms for free!
Fast endomorphisms speed up arithmetic.

Who wins?

# Complexity, complexity...

The complexity of curve arithmetic increases with genus – difficult to find efficient formulas!

Can we find simpler alternatives?

Maybe recycling genus one (and two) arithmetic for higher-dimensional varieties?

It turns out to be feasible, but *field* implementation complexity will increase.

Bonus: Some nice endomorphisms for free!
Fast endomorphisms speed up arithmetic.

Who wins?

# Complexity, complexity...

The complexity of curve arithmetic increases with genus – difficult to find efficient formulas!

Can we find simpler alternatives?

Maybe recycling genus one (and two) arithmetic for higher-dimensional varieties?

It turns out to be feasible, but *field* implementation complexity will increase.

Bonus: Some nice endomorphisms for free!
Fast endomorphisms speed up arithmetic.

Who wins?

# Outline

# $TZV$

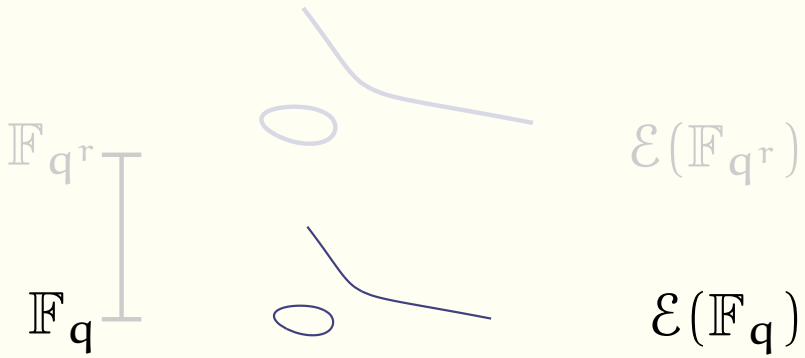
# What are Trace Zero Varieties?

Start with genus $g$ hyperelliptic curve $\mathcal{C}$ over $\mathbb{F}_q$

Trace Zero (sub)Variety of $\mathcal{C}$ over a field ext of deg $r$:

- Subgroup of divisor class group $\mathsf{Cl}(\mathcal{C}/\mathbb{F}_{q^r})$ of $\mathcal{C}$ over $\mathbb{F}_{q^r}$
- Isomorphic to quotient group $\dfrac{\mathsf{Cl}(\mathcal{C}/\mathbb{F}_{q^r})}{\mathsf{Cl}(\mathcal{C}/\mathbb{F}_q)}$
- Hence dimension over $\mathbb{F}_q$ is $g(r-1)$.
  $\approx q^{g(q-1)}$ group elements.
- We shall speak of the big and small divisor class group and say we work in the big group modulo the small one.

# Construction (From Elliptic Curves)




$\sigma \in \operatorname{End} \mathcal{E}$
$P = (x, y) \mapsto (x^q, y^q)$

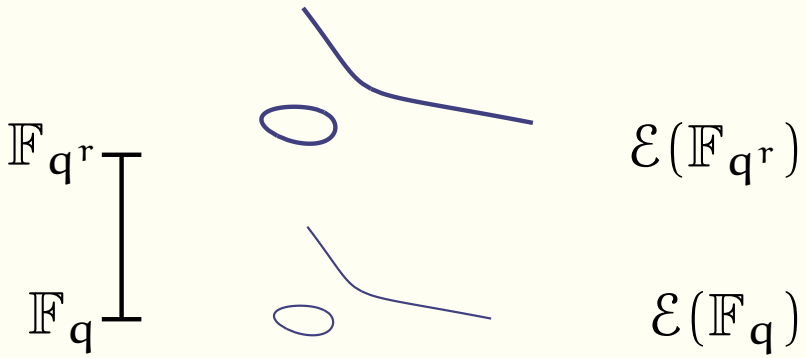
$$\operatorname{Tr} := [1] + \sigma + \cdots + \sigma^{r-1} \in \operatorname{End} \mathcal{E}(\mathbb{F}_{q^r})$$

Definition: The trace-zero subgroup of $\mathcal{E}(\mathbb{F}_{q^r})$ is:

$$G := \mathcal{E}_r(\mathbb{F}_q) = \operatorname{Ker} \operatorname{Tr} = \{P \in \mathcal{E}(\mathbb{F}_{q^r}) : \operatorname{Tr} P = \mathcal{O}\} \cong \frac{\mathcal{E}(\mathbb{F}_{q^r})}{\mathcal{E}(\mathbb{F}_q)}$$

(possibly factor out some $r$-torsion).
Now: "Recycle" group arithmetic from $\mathcal{E}$.

# Construction (From Elliptic Curves)




$\sigma \in \operatorname{End} \mathcal{E}$
$P = (x, y) \mapsto (x^q, y^q)$

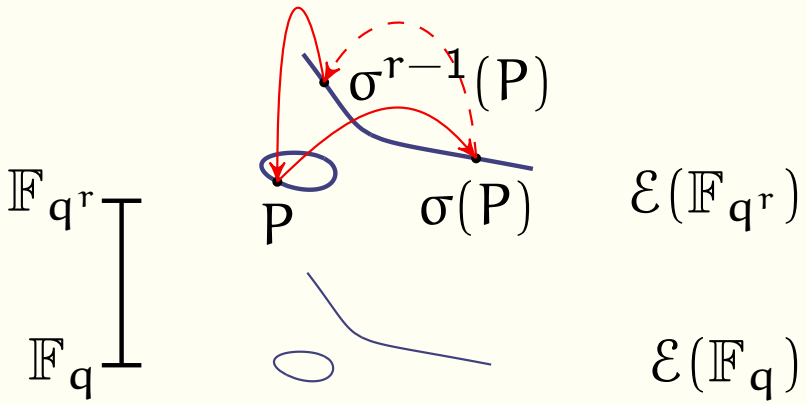
$$\operatorname{Tr} := [1] + \sigma + \cdots + \sigma^{r-1} \in \operatorname{End} \mathcal{E}(\mathbb{F}_{q^r})$$

Definition: The trace-zero subgroup of $\mathcal{E}(\mathbb{F}_{q^r})$ is:

$$G := \mathcal{E}_r(\mathbb{F}_q) = \operatorname{Ker} \operatorname{Tr} = \{P \in \mathcal{E}(\mathbb{F}_{q^r}) : \operatorname{Tr} P = \mathcal{O}\} \cong \frac{\mathcal{E}(\mathbb{F}_{q^r})}{\mathcal{E}(\mathbb{F}_q)}$$

(possibly factor out some $r$-torsion).
Now: "Recycle" group arithmetic from $\mathcal{E}$.

# Construction (From Elliptic Curves)




$\sigma \in \operatorname{End} \mathcal{E}$
$P = (x, y) \mapsto (x^q, y^q)$

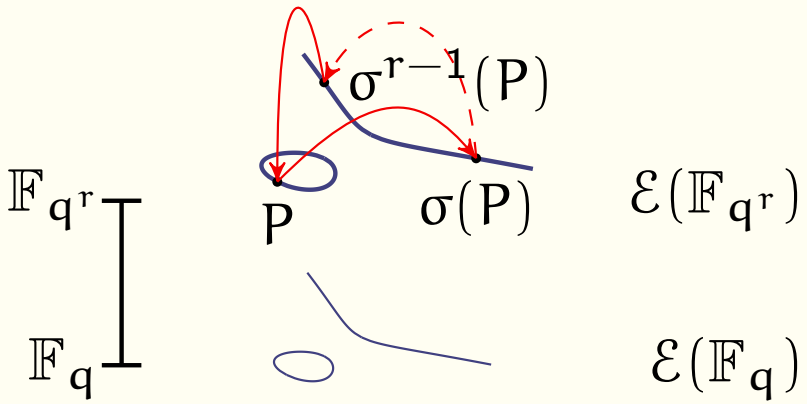
$$\operatorname{Tr} := [1] + \sigma + \cdots + \sigma^{r-1} \in \operatorname{End} \mathcal{E}(\mathbb{F}_{q^r})$$

Definition: The trace-zero subgroup of $\mathcal{E}(\mathbb{F}_{q^r})$ is:

$$G := \mathcal{E}_r(\mathbb{F}_q) = \operatorname{Ker} \operatorname{Tr} = \{P \in \mathcal{E}(\mathbb{F}_{q^r}) : \operatorname{Tr} P = \mathcal{O}\} \cong \frac{\mathcal{E}(\mathbb{F}_{q^r})}{\mathcal{E}(\mathbb{F}_q)}$$

(possibly factor out some $r$-torsion).
Now: "Recycle" group arithmetic from $\mathcal{E}$.

# Construction (From Elliptic Curves)




$\sigma \in \mathrm{End}\,\mathcal{E}$
$P = (x, y) \mapsto (x^q, y^q)$

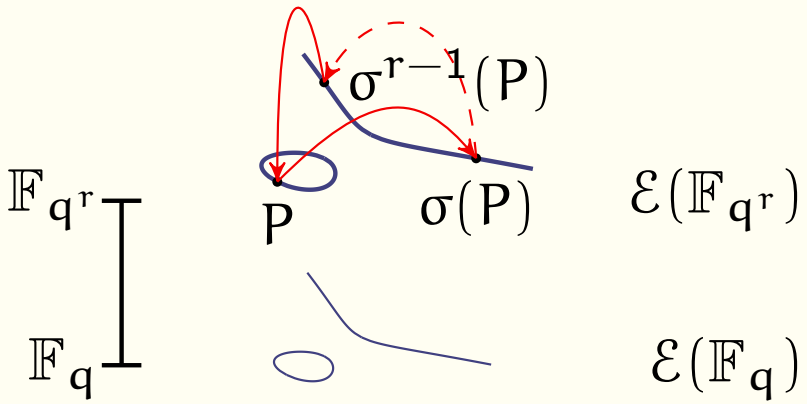
$$\mathrm{Tr} := [1] + \sigma + \cdots + \sigma^{r-1} \in \mathrm{End}\,\mathcal{E}(\mathbb{F}_{q^r})$$

Definition: The trace-zero subgroup of $\mathcal{E}(\mathbb{F}_{q^r})$ is:

$$G := \mathcal{E}_r(\mathbb{F}_q) = \mathrm{Ker}\,\mathrm{Tr} = \{P \in \mathcal{E}(\mathbb{F}_{q^r}) : \mathrm{Tr}\,P = \mathcal{O}\} \cong \frac{\mathcal{E}(\mathbb{F}_{q^r})}{\mathcal{E}(\mathbb{F}_q)}$$

(possibly factor out some $r$-torsion).
Now: "Recycle" group arithmetic from $\mathcal{E}$.

# Construction (From Elliptic Curves)




$\sigma \in \mathrm{End}\,\mathcal{E}$
$P = (x, y) \mapsto (x^q, y^q)$

$$\mathrm{Tr} := [1] + \sigma + \cdots + \sigma^{r-1} \in \mathrm{End}\,\mathcal{E}(\mathbb{F}_{q^r})$$

Definition: The trace-zero subgroup of $\mathcal{E}(\mathbb{F}_{q^r})$ is:

$$G := \mathcal{E}_r(\mathbb{F}_q) = \mathrm{Ker}\,\mathrm{Tr} = \{P \in \mathcal{E}(\mathbb{F}_{q^r}) : \mathrm{Tr}\,P = \mathcal{O}\} \cong \frac{\mathcal{E}(\mathbb{F}_{q^r})}{\mathcal{E}(\mathbb{F}_q)}$$

(possibly factor out some $r$-torsion).
Now: "Recycle" group arithmetic from $\mathcal{E}$.

# The Frobenius

Let $\mathbb{F}_{q^r}$, $q$ = prime, $2^d$ (or $3^d$), $q \geqslant 5$ and not a square.

$\mathcal{C} : y^2 + h(x)y = f(x), \quad f, g \in \mathbb{F}_q[x], \quad \deg f = 2g+1, \quad \deg h \leqslant g$

equation of non-singular genus $g$ HEC.

Consider *divisor class groups* of $\mathrm{Cl}(\mathcal{C}/\mathbb{F}_q)$ and $\mathrm{Cl}(\mathcal{C}/\mathbb{F}_{q^r})$.

Frobenius automorphism of field extension, $\sigma : x \mapsto x^q$, induces endomorphism of big divisor class group, where the elements of the small divisor class group are fixed.

On elliptic curves: $(x, y) \mapsto (\sigma x, \sigma y)$.
Mumford representation: $[U, V] \mapsto [\sigma U, \sigma V]$ in $\mathrm{Cl}(\mathcal{C}/\mathbb{F}_{q^r})$.

# The Frobenius

Let $\mathbb{F}_{q^r}$, $q$ = prime, $2^d$ (or $3^d$), $q \geqslant 5$ and not a square.

$\mathcal{C} : y^2 + h(x)y = f(x), \quad f, g \in \mathbb{F}_q[x], \quad \deg f = 2g+1, \quad \deg h \leqslant g$

equation of non-singular genus $g$ HEC.

Consider *divisor class groups* of $\mathrm{Cl}(\mathcal{C}/\mathbb{F}_q)$ and $\mathrm{Cl}(\mathcal{C}/\mathbb{F}_{q^r})$.

Frobenius automorphism of field extension, $\sigma : x \mapsto x^q$, induces endomorphism of big divisor class group, where the elements of the small divisor class group are fixed.

On elliptic curves: $(x, y) \mapsto (\sigma x, \sigma y)$.
Mumford representation: $[U, V] \mapsto [\sigma U, \sigma V]$ in $\mathrm{Cl}(\mathcal{C}/\mathbb{F}_{q^r})$.

# The Frobenius

Let $\mathbb{F}_{q^r}$, $q$ = prime, $2^d$ (or $3^d$), $q \geqslant 5$ and not a square.

$\mathcal{C} : y^2 + h(x)y = f(x), \quad f, g \in \mathbb{F}_q[x], \quad \deg f = 2g+1, \quad \deg h \leqslant g$

equation of non-singular genus $g$ HEC.

Consider *divisor class groups* of $\mathrm{Cl}(\mathcal{C}/\mathbb{F}_q)$ and $\mathrm{Cl}(\mathcal{C}/\mathbb{F}_{q^r})$.

Frobenius automorphism of field extension, $\sigma : x \mapsto x^q$, induces endomorphism of big divisor class group, where the elements of the small divisor class group are fixed.

On elliptic curves: $(x, y) \mapsto (\sigma x, \sigma y)$.
Mumford representation: $[U, V] \mapsto [\sigma U, \sigma V]$ in $\mathrm{Cl}(\mathcal{C}/\mathbb{F}_{q^r})$.

# Frobenius and Group Order

## Characteristic polynomial

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

Let $\tau_1, ..., \tau_{2g}$ be the roots. Group order over any extension field

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

If $r = 1$, this is $P(1)$. For $r > 1$, easy to make explicit.

Point counting easier (esp. for $q$ prime), since we have to compute characteristic polynomial for smaller $q$ than with elliptic curves.

# Frobenius and Group Order

Characteristic polynomial

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

Let $\tau_1, ..., \tau_{2g}$ be the roots. Group order over any extension field

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

If $r = 1$, this is $P(1)$. For $r > 1$, easy to make explicit.

Point counting easier (esp. for $q$ prime), since we have to compute characteristic polynomial for smaller $q$ than with elliptic curves.

# Frobenius and Group Order

Characteristic polynomial

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

Let $\tau_1, ..., \tau_{2g}$ be the roots. Group order over any extension field

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

If $r = 1$, this is $P(1)$. For $r > 1$, easy to make explicit.

Point counting easier (esp. for $q$ prime), since we have to compute characteristic polynomial for smaller $q$ than with elliptic curves.

# Which cases

For security and practical reasons we limit ourselves to:

- $g = 1$ **with** $r = 3$ , dimension 2 over $\mathbb{F}_q$ ,
- $g = 1$ **with** $r = 5$ , dimension 4 over $\mathbb{F}_q$ ,
- $g = 2$ **with** $r = 3$ , dimension 4 over $\mathbb{F}_q$ .

We shall return later to security considerations.

# Trouble Ahead

Let $\mathcal{C}$ be a HEC of genus $g$ over $\mathbb{F}_q$ and $\mathcal{H}$ be a HEC of genus $g$ over $\mathbb{F}_{q'}$, with $q' \approx q^{r-1}$.

Then $|\mathrm{Cl}(\mathcal{H}/\mathbb{F}_{q'})| \approx q^{(r-1)g}$, and $\left|\frac{\mathrm{Cl}(\mathcal{C}/\mathbb{F}_{q^r})}{\mathrm{Cl}(\mathcal{C}/\mathbb{F}_q)}\right| \approx q^{(r-1)g}$.

Hence $\mathrm{Cl}(\mathcal{H}/\mathbb{F}_{q'})$ has $\approx$ as many points as $\frac{\mathrm{Cl}(\mathcal{C}/\mathbb{F}_{q^r})}{\mathrm{Cl}(\mathcal{C}/\mathbb{F}_q)}$.

Suppose both orders are cryptographically good (almost primes).

We can compute on both using the same group laws but the field in the second case will be larger.

**Lower performance? How can we save the day?**

# Trouble Ahead

Let $\mathcal{C}$ be a HEC of genus $g$ over $\mathbb{F}_q$ and $\mathcal{H}$ be a HEC of genus $g$ over $\mathbb{F}_{q'}$, with $q' \approx q^{r-1}$.

Then $|\mathrm{Cl}(\mathcal{H}/\mathbb{F}_{q'})| \approx q^{(r-1)g}$, and $\left|\frac{\mathrm{Cl}(\mathcal{C}/\mathbb{F}_{q^r})}{\mathrm{Cl}(\mathcal{C}/\mathbb{F}_q)}\right| \approx q^{(r-1)g}$.

Hence $\mathrm{Cl}(\mathcal{H}/\mathbb{F}_{q'})$ has $\approx$ as many points as $\frac{\mathrm{Cl}(\mathcal{C}/\mathbb{F}_{q^r})}{\mathrm{Cl}(\mathcal{C}/\mathbb{F}_q)}$.

Suppose both orders are cryptographically good (almost primes).

We can compute on both using the same group laws but the field in the second case will be larger.

**Lower performance? How can we save the day?**

# Trouble Ahead

Let $\mathcal{C}$ be a HEC of genus $g$ over $\mathbb{F}_q$ and $\mathcal{H}$ be a HEC of genus $g$ over $\mathbb{F}_{q'}$, with $q' \approx q^{r-1}$.

Then $|\mathrm{Cl}(\mathcal{H}/\mathbb{F}_{q'})| \approx q^{(r-1)g}$, and $\left|\frac{\mathrm{Cl}(\mathcal{C}/\mathbb{F}_{q^r})}{\mathrm{Cl}(\mathcal{C}/\mathbb{F}_q)}\right| \approx q^{(r-1)g}$.

Hence $\mathrm{Cl}(\mathcal{H}/\mathbb{F}_{q'})$ has $\approx$ as many points as $\frac{\mathrm{Cl}(\mathcal{C}/\mathbb{F}_{q^r})}{\mathrm{Cl}(\mathcal{C}/\mathbb{F}_q)}$.

Suppose both orders are cryptographically good (almost primes).

We can compute on both using the same group laws but the field in the second case will be larger.

**Lower performance? How can we save the day?**

# Trouble Ahead

Let $\mathcal{C}$ be a HEC of genus $g$ over $\mathbb{F}_q$ and $\mathcal{H}$ be a HEC of genus $g$ over $\mathbb{F}_{q'}$, with $q' \approx q^{r-1}$.

Then $|\mathrm{Cl}(\mathcal{H}/\mathbb{F}_{q'})| \approx q^{(r-1)g}$, and $\left|\frac{\mathrm{Cl}(\mathcal{C}/\mathbb{F}_{q^r})}{\mathrm{Cl}(\mathcal{C}/\mathbb{F}_q)}\right| \approx q^{(r-1)g}$.

Hence $\mathrm{Cl}(\mathcal{H}/\mathbb{F}_{q'})$ has $\approx$ as many points as $\frac{\mathrm{Cl}(\mathcal{C}/\mathbb{F}_{q^r})}{\mathrm{Cl}(\mathcal{C}/\mathbb{F}_q)}$.

Suppose both orders are cryptographically good (almost primes).

We can compute on both using the same group laws but the field in the second case will be larger.

**Lower performance? How can we save the day?**

# Trouble Ahead

Let $\mathcal{C}$ be a HEC of genus $g$ over $\mathbb{F}_q$ and $\mathcal{H}$ be a HEC of genus $g$ over $\mathbb{F}_{q'}$, with $q' \approx q^{r-1}$.

Then $|\mathrm{Cl}(\mathcal{H}/\mathbb{F}_{q'})| \approx q^{(r-1)g}$, and $\left|\frac{\mathrm{Cl}(\mathcal{C}/\mathbb{F}_{q^r})}{\mathrm{Cl}(\mathcal{C}/\mathbb{F}_q)}\right| \approx q^{(r-1)g}$.

Hence $\mathrm{Cl}(\mathcal{H}/\mathbb{F}_{q'})$ has $\approx$ as many points as $\frac{\mathrm{Cl}(\mathcal{C}/\mathbb{F}_{q^r})}{\mathrm{Cl}(\mathcal{C}/\mathbb{F}_q)}$.

Suppose both orders are cryptographically good (almost primes).

We can compute on both using the same group laws but the field in the second case will be larger.

Lower performance? How can we save the day?

# Trouble Ahead

Let $\mathcal{C}$ be a HEC of genus $g$ over $\mathbb{F}_q$ and $\mathcal{H}$ be a HEC of genus $g$ over $\mathbb{F}_{q'}$, with $q' \approx q^{r-1}$.

Then $|\mathrm{Cl}(\mathcal{H}/\mathbb{F}_{q'})| \approx q^{(r-1)g}$, and $\left|\frac{\mathrm{Cl}(\mathcal{C}/\mathbb{F}_{q^r})}{\mathrm{Cl}(\mathcal{C}/\mathbb{F}_q)}\right| \approx q^{(r-1)g}$.

Hence $\mathrm{Cl}(\mathcal{H}/\mathbb{F}_{q'})$ has $\approx$ as many points as $\frac{\mathrm{Cl}(\mathcal{C}/\mathbb{F}_{q^r})}{\mathrm{Cl}(\mathcal{C}/\mathbb{F}_q)}$.

Suppose both orders are cryptographically good (almost primes).

We can compute on both using the same group laws but the field in the second case will be larger.

**Lower performance? How can we save the day?**

# Frobenius and Scalar Multiplication

Simple scalar multiplication

- Compute $[m]P$ — $m$ integer, $P$ point
- Double-and-add algorithms — $\log m$ doublings
- Windowing, multiple bases, give incremental improvements.

Use Frobenius endomorphism $\sigma$ to speed up scalar multiplication

- $\sigma(P) = [s]P$ — $s$ depends on the curve
- For $r = 3$ write $[m]P = [m_0 + m_1 s]P$ — $m_0, m_1 \approx \sqrt{m}$
- Compute $[m_0]P + [m_1]\sigma(P)$ concurrently — almost half dbls
- For $r = 5$ split scalar in four parts — $\approx \sqrt[4]{m}$
- Can this be done? Yes – more details follow.

# Frobenius and Scalar Multiplication

Simple scalar multiplication

- Compute $[m]P$ — $m$ integer, $P$ point
- Double-and-add algorithms — $\log m$ doublings
- Windowing, multiple bases, give incremental improvements.

Use Frobenius endomorphism $\sigma$ to speed up scalar multiplication

- $\sigma(P) = [s]P$ — $s$ depends on the curve
- For $r = 3$ write $[m]P = [m_0 + m_1 s]P$ — $m_0, m_1 \approx \sqrt{m}$
- Compute $[m_0]P + [m_1]\sigma(P)$ concurrently — almost half dbls
- For $r = 5$ split scalar in four parts — $\approx \sqrt[4]{m}$
- Can this be done? Yes – more details follow.

# Outline

# Scalar Multiplication

# Expand to base of "σ"?

Use subgroup $G_0$ of "large" prime order (small index) $\ell$ of $G$.
$\sigma G_0 = G_0 \Rightarrow$ exists integer $s$ s.t. $sg = \sigma g$ for all $g \in G_0$.

Why don't we write $m = \sum m_i s^i$ and compute $mg$ as $\sum m_i \sigma^i(g)$?

We can give $s$ explicitly:

- for $g = 1$, $r = 3$: $s = \frac{q-1}{1-a_1} \bmod \ell$
- for $g = 1$, $r = 5$: $s = \frac{q^2 - q - a_1^2 q + a_1 q + 1}{q - 2a_1 q + a_1^3 - a_1^2 + a_1 - 1} \bmod \ell$
- for $g = 2$, $r = 3$: $s = \frac{q^2 - a_2 + a_1}{a_2 - a_1 q - 1} \bmod \ell$ .



**Problem: size of $s$ usually around $\ell$.**

# Expand to base of "σ"?

Use subgroup $G_0$ of "large" prime order (small index) $\ell$ of $G$.
$\sigma G_0 = G_0 \Rightarrow$ exists integer $s$ s.t. $sg = \sigma g$ for all $g \in G_0$.

Why don't we write $m = \sum m_i s^i$ and compute $mg$ as $\sum m_i \sigma^i(g)$?

We can give $s$ explicitly:

- for $g = 1$, $r = 3$: $s = \frac{q-1}{1-a_1} \bmod \ell$
- for $g = 1$, $r = 5$: $s = \frac{q^2 - q - a_1^2 q + a_1 q + 1}{q - 2a_1 q + a_1^3 - a_1^2 + a_1 - 1} \bmod \ell$
- for $g = 2$, $r = 3$: $s = \frac{q^2 - a_2 + a_1}{a_2 - a_1 q - 1} \bmod \ell$ .



**Problem: size of $s$ usually around $\ell$.**

# Expand to base of "σ"?

Use subgroup $G_0$ of "large" prime order (small index) $\ell$ of G.
$\sigma G_0 = G_0 \Rightarrow$ exists integer $s$ s.t. $sg = \sigma g$ for all $g \in G_0$.

Why don't we write $m = \sum m_i s^i$ and compute $mg$ as $\sum m_i \sigma^i(g)$?

We can give $s$ explicitly:

- for $g = 1$, $r = 3$: $s = \frac{q-1}{1-a_1} \bmod \ell$
- for $g = 1$, $r = 5$: $s = \frac{q^2 - q - a_1^2 q + a_1 q + 1}{q - 2a_1 q + a_1^3 - a_1^2 + a_1 - 1} \bmod \ell$
- for $g = 2$, $r = 3$: $s = \frac{q^2 - a_2 + a_1}{a_2 - a_1 q - 1} \bmod \ell$ .



**Problem: size of $s$ usually around $\ell$.**

# Expand to base of "σ"?

Use subgroup $G_0$ of "large" prime order (small index) $\ell$ of G.
$\sigma G_0 = G_0 \Rightarrow$ exists integer $s$ s.t. $sg = \sigma g$ for all $g \in G_0$.

Why don't we write $m = \sum m_i s^i$ and compute $mg$ as $\sum m_i \sigma^i(g)$?

We can give $s$ explicitly:

- for $g = 1$, $r = 3$: $s = \frac{q-1}{1-a_1} \bmod \ell$
- for $g = 1$, $r = 5$: $s = \frac{q^2 - q - a_1^2 q + a_1 q + 1}{q - 2a_1 q + a_1^3 - a_1^2 + a_1 - 1} \bmod \ell$
- for $g = 2$, $r = 3$: $s = \frac{q^2 - a_2 + a_1}{a_2 - a_1 q - 1} \bmod \ell$ .

**Problem: size of $s$ usually around $\ell$.**

# Scalar multiplication: Scalar Splitting

**Theorem.** For the three cases which we consider, there exists an efficient technique for expressing a scalar $m$ in the form $m \equiv \sum_{i=0}^{r-2} m_i s^i \bmod \ell$ where $m_i = O(q^g)$. We have:

- If $g = 1, r = 3$, then $|m_i| < 4q$ for $q \geqslant 79$.
- *Similar bounds for other two cases.*

Then compute $mg$ as $\sum_{i=0}^{r-2} m_i \sigma^i(g)$ ( which is $= \sum_{i=0}^{r-2} m_i s^i g$ )

*Use multi-exponentiation techniques, windowing, interleaving. As many additions as in double-and-add for original $m$, but only about $\frac{1}{r-1}$ as many doublings.*

# Sketch of Proof for $g = 1$ and $r = 3$

Using $P(s) \equiv 0 \bmod \ell$ we obtain

$$m \equiv n_0 + \frac{m - n_0}{q}(-a_1 s - s^2) \bmod \ell \quad \text{with} \quad |n_0| < \frac{q}{2} \ .$$

Expand again: i.e. write $n_1 \equiv -a_1 \frac{m-n_0}{q} \bmod \ell$ with $|n_1| < q/2$, then

$$\begin{aligned} m &\equiv n_0 + \tfrac{m-n_0}{q}(-a_1 s - s^2) \\ &\equiv n_0 + n_1 s + \frac{-a_1 \frac{m-n_0}{q} - n_1}{q}(-a_1 s^2 - s^3) - \tfrac{m-n_0}{q} s^2 \bmod \ell \ . \end{aligned}$$

Use $s^2 + s + 1 \equiv 0 \bmod \ell$ to replace $s^2$ and $s^3$. We obtain $m_0, m_1$ s.t.

$$m \equiv m_0 + m_1 s \bmod \ell \ .$$

Easy to prove that $|m_0|$ and $|m_1| < 4\,q$ for all $q > 73$. □

(It does really cost just a few tens of multiplications mod $q$.)

# Scalar multiplication: Random Multi-Scalars

**Theorem.** *Let* $g$ *be a generator of* $G_0$. *Then the* $B^{r-1}$ *elements*

$$m_0 g + m_1 \sigma g + \cdots + m_{r-2} \sigma^{r-2}(g)$$

*are pairwise distinct for* $0 \leqslant m_i < B$, *where:*

1. *for* $g = 1$, $r = 3$, $B := \min\left\{ \frac{\ell}{q - a_1}, \frac{q-1}{\gcd(q-1, a_1 - 1)} \right\} = O(q)$ ;
2. *for* $g = 1$, $r = 5$, $B :=$ a more complicated expression $= O(q)$ ;
3. *for* $g = 2$, $r = 3$, $B :=$ something even worse $= O(q^2)$ .

Then pick the $m_i$ at random, use multi-exponentiation.

Problem: $B$ can be *much* smaller than $q$.
Scalar splitting preferred method in even characteristic.

# Scalar multiplication: Random Multi-Scalars

**Theorem.** *Let* $g$ *be a generator of* $G_0$. *Then the* $B^{r-1}$ *elements*

$$m_0 g + m_1 \sigma g + \cdots + m_{r-2} \sigma^{r-2}(g)$$

*are pairwise distinct for* $0 \leqslant m_i < B$, *where:*

1. *for* $g = 1$, $r = 3$, $B := \min\left\{ \frac{\ell}{q - a_1}, \frac{q-1}{\gcd(q-1, a_1 - 1)} \right\} = O(q)$ ;
2. *for* $g = 1$, $r = 5$, $B :=$ a more complicated expression $= O(q)$ ;
3. *for* $g = 2$, $r = 3$, $B :=$ something even worse $= O(q^2)$ .

Then pick the $m_i$ at random, use multi-exponentiation.

Problem: $B$ can be *much* smaller than $q$.
Scalar splitting preferred method in even characteristic.

# Scalar multiplication: Random Multi-Scalars

**Theorem.** *Let* $g$ *be a generator of* $G_0$. *Then the* $B^{r-1}$ *elements*

$$m_0 g + m_1 \sigma g + \cdots + m_{r-2} \sigma^{r-2}(g)$$

*are pairwise distinct for* $0 \leqslant m_i < B$, *where:*

1. *for* $g = 1$, $r = 3$, $B := \min\left\{ \frac{\ell}{q - a_1}, \frac{q-1}{\gcd(q-1, a_1 - 1)} \right\} = O(q)$ ;
2. *for* $g = 1$, $r = 5$, $B :=$ a more complicated expression $= O(q)$ ;
3. *for* $g = 2$, $r = 3$, $B :=$ something even worse $= O(q^2)$ .

Then pick the $m_i$ at random, use multi-exponentiation.

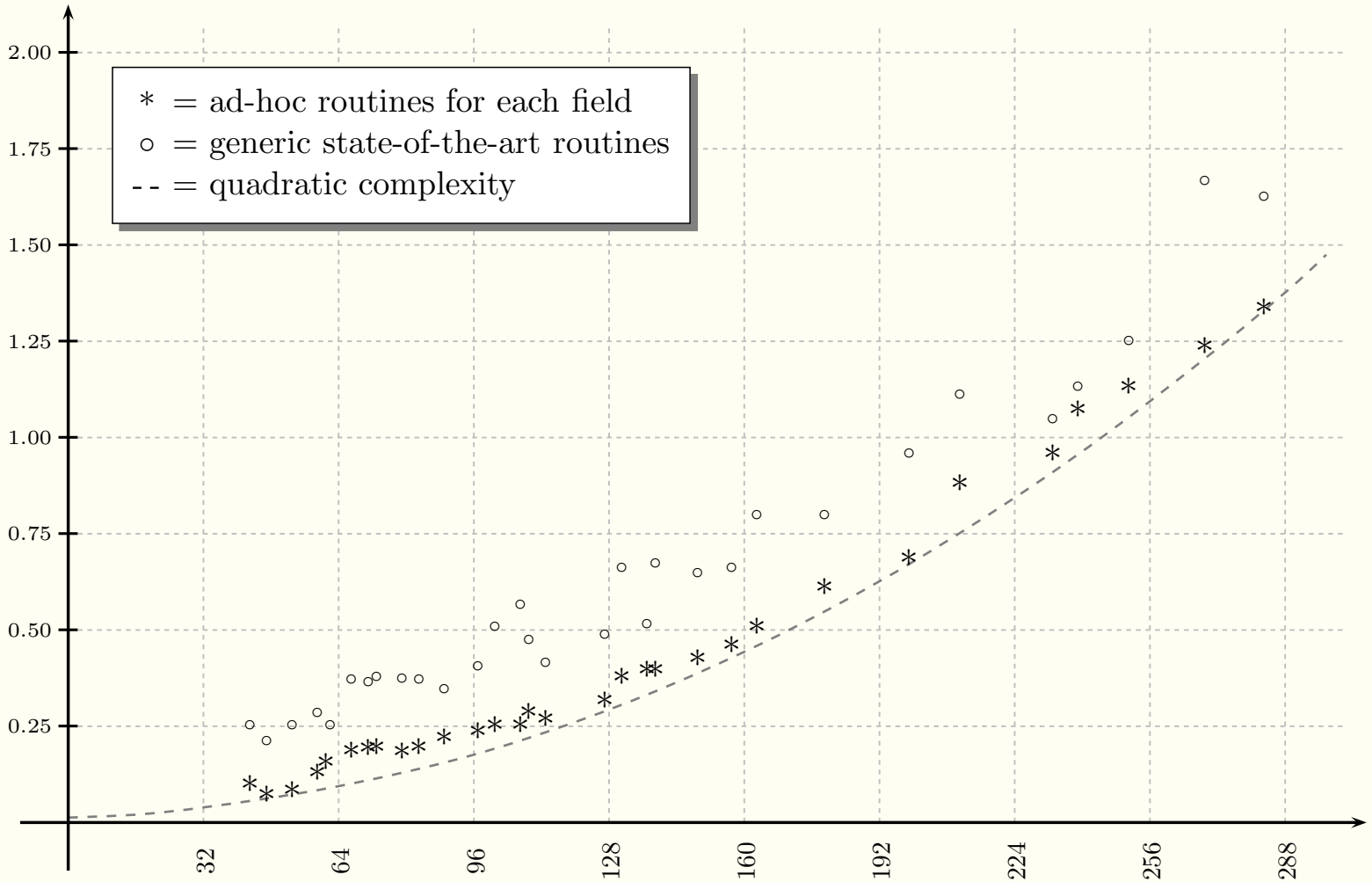
Problem: $B$ can be *much* smaller than $q$.
Scalar splitting preferred method in even characteristic.

# Implementation of Finite Fields

# Arithmetic in $\mathbb{F}_q$

- For $q = 2^d$:
  - Efficient new implementation for 32 and 64 bit machines.
  - 32 bit version (A. & Thériault) presented at WAIFI 2007.
  - Includes also square roots, new algorithms for trace and half-trace computations, and new families of polynomials for very fast square root computation (A. SAC 2007).
- For $q = p$ prime:
  - New version of `MONGO` with x86-64 support.
  - 32 and 64 bit code.
  - All fundamental operations hand optimised in assembler.
- $q = 3^d$ is work in progress.

# For instance: performance of binary fields on 32-bit cpu



# Approach of $\mathbb{F}_{2^d}$ library

- Library written in C.
- For GF(2) arithmetic only logic and shift operations.
- Purpose: make fairer comparisons for more complex objects, not pure performance. Yet also performance is there.
- Portable. Hence, no vector extensions – the results would be similar, only with bigger granularity.
- Provide as many fields as possible (15 fields hand optimised).
- `wc -l *.c *.h` outputs: *47,843 lines of code.*
- But, some parts generated automatically.

# Approach of $\mathbb{F}_{2^d}$ library

- Library written in C.
- For GF(2) arithmetic only logic and shift operations.
- Purpose: make fairer comparisons for more complex objects, not pure performance. Yet also performance is there.
- Portable. Hence, no vector extensions – the results would be similar, only with bigger granularity.
- Provide as many fields as possible (15 fields hand optimised).
- `wc -l *.c *.h` outputs: *47,843 lines of code.*
- But, some parts generated automatically.

# Approach of $\mathbb{F}_{2^d}$ library

- Library written in C.
- For GF(2) arithmetic only logic and shift operations.
- Purpose: make fairer comparisons for more complex objects, not pure performance. Yet also performance is there.
- Portable. Hence, no vector extensions – the results would be similar, only with bigger granularity.
- Provide as many fields as possible (15 fields hand optimised).
- `wc -l *.c *.h` outputs: *47,843 lines of code.*
- But, some parts generated automatically.

# Approach of $\mathbb{F}_{2^d}$ library

- Library written in C.
- For GF(2) arithmetic only logic and shift operations.
- Purpose: make fairer comparisons for more complex objects, not pure performance. Yet also performance is there.
- Portable. Hence, no vector extensions – the results would be similar, only with bigger granularity.
- Provide as many fields as possible (15 fields hand optimised).
- `wc -l *.c *.h` outputs: *47,843 lines of code.*
- But, some parts generated automatically.

# Approach of $\mathbb{F}_{2^d}$ library

- Library written in C.
- For GF(2) arithmetic only logic and shift operations.
- Purpose: make fairer comparisons for more complex objects, not pure performance. Yet also performance is there.
- Portable. Hence, no vector extensions – the results would be similar, only with bigger granularity.
- Provide as many fields as possible (15 fields hand optimised).
- `wc -l *.c *.h` outputs: *47,843 lines of code.*
- But, some parts generated automatically.

# Approach of $\mathbb{F}_{2^d}$ library

- Library written in C.
- For GF(2) arithmetic only logic and shift operations.
- Purpose: make fairer comparisons for more complex objects, not pure performance. Yet also performance is there.
- Portable. Hence, no vector extensions – the results would be similar, only with bigger granularity.
- Provide as many fields as possible (15 fields hand optimised).
- `wc -l *.c *.h` outputs: *47,843 lines of code.*
- But, some parts generated automatically.

# Approach of $\mathbb{F}_{2^d}$ library

- Library written in C.
- For GF(2) arithmetic only logic and shift operations.
- Purpose: make fairer comparisons for more complex objects, not pure performance. Yet also performance is there.
- Portable. Hence, no vector extensions – the results would be similar, only with bigger granularity.
- Provide as many fields as possible (15 fields hand optimised).
- `wc -l *.c *.h` outputs: *47,843 lines of code.*
- But, some parts generated automatically.

# Multiplications, and Multiplications in $\mathbb{F}_{2^d}$

- Optimal routines for each operand size.
- Multiplication - Lopez-Dahab's comb method.
- For extensions of $\mathbb{F}_{2^d}$: As degrees increase, we find more products of many elements time a (locally) fixed element.
- Separate precomputation from shift-and-xor loop in Lopez-Dahab, reuse them for several multiplications. Call this *serial multiplications*.
  - 2 muls cost $\approx$ 1.50 – 1.90 single muls (1.75 avg.)
  - 3 muls cost $\approx$ 2.05 – 2.50 single muls (2.35 avg.)
  - 4 muls cost $\approx$ 2.55 – 3.25 single muls (3.00 avg.)
  - 5 muls cost $\approx$ 3.05 – 4.00 single muls (3.65 avg.)
- Adapt comb width to amount of mults to be performed.

# Multiplications, and Multiplications in $\mathbb{F}_{2^d}$

- Optimal routines for each operand size.
- Multiplication - Lopez-Dahab's comb method.
- For extensions of $\mathbb{F}_{2^d}$: As degrees increase, we find more products of many elements time a (locally) fixed element.
- Separate precomputation from shift-and-xor loop in Lopez-Dahab, reuse them for several multiplications. Call this *serial multiplications*.
  - 2 muls cost $\approx$ 1.50 – 1.90 single muls (1.75 avg.)
  - 3 muls cost $\approx$ 2.05 – 2.50 single muls (2.35 avg.)
  - 4 muls cost $\approx$ 2.55 – 3.25 single muls (3.00 avg.)
  - 5 muls cost $\approx$ 3.05 – 4.00 single muls (3.65 avg.)
- Adapt comb width to amount of mults to be performed.

# Multiplications, and Multiplications in $\mathbb{F}_{2^d}$

- Optimal routines for each operand size.
- Multiplication - Lopez-Dahab's comb method.
- For extensions of $\mathbb{F}_{2^d}$: As degrees increase, we find more products of many elements time a (locally) fixed element.
- Separate precomputation from shift-and-xor loop in Lopez-Dahab, reuse them for several multiplications. Call this *serial multiplications*.
  - 2 muls cost $\approx$ 1.50 – 1.90 single muls (1.75 avg.)
  - 3 muls cost $\approx$ 2.05 – 2.50 single muls (2.35 avg.)
  - 4 muls cost $\approx$ 2.55 – 3.25 single muls (3.00 avg.)
  - 5 muls cost $\approx$ 3.05 – 4.00 single muls (3.65 avg.)
- Adapt comb width to amount of mults to be performed.

# Multiplications, and Multiplications in $\mathbb{F}_{2^d}$

- Optimal routines for each operand size.
- Multiplication - Lopez-Dahab's comb method.
- For extensions of $\mathbb{F}_{2^d}$: As degrees increase, we find more products of many elements time a (locally) fixed element.
- Separate precomputation from shift-and-xor loop in Lopez-Dahab, reuse them for several multiplications. Call this *serial multiplications*.
  - 2 muls cost $\approx$ 1.50 – 1.90 single muls (1.75 avg.)
  - 3 muls cost $\approx$ 2.05 – 2.50 single muls (2.35 avg.)
  - 4 muls cost $\approx$ 2.55 – 3.25 single muls (3.00 avg.)
  - 5 muls cost $\approx$ 3.05 – 4.00 single muls (3.65 avg.)
- Adapt comb width to amount of mults to be performed.

# Multiplications, and Multiplications in $\mathbb{F}_{2^d}$

- Optimal routines for each operand size.
- Multiplication - Lopez-Dahab's comb method.
- For extensions of $\mathbb{F}_{2^d}$: As degrees increase, we find more products of many elements time a (locally) fixed element.
- Separate precomputation from shift-and-xor loop in Lopez-Dahab, reuse them for several multiplications. Call this *serial multiplications*.
  - 2 muls cost $\approx$ 1.50 – 1.90 single muls (1.75 avg.)
  - 3 muls cost $\approx$ 2.05 – 2.50 single muls (2.35 avg.)
  - 4 muls cost $\approx$ 2.55 – 3.25 single muls (3.00 avg.)
  - 5 muls cost $\approx$ 3.05 – 4.00 single muls (3.65 avg.)
- Adapt comb width to amount of mults to be performed.

# Example: $\mathbb{F}_{2^{83}}$ on 64-bit cpu

```
void bf_mul_83( limb_t * r , const limb_t * a , const limb_t * b )
{
    int     i, idx;
    limb_t  tab[16][2];
    limb_t  r2, r1, r0;

    /* Compute the entries for the table tab[16][2] */

    for (i = 0; i < 16; i++) tab[i] = i * a;        /* view i as a polynomial over GF(2) */

    /* Do the actual multiplication */

    idx = b[0] >> 60;  [r0,r1] = tab[idx][0,1]; r2 = 0; r3 = 0;

    for (i = 56; i >= 20; i -= 4) {
        [r1,r0] <<= 4;                                          /* shift left by 4 bits */
        idx = (b[0] >> i) & 0xf;   [r0,r1] ^= tab[idx][0,1];              /* accumulate */
    }

    for (      ; i >=  0; i -= 4) {
        [r2,r1,r0] <<= 4;                                       /* shift left by 4 bits */
        idx = (b[0] >> i) & 0xf;   [r0,r1] ^= tab[idx][0,1];              /* accumulate */
        idx = (b[1] >> i) & 0xf;   [r1,r2] ^= tab[idx][0,1];              /* accumulate */
    }

    bf_mod_83(r2,r1,r0);                                                    /* reduce */

    r[1] = r1; r[0] = r0;                                                /* write back */
}
```

# And MONGO ?

If you have seen the output of `mpfq` or some of the innards of Miracl, you get the idea. But code original.

For intel compatible CPUs there are hand crafted asm sections, but some generic code is also provided.

For $(n + \frac{1}{2})$-limbs moduli, some time and memory accesses are saved by taking this into account (also the `REDC` is defined in a non-standard way).

Important feature: delayed reduction. Also in Montgomery representation you can add *unreduced* products to each other.
If there is an overflow into a third register, we can still subtract the modulus and the final result will be correct (see A. CHES 2004).

# Arithmetic in Extension Fields I

Example: $r = 3$ ($r = 5$ is just a bit different), binary field.

$\mathbb{F}_{q^3} = \mathbb{F}_q[\alpha]$, where $\alpha$ is a root of $X^3 + X + 1$ irreducible, $q = 2^d$.
(For $r = 5$ use $X^5 + X^2 + 1$ or $X^5 + X^3 + 1$.)

Elements of $\mathbb{F}_{q^3}$ represented by polynomials.

Multiplication/Squaring Karatsuba-like

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

Now observe

$$\alpha^3 = \alpha + 1 \qquad \text{and} \qquad \alpha^4 = \alpha^2 + \alpha \ .$$

6 Mults in $\mathbb{F}_q$ and a few adds (5-mult Toom-Cook seemed not faster for our field sizes at first. Giving now a second try with Bodrato.)

# Arithmetic in Extension Fields II

Multiplication of elements of $\mathbb{F}_{p^r}$ split into multiplication of the corresponding polynomials in $X$ and then reduction modulo $X^r - c$ with root $\alpha$ and $c$ small integer ($2, 3$ or $5$).

Multiplication in degree $r = 3$ extensions as before

$$\begin{aligned}(a_0 + a_1\alpha + a_2\alpha^2)(b_0 + b_1\alpha + b_2\alpha^2) = \\ = a_0b_0 + \big((a_0 + a_1)(b_0 + b_1) - a_0b_0 - a_1b_1\big)\alpha \\ + \big((a_0 + a_2)(b_0 + b_2) - a_0b_0 - a_2b_2 + a_1b_1\big)\alpha^2 \\ + \big((a_1 + a_2)(b_1 + b_2) - a_1b_1 - a_2b_2\big)\underbrace{\alpha^3}_{c} + (a_2b_2)\underbrace{\alpha^4}_{c\,\alpha}\ .\end{aligned}$$

By delaying all modular reductions and using incomplete reduction (A. and Mihailescu SAC 2003) we need just 3 modular reductions.

Toom-Cook with just 5 $\mathbb{F}_p$-mults not more efficient (because of the small divisions involved and the increased number of mod reds).

# Arithmetic in Extension Fields III

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

Let $b = b_0+b_1\alpha+b_2\alpha^2$ $(b_0, b_1, b_2 \in \mathbb{F}_q) = a^{-1}$ with $a = a_0+a_1\alpha+a_2\alpha^2 \in \mathbb{F}_{q^3}$.
Now $ab = 1$, w.r.t. basis $\{1, \alpha, \alpha^2\}$ can be written as:

$$\underbrace{\begin{bmatrix} a_0 & a_2 & a_1 \\ a_1 & a_0+a_2 & a_1+a_2 \\ a_2 & a_1 & a_0+a_2 \end{bmatrix}}_{A} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}.$$

$$\begin{bmatrix} b_0 \\ b_1 \\ b_2 \end{bmatrix} = \begin{bmatrix} a_0 & a_2 & a_1 \\ a_1 & a_0+a_2 & a_1+a_2 \\ a_2 & a_1 & a_0+a_2 \end{bmatrix}^{-1} \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} = (\det A)^{-1} \begin{bmatrix} a_0^2+a_1^2+a_2^2+a_1a_2 \\ a_2^2+a_0a_1 \\ a_1^2+a_2^2+a_0a_2 \end{bmatrix}$$

where $\det A = (a_0(a_0^2+a_1^2+a_2^2+a_1a_2)+a_1^3+a_2(a_1a_2+a_2^2))$

We need only one inversion (that of $\det A$).

Inversion in $\mathbb{F}_{q^3}$ is fast. In fact, affine coordinates are the fastest over binary fields.

The case $r = 5$ uses Itoh-Tsuji inversion. Very fast.

# Choice of Coordinate Systems

# More Details

Choice of coordinate systems:

- For curves over prime fields:
  - Mixed coordinates for EC (order prime, no Edwards, but performance very close), and HEC in genus 2 (Lange's collection of formulas).
  - Affine, Projective, Jacobian (+ modified) and compressed Jacobian (+ modified) for TZV.
- For curves over binary fields:
  - In $\mathbb{F}_{2^d}$ inversion costs $\approx$ 5-7 multiplications, hence we are using affine coordinates for EC, HEC.
    For genus 2 HEC use Lange-Stevens wicked doubling.
  - In $\mathbb{F}_{2^{rd}}$ inversion costs $\approx$ 3 multiplications.
    Affine is thus a no brainer for TZV.

# Compressed Jacobian Coordinates

The equation of $\mathcal{E}(\mathbb{F}_{p^r})$ in Compressed Jacobian Coordinates is

$$Y^2 = X^3 + a_4 X z^4 + a_6 z^6 \ .$$

$[X : Y : z] \in \mathbb{F}_{p^r} \times \mathbb{F}_{p^r} \times \mathbb{F}_p$ corresponds to affine point $(X/z^2, Y/z^3)$.

Due to Hoshino, Kobayashi and Aoki (Vietcrypt 2006).
We use it for TZV over prime fields.

Many multiplications in explicit formulae for Jacobian coordinates become multiplications in $\mathbb{F}_p$ or multiplications of elements of $\mathbb{F}_{p^r}$ by elements of $\mathbb{F}_p$.

We need *pseudo-inversion*, essentially almost-inverse by another name.
We compute the inverse in $\mathbb{F}_{p^r}$ up to a constant in $\mathbb{F}_p$ – essentially saving the inversion in $\mathbb{F}_p$.

We also "compress" Cohen's modified Jacobian coordinates (keep $a_4 z^4$).

Open question: "compressed" Edwards?

# Performance Results, Part I

# Performance Results: Scalar Multiplication

Timings (μsec) of ordinary binary curves and TZV at 160 bit level, 32 bit code

| | $d(\times r) =$ | EC 163 | TZV g1 $83 \times 3$ | TZV g1 $41 \times 5$ | HEC g2 83 | TZV g2 $41 \times 3$ |
|---|---|---|---|---|---|---|
| operation | 2 | 3.757 | 3.652 | 3.483 | 1.660 | 2.329 |
| | $+$ | 3.825 | 3.652 | 3.483 | 3.513 | 6.664 |
| | $\sigma$ | – | 0.029 | 0.032 | – | 0.036 |
| full scalar | bin | 1130.5 | 961.2 | 872.5 | 640.4 | 960.2 |
| | NAF | 1029.3 | 869.1 | 784.5 | 545.3 | 779.2 |
| | $w$-NAF | 933.1 | 788.8 | 709.2 | 466.6 | 639.7 |
| split scalar | bin | – | 569.2 | 427.4 | – | 601.7 |
| | NAF | – | 517.2 | 349.0 | – | 523.3 |
| | JSF | – | 505.4 | – | – | 496.2 |
| | $w$-NAF | – | 467.6 | 278.5 | – | 445.5 |

2 Ghz Quad Core Xeon running 32-bit code

# Performance Results: Scalar Multiplication

Timings (μsec) of ordinary binary curves and TZV at 192 bit level, 32 bit code

| | $d(\times r) =$ | EC 191 | TZV g1 $97 \times 3$ | TZV g1 $47 \times 5$ | HEC g2 97 | TZV g2 $47 \times 3$ |
|---|---|---|---|---|---|---|
| operation | 2 | 4.781 | 4.401 | 3.921 | 2.010 | 2.519 |
| | $+$ | 4.817 | 4.446 | 3.913 | 4.132 | 7.188 |
| | $\sigma$ | – | 0.033 | 0.036 | – | 0.036 |
| full scalar | bin | 1694.2 | 1487.0 | 1119.2 | 968.9 | 1182.3 |
| | NAF | 1530.2 | 1354.8 | 1020.4 | 820.1 | 959.6 |
| | $w$-NAF | 1380.1 | 1209.6 | 913.1 | 689.6 | 786.1 |
| split scalar | bin | – | 880.6 | 582.7 | – | 777.0 |
| | NAF | – | 787.0 | 439.7 | – | 646.8 |
| | JSF | – | 767.7 | – | – | 609.7 |
| | $w$-NAF | – | 712.9 | 347.1 | – | 538.5 |

2 Ghz Quad Core Xeon running 32-bit code

# Performance Results: Scalar Multiplication

Timings (μsec) of ordinary binary curves and TZV at 160 bit level, 64 bit code

| | $d(\times r) =$ | EC 163 | TZV g1 $83 \times 3$ | TZV g1 $41 \times 5$ | HEC g2 83 | TZV g2 $41 \times 3$ |
|---|---|---|---|---|---|---|
| operation | 2 | 2.267 | 2.732 | 2.327 | 1.227 | 1.729 |
| | $+$ | 2.316 | 2.742 | 2.351 | 2.577 | 4.485 |
| | $\sigma$ | – | 0.019 | 0.019 | – | 0.020 |
| full scalar | bin | 704.4 | 734.5 | 603.5 | 467.4 | 680.1 |
| | NAF | 618.4 | 649.5 | 534.7 | 402.8 | 560.8 |
| | $w$-NAF | 562.7 | 588.0 | 481.9 | 340.6 | 461.6 |
| split scalar | bin | – | 425.1 | 312.2 | – | 431.7 |
| | NAF | – | 387.3 | 230.5 | – | 375.2 |
| | JSF | – | 373.1 | – | – | 356.7 |
| | $w$-NAF | – | 349.5 | 188.9 | – | 318.1 |

2 Ghz Quad Core Xeon running 64-bit code

# Performance Results: Scalar Multiplication

Timings (μsec) of ordinary binary curves and TZV at 192 bit level, 64 bit code

| | $d(\times r) =$ | EC 191 | TZV g1 $97 \times 3$ | TZV g1 $47 \times 5$ | HEC g2 97 | TZV g2 $47 \times 3$ |
|---|---|---|---|---|---|---|
| operation | 2 | 2.714 | 3.149 | 2.686 | 1.415 | 1.955 |
| | $+$ | 2.714 | 3.160 | 2.713 | 2.927 | 5.125 |
| | $\sigma$ | – | 0.019 | 0.021 | – | 0.020 |
| full scalar | bin | 975.6 | 987.1 | 795.2 | 632.6 | 868.7 |
| | NAF | 868.6 | 888.1 | 706.9 | 531.5 | 714.9 |
| | $w$-NAF | 783.8 | 800.0 | 637.4 | 450.5 | 590.7 |
| split scalar | bin | – | 583.8 | 401.8 | – | 553.6 |
| | NAF | – | 522.5 | 319.7 | – | 480.1 |
| | JSF | – | 496.7 | – | – | 455.9 |
| | $w$-NAF | – | 467.7 | 252.4 | – | 403.6 |

2 Ghz Quad Core Xeon running 64-bit code

# Performance Results: Scalar Multiplication

Timings (μsec) of curves and TZV over prime fields and TZV at 160 bit level, 32 bit code

| $\log_2 p(\times r) =$ | | EC 160 | HEC g2 80 | HEC g3 54 | TZV $80 \times 3$ | TZV $40 \times 5$ | TZV g2 $40 \times 3$ |
|---|---|---|---|---|---|---|---|
| affine coordinates | bin | 1106.7 | 913.8 | 1203.8 | 660.3 | 644.8 | 1586.0 |
| | NAF | 974.6 | 809.1 | 1035.4 | 616.4 | 617.8 | 1473.6 |
| | JSF | – | – | – | 563.4 | – | 1274.8 |
| | $w$-NAF | 893.3 | 733.6 | 930.8 | 539.0 | 481.3 | 1223.1 |
| other coordinates | bin | 529.2 | 833.7 | – | 568.8 | 668.3 | – |
| | NAF | 442.0 | 739.5 | – | 529.5 | 637.8 | – |
| | JSF | – | – | – | 479.8 | – | – |
| | $w$-NAF | 400.1 | 672.9 | – | 467.6 | 491.6 | – |

scalar always split for TZV

2 Ghz Quad Core Xeon running 32-bit code

# Performance Results: Scalar Multiplication

Timings (μsec) of curves and TZV over prime fields and TZV at 192 bit level, 32 bit code

| $\log_2 p(\times r) =$ | | EC 192 | HEC g2 96 | HEC g3 64 | TZV $96 \times 3$ | TZV $48 \times 5$ | TZV g2 $48 \times 3$ |
|---|---|---|---|---|---|---|---|
| affine coordinates | bin | 1949.2 | 1163.3 | 1524.6 | 924.2 | 807.0 | 1926.5 |
| | NAF | 1740.9 | 1043.4 | 1370.9 | 850.5 | 768.3 | 1763.1 |
| | JSF | – | – | – | 788.0 | – | 1559.2 |
| | $w$-NAF | 1569.8 | 944.2 | 1238.9 | 750.8 | 586.9 | 1531.8 |
| other coordinates | bin | 870.4 | 1065.6 | – | 824.5 | 810.0 | – |
| | NAF | 742.5 | 948.1 | – | 754.7 | 769.5 | – |
| | JSF | – | – | – | 692.3 | – | – |
| | $w$-NAF | 655.3 | 849.5 | – | 660.8 | 584.2 | – |

scalar always split for TZV

2 Ghz Quad Core Xeon running 32-bit code

# Performance Results: Scalar Multiplication

Timings (µsec) of curves and TZV over prime fields and TZV at 256 bit level, 32 bit code

| $\log_2 p(\times r) =$ | | EC 256 | HEC g2 128 | HEC g3 84 | TZV $128 \times 3$ | TZV $64 \times 5$ | TZV g2 $64 \times 3$ |
|---|---|---|---|---|---|---|---|
| affine coordinates | bin | 4365.3 | 2453.8 | 3210.9 | 1920.6 | 1166.1 | 2726.3 |
| | NAF | 3892.8 | 2199.5 | 2850.2 | 1779.8 | 1097.4 | 2516.5 |
| | JSF | – | – | – | 1638.1 | – | 2572.6 |
| | $w$-NAF | 3486.6 | 1974.3 | 2561.1 | 1541.0 | 831.2 | 2192.5 |
| other coordinates | bin | 1977.4 | 2151.6 | – | 1738.4 | 1175.0 | – |
| | NAF | 1674.2 | 1910.6 | – | 1638.9 | 1102.7 | – |
| | JSF | – | – | – | 1500.9 | – | – |
| | $w$-NAF | 1452.7 | 1712.6 | – | 1397.6 | 826.1 | – |

scalar always split for TZV

2 Ghz Quad Core Xeon running 32-bit code

# Performance Results: Scalar Multiplication

Timings (µsec) of curves and TZV over prime fields and TZV at 160 bit level, 64 bit code

| $\log_2 p(\times r) =$ | | EC 160 | HEC g2 80 | HEC g3 54 | TZV $80 \times 3$ | TZV $40 \times 5$ | TZV g2 $40 \times 3$ |
|---|---|---|---|---|---|---|---|
| affine coordinates | bin | 631.6 | 501.2 | 446.1 | 360.7 | 280.7 | 660.9 |
| | NAF | 567.3 | 428.7 | 399.2 | 338.9 | 269.9 | 617.1 |
| | JSF | – | – | – | 310.2 | – | 565.8 |
| | $w$-NAF | 514.8 | 387.2 | 366.9 | 300.6 | 215.0 | 505.8 |
| other coordinates | bin | 197.2 | 422.8 | – | 303.8 | 246.5 | – |
| | NAF | 167.6 | 377.9 | – | 286.6 | 238.1 | – |
| | JSF | – | – | – | 259.3 | – | – |
| | $w$-NAF | 155.8 | 343.8 | – | 258.5 | 191.0 | – |

scalar always split for TZV

2 Ghz Quad Core Xeon running 64-bit code

# Performance Results: Scalar Multiplication

Timings (µsec) of curves and TZV over prime fields and TZV at 192 bit level, 64 bit code

| $\log_2 p(\times r) =$ | | EC 192 | HEC g2 96 | HEC g3 64 | TZV $96 \times 3$ | TZV $48 \times 5$ | TZV g2 $48 \times 3$ |
|---|---|---|---|---|---|---|---|
| affine coordinates | bin | 920.4 | 628.0 | 619.0 | 462.2 | 347.4 | 832.3 |
| | NAF | 804.4 | 547.1 | 556.3 | 434.7 | 334.4 | 766.1 |
| | JSF | – | – | – | 397.9 | – | 723.1 |
| | $w$-NAF | 729.8 | 495.8 | 505.4 | 385.9 | 258.8 | 641.0 |
| other coordinates | bin | 295.5 | 502.2 | – | 365.2 | 312.1 | – |
| | NAF | 251.4 | 446.4 | – | 342.4 | 288.7 | – |
| | JSF | – | – | – | 311.3 | – | – |
| | $w$-NAF | 232.0 | 403.5 | – | 301.6 | 224.1 | – |

scalar always split for TZV

2 Ghz Quad Core Xeon running 64-bit code

# Performance Results: Scalar Multiplication

Timings (μsec) of curves and TZV over prime fields and TZV at 256 bit level, 64 bit code

| $\log_2 p(\times r) =$ | | EC 256 | HEC g2 128 | HEC g3 84 | TZV $128 \times 3$ | TZV $64 \times 5$ | TZV g2 $64 \times 3$ |
|---|---|---|---|---|---|---|---|
| affine coordinates | bin | 1891.4 | 1043.4 | 2098.7 | 812.4 | 628.9 | 1401.6 |
| | NAF | 1700.9 | 901.2 | 1775.6 | 757.7 | 586.0 | 1307.9 |
| | JSF | – | – | – | 696.0 | – | 1182.1 |
| | $w$-NAF | 1523.2 | 773.8 | 1537.3 | 654.9 | 431.7 | 1015.0 |
| other coordinates | bin | 565.5 | 756.0 | – | 600.7 | 543.1 | – |
| | NAF | 486.6 | 640.9 | – | 560.6 | 508.2 | – |
| | JSF | – | – | – | 511.9 | – | – |
| | $w$-NAF | 435.8 | 568.9 | – | 491.2 | 401.8 | – |

scalar always split for TZV

2 Ghz Quad Core Xeon running 64-bit code

# Outline

# Pairing in cryptography

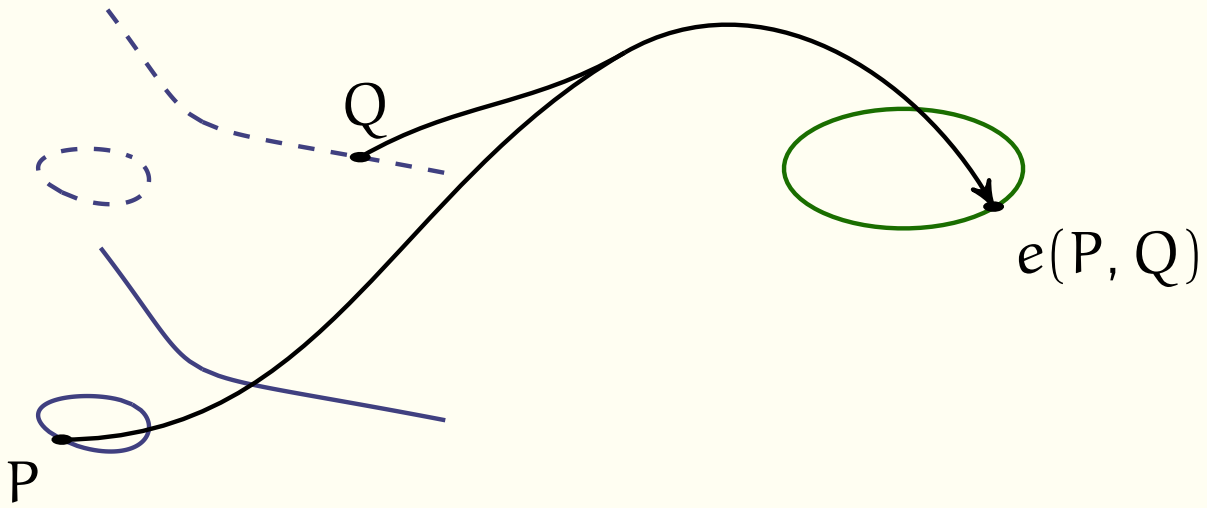
A (no longer so) new tool for cryptographers:

- $e$: $\mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$
- Bilinear: $e(aP, bQ) = e(P, Q)^{ab}$
- Non degenerate: there exists $P, Q$: $e(P, Q) \neq 1$
- Efficiently computable

# Pairing in cryptography

A (no longer so) new tool for cryptographers:

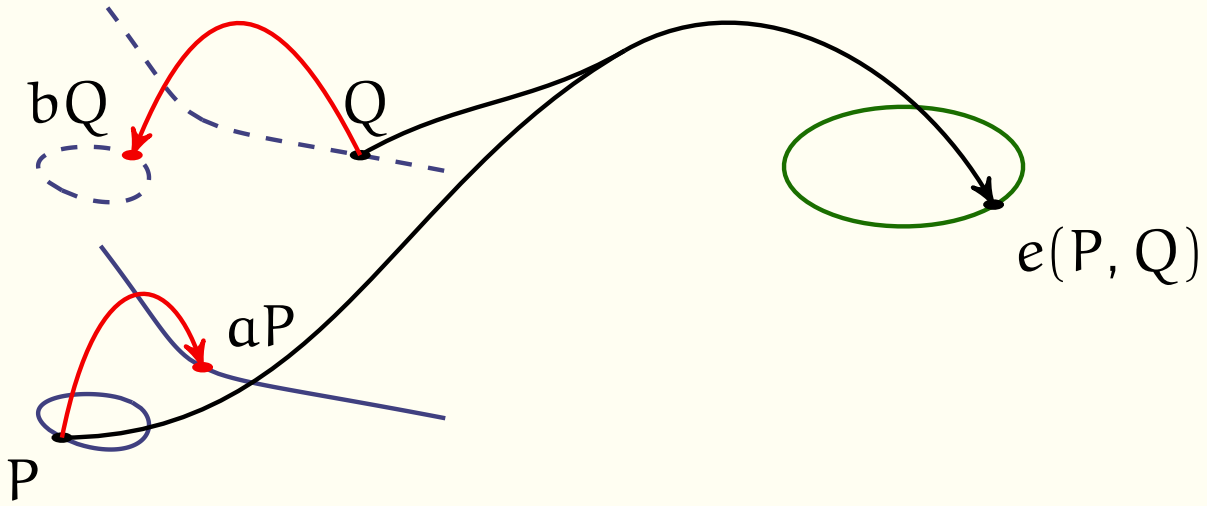
- $e\colon \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$
- Bilinear: $e(aP, bQ) = e(P, Q)^{ab}$
- Non degenerate: there exists $P, Q$: $e(P, Q) \neq 1$
- Efficiently computable



# Pairing in cryptography

A (no longer so) new tool for cryptographers:

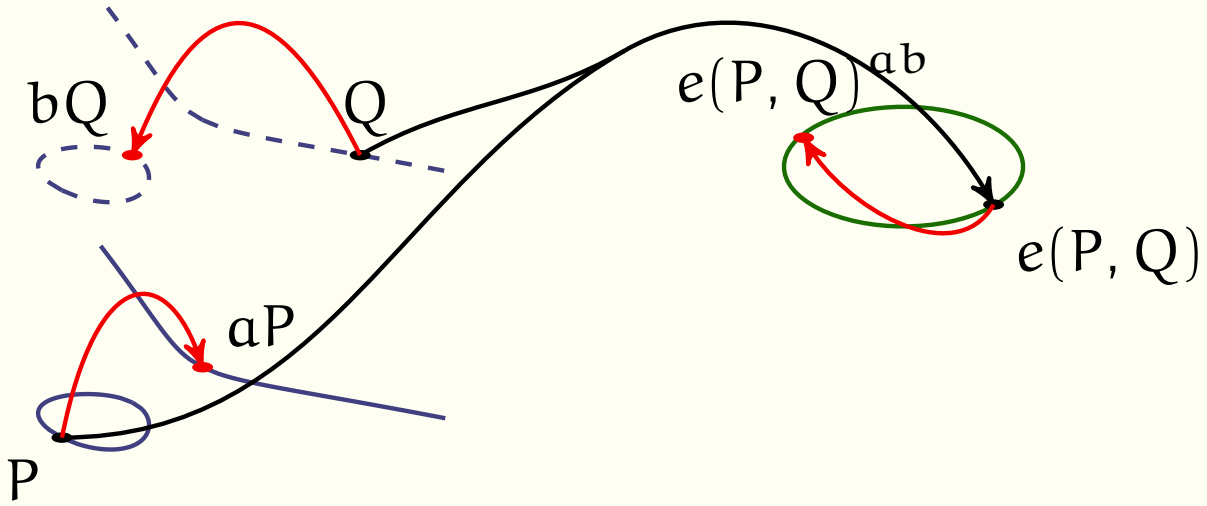
- $e\colon \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$
- Bilinear: $e(aP, bQ) = e(P, Q)^{ab}$
- Non degenerate: there exists $P, Q$: $e(P, Q) \neq 1$
- Efficiently computable



# Pairing in cryptography

A (no longer so) new tool for cryptographers:

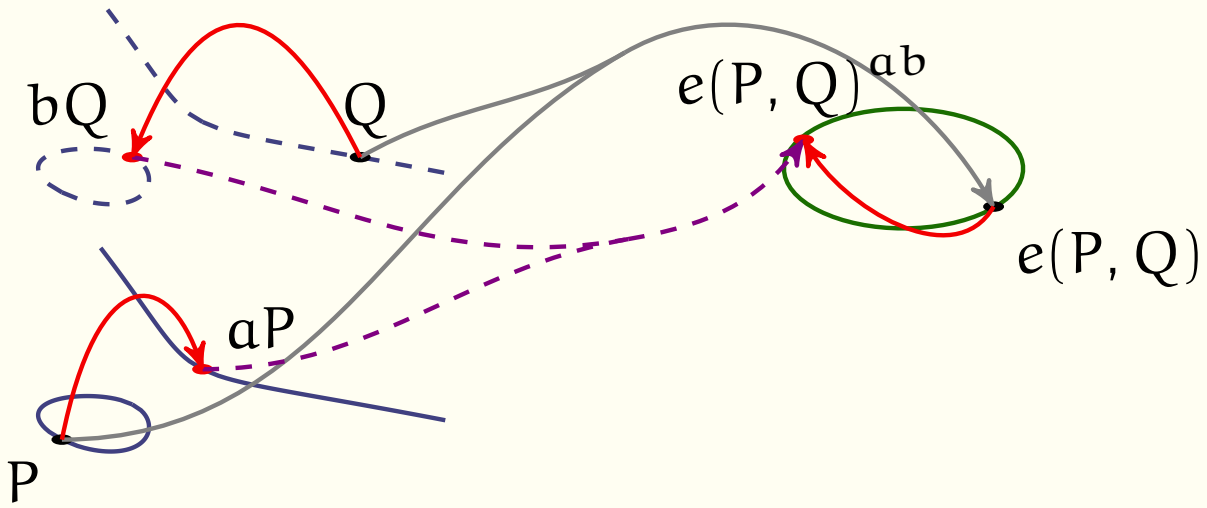
- $e\colon \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$
- Bilinear: $e(aP, bQ) = e(P, Q)^{ab}$
- Non degenerate: there exists $P, Q$: $e(P, Q) \neq 1$
- Efficiently computable



# Pairing in cryptography

A (no longer so) new tool for cryptographers:

- $e\colon \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$
- Bilinear: $e(aP, bQ) = e(P, Q)^{ab}$
- Non degenerate: there exists $P, Q$: $e(P, Q) \neq 1$
- Efficiently computable



# Why Pairing on Trace Zero Varieties?

Main observation from Karl Rubin and Alice Silverberg, 2002:

- Allow to obtain higher MOV security per bit than EC
- Boost the security parameter by a factor $r/\phi(r)$
- Application to pairing-based cryptography...

Supersingular EC over $\mathbb{F}_q$, $q = 2^d$

$\mathcal{E}: y^2 + y = x^3 + x + b \ , \qquad b \in \mathbb{F}_2$

- $\#\mathcal{E} = 2^d \pm 2^{(d+1)/2} + 1$
- embedding degree $k = 4$
- 1200-bit security: take $d \approx 1200/k \approx 307$

# Why Pairing on Trace Zero Varieties?

Main observation from Karl Rubin and Alice Silverberg, 2002:

- Allow to obtain higher MOV security per bit than EC
- Boost the security parameter by a factor $r/\phi(r)$
- Application to pairing-based cryptography...

## Supersingular TZV over $\mathbb{F}_q$, $q = 2^d$

$\mathcal{E}: y^2 + y = x^3 + x + b \ , \qquad b \in \mathbb{F}_2$

- $\#\mathcal{E}_3 = 2^{2d} \mp 2^{(3d+1)/2} + 2^d \mp 2^{(d+1)/2} + 1$
- embedding degree $k = 3 \cdot 4 = 12$
- 1200-bit security: take $d \approx 1200/12 \approx 103$

# How to compute a pairing?

Compute $f_P(Q)$:

- Recall $f_P \in \mathbb{F}_q(\mathcal{E})$, with divisor $\ell\,(P) - \ell\,\mathcal{O}$
- Rational function of degree $\approx \ell$
- Storing coefficients, evaluate it: unfeasible!

# Miller function

Idea: combine simple functions with small support to "build" $f_P$.

Miller function: is a function $f_{n,P} \in \mathbb{F}_q(\mathcal{E})$ with divisor:

$$(f_{n,P}) = n(P) - ([n]P) - (n-1)\mathcal{O}$$

Properties:

1. $f_{\lambda+\mu,P} = f_{\lambda,P} \cdot f_{\mu,P} \cdot \frac{l_{[\lambda]P,[\mu]P}}{v_{[\lambda+\mu]P}}$
2. $f_{\lambda\mu,P} = f_{\lambda,P}^{\mu} \cdot f_{\mu,[\lambda]P}$

$$\begin{cases} f_{n+1,P} = f_{n,P} \cdot \overset{=1}{f_{1,P}} \cdot \frac{l_{[n]P,P}}{v_{[n+1]P}} \\ f_{2n,P} = f_{n,P}^2 \cdot \frac{l_{[n]P,[n]P}}{v_{[2n]P}} \end{cases}$$

where $l_{P,P'}$ is the line through $P$ and $P'$
$v_{[\lambda+\mu]P}$ is the vertical line through $[\lambda+\mu]P$

Further: *directly compute evaluation at* $Q$.

Note that $v_{[\lambda+\mu]P}(Q) = x(Q) - x([\lambda+\mu]P)$.

# Miller function

Idea: combine simple functions with small support to "build" $f_P$.

Miller function: is a function $f_{n,P} \in \mathbb{F}_q(\mathcal{E})$ with divisor:

$$(f_{n,P}) = n(P) - ([n]P) - (n-1)\mathcal{O}$$

Properties:

1. $f_{\lambda+\mu,P} = f_{\lambda,P} \cdot f_{\mu,P} \cdot \frac{l_{[\lambda]P,[\mu]P}}{\nu_{[\lambda+\mu]P}}$
2. $f_{\lambda\mu,P} = f_{\lambda,P}^{\mu} \cdot f_{\mu,[\lambda]P}$

$$\begin{cases} f_{n+1,P} = f_{n,P} \cdot \overset{=1}{f_{1,P}} \cdot \frac{l_{[n]P,P}}{\nu_{[n+1]P}} \\ f_{2n,P} = f_{n,P}^2 \cdot \frac{l_{[n]P,[n]P}}{\nu_{[2n]P}} \end{cases}$$

where $l_{P,P'}$ is the line through $P$ and $P'$
$\nu_{[\lambda+\mu]P}$ is the vertical line through $[\lambda+\mu]P$

Further: *directly compute evaluation at* $Q$.

Note that $\nu_{[\lambda+\mu]P}(Q) = x(Q) - x([\lambda+\mu]P)$.

# Miller function

Idea: combine simple functions with small support to "build" $f_P$.

Miller function: is a function $f_{n,P} \in \mathbb{F}_q(\mathcal{E})$ with divisor:

$$(f_{n,P}) = n(P) - ([n]P) - (n-1)\mathcal{O}$$

Properties:

1. $f_{\lambda+\mu,P} = f_{\lambda,P} \cdot f_{\mu,P} \cdot \frac{l_{[\lambda]P,[\mu]P}}{v_{[\lambda+\mu]P}}$
2. $f_{\lambda\mu,P} = f_{\lambda,P}^{\mu} \cdot f_{\mu,[\lambda]P}$

$$\begin{cases} f_{n+1,P} = f_{n,P} \cdot \overset{=1}{f_{1,P}} \cdot \frac{l_{[n]P,P}}{v_{[n+1]P}} \\ f_{2n,P} = f_{n,P}^2 \cdot \frac{l_{[n]P,[n]P}}{v_{[2n]P}} \end{cases}$$

where $l_{P,P'}$ is the line through $P$ and $P'$
$v_{[\lambda+\mu]P}$ is the vertical line through $[\lambda+\mu]P$

Further: *directly compute evaluation at* $Q$.

Note that $v_{[\lambda+\mu]P}(Q) = x(Q) - x([\lambda+\mu]P)$.

# Miller's algorithm (elliptic curves)

**Input**: $P, Q \in \mathcal{E}[\ell]$ , $\mathbb{N} \ni n = \sum_{i=0}^{L} n_i 2^i$, with $n_j = 0, 1$.

**Output**: $f_{n,P}(Q)$

1 $T \leftarrow P$ , $f \leftarrow 1$
2 **for** $j = L - 1$ **downto** $0$ **do**
3 $\quad f \leftarrow f^2 \cdot l_{T,T}(Q) \,/v_{[2]T}(Q)$
4 $\quad T \leftarrow [2]T$
5 $\quad$ **if** $n_j = 1$ **then**
6 $\quad\quad f \leftarrow f \cdot l_{T,P}(Q) \,/v_{T+P}(Q)$
7 $\quad\quad T \leftarrow T + P$
8 $\quad$ **endif**
9 **endfor**
10 **return** $f$

# Miller's algorithm (elliptic curves)

**Input**: $P, Q \in \mathcal{E}[\ell]$ , $\mathbb{N} \ni n = \sum_{i=0}^{L} n_i 2^i$, with $n_j = 0, 1$.

**Output**: $f_{n,P}(Q)$

1 $T \leftarrow P$ , $f \leftarrow 1$
2 **for** $j = L - 1$ **downto** $0$ **do**
3 $\quad f \leftarrow f^2 \cdot l_{T,T}(Q) \,/v_{[2]T}(Q)$
4 $\quad T \leftarrow [2]T$
5 $\quad$ **if** $n_j = 1$ **then**
6 $\quad\quad f \leftarrow f \cdot l_{T,P}(Q) \,/v_{T+P}(Q)$
7 $\quad\quad T \leftarrow T + P$
8 $\quad$ **endif**
9 **endfor**
10 **return** $f$

# Miller's algorithm (elliptic curves)

**Input**: $P, Q \in \mathcal{E}[\ell]$ , $\mathbb{N} \ni n = \sum_{i=0}^{L} n_i 2^i$, with $n_j = 0, 1$.

**Output**: $f_{n,P}(Q)$

1 $T \leftarrow P$ , $f \leftarrow 1$
2 **for** $j = L - 1$ **downto** $0$ **do**
3 $\quad f \leftarrow f^2 \cdot l_{T,T}(Q) \,/v_{[2]T}(Q)$
4 $\quad T \leftarrow [2]T$
5 $\quad$ **if** $n_j = 1$ **then**
6 $\quad\quad f \leftarrow f \cdot l_{T,P}(Q) \,/v_{T+P}(Q)$
7 $\quad\quad T \leftarrow T + P$
8 $\quad$ **endif**
9 **endfor**
10 **return** $f$

# Miller's algorithm (elliptic curves)

**Input**: $P, Q \in \mathcal{E}[\ell]$ , $\mathbb{N} \ni n = \sum_{i=0}^{L} n_i 2^i$, with $n_j = 0, 1$.

**Output**: $f_{n,P}(Q)$

1 $T \leftarrow P$ , $f \leftarrow 1$
2 **for** $j = L - 1$ **downto** $0$ **do**
3 $\quad f \leftarrow f^2 \cdot l_{T,T}(Q) \; / v_{[2]T}(Q)$
4 $\quad T \leftarrow [2]T$
5 $\quad$ **if** $n_j = 1$ **then**
6 $\quad\quad f \leftarrow f \cdot l_{T,P}(Q) \; / v_{T+P}(Q)$
7 $\quad\quad T \leftarrow T + P$
8 $\quad$ **endif**
9 **endfor**
10 **return** $f$

denominator elimination

# Miller's algorithm (elliptic curves)

**Input**: $P, Q \in \mathcal{E}[\ell]$ , $\mathbb{N} \ni n = \sum_{i=0}^{L} n_i 2^i$, with $n_j = 0, 1$.

**Output**: $f_{n,P}(Q)$

1 $T \leftarrow P$ , $f \leftarrow 1$
2 **for** $j = L - 1$ **downto** $0$ **do**
3 $\quad f \leftarrow f^2 \cdot l_{T,T}(Q)$
4 $\quad T \leftarrow [2]T$
5 $\quad$ **if** $n_j = 1$ **then**
6 $\quad\quad f \leftarrow f \cdot l_{T,P}(Q)$
7 $\quad\quad T \leftarrow T + P$
8 $\quad$ **endif**
9 **endfor**
10 **return** $f^{\textit{some power}}$

denominator elimination

# Many Improvements...

- Barreto et. al. [02–07]: $\eta$ and $\eta_T$ with supersingular (H)EC
- Rubin & Silverberg [02–08]: supersingular AV (notably TZV)
- Scott [05]: An EC endowed with an efficient endomorphism
- Hess et. al. [06]: Ate and twisted–Ate with ordinary (H)EC
- Vercauteren [08]: Optimal pairings
- Hess [08]: Pairing lattices
- ... and many more ...

# Outline

# Towards a new algorithm for the Tate pairing

Recall:

## Tate pairing

$$t(P,Q) = f_P(Q)^{\frac{q^k-1}{\ell}}$$

- $f_P(Q) = f_{\ell,P}(Q)$
- Miller's algorithm to compute $f_{n,P}(Q)$

Remark: $t(P,Q) = f_{\ell,P}(Q)^{\frac{q^k-1}{\ell}} = f_{N,P}(Q)^{\frac{q^k-1}{N}}$

- with $\ell \mid N \mid q^k - 1$
- $N = \#\mathcal{E}_r = O(q^2)$, lower Hamming weigth than $\ell$ in char 2,3
- $\eta_T$ pairing (Barreto et. al.) $N = O(q^{3/2})$
- Promise: $N = q$

# Towards a new algorithm for the Tate pairing

Recall:

Tate pairing

$$t(P,Q) = f_P(Q)^{\frac{q^k-1}{\ell}}$$

- $f_P(Q) = f_{\ell,P}(Q)$
- Miller's algorithm to compute $f_{n,P}(Q)$

Remark: $t(P,Q) = f_{\ell,P}(Q)^{\frac{q^k-1}{\ell}} = f_{N,P}(Q)^{\frac{q^k-1}{N}}$

- with $\ell \mid N \mid q^k - 1$
- $N = \#\mathcal{E}_r = O(q^2)$, lower Hamming weigth than $\ell$ in char 2,3
- $\eta_T$ pairing (Barreto et. al.) $N = O(q^{3/2})$
- Promise: $N = q$

# Towards a new algorithm for the Tate pairing

Recall:

## Tate pairing

$$t(P,Q) = f_P(Q)^{\frac{q^k-1}{\ell}}$$

- $f_P(Q) = f_{\ell,P}(Q)$
- Miller's algorithm to compute $f_{n,P}(Q)$

Remark: $t(P,Q) = f_{\ell,P}(Q)^{\frac{q^k-1}{\ell}} = f_{N,P}(Q)^{\frac{q^k-1}{N}}$

- with $\ell \mid N \mid q^k - 1$
- $N = \#\mathcal{E}_r = O(q^2)$, lower Hamming weigth than $\ell$ in char 2,3
- $\eta_T$ pairing (Barreto et. al.) $N = O(q^{3/2})$
- Promise: $N = q$

# Towards a new algorithm for the Tate pairing

Recall:

## Tate pairing

$$t(P,Q) = f_P(Q)^{\frac{q^k-1}{\ell}}$$

- $f_P(Q) = f_{\ell,P}(Q)$
- Miller's algorithm to compute $f_{n,P}(Q)$

Remark: $t(P,Q) = f_{\ell,P}(Q)^{\frac{q^k-1}{\ell}} = f_{N,P}(Q)^{\frac{q^k-1}{N}}$

- with $\ell \mid N \mid q^k - 1$
- $N = \#\mathcal{E}_r = O(q^2)$, lower Hamming weigth than $\ell$ in char 2,3
- $\eta_T$ pairing (Barreto et. al.) $N = O(q^{3/2})$
- Promise: $N = q$

# Towards a new algorithm for the Tate pairing

Recall:

## Tate pairing

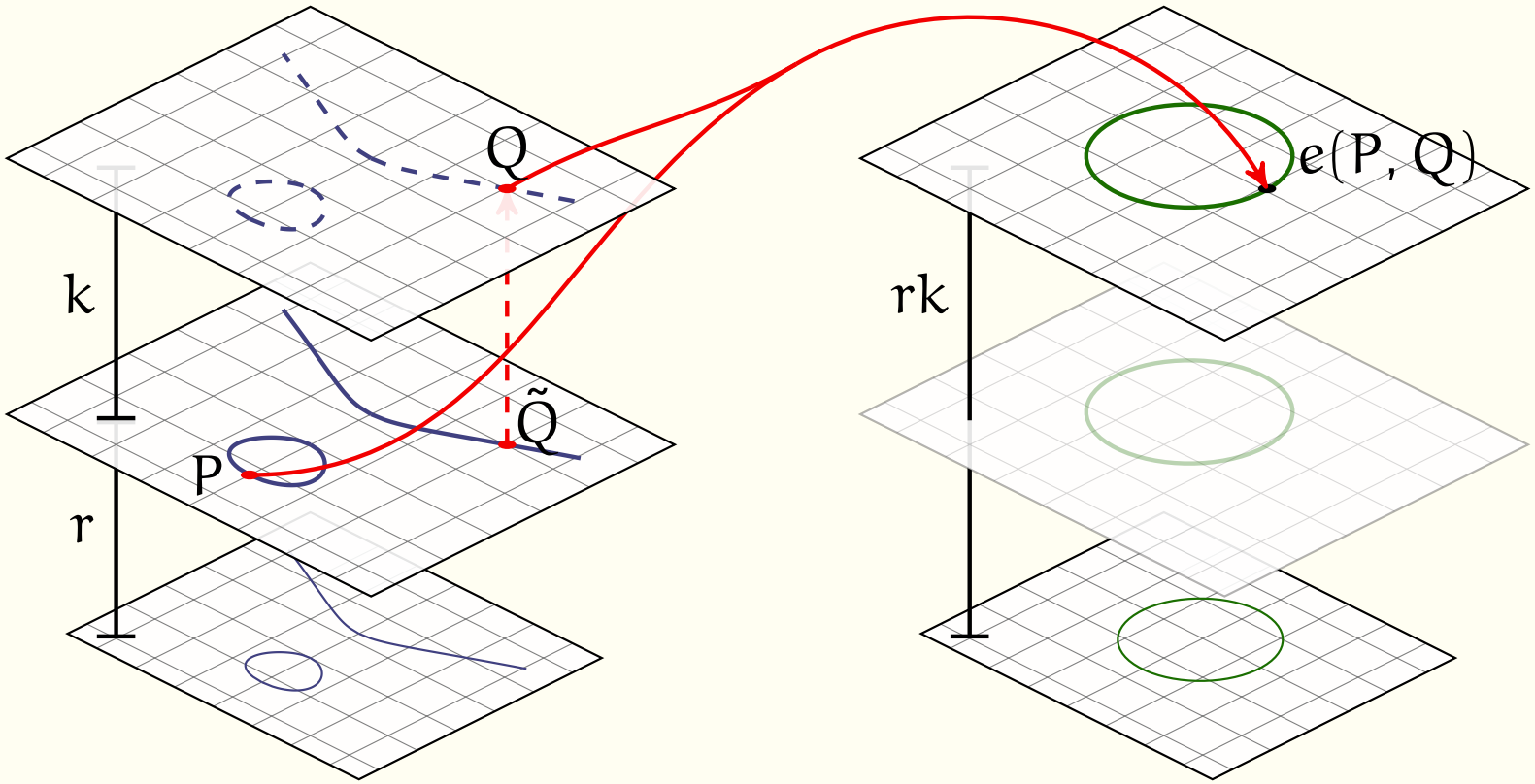
$$t(P,Q) = f_P(Q)^{\frac{q^k-1}{\ell}}$$

- $f_P(Q) = f_{\ell,P}(Q)$
- Miller's algorithm to compute $f_{n,P}(Q)$

Remark: $t(P,Q) = f_{\ell,P}(Q)^{\frac{q^k-1}{\ell}} = f_{N,P}(Q)^{\frac{q^k-1}{N}}$

- with $\ell \mid N \mid q^k - 1$
- $N = \#\mathcal{E}_r = O(q^2)$, lower Hamming weigth than $\ell$ in char 2,3
- $\eta_T$ pairing (Barreto et. al.) $N = O(q^{3/2})$
- Promise: $N = q$

# One figure says it all!



# The main result

## Theorem (Theorem 2)

*Let $\mathcal{E}_r$ be a supersingular TZV. Suppose $k$ is even and the distortion map allows for denominator elimination.*
*Then the Tate pairing can be computed as:*

$$t(P,Q) = \left( \prod_{i=0}^{r-1} f_{q,P}\left(Q^{\sigma_i}\right)^{q^{i(r+1)}} \right)^{M\frac{a}{r}q^{a-1}},$$

*where $\sigma_i = \sigma^{ij}$, $a = k/2$ and $M = q^{k/2} - 1$.*

# The new algorithm

## Formula for $t(P, Q)$:

$$\left(\prod_{i=0}^{r-1} f_{q,P}\left(Q^{\sigma_i}\right)^{q^{i(r+1)}}\right)^{M\frac{a}{r}q^{a-1}}$$

## Parallelization:

- $r$ processors
- loop on $q$

## Serial version:

- Save intermediate T's
- Store $O(\log q)$ points or interleave

**Input**: $P \in \mathbb{G}_1$ , $Q \in \mathbb{G}_2$
**Output**: $t_{TZV}(P, Q)$

1 $f \leftarrow 1$
2 **for** $i = 0$ **to** $r - 1$ **do**
3 $\quad f \leftarrow f \cdot \left(f_{q,P}\left(Q^{\sigma_i}\right)\right)^{q^{i(r+1)}}$
4 **endfor**
5 $f \leftarrow \left(f^{\frac{a}{r}}\right)^{q^{a-1}}$
6 **return** $f^{M+1}/f$

Without line 5: still bilinear

# The new algorithm

## Formula for $t(P,Q)$:

$$\left(\prod_{i=0}^{r-1} f_{q,P}\left(Q^{\sigma_i}\right)^{q^{i(r+1)}}\right)^{M\frac{a}{r}q^{a-1}}$$

## Parallelization:

- $r$ processors
- loop on $q$

## Serial version:

- Save intermediate T's
- Store $O(\log q)$ points or interleave

**Input**: $P \in \mathbb{G}_1$ , $Q \in \mathbb{G}_2$
**Output**: $t_{TZV}(P,Q)$

1 $f \leftarrow 1$
2 **for** $i = 0$ **to** $r-1$ **do**
3 $\quad f \leftarrow f \cdot \left(f_{q,P}\left(Q^{\sigma_i}\right)\right)^{q^{i(r+1)}}$
4 **endfor**
5 $f \leftarrow \left(f^{\frac{a}{r}}\right)^{q^{a-1}}$
6 **return** $f^{M+1}/f$

Without line 5: still bilinear

# The main result (proof)

## Tate pairing

$$t(P,Q) = f_P(Q)^{\frac{q^k-1}{\ell}} \qquad \left(\prod_{i=0}^{r-1} f_{q,P}\left(Q^{\sigma_i}\right)^{q^{i(r+1)}}\right)^{M\frac{a}{r}q^{a-1}}$$

1. Arithmetic of the Miller function
2. Use of the $q^r$-Frobenius $\pi$
3. Use of the $q$-Frobenius $\sigma$

# Endomorphisms and Miller's function

If $\phi \in \operatorname{End} E$ is purely inseparable of degree $T$,

$$f_{n,\phi(P)} \circ \phi = f_{n,P}^{T}$$

## Dual of $q^r$-Frobenius $\pi$

Let $\hat{\pi}$ be the dual of the $q^r$-Frobenius $\pi$

- purely inseparable of degree $q^r$ (supersingular curves)
- $P \in \operatorname{Ker}(\pi - [1])$ $\quad \hat{\pi}(P) = [q^r]P$
- $Q \in \operatorname{Ker}(\pi - [q^r])$ $\quad \hat{\pi}(Q) = Q$

$$f_{n,[q^r]P}(Q) = f_{n,\hat{\pi}P}(\hat{\pi}Q) = f_{n,\hat{\pi}P} \circ \hat{\pi}(Q) = f_{n,P}(Q)^{q^r}$$

# Endomorphisms and Miller's function

If $\phi \in \operatorname{End} E$ is purely inseparable of degree $T$,

$$f_{n,\phi(P)} \circ \phi = f_{n,P}^{T}$$

## Dual of q-Frobenius $\sigma$

Let $\hat{\sigma}$ be the dual of the q-Frobenius $\sigma$.

- purely inseparable of degree $q$ (supersingular curves)
- $P \in \operatorname{Ker}(\pi - [1])$ $\quad \hat{\sigma}(P) = [q/s]P$
- $Q \in \operatorname{Ker}(\pi - [q^r])$ $\quad \hat{\sigma}(Q) = [q^{1-\Sigma}]Q \quad (\Sigma = \frac{r^2+1}{2})$

$$f_{n,[q]P}(Q) = f_{n,\hat{\sigma}^{r+2}(P)}\left((\hat{\sigma}^{r+2} \circ \sigma^{j})(Q)\right) =$$

$$= f_{n,\hat{\sigma}^{r+2}(P)} \circ \hat{\sigma}^{r+2}\left(Q^{\sigma^{j}}\right) = f_{n,P}\left(Q^{\sigma^{j}}\right)^{q^{r+2}}$$

# The main result (proof)

## Tate pairing

$$t(P,Q) = f_P(Q)^{\frac{q^k-1}{\ell}} \qquad \left(\prod_{i=0}^{r-1} f_{q,P}\left(Q^{\sigma_i}\right)^{q^{i(r+1)}}\right)^{M\frac{a}{r}q^{a-1}}$$

1. Arithmetic of the Miller's function
2. Use of the $q^r$-Frobenius $\pi$ $\qquad f_{n,[q^r]P}(Q) = f_{n,P}(Q)^{q^r}$
3. Use of the $q$-Frobenius $\sigma$

$$f_{n,[q]P}(Q) = f_{n,P}\left(Q^{\sigma^j}\right)^{q^{r+2}}$$

# The main result (proof)

## Tate pairing

$$t(P,Q) = f_P(Q)^{\frac{q^k-1}{\ell}} \qquad \left(\prod_{i=0}^{r-1} f_{q,P}\left(Q^{\sigma_i}\right)^{q^{i(r+1)}}\right)^{M\frac{a}{r}q^{a-1}}$$

1. Arithmetic of the Miller's function
2. Use of the $q^r$-Frobenius $\pi$ $\qquad f_{n,[q^r]P}(Q) = f_{n,P}(Q)^{q^r}$
3. Use of the $q$-Frobenius $\sigma$

$$f_{n,[q]P}(Q) = f_{n,P}\left(Q^{\sigma^j}\right)^{q^{r+2}}$$

$$t(P,Q) = f_{\ell,P}(Q)^{(q^k-1)/\ell} = f_{q^{k/2}+1,P}(Q)^{q^{k/2}-1} = f_{q^{k/2},P}(Q)^{q^{k/2}-1} = f_{q^a,P}(Q)^M$$

# The main result (proof)

## Tate pairing

$$t(P,Q) = f_P(Q)^{\frac{q^k-1}{\ell}} \qquad \left(\prod_{i=0}^{r-1} f_{q,P}\left(Q^{\sigma_i}\right)^{q^{i(r+1)}}\right)^{M\frac{a}{r}q^{a-1}}$$

1. Arithmetic of the Miller's function
2. Use of the $q^r$-Frobenius $\pi$ $\qquad f_{n,[q^r]P}(Q) = f_{n,P}(Q)^{q^r}$
3. Use of the $q$-Frobenius $\sigma$

$$f_{n,[q]P}(Q) = f_{n,P}\left(Q^{\sigma^j}\right)^{q^{r+2}}$$

$$t(P,Q) = f_{\ell,P}(Q)^{(q^k-1)/\ell} = f_{q^{k/2}+1,P}(Q)^{q^{k/2}-1} = f_{q^{k/2},P}(Q)^{q^{k/2}-1} = f_{q^a,P}(Q)^M$$

k even: $q^k - 1 = (q^{k/2}-1)(q^{k/2}+1)$; k minimal

# The main result (proof)

## Tate pairing

$$t(P,Q) = f_P(Q)^{\frac{q^k-1}{\ell}} \qquad \left(\prod_{i=0}^{r-1} f_{q,P}\left(Q^{\sigma_i}\right)^{q^{i(r+1)}}\right)^{M\frac{a}{r}q^{a-1}}$$

1. Arithmetic of the Miller's function
2. Use of the $q^r$-Frobenius $\pi$ $\qquad f_{n,[q^r]P}(Q) = f_{n,P}(Q)^{q^r}$
3. Use of the $q$-Frobenius $\sigma$

$$f_{n,[q]P}(Q) = f_{n,P}\left(Q^{\sigma^j}\right)^{q^{r+2}}$$

$$t(P,Q) = f_{\ell,P}(Q)^{(q^k-1)/\ell} = f_{q^{k/2}+1,P}(Q)^{q^{k/2}-1} = f_{q^{k/2},P}(Q)^{q^{k/2}-1} = f_{q^a,P}(Q)^M$$

$$f_{N,P}(Q) = f_{N-1,P}(Q)\cdot f_{1,P}(Q)\cdot l_{[N-1]P,P}(Q)/v_{[N]P}(Q)$$

- $l_{[N-1]P,P}$ is the vertical line through $P$ (denom. elimination)

# The main result (proof)

## Tate pairing

$$t(P,Q) = f_P(Q)^{\frac{q^k-1}{\ell}} \qquad \left(\prod_{i=0}^{r-1} f_{q,P}\left(Q^{\sigma_i}\right)^{q^{i(r+1)}}\right)^{M\frac{a}{r}q^{a-1}}$$

1. Arithmetic of the Miller's function
2. Use of the $q^r$-Frobenius $\pi$ $\qquad f_{n,[q^r]P}(Q) = f_{n,P}(Q)^{q^r}$
3. Use of the $q$-Frobenius $\sigma$

$$f_{n,[q]P}(Q) = f_{n,P}\left(Q^{\sigma^j}\right)^{q^{r+2}}$$

$$t(P,Q) = f_{\ell,P}(Q)^{(q^k-1)/\ell} = f_{q^{k/2}+1,P}(Q)^{q^{k/2}-1} = f_{q^{k/2},P}(Q)^{q^{k/2}-1} = f_{q^a,P}(Q)^M$$

# The main result (proof)

## Tate pairing

$$t(P,Q) = f_P(Q)^{\frac{q^k-1}{\ell}} \qquad \left(\prod_{i=0}^{r-1} f_{q,P}\left(Q^{\sigma_i}\right)^{q^{i(r+1)}}\right)^{M\frac{a}{r}q^{a-1}}$$

1. Arithmetic of the Miller's function
2. Use of the $q^r$-Frobenius $\pi$ $\qquad f_{n,[q^r]P}(Q) = f_{n,P}(Q)^{q^r}$
3. Use of the $q$-Frobenius $\sigma$ $\qquad f_{n,[q]P}(Q) = f_{n,P}\left(Q^{\sigma^j}\right)^{q^{r+2}}$

$$\begin{aligned} f_{q^a,P}(Q)^M &= \left(f_{q,P}(Q)^{q^{a-1}} \cdot f_{q,[q]P}(Q)^{q^{a-2}} \cdots f_{q,[q^{a-1}]P}(Q)\right)^M = \\ &= \left(f_{q,P}(Q)^{\frac{a}{r}q^{(a-1)}} \cdot f_{q,[q]P}(Q)^{\frac{a}{r}q^{(a-2)}} \cdots f_{q,[q^{r-1}]P}(Q)^{\frac{a}{r}q^{(a-r)}}\right)^M = \\ &= \left(f_{q,P}(Q) \cdot f_{q,[q]P}(Q)^{q^{-1}} \cdots f_{q,[q^{r-1}]P}(Q)^{q^{-(r-1)}}\right)^{M\frac{a}{r}q^{a-1}} \end{aligned}$$

# The main result (proof)

## Tate pairing

$$t(P,Q) = f_P(Q)^{\frac{q^k-1}{\ell}} \qquad \left(\prod_{i=0}^{r-1} f_{q,P}\left(Q^{\sigma_i}\right)^{q^{i(r+1)}}\right)^{M\frac{a}{r}q^{a-1}}$$

1. Arithmetic of the Miller's function
2. Use of the $q^r$-Frobenius $\pi$ $\qquad f_{n,[q^r]P}(Q) = f_{n,P}(Q)^{q^r}$
3. Use of the q-Frobenius $\sigma$ $\qquad f_{n,[q]P}(Q) = f_{n,P}\left(Q^{\sigma^j}\right)^{q^{r+2}}$

$$t(P,Q) = \left(\prod_{i=0}^{r-1} f_{q,[q^i]P}\left(Q\right)^{q^{-i}}\right)^{M\frac{a}{r}q^{a-1}} = \left(\prod_{i=0}^{r-1} f_{q,P}\left(Q^{\sigma_i}\right)^{q^{i(r+1)}}\right)^{M\frac{a}{r}q^{a-1}}$$

# The main result (proof)

## Tate pairing

$$t(P,Q) = f_P(Q)^{\frac{q^k-1}{\ell}} \qquad \left(\prod_{i=0}^{r-1} f_{q,P}\left(Q^{\sigma_i}\right)^{q^{i(r+1)}}\right)^{M\frac{a}{r}q^{a-1}}$$

1. Arithmetic of the Miller's function
2. Use of the $q^r$-Frobenius $\pi$ $\qquad f_{n,[q^r]P}(Q) = f_{n,P}(Q)^{q^r}$
3. Use of the q-Frobenius $\sigma$

$$f_{n,[q]P}(Q) = f_{n,P}\left(Q^{\sigma^j}\right)^{q^{r+2}}$$

$$t(P,Q) = \left(\prod_{i=0}^{r-1} f_{q,[q^i]P}\left(Q\right)^{q^{-i}}\right)^{M\frac{a}{r}q^{a-1}} = \left(\prod_{i=0}^{r-1} f_{q,P}\left(Q^{\sigma_i}\right)^{q^{i(r+1)}}\right)^{M\frac{a}{r}q^{a-1}}$$

# Outline

# Performance Results, Part II

Table: Pairings on (μsec) EC and TZV ($r = 3$) over $\mathbb{F}_{2^d}$ (32 bit)

| Pairing | Loop Length | EC $d = 239$ | TZV $d = 79$ | EC $d = 307$ | TZV $d = 103$ | EC $d = 457$ | TZV $d = 157$ |
|---|---|---|---|---|---|---|---|
| $t_N$ | $N = O(q^2)$ | 1280.8 | 840.5 | 2231.7 | 1524.1 | 9006.5 | 3914.8 |
| $\eta$ | $q^3$ | 1226.9 | 1168.1 | 2141.9 | 2123.1 | 8589.1 | 5610.5 |
| $\eta_T$ | $2^{(3d+1)/2} - 1$ | 667.7 | 633.3 | 1163.0 | 1139.7 | 4692.5 | 2960.4 |
| $a_{opt}$ | $2^{(3d-1)/2}$ | 685.2 | 625.4 | 1185.5 | 1138.5 | 4889.9 | 2886.9 |
| $\eta$ (HLV) | $q^3$ | 1197.8 | 1239.2 | 2376.9 | 2359.6 | 8159.3 | 5757.0 |
| $\eta_T$ (HLV) | $2^{(3d+1)/2} - 1$ | 656.1 | 609.0 | 1120.2 | 1097.8 | 4403.5 | 2864.4 |
| $t_{TZV}$ | $3 \times q$ | – | 1062.6 | – | 1967.1 | – | 5086.4 |
| $t_\sigma$ | $2 \times s$ | – | 1336.8 | – | 2421.4 | – | 6028.8 |
| $t_{TZV}$ (Par) | $3 \times q$ | – | 435.0 | – | 807.2 | – | 1959.1 |
| $t_{\mathcal{E}_3}$ (Par) | $2 \times O(q)$ | – | 501.0 | – | 867.5 | – | 2154.6 |

2 Ghz Quad Core Xeon running 32-bit code

Table: Pairings ($\mu$sec) on EC and TZV ($r = 3$) over $\mathbb{F}_{2^d}$ (64 bit)

| Pairing | Loop Length | EC $d = 239$ | TZV $d = 79$ | EC $d = 307$ | TZV $d = 103$ | EC $d = 457$ | TZV $d = 157$ |
|---|---|---|---|---|---|---|---|
| $t_N$ | $N = O(q^2)$ | 617.9 | 699.1 | 1058.6 | 1111.4 | 3587.8 | 2816.9 |
| $\eta$ | $q^3$ | 580.0 | 949.1 | 1001.3 | 1527.4 | 3455.2 | 4028.2 |
| $\eta_T$ | $2^{(3d+1)/2} - 1$ | 326.2 | 526.8 | 563.6 | 861.5 | 1849.6 | 2157.4 |
| $a_{opt}$ | $2^{(3d-1)/2}$ | 333.9 | 512.9 | 573.6 | 817.4 | 1875.7 | 2104.4 |
| $\eta$ (HLV) | $q^3$ | 634.9 | 1022.3 | 1056.0 | 1615.8 | 3737.1 | 4066.3 |
| $\eta_T$ (HLV) | $2^{(3d+1)/2} - 1$ | 309.6 | 495.0 | 521.2 | 788.9 | 1771.1 | 2058.8 |
| $t_{TZV}$ | $3 \times q$ | – | 860.8 | – | 1392.1 | – | 3710.9 |
| $t_\sigma$ | $2 \times s$ | – | 1114.4 | – | 1702.9 | – | 4374.2 |
| $t_{TZV}$ (Par) | $3 \times q$ | – | 343.5 | – | 547.8 | – | 1424.9 |
| $t_{\mathcal{E}_3}$ (Par) | $2 \times O(q)$ | – | 415.7 | – | 638.0 | – | 1565.8 |

2 Ghz Quad Core Xeon running 64-bit code

Table: Scalar multiplication ($\mu$sec) on SS EC and TZV ($r = 3$) over $\mathbb{F}_{2^d}$ (32 bit)

| | | EC $d = 239$ | TZV $d = 79$ | EC $d = 307$ | TZV $d = 103$ | EC $d = 457$ | TZV $d = 157$ |
|---|---|---|---|---|---|---|---|
| operation | 2 | 0.193 | 0.228 | 0.309 | 0.286 | 1.140 | 0.555 |
| | $+$ | 8.013 | 3.341 | 12.515 | 4.709 | 27.510 | 8.254 |
| | $\sigma$ | – | 0.029 | – | 0.033 | – | 0.039 |
| full scalar | bin | 913.0 | 320.5 | 1720.4 | 509.1 | 6865.5 | 924.3 |
| | NAF | 673.9 | 235.0 | 1218.6 | 363.6 | 4884.6 | 588.2 |
| | $w$-NAF | 391.3 | 149.5 | 716.8 | 218.1 | 2871.9 | 420.1 |
| split scalar | bin | – | 277.8 | – | 363.6 | – | 756.3 |
| | NAF | – | 213.7 | – | 290.9 | – | 672.2 |
| | JSF | – | 213.7 | – | 254.5 | – | 504.2 |
| | $w$-NAF | – | 170.9 | – | 218.1 | – | 420.1 |

2 Ghz Quad Core Xeon running 32-bit code

Table: Scalar multiplication ($\mu$sec) on SS EC and TZV ($r = 3$) over $\mathbb{F}_{2^d}$ (64 bit)

| | | EC $d = 239$ | TZV $d = 79$ | EC $d = 307$ | TZV $d = 103$ | EC $d = 457$ | TZV $d = 157$ |
|---|---|---|---|---|---|---|---|
| operation | 2 | 0.167 | 0.236 | 0.259 | 0.280 | 0.341 | 0.421 |
| | $+$ | 4.143 | 2.619 | 6.258 | 3.273 | 14.590 | 5.832 |
| | $\sigma$ | – | 0.019 | – | 0.019 | – | 0.029 |
| full scalar | bin | 522.1 | 226.8 | 1062.8 | 396.5 | 3747.0 | 734.7 |
| | NAF | 369.8 | 164.7 | 725.3 | 280.1 | 2442.3 | 532.0 |
| | $w$-NAF | 232.8 | 117.6 | 445.2 | 185.5 | 1408.1 | 346.2 |
| split scalar | bin | – | 164.7 | – | 283.7 | – | 532.0 |
| | NAF | – | 128.3 | – | 225.5 | – | 422.2 |
| | JSF | – | 119.8 | – | 203.7 | – | 388.5 |
| | $w$-NAF | – | 102.7 | – | 170.9 | – | 312.5 |

2 Ghz Quad Core Xeon running 64-bit code

# Outline

# Security of Trace Zero Varieties

We now turn out attention to the security of TZV.

The case $g = 1, r = 3$ is simple: Dimension of TZV is 2. An attacker could hope for it to be contained in the Jacobian of a curve of genus at least 2.

This means that the best known attack has complexity not worse than $O(q)$ and this meets the square-root bound.

# The Fourth Dimension

Let us consider now the cases $g = 1, r = 5$ and $g = 2, r = 3$, i.e. 4-dimensional TZVs.

By the arguments by Lange; A. & Lange; Diem & Scholten; and under the (pessimistic) assumption that these results can also be adapted to fields of characteristic 2, we cannot exclude that G is contained in the Jacobian of a hyperelliptic curve of genus 6.

In this case attacks have complexity $\tilde{O}(q^{5/3})$, which means that security is reduced by at most one sixth of the bit length (using double LP IC).

# The Fourth Dimension

Let us consider now the cases $g = 1, r = 5$ and $g = 2, r = 3$, i.e. 4-dimensional TZVs.

Even worse, the group $G$ may be also contained in the Jacobian of a non-hyperelliptic curve (via Smith's attack), the worst case being a $C_{3,5}$ curve. In this case one may want to apply Diem's attack on planar curves, that has a complexity $\tilde{O}(q^{4/3})$ (with probability $O(1/q^2)$ of actually finding a planar model).

Hence, for the DLP in the TZV the security is in fact reduced, but for pairing applications we are using larger groups anyway.

# The Fourth Dimension

Let us consider now the cases $g = 1, r = 5$ and $g = 2, r = 3$, i.e. 4-dimensional TZVs.

Assume the $\tilde{O}(q^{4/3})$ attacks is feasible.

Let us consider a (SS) TZV $\mathcal{E}_5(\mathbb{F}_{2^{89}})$. Pairing lands in a 1780-bit field, offers about 160 bit EC security (broken in $O(2^{80})$ steps remember Coppersmith's attack has complexity $L_q[\frac{1}{3}, 1.4]$).

The order of $\mathcal{E}_5(\mathbb{F}_{2^{89}})$ has 356 bits (can use a small subgroup). The attack has complexity $\tilde{O}\big((2^{89})^{4/3}\big) \approx \tilde{O}(2^{118})$.

The complexity of attack is still more than complexity of attacking the system in the finite field in which the pairing takes its values.

Other cases similar.

# The Fourth Dimension

Let us consider now the cases $g = 1, r = 5$ and $g = 2, r = 3$, i.e. 4-dimensional TZVs.

Pairing on $\mathcal{E}_5(\mathbb{F}_{3^{43}})$ lands in a 2044.6-bit field ($\mathbb{F}_{3^{1290}}$). Field not binary, hence we have 233 bit EC security (broken in $O(2^{116.6})$ steps - attack has complexity $L_q[\frac{1}{3}, 1.92]$).

The group is a 272 bit group. There is no 2-torsion (hence no Ben Smith). The best attack we can hope to mount is Diem's attack with complexity $\tilde{O}(q^{8/5})$, hence $\tilde{O}(2^{109})$. Note that there is a $\log q$ in the $\tilde{O}$ so we in fact have at least $O(2^{114})$.

# Outline

# Conclusions

Trace Zero Varieties offer good performance:

- Scalar multiplication, ordinary curves
  - Binary TZV from EC 2 to 4 times faster than comparable EC.
  - Binary TZV from HEC smaller gain over HEC ($g = 2$).
  - Varieties over prime fields: performance in same ballpark as corresponding curves, often faster.
- Pairing on (binary) supersingular curves:
  - Serial versions, comparable performance to curves.
  - New algorithm up to 30% faster in 32-bit code.
  - **But** in these cases scalar multiplication *much* faster than on the curves (up to 7 times faster).

On the algorithm for the Tate pairing with supersingular TZV:

- Easy parallelization with $r$ processors, Miller loop on $q$

Thank you for your attention! Any questions?