# Isogenies in SageMath: Past, Present, Future

Lorenz Panny

Academia Sinica, Taipei, Taiwan

Leuven Isogeny Days, Leuven, 22 September 2022

# Part 0: *The Dream*

```
sage: E0 = EllipticCurve(...)
sage: R = E0.endomorphism_ring()
sage: IA = R.random_ideal()
sage: secret_key = IA
sage: EA = IA.isogeny_codomain()
sage: public_key = EA
```

# Part 0: *The Dream*

```
sage: E0 = EllipticCurve(...)
sage: R = E0.endomorphism_ring()
sage: IA = R.random_ideal()
sage: secret_key = IA
sage: EA = IA.isogeny_codomain()
sage: public_key = EA
```

```
sage: psi = E0.isogeny(E0.random_point())
sage: E1 = psi.codomain()
sage: commitment = E1
```

# Part 0: *The Dream*

```
sage: E0 = EllipticCurve(...)
sage: R = E0.endomorphism_ring()
sage: IA = R.random_ideal()
sage: secret_key = IA
sage: EA = IA.isogeny_codomain()
sage: public_key = EA
```

```
sage: psi = E0.isogeny(E0.random_point())
sage: E1 = psi.codomain()
sage: commitment = E1
```

```
sage: phi = E1.isogeny(E1.random_point())
sage: challenge = phi
```

# Part 0: *The Dream*

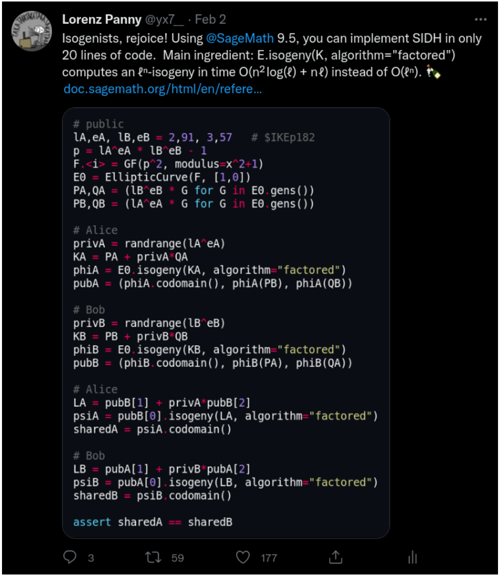
```
sage: E0 = EllipticCurve(...)
sage: R = E0.endomorphism_ring()
sage: IA = R.random_ideal()
sage: secret_key = IA
sage: EA = IA.isogeny_codomain()
sage: public_key = EA
```

```
sage: psi = E0.isogeny(E0.random_point())
sage: E1 = psi.codomain()
sage: commitment = E1
```

```
sage: phi = E1.isogeny(E1.random_point())
sage: challenge = phi
```

```
sage: I1 = R.ideal_from_isogeny(psi)
sage: S = EA.endomorphism_ring(IA)
sage: I2 = S.ideal_from_isogeny(phi)
sage: I = I2 * I1 * IA.conjugate()
sage: J = I.equivalent_smooth_ideal()
sage: response = J.isogeny()
```

# The rude awakening



Part 1: *Past*

Part 2: *Present*

Part 3: *Future*

# The past

Situation $\approx 2020$:

- `EllipticCurveIsogeny`: implementing Vélu and Kohel.
  (Space and time requirement both linear the degree.)
- `WeierstrassIsomorphism`: implementing isomorphisms.
  (Effectively a tuple $(u, r, s, t)$ with some helper methods.)

# The past

Situation $\approx 2020$:

- `EllipticCurveIsogeny`: implementing Vélu and Kohel.
  (Space and time requirement both linear the degree.)
- `WeierstrassIsomorphism`: implementing isomorphisms.
  (Effectively a tuple $(u, r, s, t)$ with some helper methods.)

However:

- Some things exponentially slower than they need to be.

# The past

Situation $\approx 2020$:

- `EllipticCurveIsogeny`: implementing Vélu and Kohel.
  (Space and time requirement both linear the degree.)
- `WeierstrassIsomorphism`: implementing isomorphisms.
  (Effectively a tuple $(u, r, s, t)$ with some helper methods.)

However:

- Some things exponentially slower than they need to be.
- Almost no non-trivial operations on isogenies ($\circ$, $+$, ...).

# The past

Situation $\approx$ 2020:

- `EllipticCurveIsogeny`: implementing Vélu and Kohel.
  (Space and time requirement both linear the degree.)
- `WeierstrassIsomorphism`: implementing isomorphisms.
  (Effectively a tuple $(u, r, s, t)$ with some helper methods.)

However:

- Some things exponentially slower than they need to be.
- Almost no non-trivial operations on isogenies ($\circ$, $+$, ...).
- No unified interface for isogenies & isomorphisms.

# The past

Situation $\approx 2020$:

- `EllipticCurveIsogeny`: implementing Vélu and Kohel.
  (Space and time requirement both linear the degree.)
- `WeierstrassIsomorphism`: implementing isomorphisms.
  (Effectively a tuple $(u, r, s, t)$ with some helper methods.)

However:

- Some things exponentially slower than they need to be.
- Almost no non-trivial operations on isogenies ($\circ$, $+$, ...).
- No unified interface for isogenies & isomorphisms.
- No tools for endomorphisms; they are just self-isogenies.

# The past

Situation $\approx 2020$:

- `EllipticCurveIsogeny`: implementing Vélu and Kohel.
  (Space and time requirement both linear the degree.)
- `WeierstrassIsomorphism`: implementing isomorphisms.
  (Effectively a tuple $(u, r, s, t)$ with some helper methods.)

However:

- Some things exponentially slower than they need to be.
- Almost no non-trivial operations on isogenies ($\circ$, $+$, ...).
- No unified interface for isogenies & isomorphisms.
- No tools for endomorphisms; they are just self-isogenies.
- Fair share of bugs!

## The past: Some things exponentially slower than they need to be.

- For $E(\mathbb{F}_q) = \mathbb{Z}/n \times \mathbb{Z}/m$ with $m \mid n$, find a basis $(P, Q)$.

# The past: Some things exponentially slower than they need to be.

- For $E(\mathbb{F}_q) = \mathbb{Z}/n \times \mathbb{Z}/m$ with $m \mid n$, find a basis $(P, Q)$.
  Old ad-hoc algorithm becomes slow when $m$ is big.
  New algorithm is slow only for $E(\mathbb{F}_q)[\ell^\infty] = \mathbb{Z}/\ell^r \times \mathbb{Z}/\ell^s$ with $r > s > 0$.

# The past: Some things exponentially slower than they need to be.

- For $E(\mathbb{F}_q) = \mathbb{Z}/n \times \mathbb{Z}/m$ with $m \mid n$, find a basis $(P, Q)$.
  Old ad-hoc algorithm becomes slow when $m$ is big.
  New algorithm is slow only for $E(\mathbb{F}_q)[\ell^\infty] = \mathbb{Z}/\ell^r \times \mathbb{Z}/\ell^s$ with $r > s > 0$.
- Given $R \in E(\mathbb{F}_q)$, find $(a, b) \in \mathbb{Z}^2$ with $R = [a]P + [b]Q$.

## The past: Some things exponentially slower than they need to be.

- For $E(\mathbb{F}_q) = \mathbb{Z}/n \times \mathbb{Z}/m$ with $m \mid n$, find a basis $(P, Q)$.
  Old ad-hoc algorithm becomes slow when $m$ is big.
  New algorithm is slow only for $E(\mathbb{F}_q)[\ell^\infty] = \mathbb{Z}/\ell^r \times \mathbb{Z}/\ell^s$ with $r > s > 0$.

- Given $R \in E(\mathbb{F}_q)$, find $(a, b) \in \mathbb{Z}^2$ with $R = [a]P + [b]Q$.

```
def _discrete_log(self,x):
    ...
    # EVEN DUMBER IMPLEMENTATION!
    ...
    u = [y for y in self.list() if y.element() == x]
    if len(u) == 0: raise TypeError("Not in group")
    if len(u) > 1: raise NotImplementedError
    return u[0].vector()
```

# The past: Some things exponentially slower than they need to be.

- For $E(\mathbb{F}_q) = \mathbb{Z}/n \times \mathbb{Z}/m$ with $m \mid n$, find a basis $(P, Q)$.
  Old ad-hoc algorithm becomes slow when $m$ is big.
  New algorithm is slow only for $E(\mathbb{F}_q)[\ell^\infty] = \mathbb{Z}/\ell^r \times \mathbb{Z}/\ell^s$ with $r > s > 0$.
- Given $R \in E(\mathbb{F}_q)$, find $(a, b) \in \mathbb{Z}^2$ with $R = [a]P + [b]Q$.

```
def _discrete_log(self,x):
    ...
    # EVEN DUMBER IMPLEMENTATION!
    ...
    u = [y for y in self.list() if y.element() == x]
    if len(u) == 0: raise TypeError("Not in group")
    if len(u) > 1: raise NotImplementedError
    return u[0].vector()
```

Sage $\geq$ 9.6:

```
sage: a,b = E.abelian_group().discrete_log(R)
```

# The past: Almost no non-trivial operations on isogenies ($\circ$, $+$, ...).

Composing two `EllipticCurveIsogeny` objects "works",
but is not overly useful:

```
sage: phi = phi2 * phi1
sage: type(phi)
<class 'sage.categories.map.FormalCompositeMap'>
sage: phi.degree()
AttributeError: ...
sage: phi.rational_maps()
AttributeError: ...
```

## The past: Almost no non-trivial operations on isogenies ($\circ$, $+$, ...).

Composing two `EllipticCurveIsogeny` objects "works", but is not overly useful:

```
sage: phi = phi2 * phi1
sage: type(phi)
<class 'sage.categories.map.FormalCompositeMap'>
sage: phi.degree()
AttributeError: ...
sage: phi.rational_maps()
AttributeError: ...
```

Addition of isogenies is not implemented at all:

```
sage: phi + phi
TypeError:
    unsupported operand parent(s) for +: 'Set of morphisms ...'
```

# The past: No unified interface for isogenies & isomorphisms.

Every isomorphism *is* an isogeny. However:

# The past: No unified interface for isogenies & isomorphisms.

Every isomorphism *is* an isogeny. However:

- Composing with isomorphisms rather awkward.

```
sage: tau = E.automorphisms()[1]
sage: phi = E.isogeny(...)
sage: psi = phi * tau                          # nope
TypeError: self (=Isogeny of degree ...) domain
    must equal right (=Generic endomorphism of Abelian group
    of points on Elliptic Curve defined by ...) codomain
sage: phi.set_pre_isomorphism(tau)  # okay; in-place
```

# The past: No unified interface for isogenies & isomorphisms.

Every isomorphism *is* an isogeny. However:

- Composing with isomorphisms rather awkward.

```
sage: tau = E.automorphisms()[1]
sage: phi = E.isogeny(...)
sage: psi = phi * tau                    # nope
TypeError: self (=Isogeny of degree ...) domain
    must equal right (=Generic endomorphism of Abelian group
    of points on Elliptic Curve defined by ...) codomain
sage: phi.set_pre_isomorphism(tau)   # okay; in-place
```

- Almost all of the usual isogeny methods missing: `.degree()`, `.rational_maps()`, `.formal()`, ...

## The past: No tools for endomorphisms; are just self-isogenies.

- `EllipticCurve_finite_field` has `.frobenius()`, but it just returns an element of a quadratic field:

```
sage: E = EllipticCurve(GF(101), [5,5])
sage: E.frobenius().parent()
Order in Number Field in phi
        with defining polynomial x^2 + 17*x + 101
```

## The past: No tools for endomorphisms; are just self-isogenies.

- `EllipticCurve_finite_field` has `.frobenius()`, but it just returns an element of a quadratic field:

```
sage: E = EllipticCurve(GF(101), [5,5])
sage: E.frobenius().parent()
Order in Number Field in phi
        with defining polynomial x^2 + 17*x + 101
```

Endomorphisms can of course be represented as just another `EllipticCurveIsogeny`, but it doesn't do much work for you:

## The past: No tools for endomorphisms; are just self-isogenies.

- `EllipticCurve_finite_field` has `.frobenius()`, but it just returns an element of a quadratic field:

```
sage: E = EllipticCurve(GF(101), [5,5])
sage: E.frobenius().parent()
Order in Number Field in phi
        with defining polynomial x^2 + 17*x + 101
```

Endomorphisms can of course be represented as just another `EllipticCurveIsogeny`, but it doesn't do much work for you:

- No useful composition, and no addition at all.

# The past: No tools for endomorphisms; are just self-isogenies.

- `EllipticCurve_finite_field` has `.frobenius()`, but it just returns an element of a quadratic field:

```
sage: E = EllipticCurve(GF(101), [5,5])
sage: E.frobenius().parent()
Order in Number Field in phi
        with defining polynomial x^2 + 17*x + 101
```

Endomorphisms can of course be represented as just another `EllipticCurveIsogeny`, but it doesn't do much work for you:

- No useful composition, and no addition at all.
- No compact representation; need to hand-craft each time.
  (Think things like formal linear combinations of morphisms.)

## The past: No tools for endomorphisms; are just self-isogenies.

- `EllipticCurve_finite_field` has `.frobenius()`, but it just returns an element of a quadratic field:

```
sage: E = EllipticCurve(GF(101), [5,5])
sage: E.frobenius().parent()
Order in Number Field in phi
        with defining polynomial x^2 + 17*x + 101
```

Endomorphisms can of course be represented as just another `EllipticCurveIsogeny`, but it doesn't do much work for you:

- No useful composition, and no addition at all.
- No compact representation; need to hand-craft each time.
  (Think things like formal linear combinations of morphisms.)
- Inseparable isogenies are actually irrepresentable.
  (This includes the Frobenius endomorphism in the supersingular case!)

## The past: No tools for endomorphisms; are just self-isogenies.

- `EllipticCurve_finite_field` has `.frobenius()`, but it just returns an element of a quadratic field:

```
sage: E = EllipticCurve(GF(101), [5,5])
sage: E.frobenius().parent()
Order in Number Field in phi
        with defining polynomial x^2 + 17*x + 101
```

Endomorphisms can of course be represented as just another `EllipticCurveIsogeny`, but it doesn't do much work for you:

- No useful composition, and no addition at all.
- No compact representation; need to hand-craft each time.
  (Think things like formal linear combinations of morphisms.)
- Inseparable isogenies are actually irrepresentable.
  (This includes the Frobenius endomorphism in the supersingular case!)
- No algorithm for traces or degrees, or anything else.
  Crucial tool for computing the structure of an endomorphism (sub)ring!

Part 1: *Past*

## Part 2: *Present*

Part 3: *Future*

# The present: Unified parent class for isogenies

- Sage $\geq 9.5$: Common class `EllipticCurveHom` for `EllipticCurveIsogeny`, `WeierstrassIsomorphism`, and other (new) types of elliptic-curve morphisms.

# The present: Unified parent class for isogenies

- Sage $\geq 9.5$: Common class `EllipticCurveHom` for `EllipticCurveIsogeny`, `WeierstrassIsomorphism`, and other (new) types of elliptic-curve morphisms.

- Goal: All isogenies should behave the same from an user's perspective regardless of internal representation.
  "API contract" says these objects support evaluation, composition, `.degree()`, `.rational_maps()`, `.kernel_polynomial()`, ...

# The present: Unified parent class for isogenies

- Sage $\geq 9.5$: Common class `EllipticCurveHom` for `EllipticCurveIsogeny`, `WeierstrassIsomorphism`, and other (new) types of elliptic-curve morphisms.

- <u>Goal:</u> All isogenies should behave the same from an user's perspective regardless of internal representation.
  "API contract" says these objects support evaluation, composition, `.degree()`, `.rational_maps()`, `.kernel_polynomial()`, ...

- Compose any two isogenies using the `*` operator.
  !! This is currently opt-in for some type combinations. Use `EllipticCurveHom_composite.make_default()`. Soon™ default.

# The present: Composing and decomposing isogenies

Computing a smooth isogeny efficiently (Sage $\geq 9.5$):

```
E.isogeny(K, algorithm='factored')
```

# The present: Composing and decomposing isogenies

Computing a smooth isogeny efficiently (Sage $\geq 9.5$):

```
E.isogeny(K, algorithm='factored')
```

- This takes time and space polylogarithmic in the degree.

# The present: Composing and decomposing isogenies

Computing a smooth isogeny efficiently (Sage $\geq 9.5$):

```
E.isogeny(K, algorithm='factored')
```

- This takes time and space polylogarithmic in the degree.
- Currently uses a naïve quadratic strategy.

# The present: Composing and decomposing isogenies

Computing a smooth isogeny efficiently (Sage $\geq 9.5$):

```
E.isogeny(K, algorithm='factored')
```

- This takes time and space polylogarithmic in the degree.
- Currently uses a naïve quadratic strategy.
- Patch for quasilinear strategy is ready, but stuck.

# The present: Scalar and inseparable isogenies

- Vélu/$\sqrt{\text{élu}}$ are fully general, complexity is not great.

# The present: Scalar and inseparable isogenies

- Vélu/$\sqrt{\text{élu}}$ are fully general, complexity is not great.
- Composite smooth-degree isogenies do much better.

# The present: Scalar and inseparable isogenies

- Vélu/$\sqrt{\text{élu}}$ are fully general, complexity is not great.
- Composite smooth-degree isogenies do much better.
- Luckily, there are more examples of compact isogenies with time and space complexity polynomial in $\log(\deg)$:

# The present: Scalar and inseparable isogenies

- Vélu/$\sqrt{\text{élu}}$ are fully general, complexity is not great.
- Composite smooth-degree isogenies do much better.
- Luckily, there are more examples of compact isogenies with time and space complexity polynomial in $\log(\deg)$:
  - We can represent $[m]\colon\ E \to E$ simply as a tuple $(E, m)$, enriched with a type tag and some simple helper methods.
    (The "type tag" is implicitly applied by Python when you define a class.)
    (Example: The implementation of `.degree()` is just `return` $m^2$.)

# The present: Scalar and inseparable isogenies

- Vélu/$\sqrt{\text{élu}}$ are fully general, complexity is not great.
- Composite smooth-degree isogenies do much better.
- Luckily, there are more examples of compact isogenies with time and space complexity polynomial in $\log(\deg)$:
  - We can represent $[m]\colon E \to E$ simply as a tuple $(E, m)$, enriched with a type tag and some simple helper methods.
    (The "type tag" is implicitly applied by Python when you define a class.)
    (Example: The implementation of `.degree()` is just `return` $m^2$.)
  - Similarly, we can represent $\pi_r\colon E \to E^{(p^k)}$, $(x, y) \mapsto (x^{p^k}, y^{p^k})$ simply as $(E, k)$, enriched in the same way.
    (Example: The implementation of `.degree()` is just `return` $p^k$.)

# The present: Scalar and inseparable isogenies

- Vélu/$\sqrt{\text{élu}}$ are fully general, complexity is not great.
- Composite smooth-degree isogenies do much better.
- Luckily, there are more examples of compact isogenies with time and space complexity polynomial in $\log(\deg)$:
  - We can represent $[m]\colon E \to E$ simply as a tuple $(E, m)$, enriched with a type tag and some simple helper methods.
    (The "type tag" is implicitly applied by Python when you define a class.)
    (Example: The implementation of `.degree()` is just `return` $m^2$.)
  - Similarly, we can represent $\pi_r\colon E \to E^{(p^k)}$, $(x, y) \mapsto (x^{p^k}, y^{p^k})$ simply as $(E, k)$, enriched in the same way.
    (Example: The implementation of `.degree()` is just `return` $p^k$.)

These things are implemented, but stuck.

# The present: Comparing isogenies efficiently

- Comparing isogenies is fundamental to many algorithms.

# The present: Comparing isogenies efficiently

- Comparing isogenies is fundamental to many algorithms.
- Now have many ways of representing the same isogeny.

# The present: Comparing isogenies efficiently

- Comparing isogenies is fundamental to many algorithms.
- Now have many ways of representing the same isogeny.
- Traditional Sage method: Compare domain, codomain, and `.rational_maps()`. Complexity linear in the degree; kills any speedup from compact or formal representation.

# The present: Comparing isogenies efficiently

- Comparing isogenies is fundamental to many algorithms.
- Now have many ways of representing the same isogeny.
- Traditional Sage method: Compare domain, codomain, and `.rational_maps()`. Complexity linear in the degree; kills any speedup from compact or formal representation.

(New??) polynomial-time method:

- Two isogenies $\varphi, \psi \colon E \to E'$ of equal degree $d$ which agree on $> 4d$ points must be identical.

# The present: Comparing isogenies efficiently

- Comparing isogenies is fundamental to many algorithms.
- Now have many ways of representing the same isogeny.
- Traditional Sage method: Compare domain, codomain, and `.rational_maps()`. Complexity linear in the degree; kills any speedup from compact or formal representation.

(New??) polynomial-time method:

- Two isogenies $\varphi, \psi \colon E \to E'$ of equal degree $d$ which agree on $> 4d$ points must be identical.
- Simply evaluate on generators of large enough subgroup.
(May require taking an extension of degree $O(\log d)$.)

# The present: Comparing isogenies efficiently

- Comparing isogenies is fundamental to many algorithms.
- Now have many ways of representing the same isogeny.
- Traditional Sage method: Compare domain, codomain, and `.rational_maps()`. Complexity linear in the degree; kills any speedup from compact or formal representation.

(New??) polynomial-time method:

- Two isogenies $\varphi, \psi\colon E \to E'$ of equal degree $d$ which agree on $> 4d$ points must be identical.
- Simply evaluate on generators of large enough subgroup. (May require taking an extension of degree $O(\log d)$.)
- This is essentially a version of *polynomial identity testing*, optimized for maps defining a group homomorphism.

# The present: $\sqrt{\text{élu}}$

Sage $\geq 9.7$ (released *eergisteren*!):

```
sage: l = 10000019
sage: p = 40*l - 1
sage: E = EllipticCurve(GF(p), [1,0])
sage: P = (p+1)//l * E.gens()[0]
sage: E.isogeny(P, algorithm='velusqrt')
Elliptic-curve isogeny (using √élu) of degree 10000019:
  From: Elliptic Curve defined by y^2 = x^3 + x
          over Finite Field of size 400000759
  To:   Elliptic Curve defined by
          y^2 = x^3 + 88879239*x + 195338414
          over Finite Field of size 400000759
sage: %timeit E.isogeny(P, algorithm='velusqrt')
4.11 s ± 72.9 ms per loop (...)
```

# The present: $\sqrt{\text{élu}}$

Sage $\geq 9.7$ (released *eergisteren*!):

```
sage: l = 10000019
sage: p = 40*l - 1
sage: E = EllipticCurve(GF(p), [1,0])
sage: P = (p+1)//l * E.gens()[0]
sage: E.isogeny(P, algorithm='velusqrt')
Elliptic-curve isogeny (using √élu) of degree 10000019:
  From: Elliptic Curve defined by y^2 = x^3 + x
            over Finite Field of size 400000759
  To:   Elliptic Curve defined by
            y^2 = x^3 + 88879239*x + 195338414
            over Finite Field of size 400000759
sage: %timeit E.isogeny(P, algorithm='velusqrt')
4.11 s ± 72.9 ms per loop (...)
```

- Vélu's formulas take about 8 minutes for the same isogeny.
- Speedup is even more significant as the degree grows.

Part 1: *Past*

Part 2: *Present*

Part 3: *Future*

# The future: Sums of isogenies

- Vélu/Kohel: Computing a sum of two isogenies is tricky.

# The future: Sums of isogenies

- Vélu/Kohel: Computing a sum of two isogenies is tricky.
- Best(?) solution: Store formally; "simplify" when needed.

# The future: Sums of isogenies

- Vélu/Kohel: Computing a sum of two isogenies is tricky.
- Best(?) solution: Store formally; "simplify" when needed.
- Computing the degree (and perhaps trace) is a Schoof-type algorithm; needs only evaluation. Polynomial-time.

# The future: Sums of isogenies

- Vélu/Kohel: Computing a sum of two isogenies is tricky.
- Best(?) solution: Store formally; "simplify" when needed.
- Computing the degree (and perhaps trace) is a Schoof-type algorithm; needs only evaluation. Polynomial-time.

**Q**: What does "simplify" mean? Expand everything? Group common summands? **How to apply the distributive law?**

# The future: Endomorphism (sub)rings! (1)

Good support for sums and compositions of isogenies
$\Longrightarrow$ Computing (with) endomorphism rings.

# The future: Endomorphism (sub)rings! (1)

Good support for sums and compositions of isogenies
$\implies$ Computing (with) endomorphism rings.

(Example: Embedding some explicitly given endomorphisms into a quaternion algebra essentially boils down to computing *trace pairings* of the form $\langle \varphi, \psi \rangle = \mathrm{tr}(\varphi \circ \widehat{\psi})$.)

# The future: Endomorphism (sub)rings! (1)

Good support for sums and compositions of isogenies
$\implies$ Computing (with) endomorphism rings.

(Example: Embedding some explicitly given endomorphisms into a quaternion algebra essentially boils down to computing *trace pairings* of the form $\langle \varphi, \psi \rangle = \mathrm{tr}(\varphi \circ \widehat{\psi})$.)

For usability/flexibility:

# The future: Endomorphism (sub)rings! (1)

Good support for sums and compositions of isogenies
$\Longrightarrow$ Computing (with) endomorphism rings.

(Example: Embedding some explicitly given endomorphisms into a quaternion algebra essentially boils down to computing *trace pairings* of the form $\langle\varphi, \psi\rangle = \operatorname{tr}(\varphi \circ \widehat{\psi})$.)

For usability/flexibility:

- Important to support endomorphism subrings for working with oriented supersingular curves (in particular: CSIDH), or when having only a non-full-index subring.

# The future: Endomorphism (sub)rings! (1)

Good support for sums and compositions of isogenies
$\Longrightarrow$ Computing (with) endomorphism rings.

(Example: Embedding some explicitly given endomorphisms into a quaternion algebra essentially boils down to computing *trace pairings* of the form $\langle \varphi, \psi \rangle = \mathrm{tr}(\varphi \circ \widehat{\psi})$.)

For usability/flexibility:

- Important to support endomorphism subrings for working with oriented supersingular curves (in particular: CSIDH), or when having only a non-full-index subring.
- Important to render existing algebraic tools for orders in {quadratic fields, quaternion algebras} easily applicable to endomorphism rings.

# The future: Endomorphism (sub)rings! (2)

Suggested construction:

Define a class `EllipticCurveEndRing` whose **data** are:

# The future: Endomorphism (sub)rings! (2)

Suggested construction:

Define a class `EllipticCurveEndRing` whose **data** are:

- An abstract quadratic or quaternion order $\mathcal{O}$ which embeds into an endomorphism ring $\mathrm{End}(E)$.

# The future: Endomorphism (sub)rings! (2)

Suggested construction:

Define a class `EllipticCurveEndRing` whose **data** are:

- An abstract quadratic or quaternion order $\mathcal{O}$ which embeds into an endomorphism ring $\mathrm{End}(E)$.
- Actual endomorphisms of $E$ which satisfy the relations of $\mathcal{O}$ when substituted for the generators.

# The future: Endomorphism (sub)rings! (2)

Suggested construction:

Define a class `EllipticCurveEndRing` whose **data** are:

- An abstract quadratic or quaternion order $\mathcal{O}$ which embeds into an endomorphism ring $\mathrm{End}(E)$.
- Actual endomorphisms of $E$ which satisfy the relations of $\mathcal{O}$ when substituted for the generators.

Selling points:

- Design is flexible: It wraps arbitrary subrings of $\mathrm{End}(E)$.

# The future: Endomorphism (sub)rings! (2)

Suggested construction:

Define a class `EllipticCurveEndRing` whose **data** are:

- An abstract quadratic or quaternion order $\mathcal{O}$ which embeds into an endomorphism ring $\mathrm{End}(E)$.
- Actual endomorphisms of $E$ which satisfy the relations of $\mathcal{O}$ when substituted for the generators.

Selling points:

- Design is flexible: It wraps arbitrary subrings of $\mathrm{End}(E)$.
- The defining morphisms can be user-supplied; they might well be cryptographically-sized secrets.

# The future: Endomorphism (sub)rings! (2)

Suggested construction:

Define a class `EllipticCurveEndRing` whose **data** are:

- An abstract quadratic or quaternion order $\mathcal{O}$ which embeds into an endomorphism ring $\mathrm{End}(E)$.
- Actual endomorphisms of $E$ which satisfy the relations of $\mathcal{O}$ when substituted for the generators.

Selling points:

- Design is flexible: It wraps arbitrary subrings of $\mathrm{End}(E)$.
- The defining morphisms can be user-supplied; they might well be cryptographically-sized secrets.
- We can use the full power of Sage's ideal machinery on the abstract side, then map "down" to concrete isogenies.

# The future: Endomorphism (sub)rings! (2)

<u>Suggested construction</u>:

Define a class `EllipticCurveEndRing` whose **data** are:

- An abstract quadratic or quaternion order $\mathcal{O}$ which embeds into an endomorphism ring $\mathrm{End}(E)$.
- Actual endomorphisms of $E$ which satisfy the relations of $\mathcal{O}$ when substituted for the generators.

Selling points:

- Design is flexible: It wraps arbitrary subrings of $\mathrm{End}(E)$.
- The defining morphisms can be user-supplied; they might well be cryptographically-sized secrets.
- We can use the full power of Sage's ideal machinery on the abstract side, then map "down" to concrete isogenies.
- To figure out abstract version of a concrete morphism, compute trace pairings with the defining morphisms.

# The future: Ideal-to-isogeny and back

With the abstract-and-concrete endomorphism ring structure:

# The future: Ideal-to-isogeny and back

With the abstract-and-concrete endomorphism ring structure:

- Easy to compute ideal-to-isogeny, subsuming both Deuring correspondence and CM action in a unified way.

# The future: Ideal-to-isogeny and back

With the abstract-and-concrete endomorphism ring structure:

- Easy to compute ideal-to-isogeny, subsuming both Deuring correspondence and CM action in a unified way.
- Easy to compute isogeny-to-ideal.

# The future: Ideal-to-isogeny and back

With the abstract-and-concrete endomorphism ring structure:

- Easy to compute ideal-to-isogeny, subsuming both Deuring correspondence and CM action in a unified way.
- Easy to compute isogeny-to-ideal.
- Smoothing ideals easy using KLPT or index calculus.
  (Somewhat optimized implementation of Deuring correspondence is work in progress with the *friends of quaternions*!)

# The future: Ideal-to-isogeny and back

With the abstract-and-concrete endomorphism ring structure:

- Easy to compute ideal-to-isogeny, subsuming both Deuring correspondence and CM action in a unified way.
- Easy to compute isogeny-to-ideal.
- Smoothing ideals easy using KLPT or index calculus.
  (Somewhat optimized implementation of Deuring correspondence is work in progress with the *friends of quaternions*!)
- Quadratic case requires ideals of non-maximal quadratic orders. Work in progress.

# The future: Computing endomorphisms

Algorithms for endomorphism rings come in two flavours:

# The future: Computing endomorphisms

Algorithms for endomorphism rings come in two flavours:

- Quaternionic case: See Benjamin's talk yesterday.
  Basic algorithm: Brute-force random cycles in some isogeny graph; compute relations (trace pairings); repeat until full-rank and full-index. This is exponential-time.

# The future: Computing endomorphisms

Algorithms for endomorphism rings come in two flavours:

- Quaternionic case: See Benjamin's talk yesterday.
  Basic algorithm: Brute-force random cycles in some isogeny graph; compute relations (trace pairings); repeat until full-rank and full-index.
  This is exponential-time.
- Quadratic case: Isogeny volcano walking.
  Beautiful theory; see for instance Sutherland's "Isogeny Volcanoes".
  This is efficient in many cases! [Preliminary implementation exists.]

# The future

# The future

Oh, and also: **Genus 2**, for reasons. LOL

Part 1: *Past*

Part 2: *Present*

Part 3: *Future*

- Most of the isogeny code is "just" Python. Despair not.
  $\Longrightarrow$ Coding *for* Sage is vastly the same as coding *in* Sage.

- Most of the isogeny code is “just” Python. Despair not.

  $\Longrightarrow$ Coding *for* Sage is vastly the same as coding *in* Sage.
- Some patches stuck for months for lack of reviewers.

- Most of the isogeny code is “just” Python. Despair not.
  $\Longrightarrow$ Coding *for* Sage is vastly the same as coding *in* Sage.
- Some patches stuck for months for lack of reviewers.
- Reporting bugs or missing functionality is useful too!

- Most of the isogeny code is “just” Python. Despair not.
  $\Longrightarrow$ Coding *for* Sage is vastly the same as coding *in* Sage.
- Some patches stuck for months for lack of reviewers.
- Reporting bugs or missing functionality is useful too!

You can help!

# Bonus slide: Life hacks

- Turn off *provable* primality testing:

```
sage: proof.all(False)
```

# Bonus slide: Life hacks

- Turn off *provable* primality testing:

```
sage: proof.all(False)
```

- Indicate that you're fine with non-Conway finite fields:

```
sage: F = GF(q,'t')    # can be much faster than GF(q)
sage: F.<t> = GF(q)    # alternative syntax
```

# Bonus slide: Life hacks

- Turn off *provable* primality testing:

```
sage: proof.all(False)
```

- Indicate that you're fine with non-Conway finite fields:

```
sage: F = GF(q,'t')    # can be much faster than GF(q)
sage: F.<t> = GF(q)    # alternative syntax
```

- Make sure to run the most recent version: Lots of speed improvements for elliptic curves and isogenies.