Verifiable Delay Functions from Supersingular Isogenies and Pairings

Luca De Feo

IBM Research Zürich

based on joint work with J. Burdges (@jeffburdges), S. Masson (@SimonMasson2), C. Petit, A. Sanso (@asanso)

November 12, 2020, CV Labs

Computational hardness in cryptography

boring picture of Alice, Bob and Eve goes here

How long will it take Eve to decrypt the message?

Complexity theory: (sub)exponentially more than Bob.

- Asymptotics don't say anything on constants.
- Based on an average-case analysis, ignores worst case.
- Typically based on a Turing-machine or RAM-like model, doesn't necessarily fit reality.

Real world crypto: at least 2^{128} "operations".

- But what's an "operation"?
- Often based on extrapolations (see, in particular, factoring).
- Doesn't account for parallelism.
- More a measure of cost than a measure of time.

Time-lock Puzzles

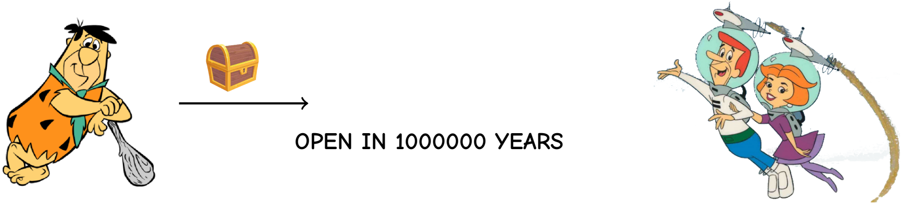
Basically a Key Derivation Function (family) with two algorithms:

KDF(T, Δ): which computes a key k given a trapdoor T and a delay Δ.

SlowKDF(Δ): which computes the same key k without knowledge of the trapdoor, in time approximately $\Delta \cdot \mathtt{constant}$.

…under the conjecture that no algorithm faster than SlowKDF can compute k from Δ with non-negligible probability.

Time-lock Puzzles



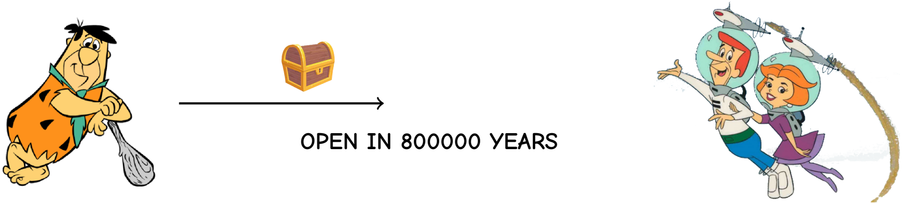
Basically a Key Derivation Function (family) with two algorithms:

KDF(T, Δ): which computes a key k given a trapdoor T and a delay Δ.

SlowKDF(Δ): which computes the same key k without knowledge of the trapdoor, in time approximately $\Delta \cdot \mathtt{constant}$.

…under the conjecture that no algorithm faster than SlowKDF can compute k from Δ with non-negligible probability.

Time-lock Puzzles



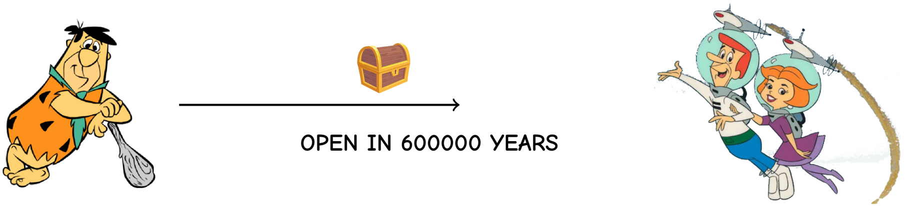
Basically a Key Derivation Function (family) with two algorithms:

KDF(T, Δ): which computes a key k given a trapdoor T and a delay Δ.

SlowKDF(Δ): which computes the same key k without knowledge of the trapdoor, in time approximately $\Delta \cdot \mathtt{constant}$.

…under the conjecture that no algorithm faster than SlowKDF can compute k from Δ with non-negligible probability.

Time-lock Puzzles



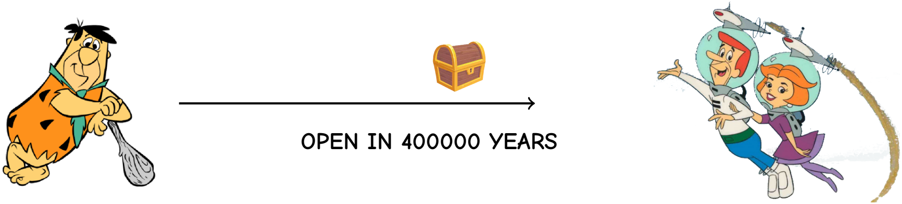
Basically a Key Derivation Function (family) with two algorithms:

KDF(T, Δ): which computes a key k given a trapdoor T and a delay Δ.

SlowKDF(Δ): which computes the same key k without knowledge of the trapdoor, in time approximately $\Delta \cdot \mathtt{constant}$.

…under the conjecture that no algorithm faster than SlowKDF can compute k from Δ with non-negligible probability.

Time-lock Puzzles



Basically a Key Derivation Function (family) with two algorithms:

KDF(T, Δ): which computes a key k given a trapdoor T and a delay Δ.

SlowKDF(Δ): which computes the same key k without knowledge of the trapdoor, in time approximately $\Delta \cdot \text{constant}$.

…under the conjecture that no algorithm faster than SlowKDF can compute k from Δ with non-negligible probability.

Time-lock Puzzles

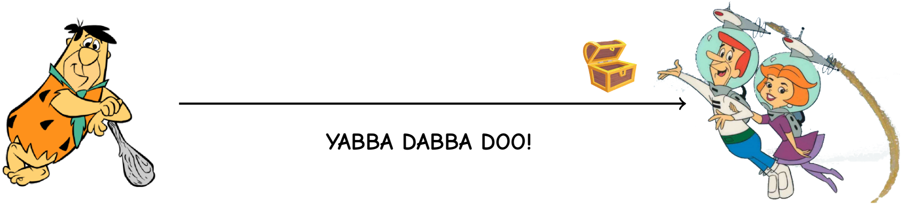
Basically a Key Derivation Function (family) with two algorithms:

KDF(T, Δ): which computes a key k given a trapdoor T and a delay Δ.

SlowKDF(Δ): which computes the same key k without knowledge of the trapdoor, in time approximately $\Delta \cdot \mathtt{constant}$.

…under the conjecture that no algorithm faster than SlowKDF can compute k from Δ with non-negligible probability.

Time-lock Puzzles



Basically a Key Derivation Function (family) with two algorithms:

KDF(T, Δ): which computes a key k given a trapdoor T and a delay Δ.

SlowKDF(Δ): which computes the same key k without knowledge of the trapdoor, in time approximately $\Delta \cdot \mathtt{constant}$.

…under the conjecture that no algorithm faster than SlowKDF can compute k from Δ with non-negligible probability.

Some applications

Sealed bid auctions

Standard solution based on encryption:

- Each bidder encrypts its bid;
- At the end of the auction each bidder reveals the key.

Problem: some bidders may refuse to reveal the key.

Especially important in Vickrey auctions (winner pays second highest bid).

Some applications

Sealed bid auctions

Standard solution based on encryption:

- Each bidder encrypts its bid;
- At the end of the auction each bidder reveals the key.

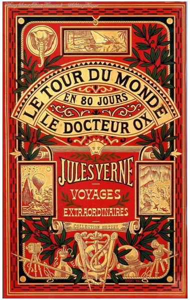
Problem: some bidders may refuse to reveal the key.
Especially important in Vickrey auctions (winner pays second highest bid).

Solution:

- Each bidder encrypts bid with a TL puzzle;
- At the end of the auction each well behaved bidder reveals its trapdoor;
- Other bids are opened with SlowKDF. (can get quite expensive, though)

Other applications: Voting, key escrow, …

Verifiable Delay Functions (Boneh, Bonneau, Bünz, Fisch 2018)



A sort of *deterministic Proof of Sequential Work*

Function (family) $f : X \to Y$ s.t.:

- Evaluating $f(x)$ takes long time:
  - predictably long time,
  - on almost all random inputs x,
  - even after having seen many values $f(x')$,
  - even given massive number of processors;
- Verifying $y = f(x)$ is efficient:
  - ideally, exponential separation between evaluation and verification.

Example application: distributed lotteries

Participants **A, B, . . . , Z** want to agree on a random winning ticket.

Flawed protocol

- Each participant x broadcasts a random string s_x;
- Winning ticket is $H(s_A, \dots, s_Z)$.

Example application: distributed lotteries

Participants **A, B, . . . , Z** want to agree on a random winning ticket.

Flawed protocol

- Each participant x broadcasts a random string s_x;
- Winning ticket is $H(s_A, \ldots, s_Z)$.

Cheating participant **Z** waits to see all other strings, then brute-forces s_Z to win lottery.

Example application: distributed lotteries

Participants **A, B, . . . , Z** want to agree on a random winning ticket.

Flawed protocol

- Each participant x broadcasts a random string s_x;
- Winning ticket is $H(s_A, \ldots, s_Z)$.

Cheating participant **Z** waits to see all other strings, then brute-forces s_Z to win lottery.

Fixes

- Make the hash function **slooooooooooooooooooooooooooooow**;
  - e.g., participants have 10 minutes to submit s_x,
  - outcome will be known after 20 minutes.

Example application: distributed lotteries

Participants **A, B, . . . , Z** want to agree on a random winning ticket.

Flawed protocol

- Each participant x broadcasts a random string s_x;
- Winning ticket is $H(s_A, \dots, s_Z)$.

Cheating participant **Z** waits to see all other strings, then brute-forces s_Z to win lottery.

Fixes

- Make the hash function **sloooooooooooooooooooooooooooow**;
  - e.g., participants have 10 minutes to submit s_x,
  - outcome will be known after 20 minutes.
- Make it possible to verify $w = H(s_A, \dots, s_Z)$ **fast**.

More applications

Randomness beacons

Goal: Generate a public stream of provably random numbers.

Standard technique: Hash output of public high entropy sources (e.g.: stock market, weather, …) at regular intervals (epochs).

Risk: Close to the end of the epoch, adversary manipulates the data (e.g., buys stock) repeatedly until they get the desired alea.

Fix: Run the hashed value through a VDF with delay longer than the epoch.

Proofs of Stake/Space

Goal: Elect epoch leader(s) in PoS blockchains.

Standard technique: PoS are assigned a "quality" (e.g.: hash of the PoS), the higher quality gets elected as leader.

Disadvantage: Requires synchronization of miners.

Fix (Chia): Run the PoS through a VDF with delay proportional to quality.

VDF Craze

Who is investing in VDFs

VDF Alliance[1]: formed by Etherereum, Protocol Labs, Tezos, Interchain, Supranational.

VDF competitions (cash prizes in the order of 100k$):

- RSA-based, run by VDF Alliance[2].
  - Squaring modulo N,
  - Distributed generation of RSA numbers.
- Class group based, run by Chia[3].
  - Class number computation.
  - Squaring in class groups.

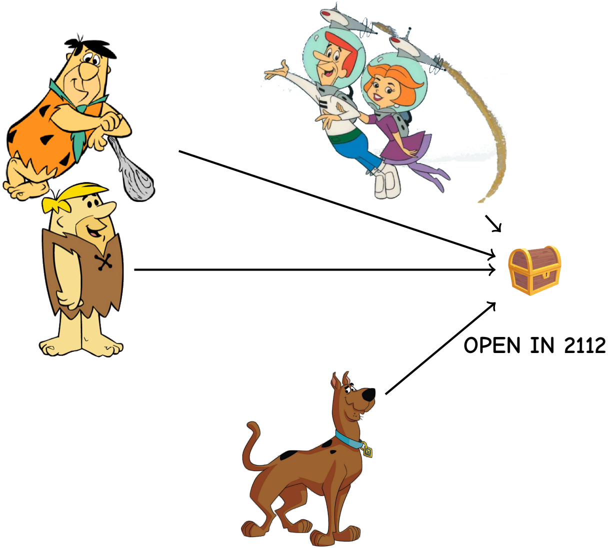
More resources: `https://vdfresearch.org/`.

[1] `https://www.vdfalliance.org/`

[2] `https://supranational.atlassian.net/wiki/spaces/VA/pages/36569208/FPGA+Competition`

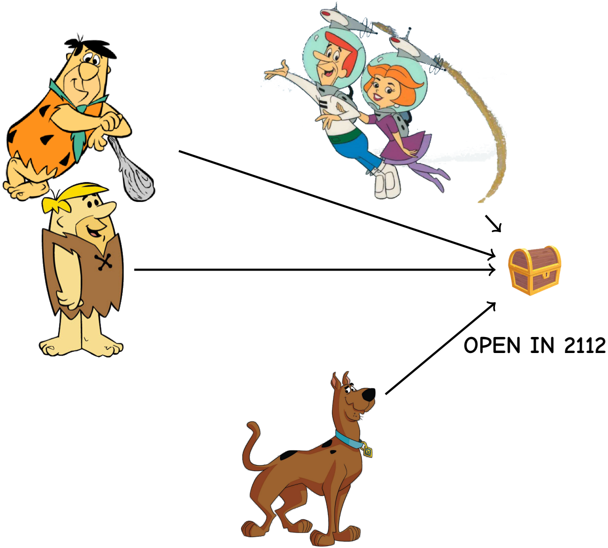
[3] `https://github.com/Chia-Network/vdf-competition/`

Delay Encryption (`https://ia.cr/2020/638`)



- Trapdoor-less time capsule.
- Delay Encryption $\Rightarrow$ Time-lock Puzzles.
- Delay Encryption $\Rightarrow$ VDF.
- Only known from isogenies.
- Applications: better auctions, voting, …

Delay Encryption (https://ia.cr/2020/638)



- Trapdoor-less time capsule.
- Delay Encryption $\Rightarrow$ Time-lock Puzzles.
- Delay Encryption $\Rightarrow$ VDF.
- Only known from isogenies.
- Applications: better auctions, voting, ...

Not this talk.

Group based Delay Functions

Rivest-Shamir-Wagner TL Puzzle ('96)

Setup

$\mathbb{Z}/N\mathbb{Z}$ with $N = pq$ an RSA modulus
N public, p, q private

Slow KDF

With delay parameter Δ:

$$\begin{aligned} f : (\mathbb{Z}/N\mathbb{Z})^\times &\longrightarrow (\mathbb{Z}/N\mathbb{Z})^\times \\ x &\longmapsto x^{2^\Delta} \end{aligned}$$

(Conjecturally) takes Δ squarings.

$\bullet\ x$

Rivest-Shamir-Wagner TL Puzzle ('96)

Setup

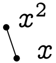
$\mathbb{Z}/N\mathbb{Z}$ with $N = pq$ an RSA modulus
N public, p, q private

Slow KDF

With delay parameter Δ:

$$
\begin{aligned}
f : (\mathbb{Z}/N\mathbb{Z})^\times &\longrightarrow (\mathbb{Z}/N\mathbb{Z})^\times \\
x &\longmapsto x^{2^\Delta}
\end{aligned}
$$

(Conjecturally) takes Δ squarings.



Rivest-Shamir-Wagner TL Puzzle ('96)

Setup

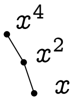
$\mathbb{Z}/N\mathbb{Z}$ with $N = pq$ an RSA modulus
N public, p, q private

Slow KDF

With delay parameter Δ:

$$
\begin{aligned}
f : (\mathbb{Z}/N\mathbb{Z})^\times &\longrightarrow (\mathbb{Z}/N\mathbb{Z})^\times \\
x &\longmapsto x^{2^\Delta}
\end{aligned}
$$

(Conjecturally) takes Δ squarings.



Rivest-Shamir-Wagner TL Puzzle ('96)

Setup

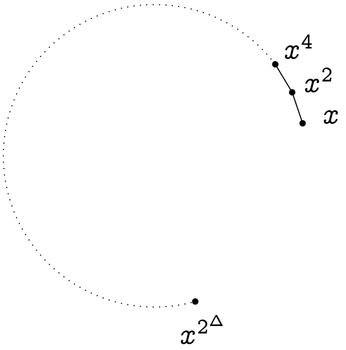
$\mathbb{Z}/N\mathbb{Z}$ with $N = pq$ an RSA modulus
N public, p, q private

Slow KDF

With delay parameter Δ:

$$f : (\mathbb{Z}/N\mathbb{Z})^\times \longrightarrow (\mathbb{Z}/N\mathbb{Z})^\times$$
$$x \longmapsto x^{2^\Delta}$$

(Conjecturally) takes Δ squarings.



Rivest-Shamir-Wagner TL Puzzle ('96)

Setup

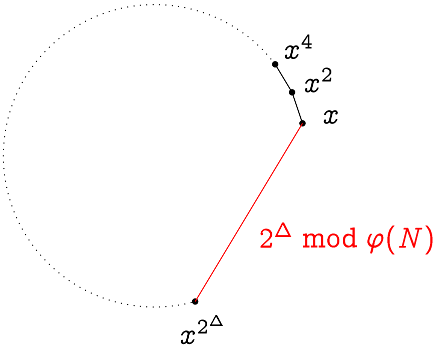
$\mathbb{Z}/N\mathbb{Z}$ with $N = pq$ an RSA modulus
N public, p, q private

Slow KDF

With delay parameter Δ:

$$\begin{aligned} f : (\mathbb{Z}/N\mathbb{Z})^\times &\longrightarrow (\mathbb{Z}/N\mathbb{Z})^\times \\ x &\longmapsto x^{2^\Delta} \end{aligned}$$

(Conjecturally) takes Δ squarings.



KDF: knowing p, q, we can take a shortcut!

VDFs from groups of unknown order

Setup

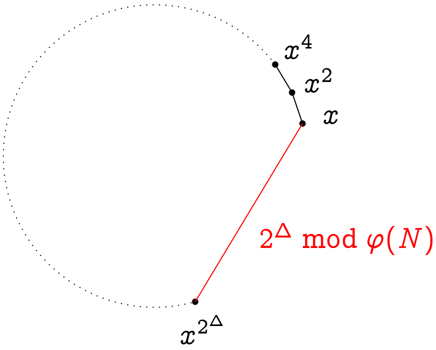
$\mathbb{Z}/N\mathbb{Z}$ with $N = pq$ an RSA modulus,
N public, p, q unknown

Evaluation

With delay parameter Δ:

$$f : (\mathbb{Z}/N\mathbb{Z})^\times \longrightarrow (\mathbb{Z}/N\mathbb{Z})^\times$$
$$x \longmapsto x^{2^\Delta}$$

(Conjecturally) takes Δ squarings.



VDFs from groups of unknown order

Setup

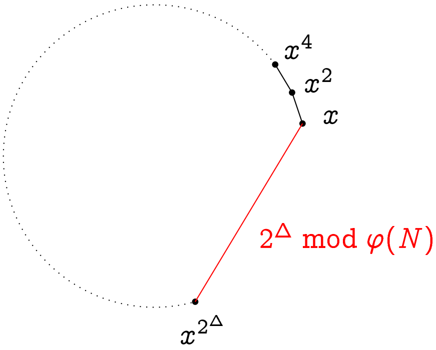
$\mathbb{Z}/N\mathbb{Z}$ with $N = pq$ an RSA modulus,
N public, p, q unknown

Evaluation

With delay parameter Δ:

$$f : (\mathbb{Z}/N\mathbb{Z})^\times \longrightarrow (\mathbb{Z}/N\mathbb{Z})^\times$$
$$x \longmapsto x^{2^\Delta}$$

(Conjecturally) takes Δ squarings.



If we knew p, q, then we could easily verify…

VDFs from groups of unknown order

Setup

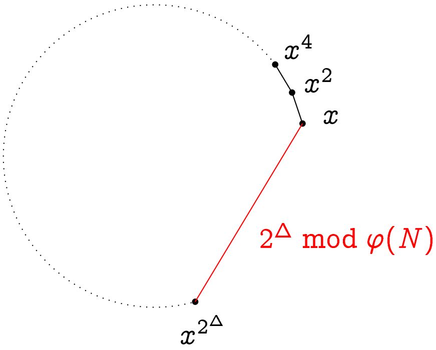
$\mathbb{Z}/N\mathbb{Z}$ with $N = pq$ an RSA modulus,
N public, p, q unknown

Evaluation

With delay parameter Δ:

$$f : (\mathbb{Z}/N\mathbb{Z})^\times \longrightarrow (\mathbb{Z}/N\mathbb{Z})^\times$$
$$x \longmapsto x^{2^\Delta}$$

(Conjecturally) takes Δ squarings.



If we knew p, q, then we could easily verify… but we don't!

VDFs from groups of unknown order

Proofs of exponentiation

Pietrzak: recursive argument, rather expensive, low order assumption.

Wesolowski: arithmetic argument, cheaper, *ad hoc* assumption.

Both made non-interactive via the Fiat-Shamir transform.

VDFs from groups of unknown order

Proofs of exponentiation

Pietrzak: recursive argument, rather expensive, low order assumption.

Wesolowski: arithmetic argument, cheaper, *ad hoc* assumption.

Both made non-interactive via the Fiat-Shamir transform.

Removing trusted third parties

- RSA setup requires trusted generation of $N = pq$ (single or distributed authority);
- The only property used by the VDFs is that the order of $(\mathbb{Z}/N\mathbb{Z})^\times$ is unknown;
- Can adapt the protocol to any cryptographic group of unknown order:
  - e.g., quadratic imaginary class groups of unknown order can be publicly generated with no trusted setup!

The passage of time

Some (problematic?) key assumptions:

- A squaring is a squaring. It cannot possibly go faster than xxx ns!
- You have a machine that computes a squaring in not much more than that!
- n squarings are n squarings. It cannot take less than $n\times$ one squaring to do them!
- Crucially, even if you have n parallel processors!

These are likely all false, but seem to hold in practice…

The passage of time

Some (problematic?) key assumptions:

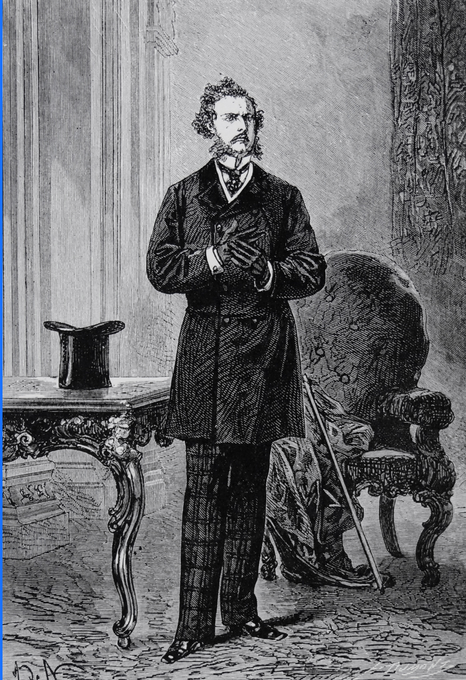
- A squaring is a squaring. It cannot possibly go faster than xxx ns!
- You have a machine that computes a squaring in not much more than that!
- n squarings are n squarings. It cannot take less than $n\times$ one squaring to do them!
- Crucially, even if you have n parallel processors!

These are likely all false, but seem to hold in practice…

Some concrete numbers:

- 1 squaring modulo a 2048-bits integer (unknown factorization)
  - takes $\approx 1\mu$s in software;
  - the current record in FPGA is 25ns.
- Some example delays:
  - 1 hour $\rightarrow \approx 2^{38}$ squarings,
  - 1 year $\rightarrow \approx 2^{51}$ squarings,
  - 1M years $\rightarrow \approx 2^{71}$ squarings.

Isogeny based Delay Functions

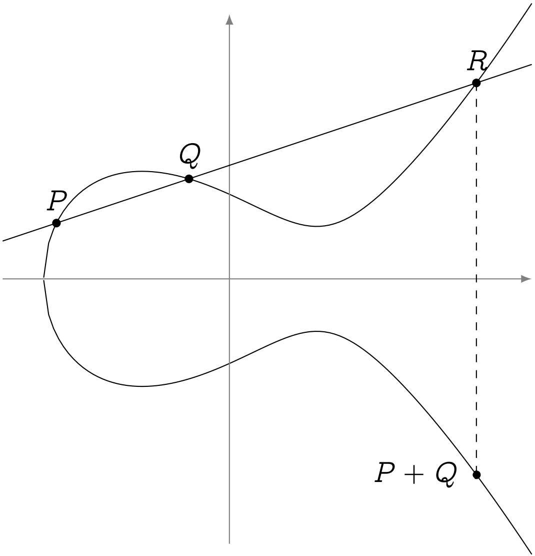


Elliptic curves and isogenies

Elliptic curves

$$y^2 = x^3 + ax + b$$

and their famous group law…



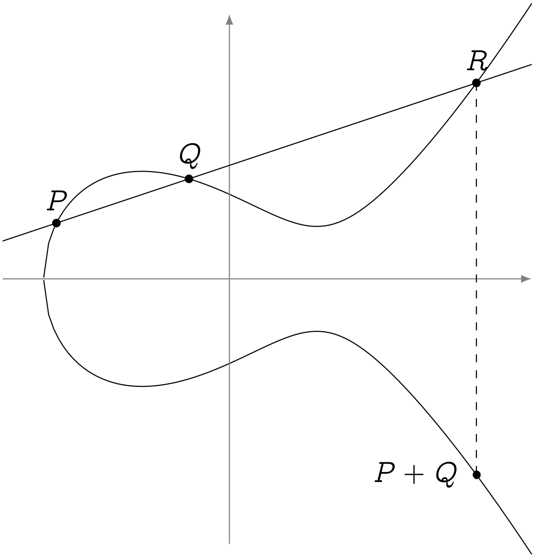
Elliptic curves and isogenies

Elliptic curves

$$y^2 = x^3 + ax + b$$

and their famous group law…

Isogenies are morphisms of elliptic curves.



Elliptic curves and isogenies

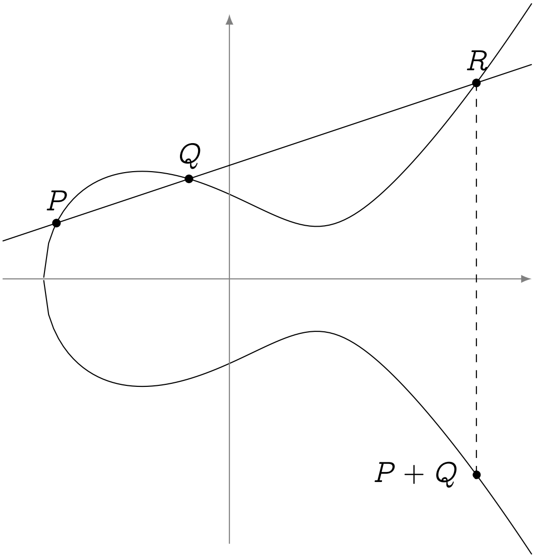
Elliptic curves

$$y^2 = x^3 + ax + b$$

and their famous group law…

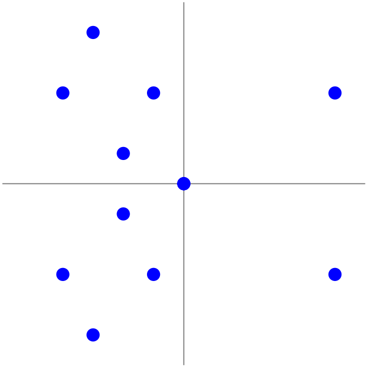
Isogenies are morphisms of elliptic curves.

$$\frac{\text{Elliptic curves}}{\text{Isogenies}} = \frac{\text{Vector spaces}}{\text{Matrices}}$$

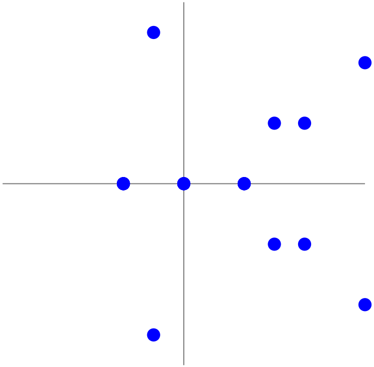


Isogenies: an example over $\mathbb{F}_{11}$

$E : y^2 = x^3 + x$

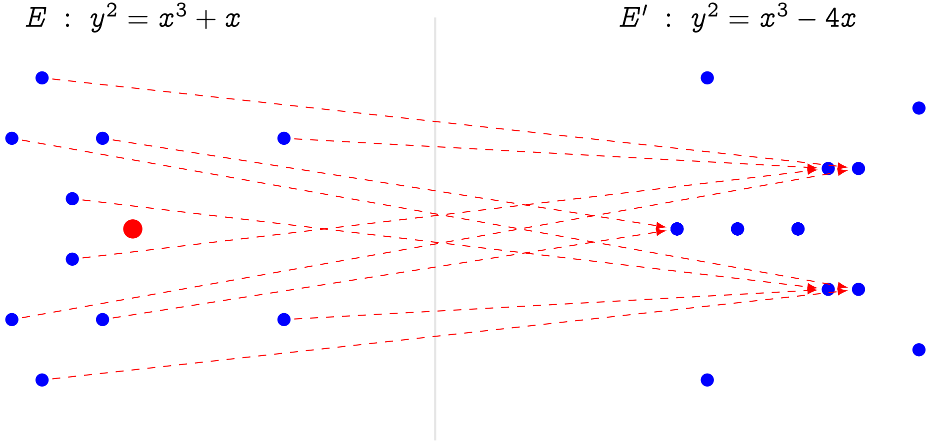


$E' : y^2 = x^3 - 4x$



$$\phi(x, y) = \left(\frac{x^2+1}{x}, \quad y\frac{x^2-1}{x^2} \right)$$

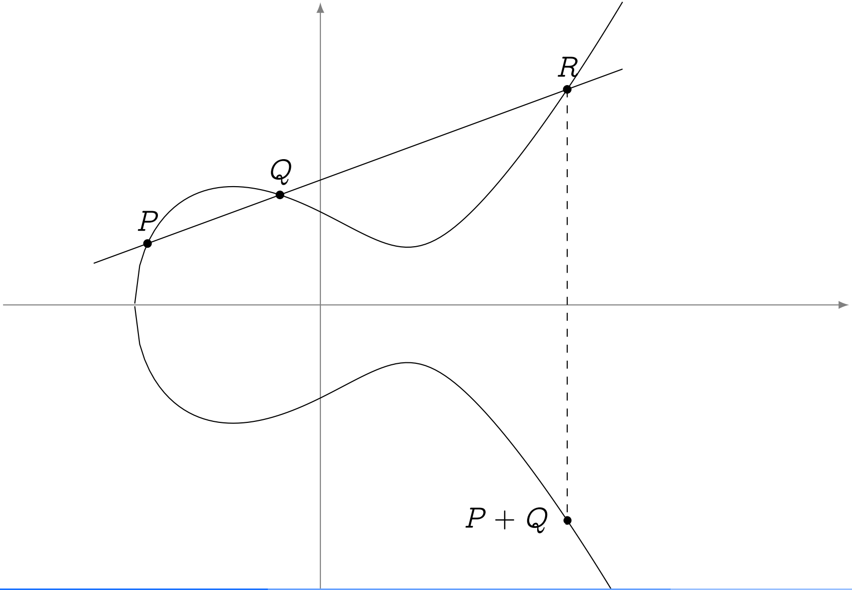
Isogenies: an example over $\mathbb{F}_{11}$



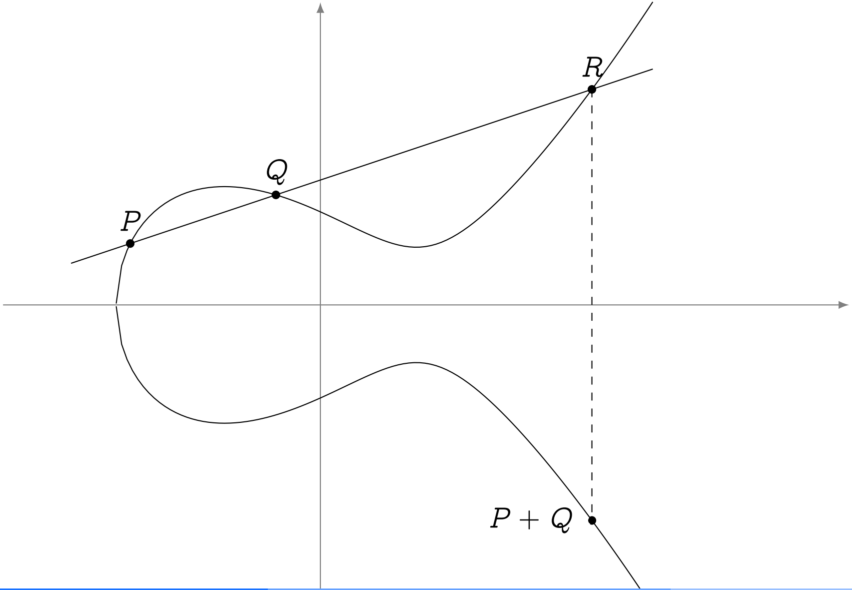
$$\phi(x,y) = \left(\frac{x^2+1}{x},\quad y\frac{x^2-1}{x^2}\right)$$

- Kernel generator in red.
- This is a degree 2 map.
- Analogous to $x \mapsto x^2$ in $\mathbb{F}_q^*$.

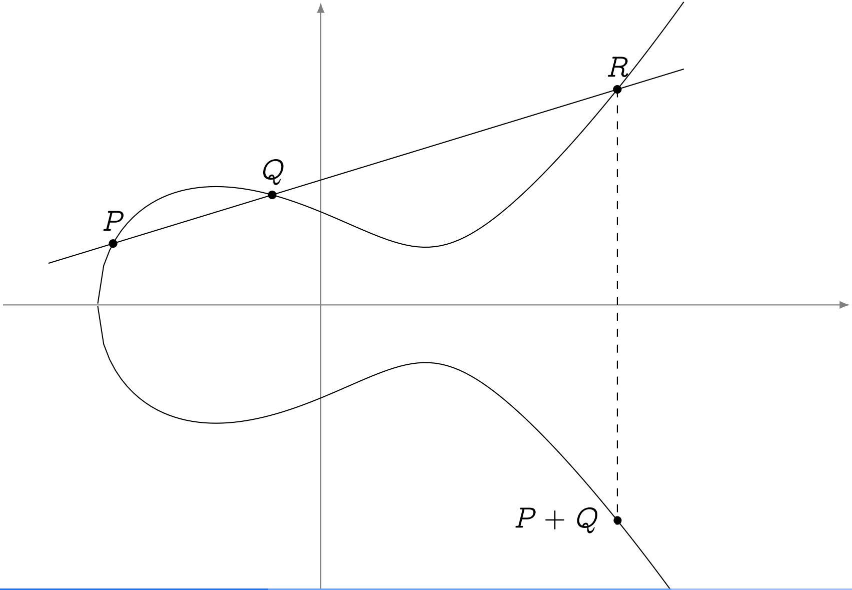
Up to isomorphism



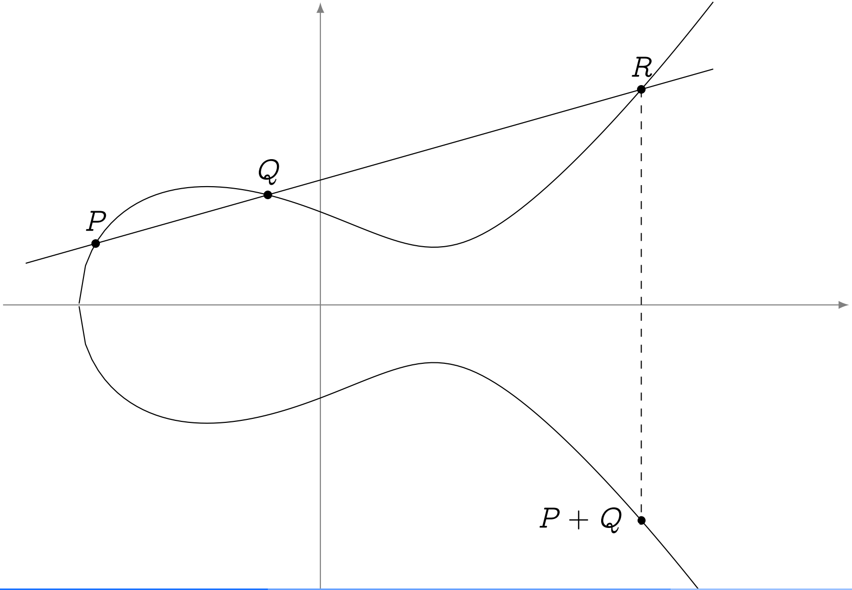
Up to isomorphism



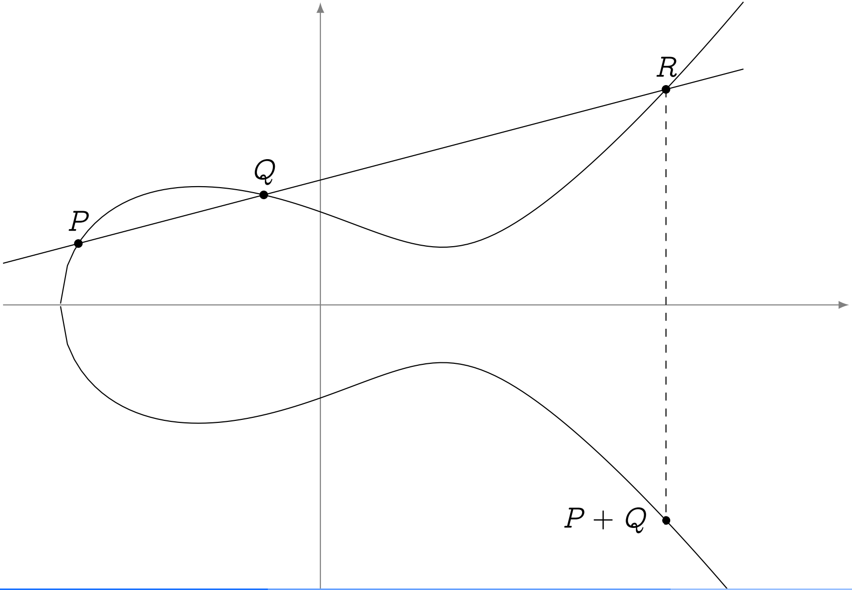
Up to isomorphism



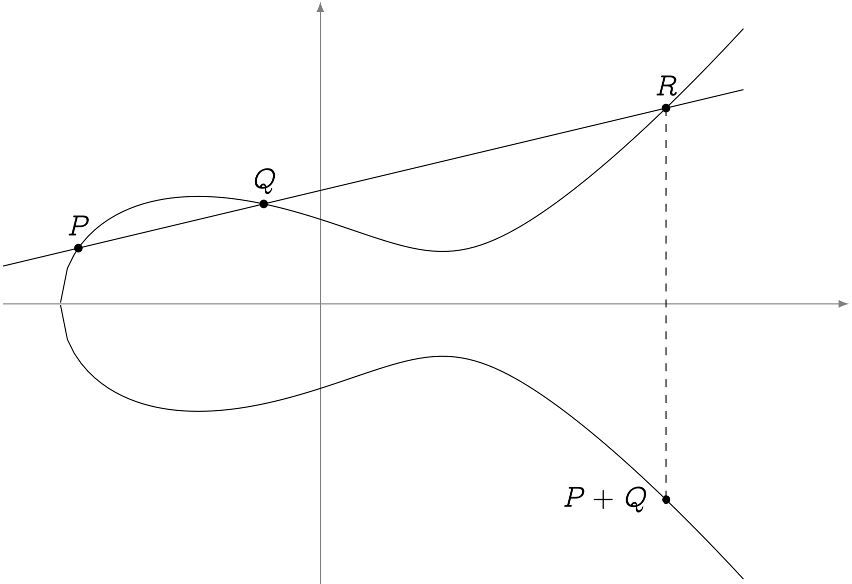
Up to isomorphism



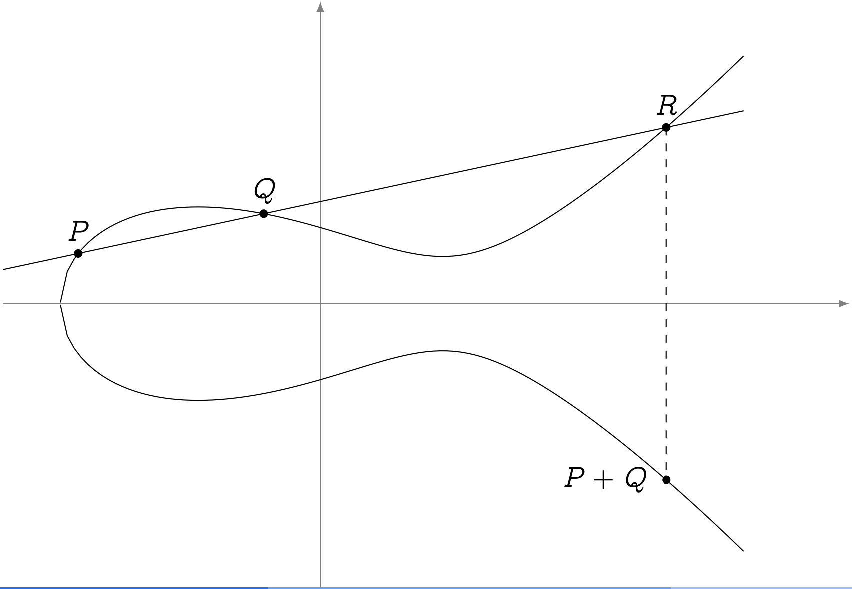
Up to isomorphism



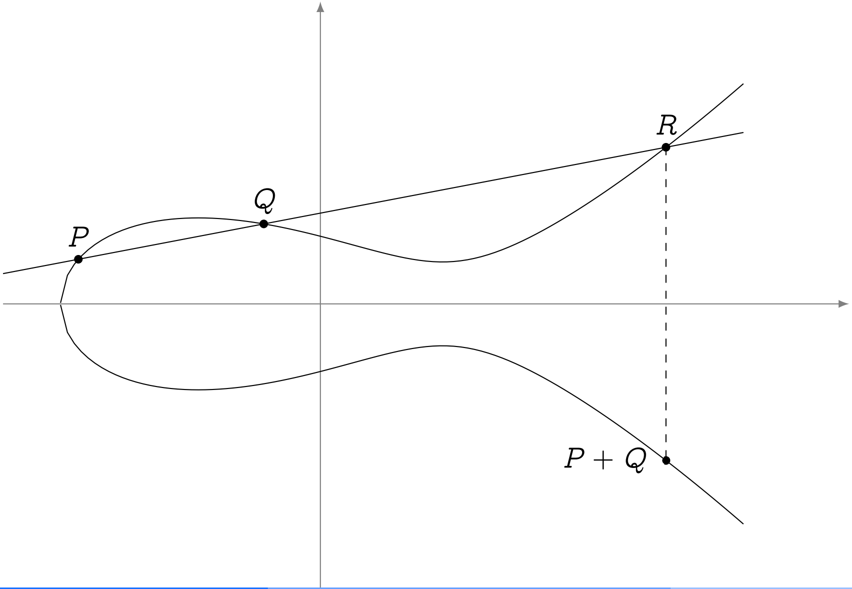
Up to isomorphism



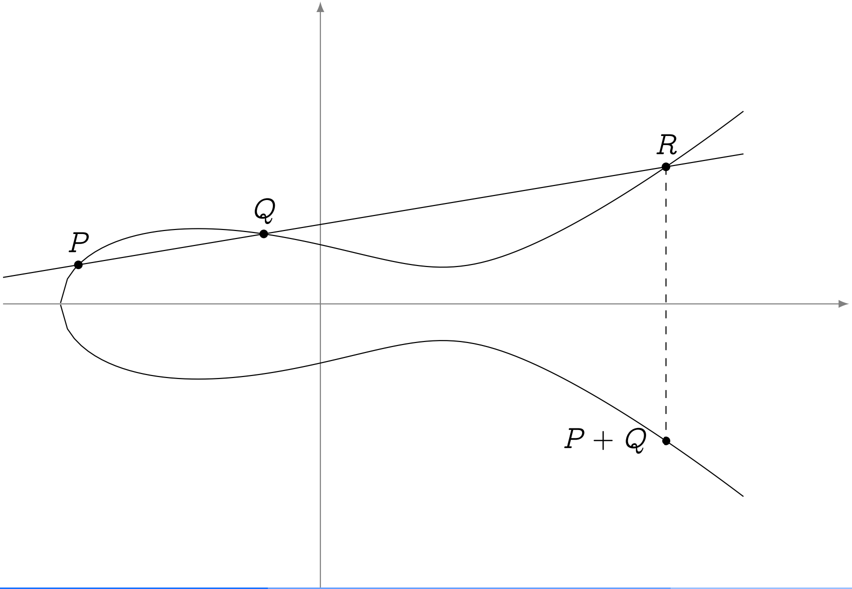
Up to isomorphism



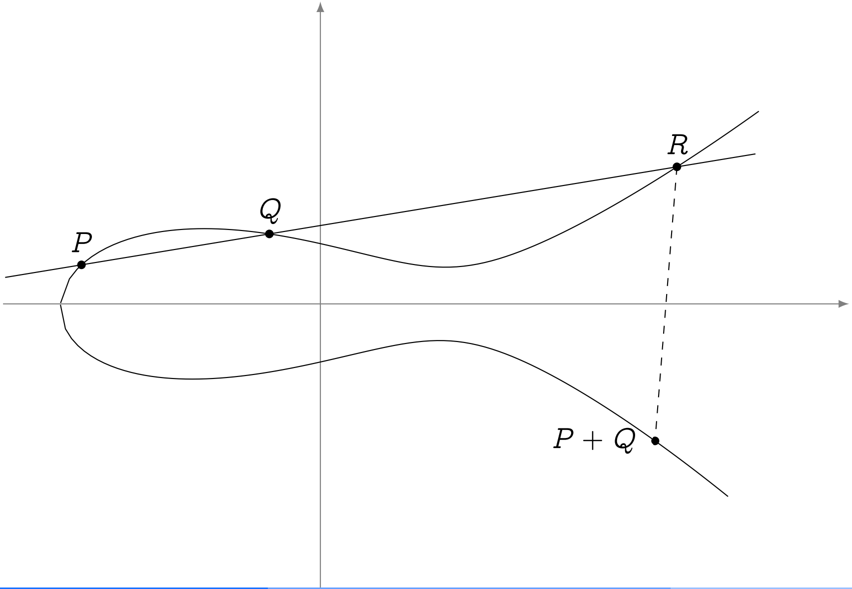
Up to isomorphism



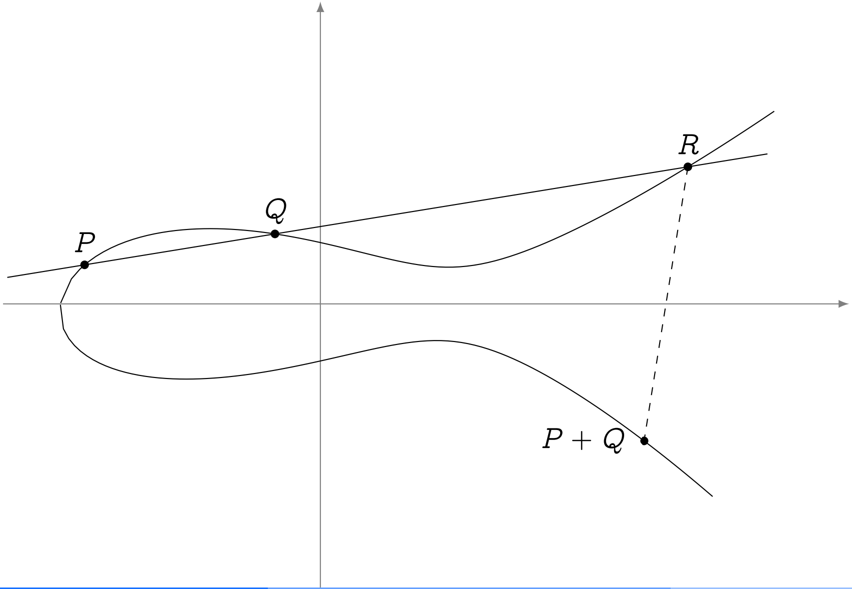
Up to isomorphism



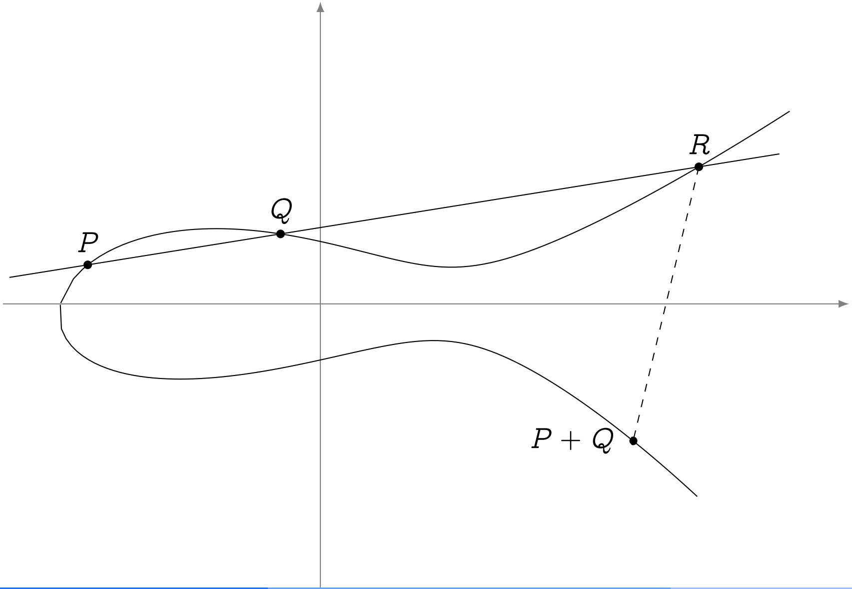
Up to isomorphism



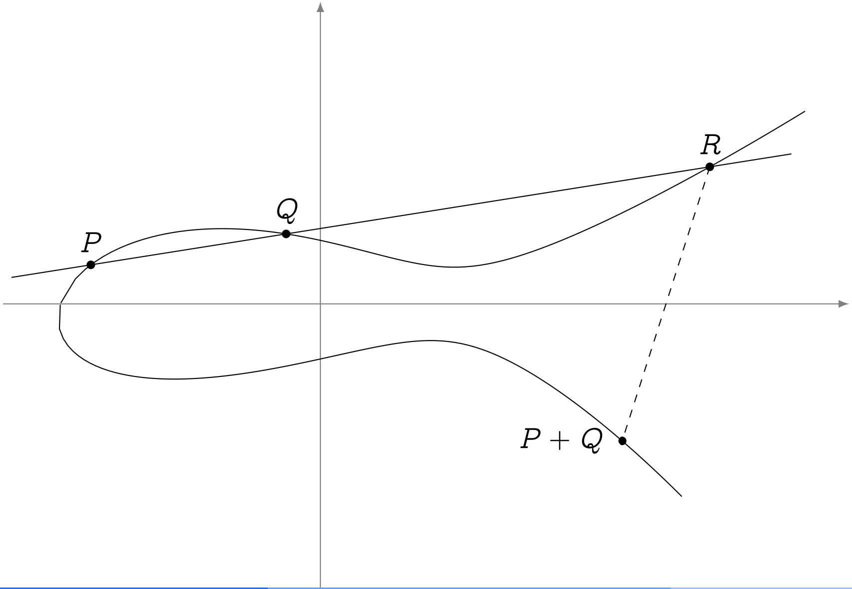
Up to isomorphism



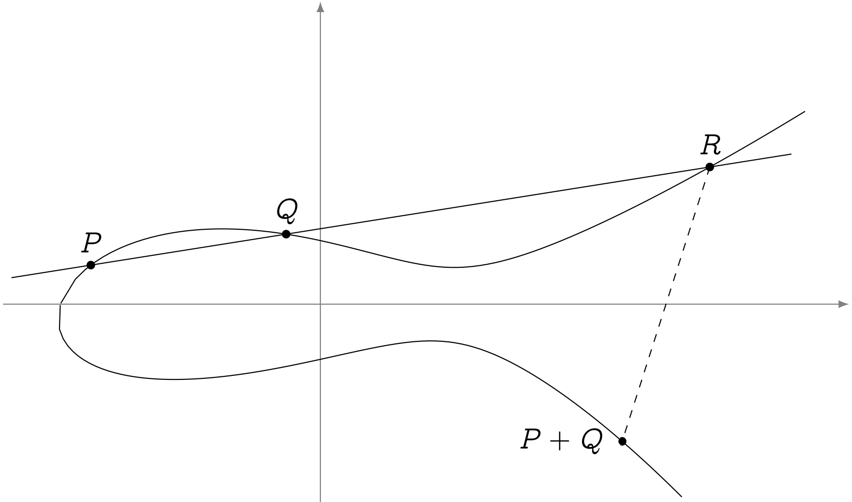
Up to isomorphism



Up to isomorphism

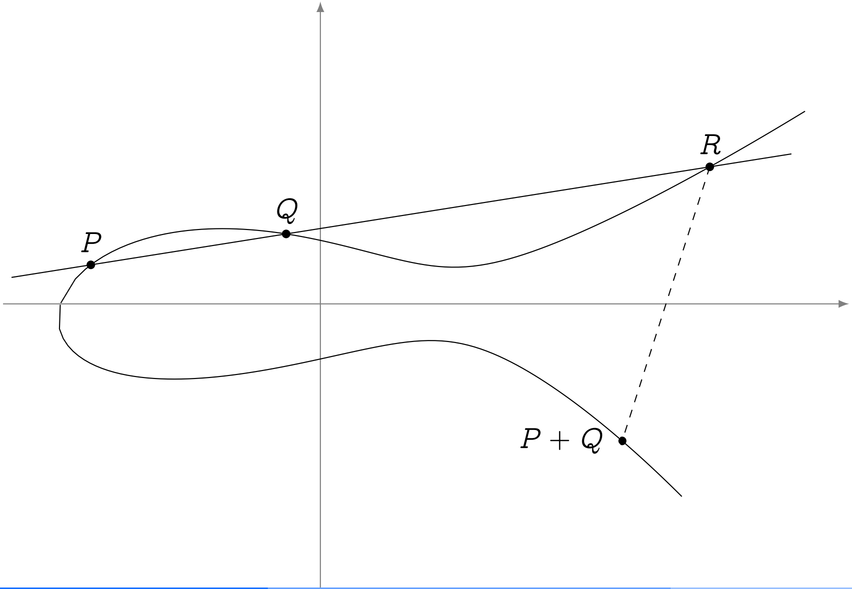


Up to isomorphism

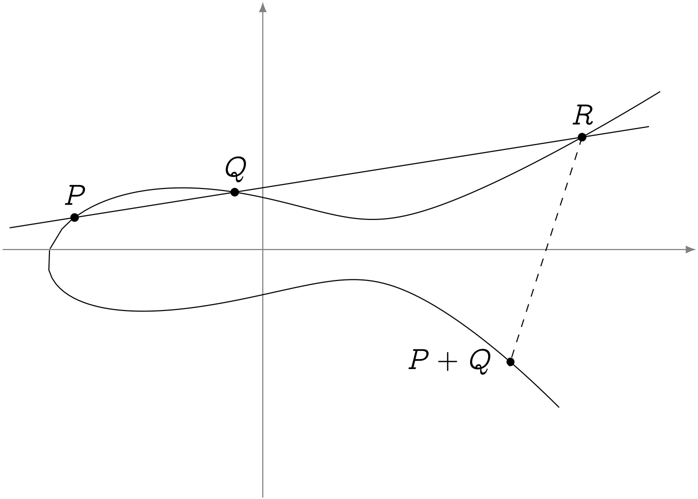


$$y^2 = x^3 + ax + b \quad \longrightarrow \quad j = 1728\frac{4a^3}{4a^3+27b^2}$$

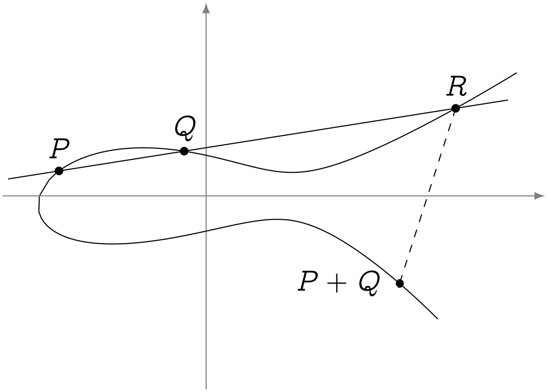
Up to isomorphism



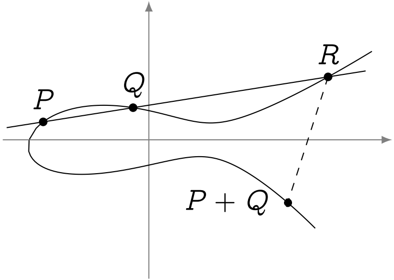
Up to isomorphism



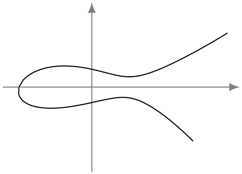
Up to isomorphism



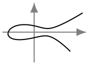
Up to isomorphism



Up to isomorphism



Up to isomorphism



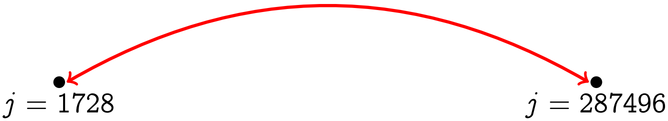
Up to isomorphism

$j = 1728$

Up to isomorphism

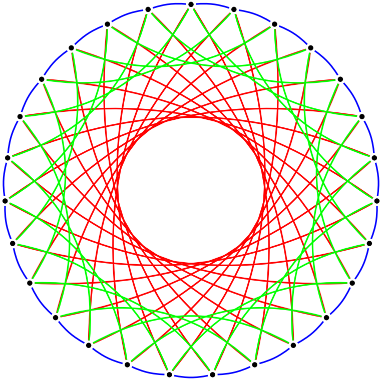
$$j = 1728 \xrightarrow{\phi} j = 287496$$

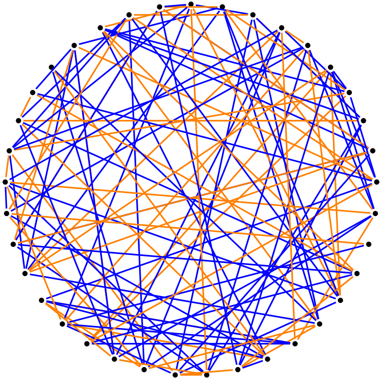
Up to isomorphism



The beauty and the beast (credit: Lorenz Panny)

Components of particular isogeny graphs look like this:

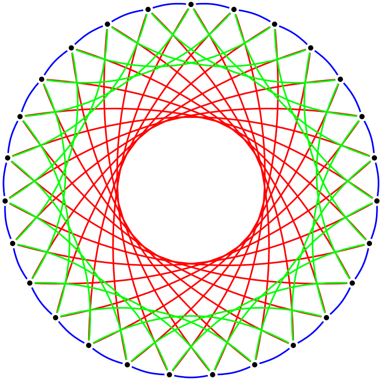


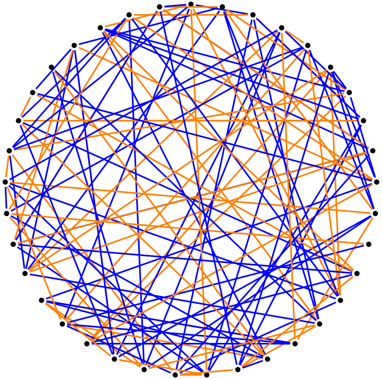


Which of these is good for VDFs?

The beauty and the beast (credit: Lorenz Panny)

Components of particular isogeny graphs look like this:





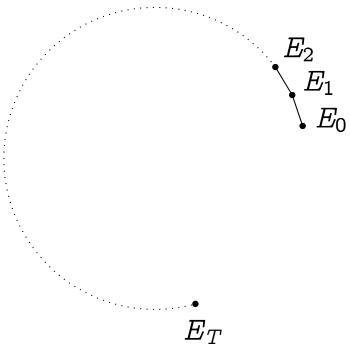
Which of these is good for VDFs? **Both!**

Sloooooooooooooooooooooow isogenies (`https://ia.cr/2019/166`)

Setup

With delay parameter T:

- A looooooooooooooooooooooooong isogeny path,

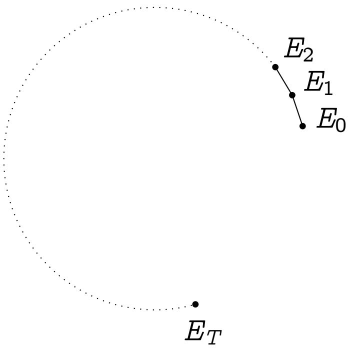


Slooooooooooooooooooooooow isogenies (`https://ia.cr/2019/166`)

Setup

With delay parameter T:

- A looooooooooooooooooooooooooong isogeny path,
- A starting curve E_0,
- An isogeny $\phi : E_0 \to E_T$ of degree 2^T.



Sloooooooooooooooooooooow isogenies (`https://ia.cr/2019/166`)

Setup

With delay parameter T:

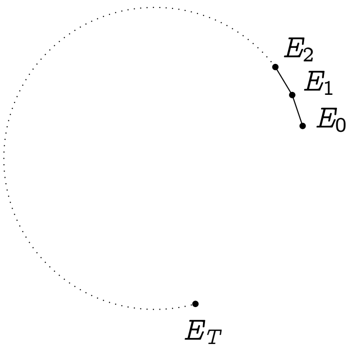
- A looooooooooooooooooooooooong isogeny path,
- A starting curve E_0,
- An isogeny $\phi : E_0 \to E_T$ of degree 2^T.

Evaluation

ϕ **is** the VDF:

$$\begin{aligned} \phi : E_0(\mathbb{F}_p) &\longrightarrow E_T(\mathbb{F}_p) \\ P &\longmapsto \phi(P) \end{aligned}$$

Conjecturally, no faster way than composing degree 2 isogenies.



Sloooooooooooooooooooooow isogenies (https://ia.cr/2019/166)

Setup

With delay parameter T:

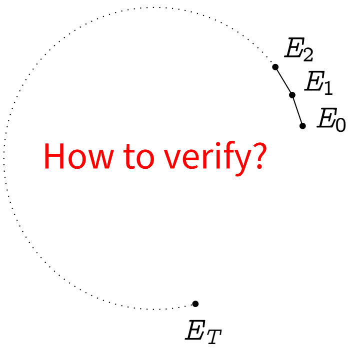
- A looooooooooooooooooooooooong isogeny path,
- A starting curve E_0,
- An isogeny $\phi : E_0 \to E_T$ of degree 2^T.

Evaluation

ϕ **is** the VDF:

$$\begin{aligned} \phi : E_0(\mathbb{F}_p) &\longrightarrow E_T(\mathbb{F}_p) \\ P &\longmapsto \phi(P) \end{aligned}$$

Conjecturally, no faster way than composing degree 2 isogenies.



Sloooooooooooooooooooooow isogenies (https://ia.cr/2019/166)

Setup

With delay parameter T:

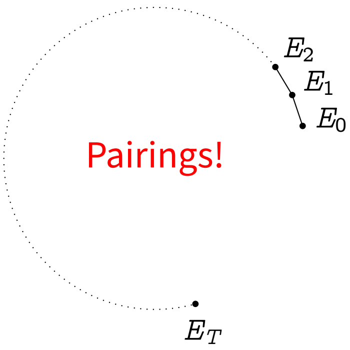
- A looooooooooooooooooooooooong isogeny path,
- A starting curve E_0,
- An isogeny $\phi : E_0 \to E_T$ of degree 2^T.

Evaluation

ϕ **is** the VDF:

$$\begin{aligned} \phi : E_0(\mathbb{F}_p) &\longrightarrow E_T(\mathbb{F}_p) \\ P &\longmapsto \phi(P) \end{aligned}$$

Conjecturally, no faster way than composing degree 2 isogenies.



Slooooooooooooooooooooooow isogenies (https://ia.cr/2019/166)

Setup

With delay parameter T:

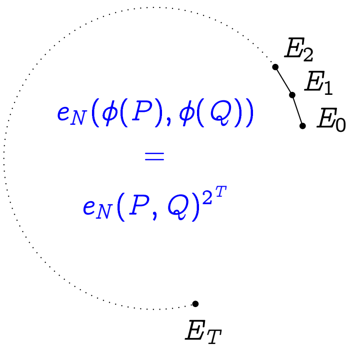
- A looooooooooooooooooooooooong isogeny path,
- A starting curve E_0,
- An isogeny $\phi : E_0 \to E_T$ of degree 2^T.

Evaluation

ϕ **is** the VDF:

$$\begin{aligned} \phi : E_0(\mathbb{F}_p) &\longrightarrow E_T(\mathbb{F}_p) \\ P &\longmapsto \phi(P) \end{aligned}$$

Conjecturally, no faster way than composing degree 2 isogenies.



Comparison

	Wesolowski		Pietrzak		Ours	
	RSA	class group	RSA	class group	$\mathbb{F}_p$	$\mathbb{F}_{p^2}$
proof size	$O(1)$	$O(1)$	$O(\log(T))$	$O(\log(T))$	—	—
aggregatable	yes	yes	yes	yes	—	—
watermarkable	yes	yes	yes	yes	(yes)	(yes)
perfect soundness	no	no	no	no	yes	yes
long setup	no	no	no	no	yes	yes
trusted setup	yes	no	yes	no	yes	yes
↳ updatable	no	—	no	—	yes	yes
↳ synchronous	yes	—	yes	—	no	no
best attack	$L_N(1/3)$	$L_N(1/2)$	$L_N(1/3)$	$L_N(1/2)$	$L_p(1/3)$	$L_p(1/3)$
quantum annoying	no	(yes)	no	(yes)	no	yes

For concreteness

Elementary step:

RSA: $$x \longmapsto x^2 \mod N$$

Isogenies: $$x \longmapsto \frac{(x+1)^2}{4\alpha_i x} \mod p$$

($\alpha_1, \dots, \alpha_T$ correspond to the isogeny steps)

Typical parameters: $\log_2 p \approx 1500$ gives security similar to $\log_2 N \approx 2048$.

Huge storage: for a 1 hour delay,

- Isogeny path of length $\approx 7 \cdot 10^{10}$,
- evaluator needs $\approx$ 16TiB for storing all $(\alpha_1, \dots, \alpha_T)$,
- Throughput of $\approx$ 4.5 GiB/s to read the α_i's from memory.
- Storage/speed compromises are available, but it's a tough call!

(My favorite) open questions

- Understand the impact of large memory requirements in evaluation; is a time/memory trade-off reasonable?
- Remove trusted setup:
  - Hash into the supersingular set, or
  - Construct ordinary pairing friendly curves with large discriminant.
- Explore more advanced pairing+delay constructions.
- Spend millions on dedicated hardware for 2-isogenies.

Just Add Isogenies™!

Thank you

@luca_defeo