CSIDH: An Efficient Post-Quantum Commutative Group Action

Chloe Martindale
`csidh.isogeny.org`

University of South Florida, 26th April 2019

The Discrete Logarithm Problem

- Most of your online data is encrypted via cryptographic protocols that rely on the discrete logarithm problem.

The Discrete Logarithm Problem

- Most of your online data is encrypted via cryptographic protocols that rely on the discrete logarithm problem.
  eg. WhatsApp messages; internet banking apps; sites using 'https'.

The Discrete Logarithm Problem

- Most of your online data is encrypted via cryptographic protocols that rely on the discrete logarithm problem.
  eg. WhatsApp messages; internet banking apps; sites using 'https'.
- What is the discrete logarithm problem?

The Discrete Logarithm Problem

- Let G be a group with group operation $*$.

The Discrete Logarithm Problem

- Let G be a group with group operation $*$.

Example: Let

$$\begin{aligned} G &= (\mathbb{Z}/23\mathbb{Z}) - \{0\} \\ &= \{1 \bmod 23,\ 2 \bmod 23,\ 3 \bmod 23, \ldots, 22 \bmod 23\}, \end{aligned}$$

then G is a group with group operation $*$ given by multiplication.

The Discrete Logarithm Problem

- Let G be a group with group operation $*$.
- The discrete logarithm problem (DLP): given $g \in G$ and $\underbrace{g * \cdots * g}_{n \text{ times}}$, find n.

Example: Let

$$\begin{aligned} G &= (\mathbb{Z}/23\mathbb{Z}) - \{0\} \\ &= \{1 \bmod 23,\ 2 \bmod 23,\ 3 \bmod 23, \ldots, 22 \bmod 23\}, \end{aligned}$$

then G is a group with group operation $*$ given by multiplication.

The Discrete Logarithm Problem

- Let G be a group with group operation $*$.
- The discrete logarithm problem (DLP): given $g \in G$ and $\underbrace{g * \cdots * g}_{n \text{ times}}$, find n.

Example: Let

$$\begin{aligned} G &= (\mathbb{Z}/23\mathbb{Z}) - \{0\} \\ &= \{1 \bmod 23,\ 2 \bmod 23,\ 3 \bmod 23, \dots,\ 22 \bmod 23\}, \end{aligned}$$

then G is a group with group operation $*$ given by multiplication.
DLP in $(\mathbb{Z}/23\mathbb{Z}) - \{0\}$: Given $g \bmod 23$ and $g^n \bmod 23$, find n.

The Discrete Logarithm Problem

The DLP is hard when, given $g \in G$:

- Given $n \in \mathbb{Z}$, computing $\underbrace{g * \cdots * g}_{n \text{ times}}$ is fast. (eg. Polynomial time).

Example: Given $g = 5 \bmod 23$:

The Discrete Logarithm Problem

The DLP is hard when, given $g \in G$:

- Given $n \in \mathbb{Z}$, computing $\underbrace{g * \cdots * g}_{n \text{ times}}$ is fast. (eg. Polynomial time).

Example: Given $g = 5 \bmod 23$:

- Let $n = 9$; compute $5^9 \bmod 23$.

The Discrete Logarithm Problem

The DLP is hard when, given $g \in G$:

- Given $n \in \mathbb{Z}$, computing $\underbrace{g * \cdots * g}_{n \text{ times}}$ is fast. (eg. Polynomial time).
- Given $\underbrace{g * \cdots * g}_{n \text{ times}}$, computing n is slow. (eg. Exponential time).

Example: Given $g = 5 \bmod 23$:

- Let $n = 9$; compute $5^9 \bmod 23$.

The Discrete Logarithm Problem

The DLP is hard when, given $g \in G$:

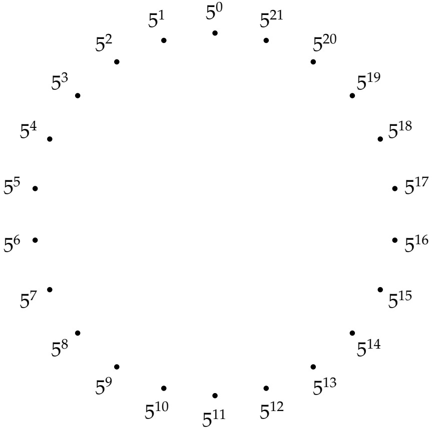
- Given $n \in \mathbb{Z}$, computing $\underbrace{g * \cdots * g}_{n \text{ times}}$ is fast. (eg. Polynomial time).
- Given $\underbrace{g * \cdots * g}_{n \text{ times}}$, computing n is slow. (eg. Exponential time).

Example: Given $g = 5 \bmod 23$:

- Let $n = 9$; compute $5^9 \bmod 23$.
- If $5^n = 11 \bmod 23$; compute n.

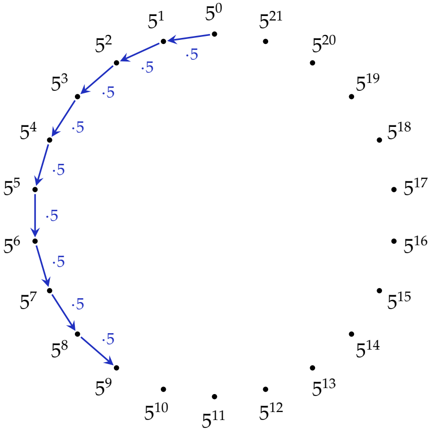
Square-and-multiply

Compute 5^9 mod 23.



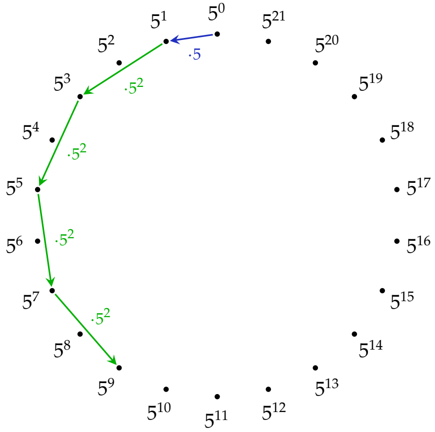
Square-and-multiply

Compute 5^9 mod 23.



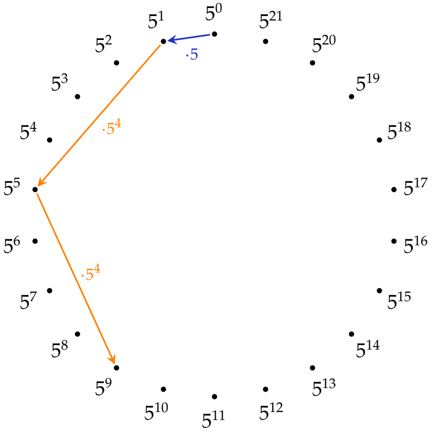
Square-and-multiply

Compute 5^9 mod 23.



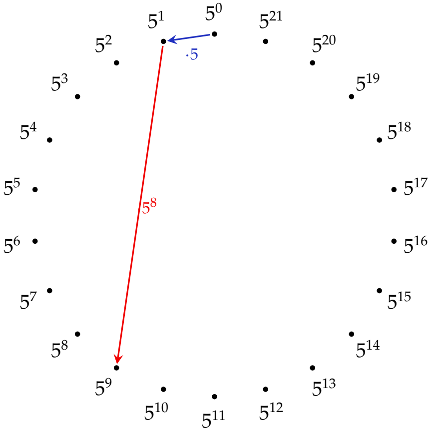
Square-and-multiply

Compute 5^9 mod 23.



Square-and-multiply

Compute 5^9 mod 23.



Square-and-multiply vs. solving DLP

- To compute $5^9 \bmod 23$, compute:
  $5 \cdot 5^8 = 5 \cdot ((5^2)^2)^2 \bmod 23$. (Fast).

Square-and-multiply vs. solving DLP

- To compute $5^9 \bmod 23$, compute:
  $5 \cdot 5^8 = 5 \cdot ((5^2)^2)^2 \bmod 23$. (Fast).
- To compute n such that $5^n \equiv 11 \bmod 23$, check:

Square-and-multiply vs. solving DLP

- To compute $5^9 \bmod 23$, compute:
  $5 \cdot 5^8 = 5 \cdot ((5^2)^2)^2 \bmod 23$. (Fast).
- To compute n such that $5^n \equiv 11 \bmod 23$, check:

$$5^2 \equiv 2 \not\equiv 11 \bmod 23$$

Square-and-multiply vs. solving DLP

- To compute 5^9 mod 23, compute:
  $5 \cdot 5^8 = 5 \cdot ((5^2)^2)^2$ mod 23. (Fast).
- To compute n such that $5^n \equiv 11$ mod 23, check:

$$\begin{aligned}
5^2 &\equiv 2 \not\equiv 11 \bmod 23 \\
5^3 &\equiv 10 \not\equiv 11 \bmod 23 \\
5^4 &\equiv 4 \not\equiv 11 \bmod 23 \\
5^5 &\equiv 20 \not\equiv 11 \bmod 23 \\
5^6 &\equiv 8 \not\equiv 11 \bmod 23 \\
5^7 &\equiv 17 \not\equiv 11 \bmod 23 \\
5^8 &\equiv 16 \not\equiv 11 \bmod 23 \\
5^9 &\equiv 11 \bmod 23.
\end{aligned}$$

Square-and-multiply vs. solving DLP

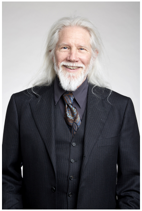
- To compute 5^9 mod 23, compute:
  $5 \cdot 5^8 = 5 \cdot ((5^2)^2)^2$ mod 23. (Fast).
- To compute n such that $5^n \equiv 11$ mod 23, check:

$$5^2 \equiv 2 \not\equiv 11 \bmod 23$$
$$5^3 \equiv 10 \not\equiv 11 \bmod 23$$
$$5^4 \equiv 4 \not\equiv 11 \bmod 23$$
$$5^5 \equiv 20 \not\equiv 11 \bmod 23$$
$$5^6 \equiv 8 \not\equiv 11 \bmod 23$$
$$5^7 \equiv 17 \not\equiv 11 \bmod 23$$
$$5^8 \equiv 16 \not\equiv 11 \bmod 23$$
$$5^9 \equiv 11 \bmod 23.$$

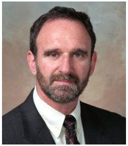
(Slow).

(There are smarter ways to do this in practise, but they're still slow).

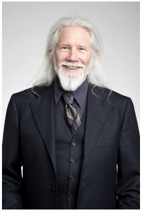
Application of DLP: Diffie-Hellman key exchange



$g \in G$

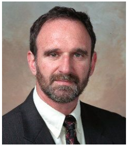


Application of DLP: Diffie-Hellman key exchange



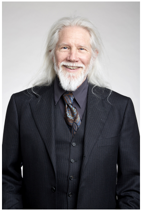
Secret key: d

$g \in G$



Secret key: h

Application of DLP: Diffie-Hellman key exchange

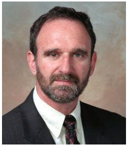


Secret key: d

$g \in G$

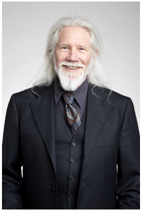
$\xrightarrow{\text{Public key: } g^d}$

$\xleftarrow{\text{Public key: } g^h}$



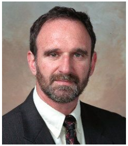
Secret key: h

Application of DLP: Diffie-Hellman key exchange



Secret key: d

$g \in G$



Secret key: h

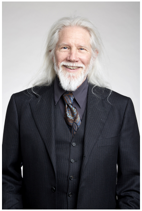
$\xrightarrow{\text{Public key: } g^d}$

$\xleftarrow{\text{Public key: } g^h}$

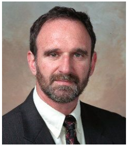
Shared secret: $s = (g^h)^d$

Shared secret: $s = (g^d)^h$

Application of DLP: Diffie-Hellman key exchange



$g \in G$



Secret key: d

$\xrightarrow{\text{Public key: } g^d}$

Secret key: h

$\xleftarrow{\text{Public key: } g^h}$

Shared secret: $s = (g^h)^d$

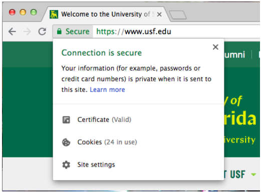
Shared secret: $s = (g^d)^h$

If DLP is hard for G, then computing the public keys and the shared secret is fast for Diffie and Hellman, and computing the secret values is slow for an adversary.

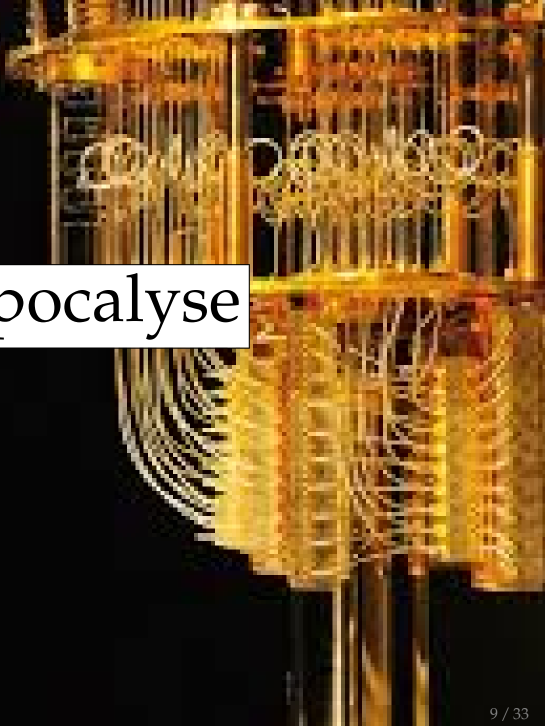
Applications of Diffie-Hellman key exchange

The Diffie-Hellman key exchange is a building block in:

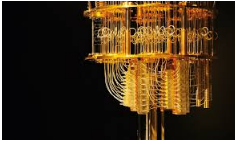
- Digital signature schemes (used for example by some online banking apps; secure websites).
- Encrypted messaging services (eg. WhatsApp).



Cryptapocalyse



Quantum cryptapocalyse

$g^n \longrightarrow$  $\longrightarrow n$

Shor's algorithm quantumly computes n from g^n and g in any group in polynomial time. (About as fast as computing g^n from n and g).

$\rightsquigarrow$ All applications of DLP are broken by quantum computers!


QUANTUM COMPUTING
~~RALPH~~
BREAKS THE
INTERNET
© 2019 Disney

Quantum cryptapocalyse

Key Finding 10: Even if a quantum computer that can decrypt current cryptographic ciphers is more than a decade off, the hazard of such a machine is high enough – and the time frame for transitioning to a new security protocol is sufficiently long and uncertain – that prioritization of the development, standardization, and deployment of post-quantum cryptography is critical for minimizing the chance of a potential security and privacy disaster.

Report by the US National Academy of Sciences, see

`http://www8.nationalacademies.org/onpinews/newsitem.aspx?RecordID=25196`

Reminder: applications of Diffie-Hellman key exchange

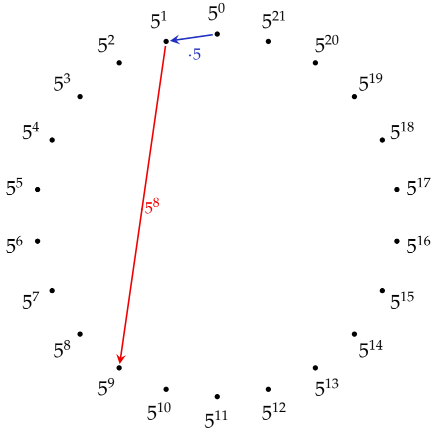
- The Diffie-Hellman key exchange (and hence DLP) is a building block in:
  - Digital signature schemes (used for example by some online banking apps; secure websites).
  - Encrypted messaging services (eg. WhatsApp).

Reminder: applications of Diffie-Hellman key exchange

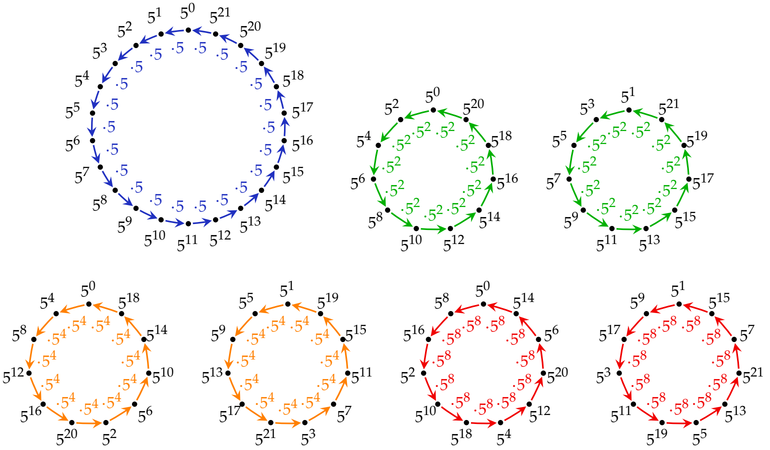
- The Diffie-Hellman key exchange (and hence DLP) is a building block in:
  - Digital signature schemes (used for example by some online banking apps; secure websites).
  - Encrypted messaging services (eg. WhatsApp).
- We need a post-quantum Diffie-Hellman-style key exchange.

Square-and-multiply

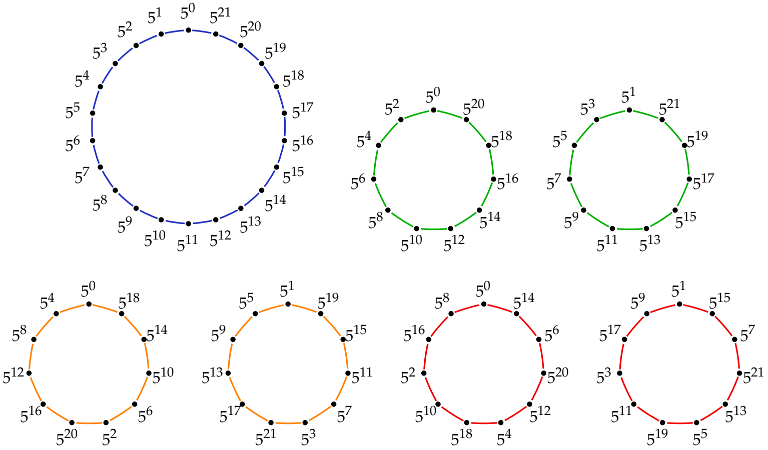
Reminder: how to compute 5^9 mod 23.



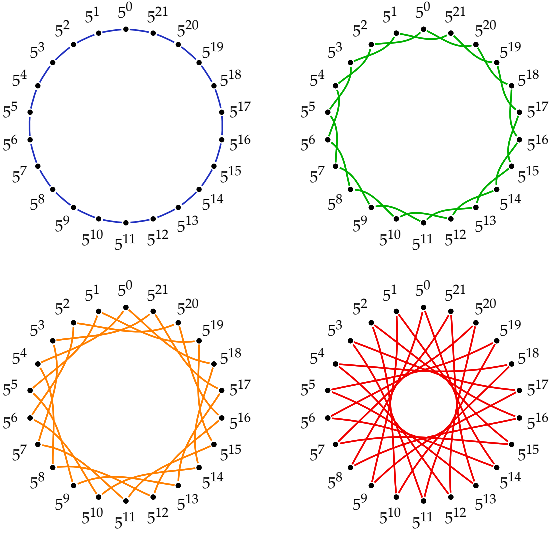
Square-and-multiply



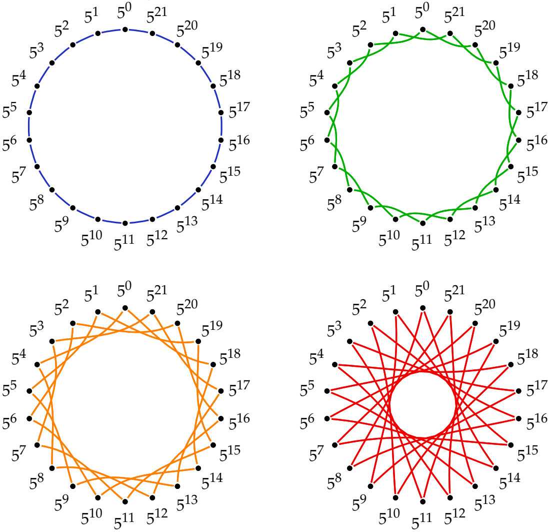
Square-and-multiply



Square-and-multiply

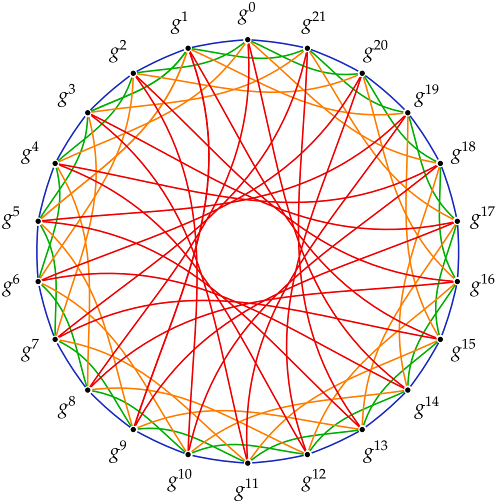


Square-and-multiply

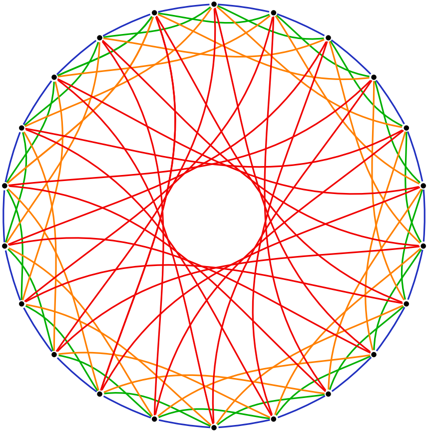


Needed for Diffie-Hellman: Cycles are compatible–
[right, then left] = [left, then right], etc. (Else $(5^a)^b \neq (5^b)^a$).

Union of cycles: rapid mixing

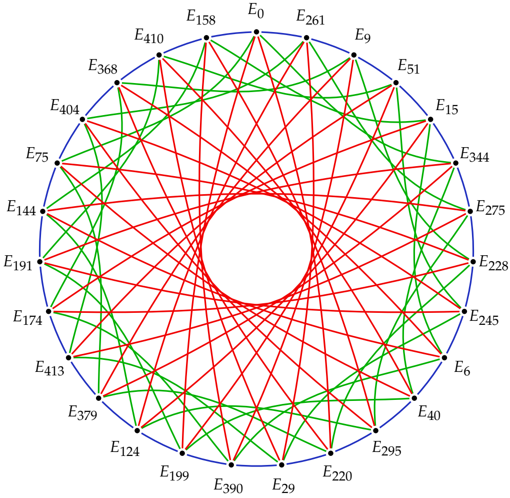


Union of cycles: rapid mixing

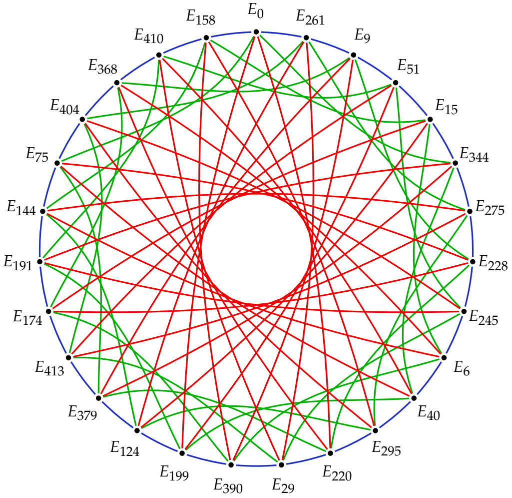


Post-quantum Diffie-Hellman: Nodes are now elliptic curves and edges are isogenies.

Graphs of elliptic curves

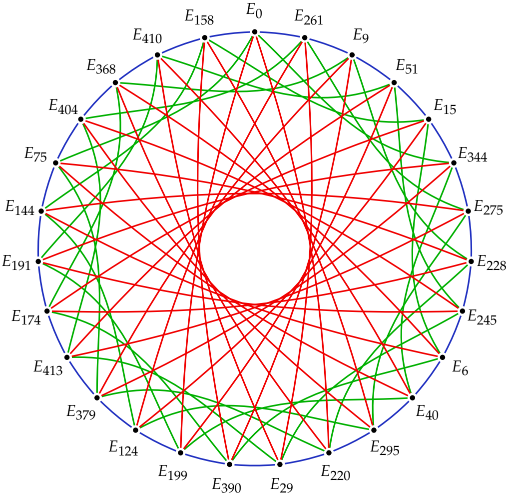


Graphs of elliptic curves



Nodes: Supersingular elliptic curves $E_A : y^2 = x^3 + Ax^2 + x$ over $\mathbb{Z}/419\mathbb{Z}$.

Graphs of elliptic curves



Nodes: Supersingular elliptic curves $E_A : y^2 = x^3 + Ax^2 + x$ over $\mathbb{Z}/419\mathbb{Z}$.
Edges: 3-, 5-, and 7-isogenies.

Interlude: supersingular elliptic curves and isogenies

Nodes: Supersingular elliptic curves $E_A : y^2 = x^3 + Ax^2 + x$ over $\mathbb{F}_{419}$.

Interlude: supersingular elliptic curves and isogenies

Nodes: Supersingular elliptic curves $E_A : y^2 = x^3 + Ax^2 + x$ over $\mathbb{F}_{419}$.

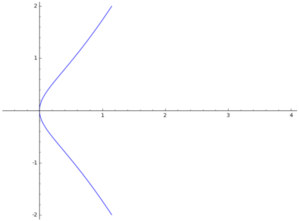
- If equation E_A is smooth (no self intersections or cusps) it represents an elliptic curve.

Interlude: supersingular elliptic curves and isogenies

Nodes: Supersingular elliptic curves $E_A : y^2 = x^3 + Ax^2 + x$ over $\mathbb{F}_{419}$.

- If equation E_A is smooth (no self intersections or cusps) it represents an elliptic curve.

$A = 1$:

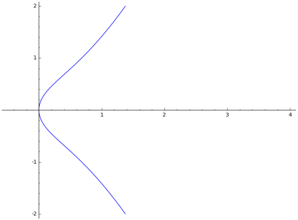


Interlude: supersingular elliptic curves and isogenies

Nodes: Supersingular elliptic curves $E_A : y^2 = x^3 + Ax^2 + x$ over $\mathbb{F}_{419}$.

- If equation E_A is smooth (no self intersections or cusps) it represents an elliptic curve.

$A = 0$:

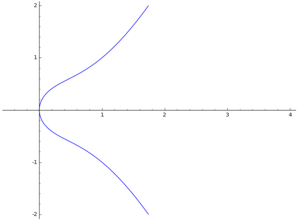


Interlude: supersingular elliptic curves and isogenies

Nodes: Supersingular elliptic curves $E_A: y^2 = x^3 + Ax^2 + x$ over $\mathbb{F}_{419}$.

- If equation E_A is smooth (no self intersections or cusps) it represents an elliptic curve.

$A = -1$:

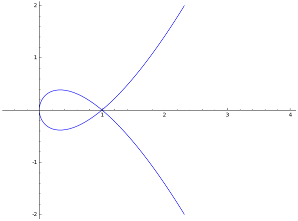


Interlude: supersingular elliptic curves and isogenies

Nodes: Supersingular elliptic curves $E_A : y^2 = x^3 + Ax^2 + x$ over $\mathbb{F}_{419}$.

- If equation E_A is smooth (no self intersections or cusps) it represents an elliptic curve.

$A = -2$:

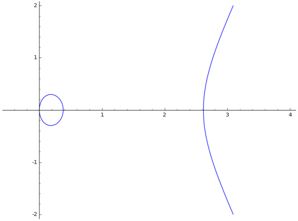


Interlude: supersingular elliptic curves and isogenies

Nodes: Supersingular elliptic curves $E_A : y^2 = x^3 + Ax^2 + x$ over $\mathbb{F}_{419}$.

- If equation E_A is smooth (no self intersections or cusps) it represents an elliptic curve.

$A = -3$:

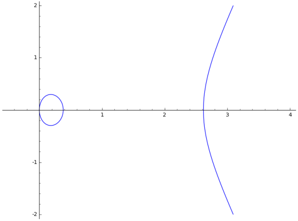


Interlude: supersingular elliptic curves and isogenies

Nodes: Supersingular elliptic curves $E_A : y^2 = x^3 + Ax^2 + x$ over $\mathbb{F}_{419}$.

- If equation E_A is smooth (no self intersections or cusps) it represents an elliptic curve.
- The set of $\mathbb{F}_p$-rational solutions (x, y) to an elliptic curve equation $E_A/\mathbb{F}_p$, together with a 'point at infinity' P_∞, forms a group with identity P_∞, notated $E_A(\mathbb{F}_p)$.

$A = -3$:

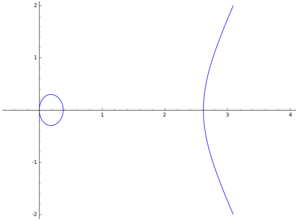


Interlude: supersingular elliptic curves and isogenies

Nodes: Supersingular elliptic curves $E_A : y^2 = x^3 + Ax^2 + x$ over $\mathbb{F}_{419}$.

- If equation E_A is smooth (no self intersections or cusps) it represents an elliptic curve.
- The set of $\mathbb{F}_p$-rational solutions (x, y) to an elliptic curve equation $E_A/\mathbb{F}_p$, together with a 'point at infinity' P_∞, forms a group with identity P_∞, notated $E_A(\mathbb{F}_p)$.
- An elliptic curve $E_A/\mathbb{F}_p$ with $p \geq 5$ such that $\#E_A(\mathbb{F}_p) = p + 1$ is supersingular.

$A = -3$:



Interlude: supersingular elliptic curves and isogenies

Edges: 3-, 5-, and 7-isogenies.

- An isogeny $E_A \to E_B$ is a non-zero morphism the preserves P_∞ ('nice map' given by rational maps).

Interlude: supersingular elliptic curves and isogenies

Edges: 3-, 5-, and 7-isogenies.

- An isogeny $E_A \to E_B$ is a non-zero morphism the preserves P_∞ ('nice map' given by rational maps).
- For $\ell \neq p$ (= 419 here), an ℓ-isogeny $f : E_A \to E_B$ is an isogeny with $\#\ker(f) = \ell$.

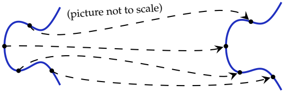
Interlude: supersingular elliptic curves and isogenies

Edges: 3-, 5-, and 7-isogenies.

- An isogeny $E_A \to E_B$ is a non-zero morphism the preserves P_∞ ('nice map' given by rational maps).
- For $\ell \neq p$ (= 419 here), an ℓ-isogeny $f : E_A \to E_B$ is an isogeny with $\#\ker(f) = \ell$.
- Every ℓ-isogeny $f : E_A \to E_B$ has a unique dual ℓ-isogeny $f : E_B \to E_A$.

Graphs of elliptic curves

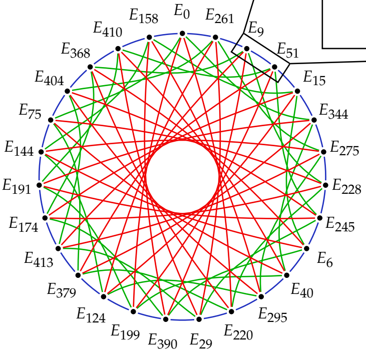
A 3-isogeny



$$E_{51}\colon y^2=x^3+51x^2+x \longrightarrow E_9\colon y^2=x^3+9x^2+x$$

$$(x,y) \longmapsto \left(\frac{97x^3-183x^2+x}{x^2-183x+97},\; y\cdot\frac{133x^3+154x^2-5x+97}{-x^3+65x^2+128x-133}\right)$$

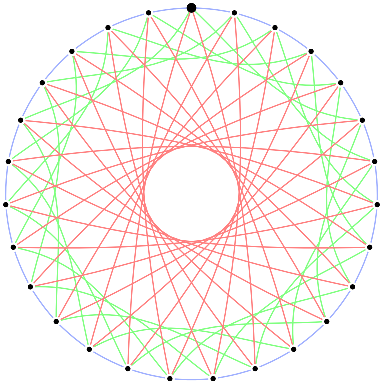
(...and its dual isogeny)



Diffie-Hellman on isogeny graphs

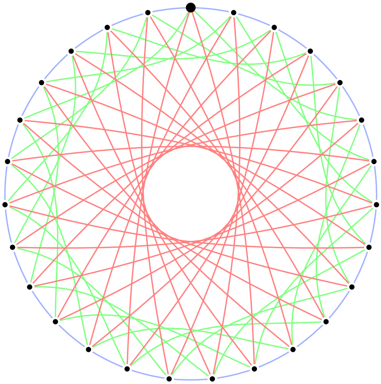
Alice

$[{\color{blue}+}, {\color{green}-}, {\color{blue}+}, {\color{red}-}]$



Bob

$[{\color{red}+}, {\color{green}+}, {\color{blue}-}, {\color{red}+}]$



Diffie-Hellman on isogeny graphs

Alice

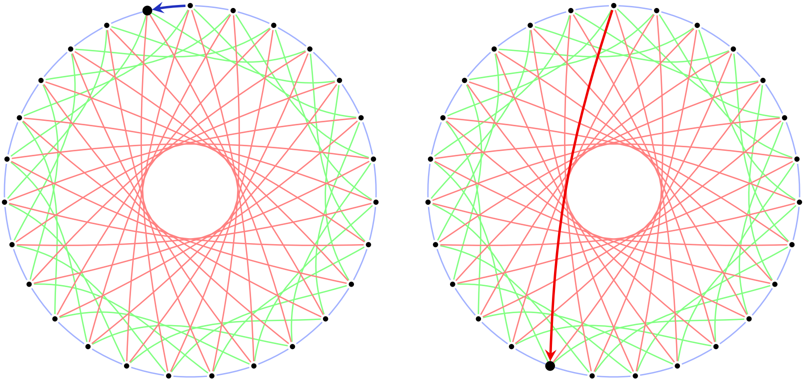
$[+, -, +, -]$

$\uparrow$

Bob

$[+, +, -, +]$

$\uparrow$



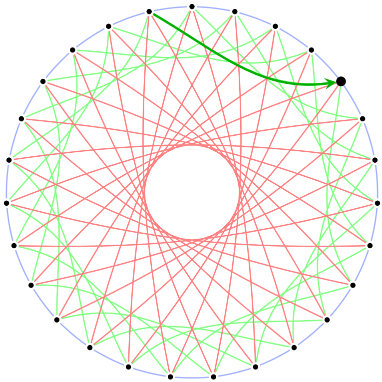
Diffie-Hellman on isogeny graphs

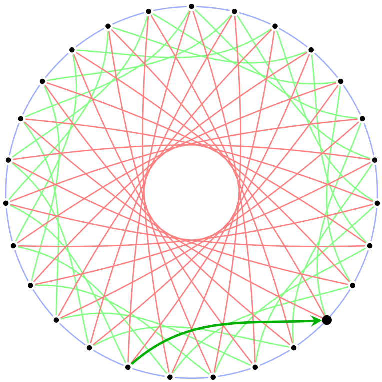
Alice

$[+, -, +, -]$
$\uparrow$

Bob

$[+, +, -, +]$
$\uparrow$





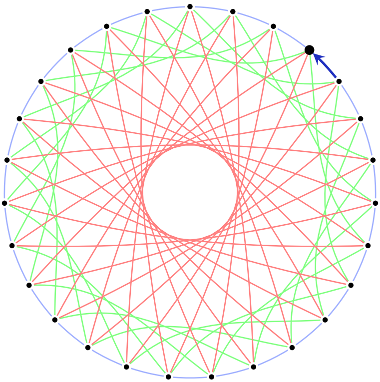
Diffie-Hellman on isogeny graphs

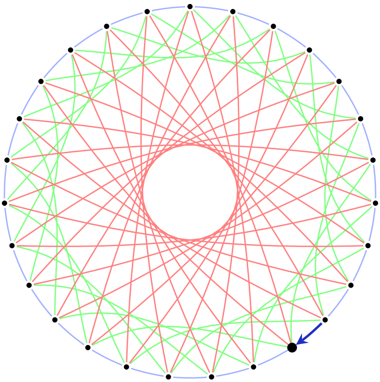
Alice

$[+, -, +, -]$
$\uparrow$

Bob

$[+, +, -, +]$
$\uparrow$

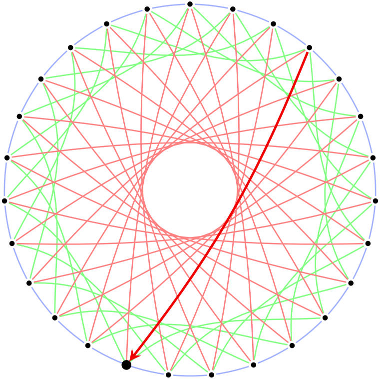




Diffie-Hellman on isogeny graphs

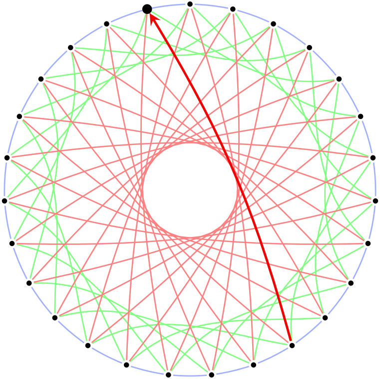
Alice

$[+, -, +, -]$
$\qquad\qquad\uparrow$



Bob

$[+, +, -, +]$
$\qquad\qquad\uparrow$



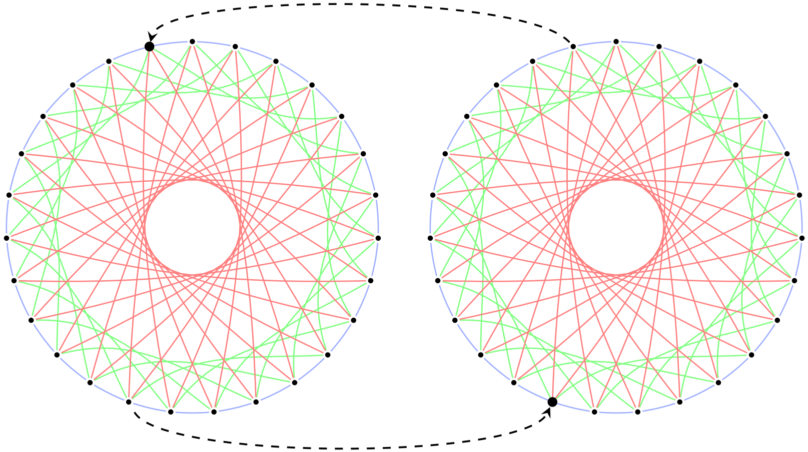
Diffie-Hellman on isogeny graphs

Alice

$[+, -, +, -]$

Bob

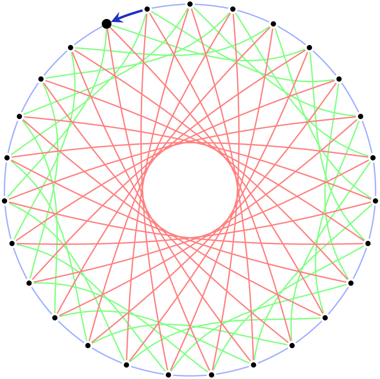
$[+, +, -, +]$



Diffie-Hellman on isogeny graphs

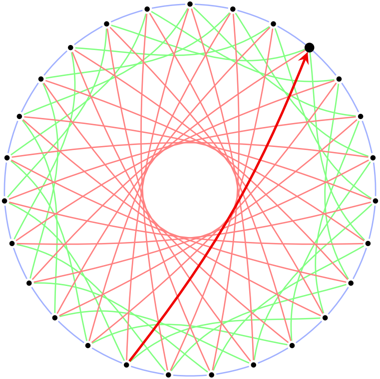
Alice

$[+, -, +, -]$
$\uparrow$



Bob

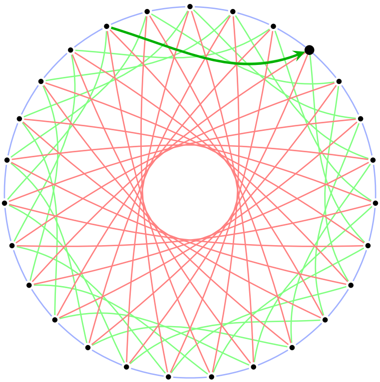
$[+, +, -, +]$
$\uparrow$



Diffie-Hellman on isogeny graphs

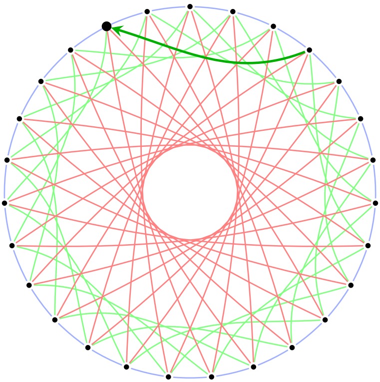
Alice

$[+, -, +, -]$



Bob

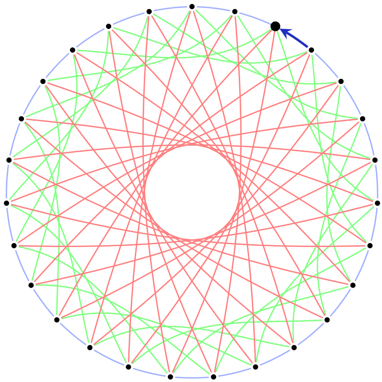
$[+, +, -, +]$



Diffie-Hellman on isogeny graphs

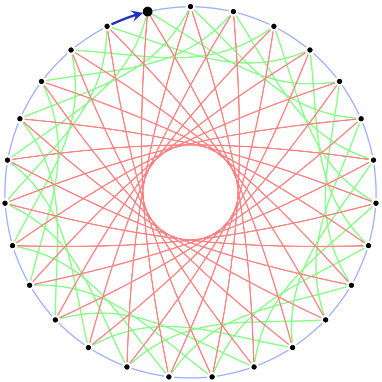
Alice

$[+, -, +, -]$
$\uparrow$



Bob

$[+, +, -, +]$
$\uparrow$



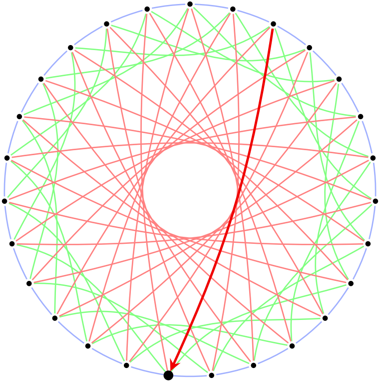
Diffie-Hellman on isogeny graphs

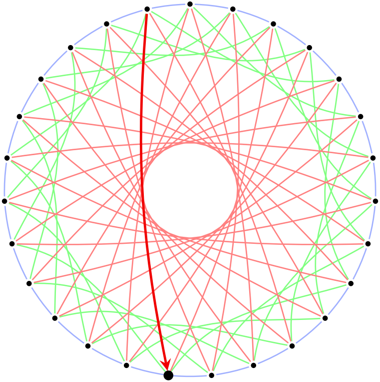
Alice

$[+, -, +, -]$
$\uparrow$

Bob

$[+, +, -, +]$
$\uparrow$

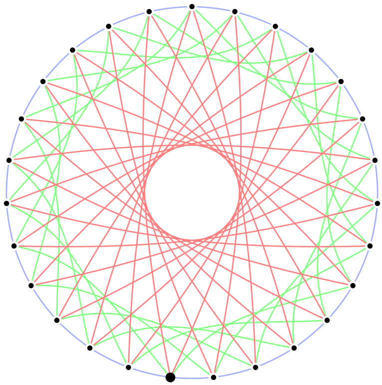




Diffie-Hellman on isogeny graphs

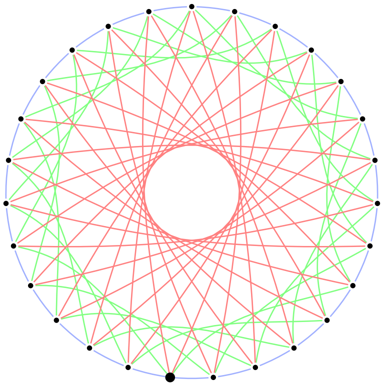
Alice

$[+, -, +, -]$



Bob

$[+, +, -, +]$



A walkable graph

Important properties for our graph:

- IP1 ▸ The graph is a composition of compatible cycles.
- IP2 ▸ We can compute neighbours in given directions.

IP1: A composition of cycles

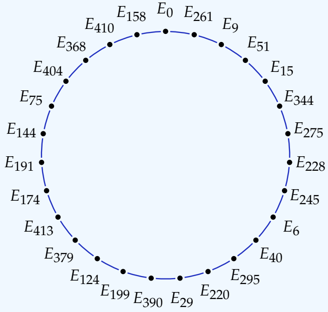
- The graph used in CSIDH is constructed as a composition of graphs G_ℓ of ℓ-isogenies.

IP1: A composition of cycles

- The graph used in CSIDH is constructed as a composition of graphs G_ℓ of ℓ-isogenies.

- In our example, these are

G_3:

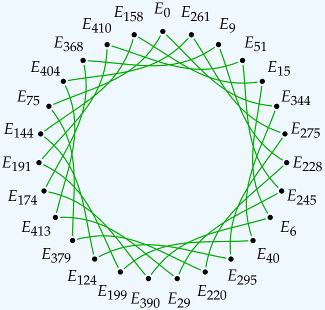


IP1: A composition of cycles

- The graph used in CSIDH is constructed as a composition of graphs G_ℓ of ℓ-isogenies.

- In our example, these are

G_5:

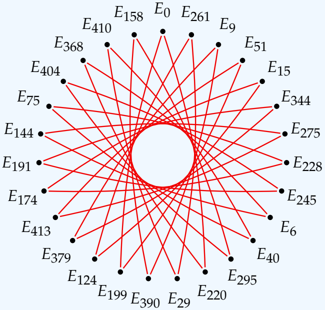


IP1: A composition of cycles

- The graph used in CSIDH is constructed as a composition of graphs G_ℓ of ℓ-isogenies.

- In our example, these are

G_{7}:

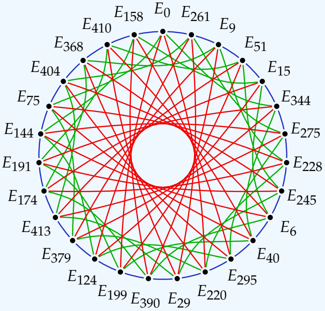


IP1: A composition of cycles

- The graph used in CSIDH is constructed as a composition of graphs G_ℓ of ℓ-isogenies.

- In our example, these are

$G_3 \cup G_5 \cup G_7$:

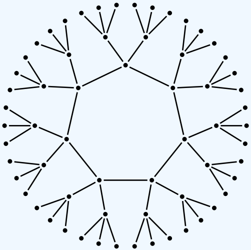


IP1: A composition of cycles

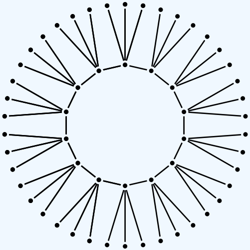
- The graph used in CSIDH is constructed as a composition of graphs G_ℓ of ℓ-isogenies.

- Generally, the G_ℓ look something like

G_3:



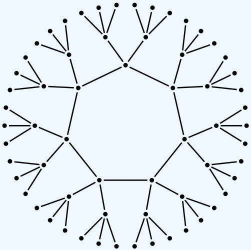
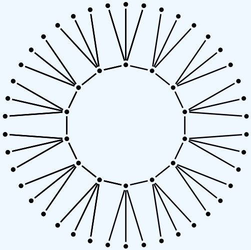
G_5:



IP1: A composition of cycles

- The graph used in CSIDH is constructed as a composition of graphs G_ℓ of ℓ-isogenies.

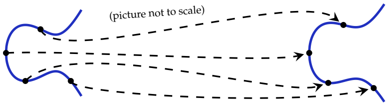
- Generally, the G_ℓ look something like

G_3:  G_5: 

- We want to make sure G_ℓ is just a cycle.

IP2: Compute neighbours in given directions

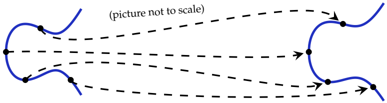
The edges of G_ℓ are ℓ-isogenies.



$$\begin{aligned} E_{51} : y^2 = x^3 + 51x^2 + x &\longrightarrow E_9 : y^2 = x^3 + 9x^2 + x \\ (x, y) &\longmapsto \left(\frac{97x^3 - 183x^2 + x}{x^2 - 183x + 97}, y \cdot \frac{133x^3 + 154x^2 - 5x + 97}{-x^3 + 65x^2 + 128x - 133} \right) \end{aligned}$$

IP2: Compute neighbours in given directions

The edges of G_ℓ are ℓ-isogenies.

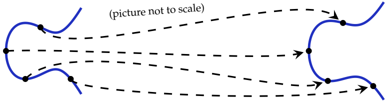


$$\begin{aligned} E_{51}: y^2 = x^3 + 51x^2 + x &\longrightarrow E_9: y^2 = x^3 + 9x^2 + x \\ (x, y) &\longmapsto \left(\frac{97x^3 - 183x^2 + x}{x^2 - 183x + 97}, y \cdot \frac{133x^3 + 154x^2 - 5x + 97}{-x^3 + 65x^2 + 128x - 133}\right) \end{aligned}$$

- The orientation of G_ℓ is mathematically well-defined (canonical way to compute the 'left' or 'right' isogeny).

IP2: Compute neighbours in given directions

The edges of G_ℓ are ℓ-isogenies.

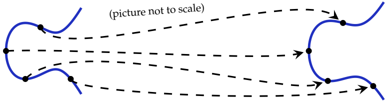


$$\begin{aligned} E_{51}: y^2 = x^3 + 51x^2 + x &\longrightarrow E_9: y^2 = x^3 + 9x^2 + x \\ (x, y) &\longmapsto \left(\tfrac{97x^3 - 183x^2 + x}{x^2 - 183x + 97}, y \cdot \tfrac{133x^3 + 154x^2 - 5x + 97}{-x^3 + 65x^2 + 128x - 133}\right) \end{aligned}$$

- The orientation of G_ℓ is mathematically well-defined (canonical way to compute the 'left' or 'right' isogeny).
- The cost grows with $\ell \rightsquigarrow$ want small ℓ.

IP2: Compute neighbours in given directions

The edges of G_ℓ are ℓ-isogenies.



$$\begin{aligned} E_{51} : y^2 = x^3 + 51x^2 + x \quad &\longrightarrow \quad E_9 : y^2 = x^3 + 9x^2 + x \\ (x, y) \quad &\longmapsto \quad \left(\tfrac{97x^3 - 183x^2 + x}{x^2 - 183x + 97}, y \cdot \tfrac{133x^3 + 154x^2 - 5x + 97}{-x^3 + 65x^2 + 128x - 133} \right) \end{aligned}$$

- The orientation of G_ℓ is mathematically well-defined (canonical way to compute the 'left' or 'right' isogeny).
- The cost grows with ℓ $\rightsquigarrow$ want small ℓ.
- Generally needs big extension fields...

Solution

1. - Choose some small odd primes $\ell_1, \ldots, \ell_n$.

Solution

1.
   - Choose some small odd primes $\ell_1, \ldots, \ell_n$.
   - Make sure $p = 4 \cdot \ell_1 \cdots \ell_n - 1$ is prime.

Solution

1.
- Choose some small odd primes $\ell_1, \ldots, \ell_n$.
- Make sure $p = 4 \cdot \ell_1 \cdots \ell_n - 1$ is prime.
- Fix the curve $E_0 \colon y^2 = x^3 + x$ over $\mathbb{F}_p$.

Solution

1.
   - Choose some small odd primes $\ell_1, \ldots, \ell_n$.
   - Make sure $p = 4 \cdot \ell_1 \cdots \ell_n - 1$ is prime.
   - Fix the curve $E_0 \colon y^2 = x^3 + x$ over $\mathbb{F}_p$.

2.
   - E_0 is supersingular $\rightsquigarrow$ has $p + 1$ points.

Solution

1.
- Choose some small odd primes $\ell_1, \ldots, \ell_n$.
- Make sure $p = 4 \cdot \ell_1 \cdots \ell_n - 1$ is prime.
- Fix the curve $E_0 \colon y^2 = x^3 + x$ over $\mathbb{F}_p$.

2.
- E_0 is supersingular $\rightsquigarrow$ has $p + 1$ points.
- Let the nodes of G_{ℓ_i} be those E_A with $p + 1$ points.

Solution

1.
- Choose some small odd primes $\ell_1, \ldots, \ell_n$.
- Make sure $p = 4 \cdot \ell_1 \cdots \ell_n - 1$ is prime.
- Fix the curve $E_0 \colon y^2 = x^3 + x$ over $\mathbb{F}_p$.

2.
- E_0 is supersingular $\rightsquigarrow$ has $p + 1$ points.
- Let the nodes of G_{ℓ_i} be those E_A with $p + 1$ points.
- Then every G_{ℓ_i} is a disjoint union of cycles.

Solution

1.
- Choose some small odd primes $\ell_1, \ldots, \ell_n$.
- Make sure $p = 4 \cdot \ell_1 \cdots \ell_n - 1$ is prime.
- Fix the curve $E_0 \colon y^2 = x^3 + x$ over $\mathbb{F}_p$.

2.
- E_0 is supersingular $\rightsquigarrow$ has $p + 1$ points.
- Let the nodes of G_{ℓ_i} be those E_A with $p + 1$ points.
- Then every G_{ℓ_i} is a disjoint union of cycles.
- All G_{ℓ_i} are compatible.

Solution

1.
   - Choose some small odd primes $\ell_1, \ldots, \ell_n$.
   - Make sure $p = 4 \cdot \ell_1 \cdots \ell_n - 1$ is prime.
   - Fix the curve $E_0 \colon y^2 = x^3 + x$ over $\mathbb{F}_p$.

2.
   - E_0 is supersingular $\rightsquigarrow$ has $p + 1$ points.
   - Let the nodes of G_{ℓ_i} be those E_A with $p + 1$ points.
   - Then every G_{ℓ_i} is a disjoint union of cycles.
   - All G_{ℓ_i} are compatible.
   - Computations need only $\mathbb{F}_p$-arithmetic (because $\ell_i | (p + 1)$).

Representing nodes of the graph

Side effect of magic:

- Every node of G_{ℓ_i} can be written as

$$E_A : y^2 = x^3 + Ax^2 + x.$$

Representing nodes of the graph

Side effect of magic:

- Every node of G_{ℓ_i} can be written as

$$E_A : y^2 = x^3 + Ax^2 + x.$$

$\Rightarrow$ Can compress every node to a single value $A \in \mathbb{F}_p$.

Representing nodes of the graph

Side effect of magic:

- Every node of G_{ℓ_i} can be written as

$$E_A : y^2 = x^3 + Ax^2 + x.$$

$\Rightarrow$ Can compress every node to a single value $A \in \mathbb{F}_p$.

$\Rightarrow$ Tiny keys!

Does any A work?

[1] This algorithm has a small chance of false positives, but we actually use a variant that *proves* that E_A has $p + 1$ points.

Does any A work?

No.

[1] This algorithm has a small chance of false positives, but we actually use a variant that *proves* that E_A has $p+1$ points.

Does any A work?

No.

- About $\sqrt{p}$ of all $A \in \mathbb{F}_p$ are valid keys.

[1] This algorithm has a small chance of false positives, but we actually use a variant that *proves* that E_A has $p + 1$ points.

Does any A work?

No.

- About $\sqrt{p}$ of all $A \in \mathbb{F}_p$ are valid keys.
- Public-key validation: Check that E_A has $p+1$ points.

  Easy Monte-Carlo algorithm: Pick random P on E_A and check $[p+1]P = \infty$.[1]

[1] This algorithm has a small chance of false positives, but we actually use a variant that *proves* that E_A has $p+1$ points.

Why CSIDH?

- Drop-in post-quantum replacement for Diffie-Hellman

Why CSIDH?

- Drop-in post-quantum replacement for Diffie-Hellman
- Non-interactive key exchange (full public-key validation); previously an open problem post-quantumly

Why CSIDH?

- Drop-in post-quantum replacement for Diffie-Hellman
- Non-interactive key exchange (full public-key validation); previously an open problem post-quantumly
- Smallest keys of all post-quantum key exchange proposals

Why CSIDH?

- Drop-in post-quantum replacement for Diffie-Hellman
- Non-interactive key exchange (full public-key validation); previously an open problem post-quantumly
- Smallest keys of all post-quantum key exchange proposals
- Competitive speed: $\sim 35\,\mathrm{ms}$ per operation

Why CSIDH?

- Drop-in post-quantum replacement for Diffie-Hellman
- Non-interactive key exchange (full public-key validation); previously an open problem post-quantumly
- Smallest keys of all post-quantum key exchange proposals
- Competitive speed: $\sim 35\,\text{ms}$ per operation
- Security is based on a well-studied mathematical problem (no added extra structure that could weaken security)

Why CSIDH?

- Drop-in post-quantum replacement for Diffie-Hellman
- Non-interactive key exchange (full public-key validation); previously an open problem post-quantumly
- Smallest keys of all post-quantum key exchange proposals
- Competitive speed: $\sim 35\,\mathrm{ms}$ per operation
- Security is based on a well-studied mathematical problem (no added extra structure that could weaken security)

Classical Security

- Security is based on the isogeny problem: given two elliptic curves, compute an isogeny between them.

Classical Security

- Security is based on the isogeny problem: given two elliptic curves, compute an isogeny between them.
- Say Alice's secret is isogeny is of degree $\ell_1^{e_1} \cdots \ell_n^{e_n}$. She knows the path, so can do only small degree isogeny computations, giving complexity $O(\sum e_i \ell_i)$.

Classical Security

- Security is based on the isogeny problem: given two elliptic curves, compute an isogeny between them.
- Say Alice's secret is isogeny is of degree $\ell_1^{e_1} \cdots \ell_n^{e_n}$. She knows the path, so can do only small degree isogeny computations, giving complexity $O(\sum e_i \ell_i)$. An attacker has to compute one isogeny of large degree.

Classical Security

- Security is based on the isogeny problem: given two elliptic curves, compute an isogeny between them.
- Say Alice's secret is isogeny is of degree $\ell_1^{e_1} \cdots \ell_n^{e_n}$. She knows the path, so can do only small degree isogeny computations, giving complexity $O(\sum e_i \ell_i)$. An attacker has to compute one isogeny of large degree.
- Alternative way of thinking about it: Alice has to compute the isogeny corresponding to one path from E_0 to E_A, whereas an attacker has compute all the possible paths from E_0.

Classical Security

- Security is based on the isogeny problem: given two elliptic curves, compute an isogeny between them.
- Say Alice's secret is isogeny is of degree $\ell_1^{e_1} \cdots \ell_n^{e_n}$. She knows the path, so can do only small degree isogeny computations, giving complexity $O(\sum e_i \ell_i)$. An attacker has to compute one isogeny of large degree.
- Alternative way of thinking about it: Alice has to compute the isogeny corresponding to one path from E_0 to E_A, whereas an attacker has compute all the possible paths from E_0.
- Best classical attacks are (variants of) meet-in-the-middle: Time $O(\sqrt[4]{p})$.

Quantum Security

- Shor's (polynomial time) algorithm does not apply because the nodes in the graph do not form a group.
- Best algorithms are Hidden-shift algorithms: Subexponential complexity (Kuperberg, Regev).

Quantum Security

- Shor's (polynomial time) algorithm does not apply because the nodes in the graph do not form a group.
- Best algorithms are Hidden-shift algorithms: Subexponential complexity (Kuperberg, Regev).
- Kuperberg's algorithm [Kup1] requires a subexponential number of queries, and a subexponential number of operations on a subexponential number of qubits.

Quantum Security

- Shor's (polynomial time) algorithm does not apply because the nodes in the graph do not form a group.
- Best algorithms are Hidden-shift algorithms: Subexponential complexity (Kuperberg, Regev).
- Kuperberg's algorithm [Kup1] requires a subexponential number of queries, and a subexponential number of operations on a subexponential number of qubits.
- Variant by Regev [Reg] uses polynomial number of qubits at the expense of time.

Quantum Security

- Shor's (polynomial time) algorithm does not apply because the nodes in the graph do not form a group.
- Best algorithms are Hidden-shift algorithms: Subexponential complexity (Kuperberg, Regev).
- Kuperberg's algorithm [Kup1] requires a subexponential number of queries, and a subexponential number of operations on a subexponential number of qubits.
- Variant by Regev [Reg] uses polynomial number of qubits at the expense of time.
- Kuperberg later [Kup2] gave more trade-off options for quantum and classical memory vs. time.

Quantum Security

- Shor's (polynomial time) algorithm does not apply because the nodes in the graph do not form a group.
- Best algorithms are Hidden-shift algorithms: Subexponential complexity (Kuperberg, Regev).

- Kuperberg's algorithm [Kup1] requires a subexponential number of queries, and a subexponential number of operations on a subexponential number of qubits.
- Variant by Regev [Reg] uses polynomial number of qubits at the expense of time.
- Kuperberg later [Kup2] gave more trade-off options for quantum and classical memory vs. time.
- Childs-Jao-Soukharev [CJS] applied Kuperberg/Regev to CRS – their attack also applies to CSIDH.

Quantum Security

- Shor's (polynomial time) algorithm does not apply because the nodes in the graph do not form a group.
- Best algorithms are Hidden-shift algorithms: Subexponential complexity (Kuperberg, Regev).

- Kuperberg's algorithm [Kup1] requires a subexponential number of queries, and a subexponential number of operations on a subexponential number of qubits.
- Variant by Regev [Reg] uses polynomial number of qubits at the expense of time.
- Kuperberg later [Kup2] gave more trade-off options for quantum and classical memory vs. time.
- Childs-Jao-Soukharev [CJS] applied Kuperberg/Regev to CRS – their attack also applies to CSIDH.
- Part of CJS attack computes many paths in superposition.

Quantum Security

- The exact cost of the Kuperberg/Regev/CJS attack is subtle – it depends on:
  - Choice of time/memory trade-off (Regev/Kuperberg)
  - Quantum evaluation of isogenies

  (and much more).

Quantum Security

- The exact cost of the Kuperberg/Regev/CJS attack is subtle – it depends on:
  - Choice of time/memory trade-off (Regev/Kuperberg)
  - Quantum evaluation of isogenies

  (and much more).
- Asymptotic complexity is relatively well understood [BIJ], [JLLR]

Quantum Security

- The exact cost of the Kuperberg/Regev/CJS attack is subtle – it depends on:
  - Choice of time/memory trade-off (Regev/Kuperberg)
  - Quantum evaluation of isogenies

  (and much more).
- Asymptotic complexity is relatively well understood [BIJ], [JLLR]
- [BLMP] gives full computer-verified simulation of quantum evaluation of isogenies $\rightsquigarrow$ concrete estimates for a given security level ('NIST level I')

Work in progress & future work

- Optimize the constant-time implementation of CSIDH.

Work in progress & future work

- Optimize the constant-time implementation of CSIDH.
- More applications of CSIDH (recall the many applications of classical Diffie-Hellman)!

  The tiny keys make CSIDH ideal for implementation on small devices.

Work in progress & future work

- Optimize the constant-time implementation of CSIDH.
- More applications of CSIDH (recall the many applications of classical Diffie-Hellman)!

  The tiny keys make CSIDH ideal for implementation on small devices.
- Explore different graph structures occuring for other curves/geometrical objects.

Work in progress & future work

- Optimize the constant-time implementation of CSIDH.
- More applications of CSIDH (recall the many applications of classical Diffie-Hellman)!

  The tiny keys make CSIDH ideal for implementation on small devices.
- Explore different graph structures occuring for other curves/geometrical objects.
- More applications exploiting new graph structures.


Thank you!

Parameters

CSIDH-log p	intended NIST level	public key size	private key size	time (full exchange)	cycles (full exchange)	stack memory	classical security
CSIDH-512	1	64 b	32 b	70 ms	212e6	4368 b	128
CSIDH-1024	3	128 b	64 b				256
CSIDH-1792	5	224 b	112 b				448

CSIDH vs SIDH?

Apart from mathematical background, SIDH and CSIDH actually have very little in common, and are likely to be useful for different applications.

Here is a comparison for (conjectured) NIST level 1:

	CSIDH	SIDH
Speed (NIST 1)	65ms (can be improved)	$\approx$ 10ms[2]
Public key size (NIST 1)	64B	378B
Key compression (speed)		$\approx$ 15ms
Key compression (size)		222B
Constant-time slowdown	$\approx \times 2.2$ (can be improved)	$\approx \times 1$
Submitted to NIST	no	yes
Maturity	11 months	8 years
Best classical attack	$p^{1/4}$	$p^{1/4}$
Best quantum attack	$L_p[1/2]$	$p^{1/4}$
Key size scales	quadratically	linearly
Security assumption	isogeny walk problem	ad hoc
Non-interactive key exchange	yes	unbearably slow
Signatures (classical)	unbearably slow[3]	seconds
Signatures (quantum)	seconds	still seconds?

[2] This is a very conservative estimate!

[3] Word on the street is that a paper is coming with a signature scheme that takes milliseconds.

References

AMW Appelbaum, Martindale, and Wu:
Tiny Wireguard Tweak
(upcoming)

BLMP Bernstein, Lange, Martindale, and Panny:
Quantum circuits for the CSIDH: optimizing quantum evaluation of isogenies
`https://quantum.isogeny.org` (Eurocrypt 2019)

CLMPR Castryck, Lange, Martindale, Panny, Renes:
CSIDH: An Efficient Post-Quantum Commutative Group Action
`https://ia.cr/2018/383` (Asiacrypt 2018)

DG De Feo, Galbraith:
SeaSign: Compact isogeny signatures from class group actions
`https://ia.cr/2018/824`

DGOPS Delpech de Saint Guilhem, Orsini, Petit, and Smart:
Secure Oblivious Transfer from Semi-Commutative Masking
`https://ia.cr/2018/648`

FTY Fujioka, Takashima, and Yoneyama:
One-Round Authenticated Group Key Exchange from Isogenies
`https://eprint.iacr.org/2018/1033`