Diffie-Hellman and its applications in a post-quantum world

Chloe Martindale
www.martindale.info

University of Bristol, UK, 13th March 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

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

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 $g \in G$:

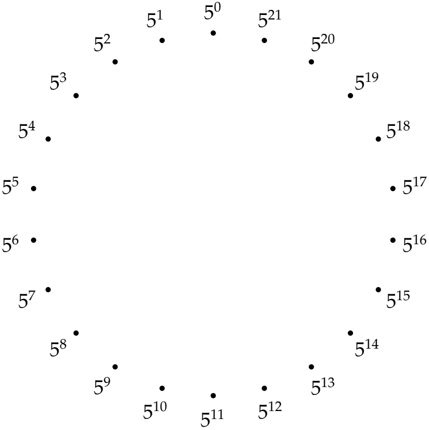
- Given $n \in \mathbb{Z}$, computing $\underbrace{g * \cdots * g}_{n \text{ times}}$ is fast. (eg. Polynomial time).
- Given $\underbrace{g * \cdots * g}_{n \text{ times}}$, computing n is slow. (eg. Exponential time).

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

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

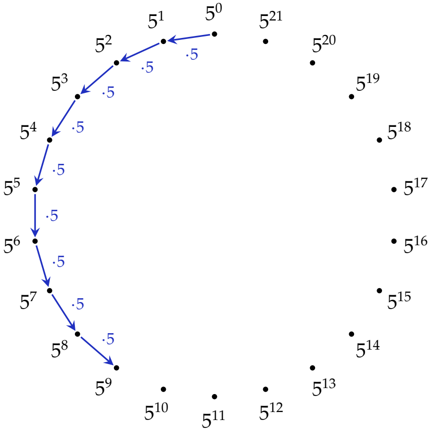
Square-and-multiply

Compute 5^9 mod 23.



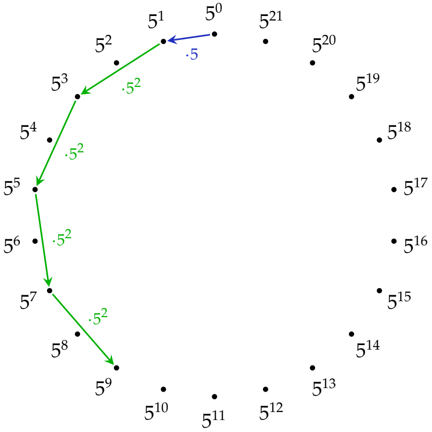
Square-and-multiply

Compute 5^9 mod 23.



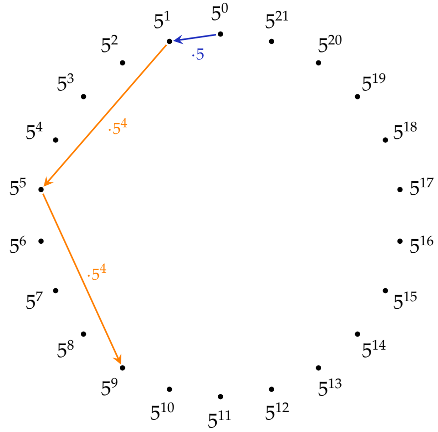
Square-and-multiply

Compute 5^9 mod 23.



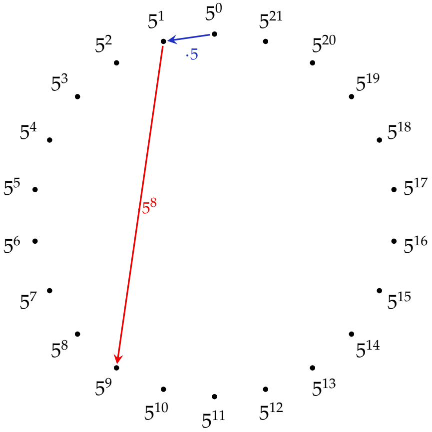
Square-and-multiply

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 mod 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:

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

$$5^3 \equiv 10 \not\equiv 11 \bmod 23$$

$$5^4 \equiv 4 \not\equiv 11 \bmod 23$$

$$5^5 \equiv 20 \not\equiv 11 \bmod 23$$

$$5^6 \equiv 8 \not\equiv 11 \bmod 23$$

$$5^7 \equiv 17 \not\equiv 11 \bmod 23$$

$$5^8 \equiv 16 \not\equiv 11 \bmod 23$$

$$5^9 \equiv 11 \bmod 23.$$

Square-and-multiply vs. solving DLP

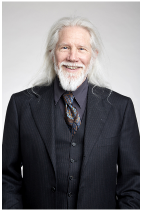
- To compute 5^9 mod 23, compute:
  $5 \cdot 5^8 = 5 \cdot ((5^2)^2)^2$ mod 23. (Fast).
- To compute n such that $5^n \equiv 11$ mod 23, check:

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

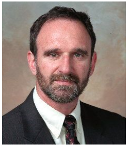
(Slow).

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

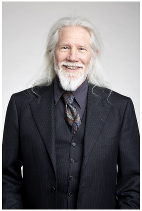
Application of DLP: Diffie-Hellman key exchange



$g \in G$

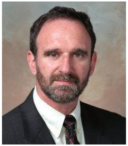


Application of DLP: Diffie-Hellman key exchange



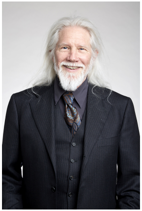
Secret key: d

$g \in G$



Secret key: h

Application of DLP: Diffie-Hellman key exchange

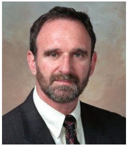


Secret key: d

$g \in G$

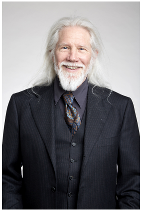
$\xrightarrow{\text{Public key: } g^d}$

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

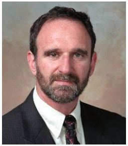


Secret key: h

Application of DLP: Diffie-Hellman key exchange



$g \in G$



Secret key: d

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

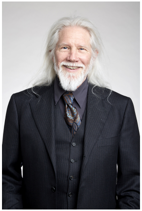
Secret key: h

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

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

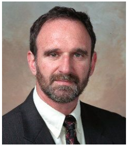
Shared secret: $s = (g^d)^h$

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

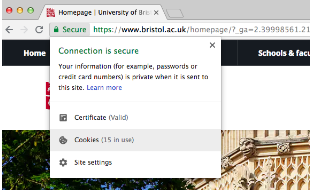
Shared secret: $s = (g^d)^h$

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:

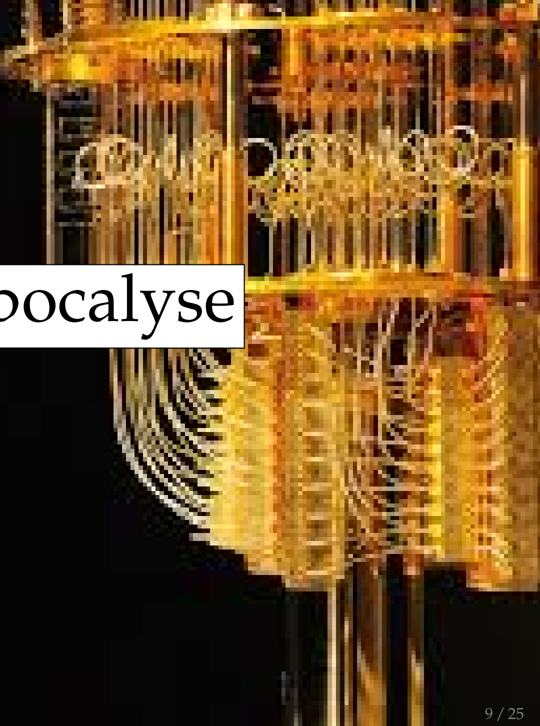
- Digital signature schemes (used for example by some online banking apps; secure websites).
- Encrypted messaging services (eg. WhatsApp; Signal; WireGuard).



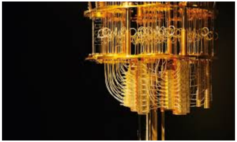
Connection is secure

Your information (for example, passwords or credit card numbers) is private when it is sent to this site. Learn more

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
Disney
~~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, Signal, WireGuard).

---

[1]

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, Signal, WireGuard).
- The WireGuard protocol's special preconditions $\rightsquigarrow$ one-line fix to protect our current messages against future quantum computers. [1]

---

[1] Recent joint work with Jacob Appelbaum and Peter Wu [AMW19].

Reminder: applications of Diffie-Hellman key exchange

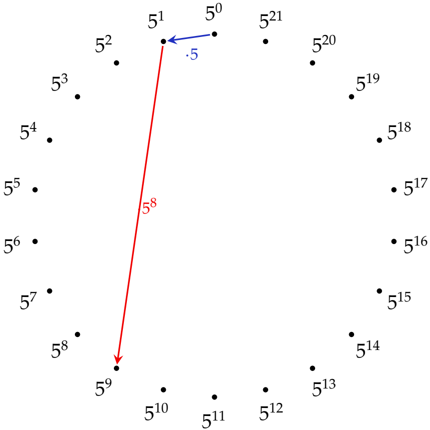
- The Diffie-Hellman key exchange (and hence DLP) is a building block in:
  - Digital signature schemes (used for example by some online banking apps; secure websites).
  - Encrypted messaging services (eg. WhatsApp, Signal, WireGuard).
- The WireGuard protocol's special preconditions $\rightsquigarrow$ one-line fix to protect our current messages against future quantum computers. [1]
- For most other applications, we need a post-quantum Diffie-Hellman-style key exchange.

---

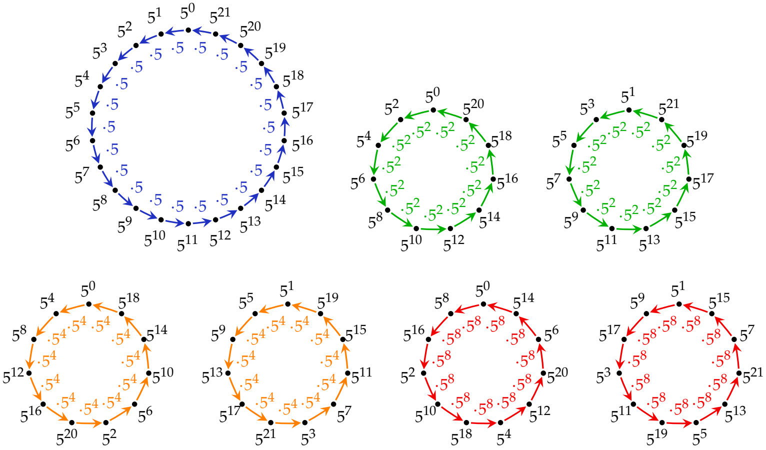
[1] Recent joint work with Jacob Appelbaum and Peter Wu [AMW19].

Square-and-multiply

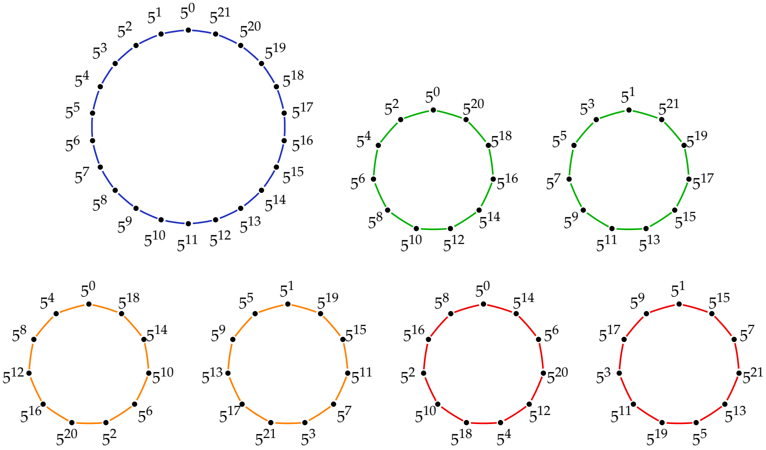
Reminder: how to compute 5^9 mod 23.



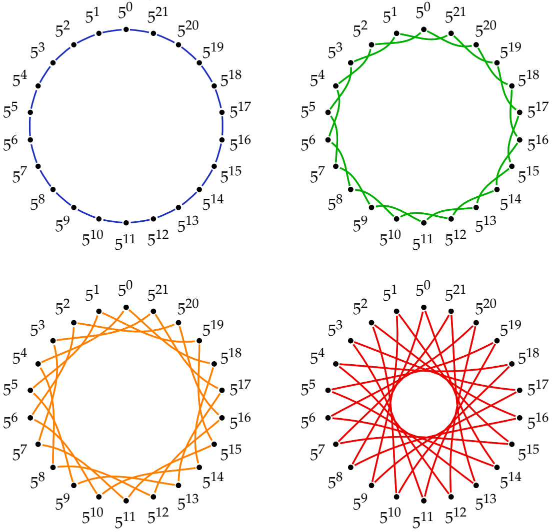
Square-and-multiply



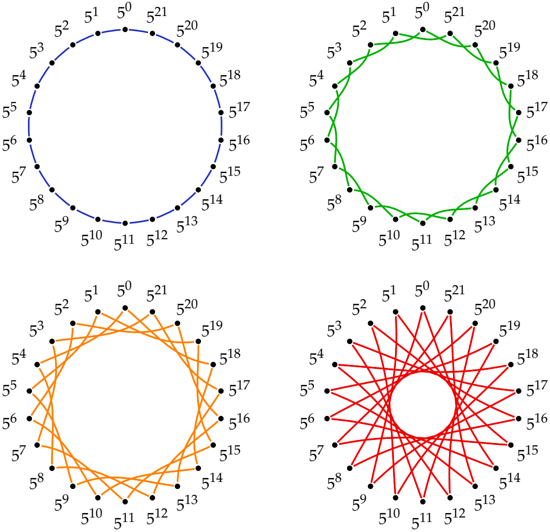
Square-and-multiply



Square-and-multiply

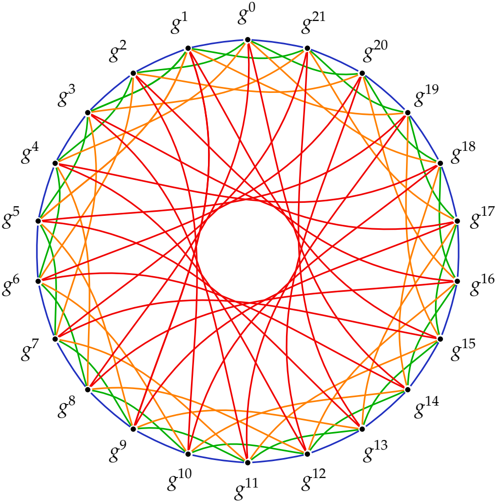


Square-and-multiply

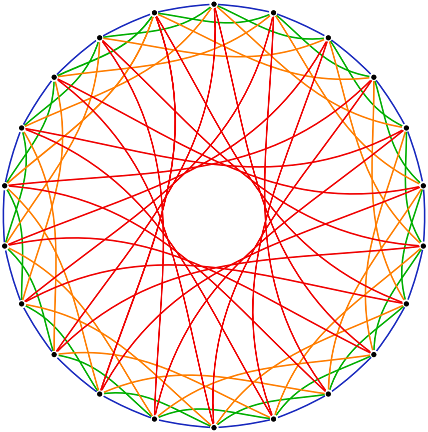


Needed for Diffie-Hellman: Cycles are compatible–
[right, then left] = [left, then right], etc. (Else $(5^a)^b \neq (5^b)^a$).

Union of cycles: rapid mixing

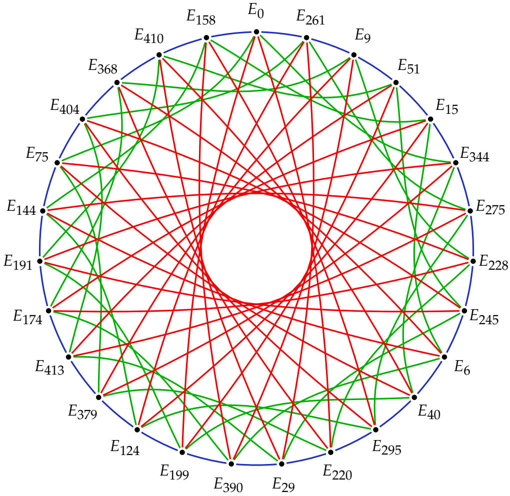


Union of cycles: rapid mixing

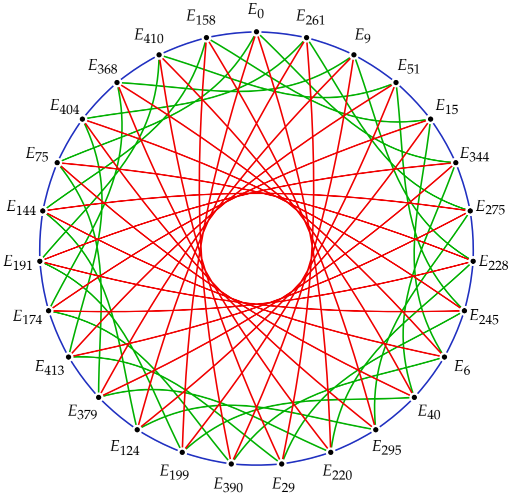


Post-quantum Diffie-Hellman: Nodes are now elliptic curves and edges are isogenies.

Graphs of elliptic curves

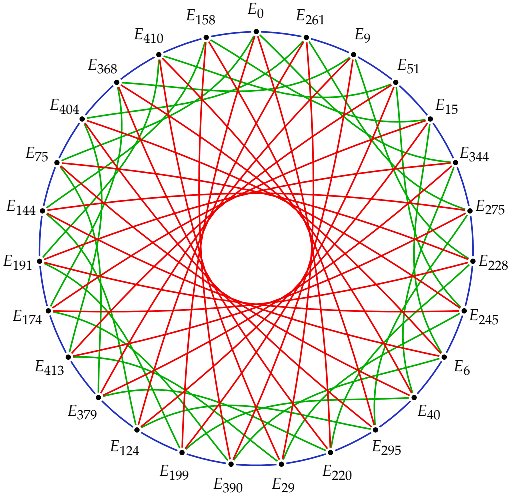


Graphs of elliptic curves



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

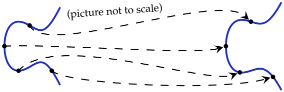
Graphs of elliptic curves



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.

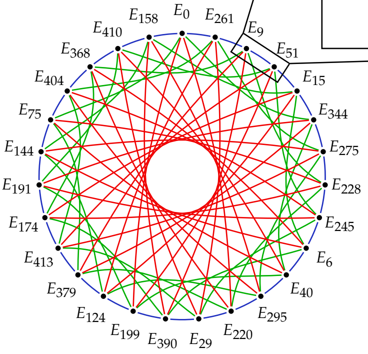
Graphs of elliptic curves

A 3-isogeny



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

(...and its dual isogeny)



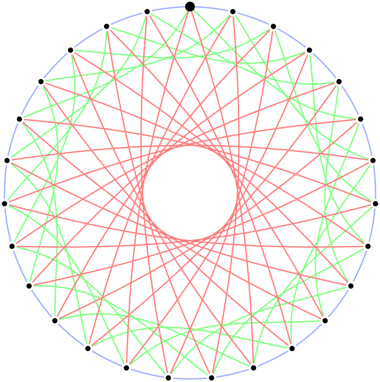
Diffie-Hellman on isogeny graphs

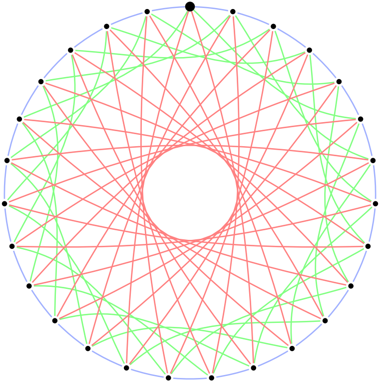
Alice

$[+, -, +, -]$

Bob

$[+, +, -, +]$





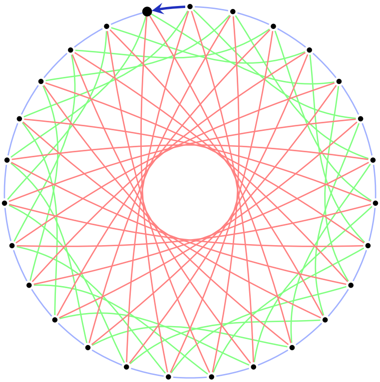
Diffie-Hellman on isogeny graphs

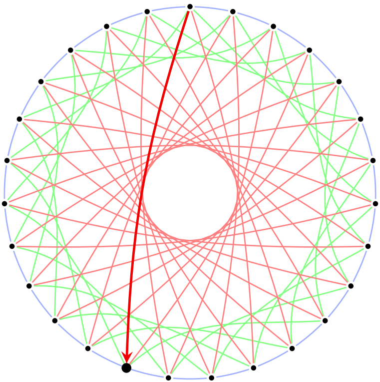
Alice

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

Bob

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

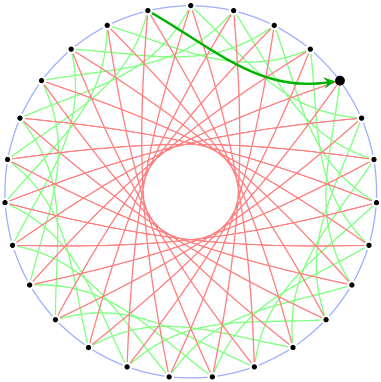




Diffie-Hellman on isogeny graphs

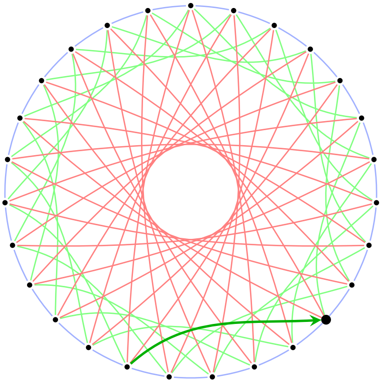
Alice

$[+, \underset{\uparrow}{-}, +, -]$



Bob

$[+, \underset{\uparrow}{+}, -, +]$



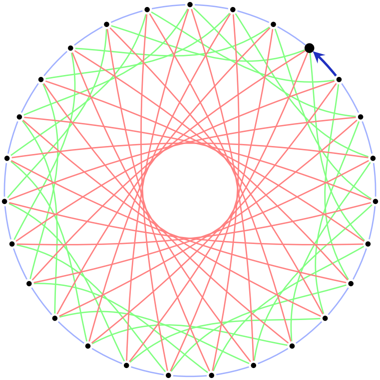
Diffie-Hellman on isogeny graphs

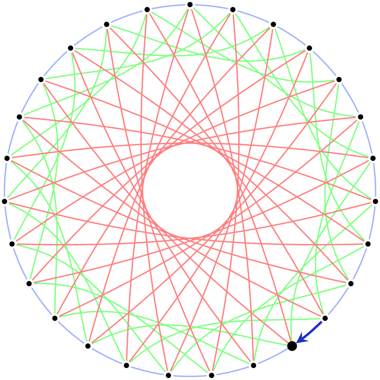
Alice

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

Bob

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





Diffie-Hellman on isogeny graphs

Alice

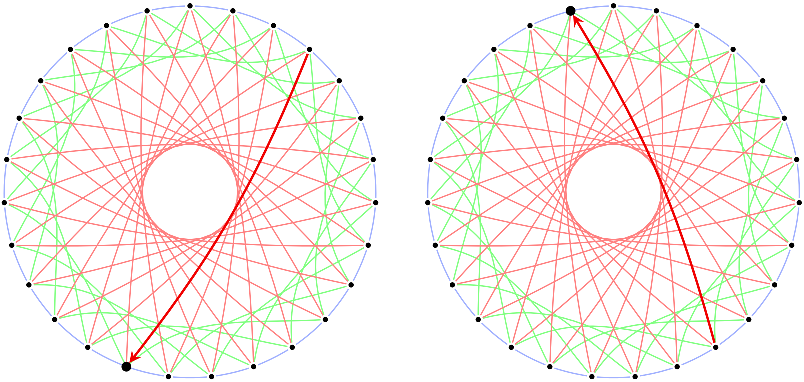
$[+, -, +, -]$

$\uparrow$

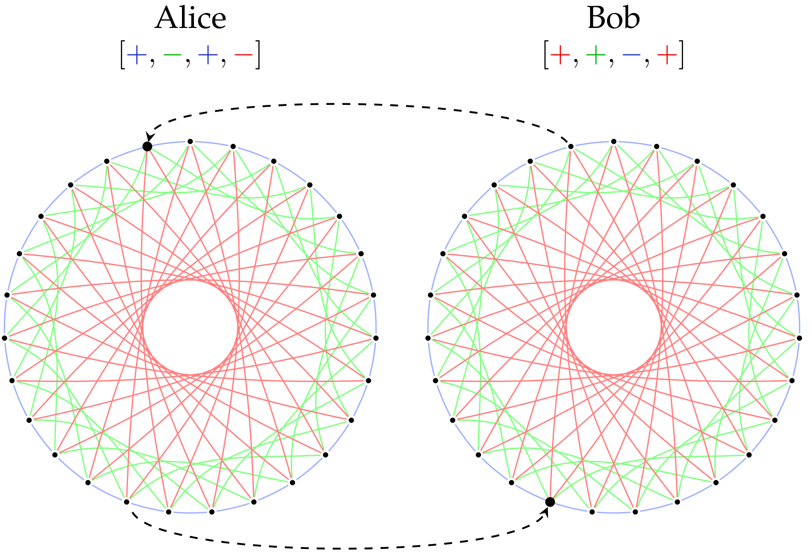
Bob

$[+, +, -, +]$

$\uparrow$



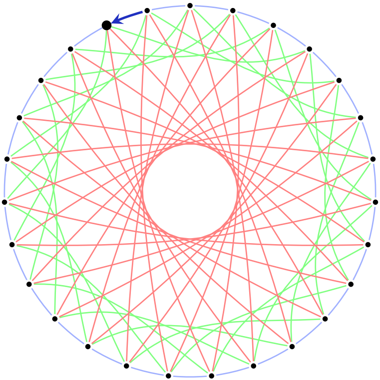
Diffie-Hellman on isogeny graphs



Diffie-Hellman on isogeny graphs

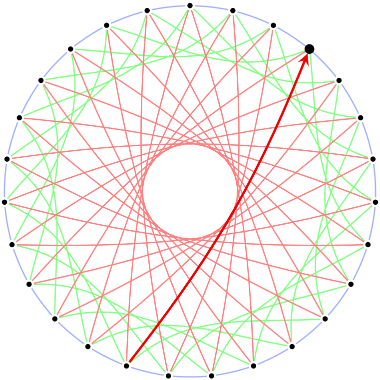
Alice

$[+, -, +, -]$
↑



Bob

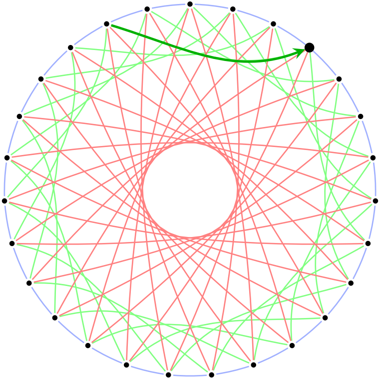
$[+, +, -, +]$
↑



Diffie-Hellman on isogeny graphs

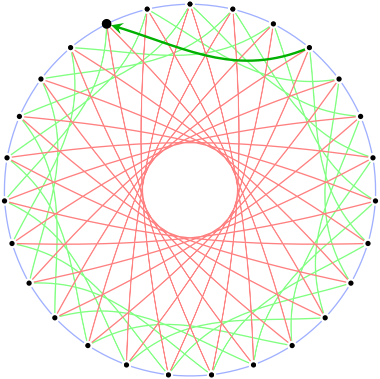
Alice

$[+, \underset{\uparrow}{-}, +, -]$



Bob

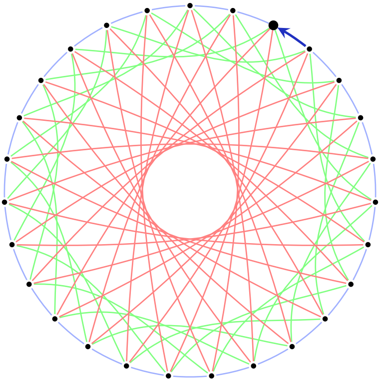
$[+, \underset{\uparrow}{+}, -, +]$



Diffie-Hellman on isogeny graphs

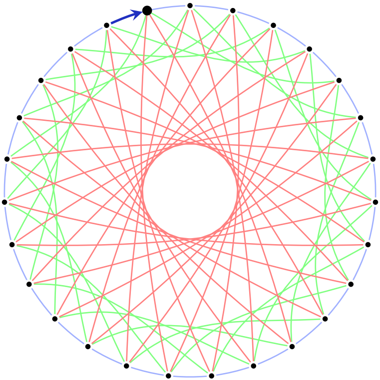
Alice

$[+, -, \underset{\uparrow}{+}, -]$



Bob

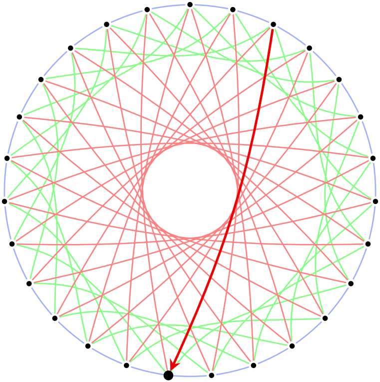
$[+, +, \underset{\uparrow}{-}, +]$



Diffie-Hellman on isogeny graphs

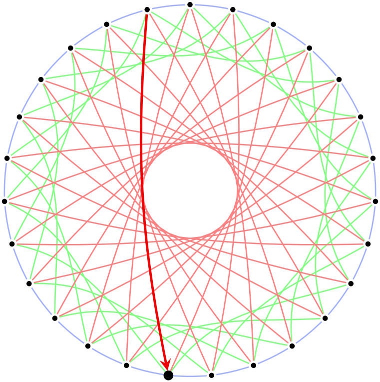
Alice

[+, −, +, −]
↑



Bob

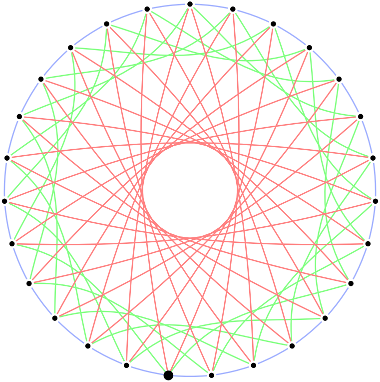
[+, +, −, +]
↑



Diffie-Hellman on isogeny graphs

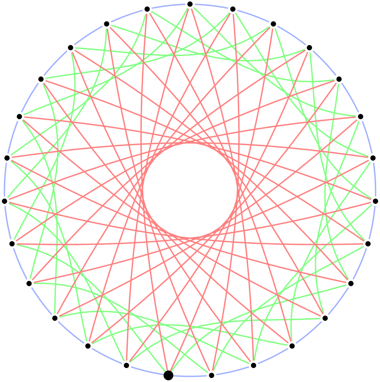
Alice

$[+, -, +, -]$



Bob

$[+, +, -, +]$



Post-quantum Diffie-Hellman key exchange

- Shor's quantum algorithm does not apply to the set of nodes E_A of the graph because they do not form a group.

Post-quantum Diffie-Hellman key exchange

- Shor's quantum algorithm does not apply to the set of nodes E_A of the graph because they do not form a group.
- In [CLMPR18] we show how to construct such examples, ie. where:
  - The graph is a composition of compatible cycles.
  - We can efficiently compute neighbours in given directions.

Post-quantum Diffie-Hellman key exchange

- Shor's quantum algorithm does not apply to the set of nodes E_A of the graph because they do not form a group.
- In [CLMPR18] we show how to construct such examples, ie. where:
  - The graph is a composition of compatible cycles.
  - We can efficiently compute neighbours in given directions.
- We give parameters for secure post-quantum non-interactive key exchange using this graph.

Commutative Supersingular Isogeny Diffie- Hellman

[ˈsiːˌsaɪd]

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
- Small keys: 64 bytes at conjectured AES-128 security level (smallest 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
- Small keys: 64 bytes at conjectured AES-128 security level (smallest 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
- Small keys: 64 bytes at conjectured AES-128 security level (smallest 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)

Why CSIDH?

- Drop-in post-quantum replacement for Diffie-Hellman
- Non-interactive key exchange (full public-key validation); previously an open problem post-quantumly
- Small keys: 64 bytes at conjectured AES-128 security level (smallest 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)
- Flexible:
  - [DG] uses CSIDH for 'SeaSign' signatures
  - [DGOPS] uses CSIDH for oblivious transfer
  - [FTY] uses CSIDH for authenticated group key exchange

Parameters

CSIDH-log p	intended NIST level[2]	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

---

[2] For the NIST level 1 parameters, in [BLMP18] we built a simulator that counts the number of bit operations in order to to analyze the fastest known quantum attack.

Work in progress & future work

- Fast and constant-time implementation of CSIDH. (We already introduced some ideas for optimizing a constant-time optimization in [BLMP]).

Work in progress & future work

- Fast and constant-time implementation of CSIDH. (We already introduced some ideas for optimizing a constant-time optimization in [BLMP]).
- 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

- Fast and constant-time implementation of CSIDH. (We already introduced some ideas for optimizing a constant-time optimization in [BLMP]).
- 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

- Fast and constant-time implementation of CSIDH. (We already introduced some ideas for optimizing a constant-time optimization in [BLMP]).
- 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.

  One aim: find a post-quantum isogeny-based bilinear map

  $\rightsquigarrow$ identity-based encryption?

Thank you!

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)	70ms (can be improved)	$\approx$ 10ms[3]
Public key size (NIST 1)	64B	378B
Key compression (speed)		$\approx$ 15ms
Key compression (size)		222B
Constant-time slowdown	$\approx \times 3$ (can be improved)	$\approx \times 1$
Submitted to NIST	no	yes
Maturity	9 months	8 years
Best classical attack	$p^{1/4}$	$p^{1/4}$
Best quantum attack	$L_p[1/2]$	$p^{1/6}$
Key size scales	quadratically	linearly
Security assumption	isogeny walk problem	ad hoc
Non-interactive key exchange	yes	unbearably slow
Signatures (classical)	unbearably slow	seconds
Signatures (quantum)	seconds	still seconds?

---

[3] This is a very conservative estimate!

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`