Elliptic curves for future-proof cryptography

Tanja Lange

Eindhoven University of Technology; Academia Sinica

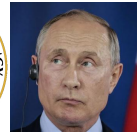
24 June 2022

Cryptography

Sender
“Alice”

Receiver
“Bob”

Cryptography



Sender
"Alice"

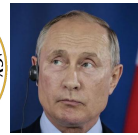
Untrustworthy network
"Eve"

Receiver
"Bob"

- Motivation #1: Communication channels are spying on our data.
- Motivation #2: Communication channels are modifying our data.

Cryptography

Sender
"Alice"



Untrustworthy network
"Eve"

Receiver
"Bob"

- Motivation #1: Communication channels are spying on our data.
- Motivation #2: Communication channels are modifying our data.
- Literal meaning of cryptography: "secret writing".
- Achieves various security goals by secretly transforming messages.
  - Confidentiality: Eve cannot infer information about the content
  - Integrity: Eve cannot modify the message without this being noticed
  - Authenticity: Bob is convinced that the message originated from Alice

Public-key vs. symmetric-key cryptography

Public-key cryptography
Each user has 2 keys:
a public key and a private key.

Public key can be posted online;
private key must be kept secret.

Often can compute public key from private key.
Other direction must be hard.

Can be used on Internet with unknown parties.
Requires mathematically hard problem.

Symmetric-key cryptography
Each pair of users shares a key.
This key is symmetric between both users.

This key must be kept secret.

Symmetric systems often faster than
public-key systems.
Use latter to get symmetric key.

Requires that users have securely shared this key.
Typically cheaper/faster than public-key crypto.

Current state of the art

- Currently used crypto (check the lock icon in your browser) starts with RSA (can be broken by factoring large integers), Diffie-Hellman in finite fields, or elliptic-curve Diffie-Hellman (both require the attacker to compute discrete logarithms in some group).
- Older standards are RSA or elliptic curves from NIST (or Brainpool), e.g. NIST P256 or ECDSA.
- Internet currently moving over to Curve25519 and Ed25519
- For symmetric crypto, TLS (the protocol behind https) uses AES or ChaCha20 and some MAC, e.g. AES-GCM or ChaCha20-Poly1305.
  High-end devices have support for AES-GCM, smaller ones do better with ChaCha20-Poly1305.
- Security is getting better. Some obstacles: bugs; untrustworthy hardware.
- Some countries make ill-advised recommendations to weaken crypto.

Algorithms for Quantum Computation: Discrete Logarithms and Factoring

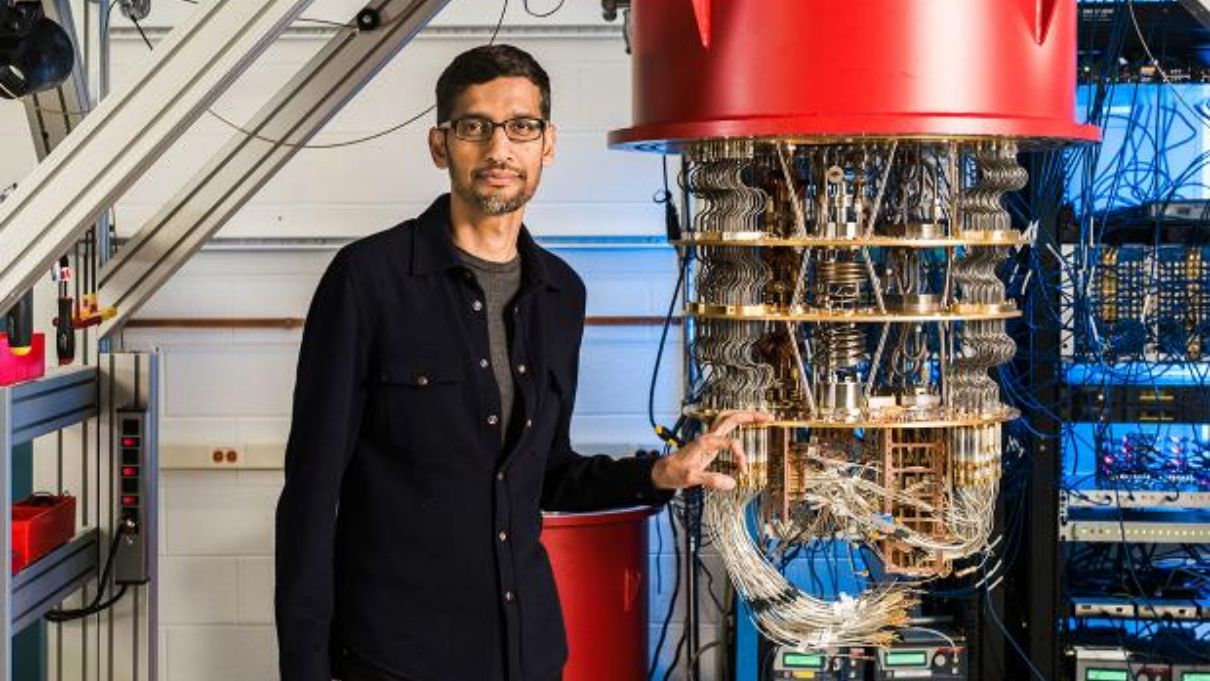
Peter W. Shor
AT&T Bell Labs
Room 2D-149
600 Mountain Ave.
Murray Hill, NJ 07974, USA

Abstract

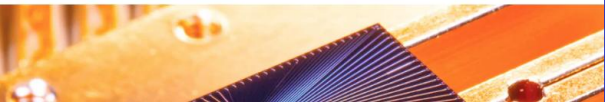
A computer is generally considered to be a universal computational device; i.e., it is believed able to simulate any physical computational device with a cost in computation time of at most a polynomial factor. It is not clear whether this is still true when quantum mechanics is taken into consideration. Several researchers, starting with David Deutsch, have developed models for quantum mechanical computers and have investigated their compu-

[1, 2]. Although he did not ask whether quantum mechanics conferred extra power to computation, he did show that a Turing machine could be simulated by the reversible unitary evolution of a quantum process, which is a necessary prerequisite for quantum computation. Deutsch [9, 10] was the first to give an explicit model of quantum computation. He defined both quantum Turing machines and quantum circuits and investigated some of their properties.

The next part of this paper discusses how quantum com-



Quantum computing could end encryption within five years, says Google boss



Mr Pichai said a combination of artificial intelligence and quantum would "help us tackle some of the biggest problems we see", but said it was important encryption evolved to match this.

"In a five to ten year time frame, quantum computing will break encryption as we know it today."

This is because current encryption methods, by which information such as texts or passwords is turned into code to make it unreadable, rely upon the fact that classic computers would take billions of years to decipher that code.

Quantum computers, with their ability to be

National Academy of Sciences (US)

4 December 2018: Report on quantum computing

Don't panic. "Key Finding 1: Given the current state of quantum computing and recent rates of progress, it is highly unexpected that a quantum computer that can compromise RSA 2048 or comparable discrete logarithm-based public key cryptosystems will be built within the next decade."

National Academy of Sciences (US)

4 December 2018: Report on quantum computing

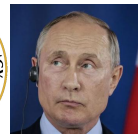
Don't panic. "Key Finding 1: Given the current state of quantum computing and recent rates of progress, it is highly unexpected that a quantum computer that can compromise RSA 2048 or comparable discrete logarithm-based public key cryptosystems will be built within the next decade."

Panic. "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."

"[Section 4.4:] In particular, all encrypted data that is recorded today and stored for future use, will be cracked once a large-scale quantum computer is developed."

Commonly used systems

Sender
"Alice"



Untrustworthy network
"Eve"

Receiver
"Bob"

Cryptography with symmetric keys
AES-128. **AES-192**. **AES-256**. **AES-GCM**. **ChaCha20**. **HMAC-SHA-256**. **Poly1305**. **SHA-2**. **SHA-3**. **Salsa20**.

Cryptography with public keys
BN-254. **Curve25519**. **DH**. **DSA**. **ECDH**. **ECDSA**. **EdDSA**. **NIST P-256**. **NIST P-384**. **NIST P-521**. **RSA encrypt**. **RSA sign**. **secp256k1**.

Commonly used systems

Sender
"Alice"

Untrustworthy network
"Eve" with quantum computer

Receiver
"Bob"

Cryptography with symmetric keys
AES-128. **AES-192**. **AES-256**. **AES-GCM**. **ChaCha20**. **HMAC-SHA-256**. **Poly1305**. **SHA-2**. **SHA-3**. **Salsa20**.

Cryptography with public keys
BN-254. **Curve25519**. **DH**. **DSA**. **ECDH**. **ECDSA**. **EdDSA**. **NIST P-256**. **NIST P-384**. **NIST P-521**. **RSA encrypt**. **RSA sign**. **secp256k1**.

Post-quantum cryptography

Cryptography under the assumption that the attacker has a quantum computer.

Post-quantum cryptography

Cryptography under the assumption that the attacker has a quantum computer.

5 Major categories of public-key post-quantum systems

- **Code-based** encryption: McEliece cryptosystem has survived since 1978. Short ciphertexts and large public keys. Security relies on hardness of decoding error-correcting codes.
- **Hash-based** signatures: very solid security and small public keys. Require only a secure hash function (hard to find second preimages).
- **Isogeny-based** encryption: new kid on the block, promising short keys and ciphertexts and non-interactive key exchange. Security relies on hardness of finding isogenies between elliptic curves over finite fields.
- **Lattice-based** encryption and signatures: possibility for balanced sizes. Security relies on hardness of finding short vectors in some (typically special) lattice.
- **Multivariate-quadratic** signatures: short signatures and large public keys. Security relies on hardness of solving systems of multivariate equations over finite fields.

Warning: These are categories of mathematical problems;
individual systems may be totally insecure if the problem is not used correctly.

We have a good understanding of what a quantum computer can do,
but new systems need more analysis.

Focus today:
isogeny-based cryptography

(with lots of slides by Chloe Martindale and Lorenz Panny)

Diffie–Hellman key exchange '76

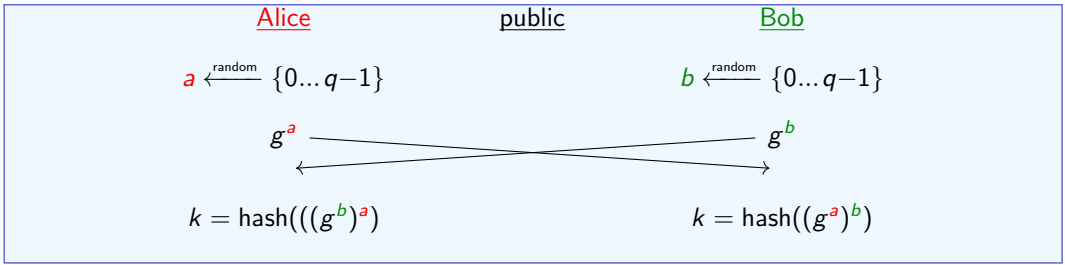
Public parameters:

- a finite group G (traditionally $\mathbb{F}_p^*$, today elliptic curves)
- an element $g \in G$ of prime order q

Diffie–Hellman key exchange '76

Public parameters:

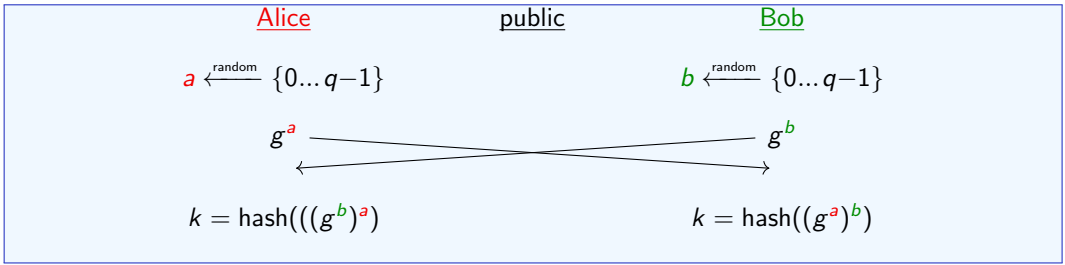
- a finite group G (traditionally $\mathbb{F}_p^*$, today elliptic curves)
- an element $g \in G$ of prime order q



Diffie–Hellman key exchange '76

Public parameters:

- a finite group G (traditionally $\mathbb{F}_p^*$, today elliptic curves)
- an element $g \in G$ of prime order q



Fundamental reason this works: $\cdot^a$ and $\cdot^b$ *commute*!

Diffie–Hellman: Bob vs. Eve

Bob

1. Set $t \leftarrow g$.
2. Set $t \leftarrow t \cdot g$.
3. Set $t \leftarrow t \cdot g$.
4. Set $t \leftarrow t \cdot g$.

...

b–2. Set $t \leftarrow t \cdot g$.

b–1. Set $t \leftarrow t \cdot g$.

b. Publish $B \leftarrow t \cdot g$.

Diffie–Hellman: Bob vs. Eve

Bob

1. Set $t \leftarrow g$.
2. Set $t \leftarrow t \cdot g$.
3. Set $t \leftarrow t \cdot g$.
4. Set $t \leftarrow t \cdot g$.

...

b–2. Set $t \leftarrow t \cdot g$.

b–1. Set $t \leftarrow t \cdot g$.

b. Publish $B \leftarrow t \cdot g$.

Is this a good idea?

Diffie–Hellman: Bob vs. Eve

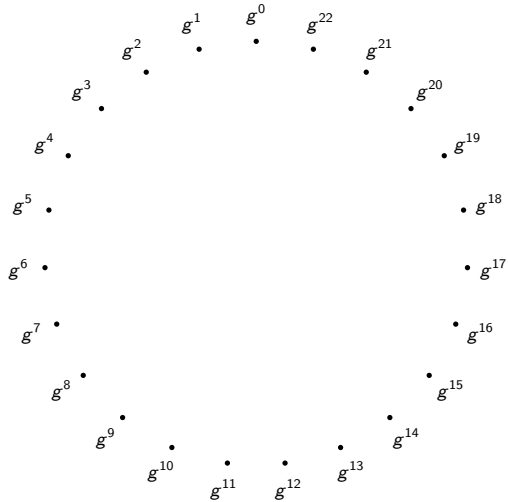
Bob	**Attacker** Eve	
1. Set $t \leftarrow g$.	1. Set $t \leftarrow g$.	If $t = B$ return 1.
2. Set $t \leftarrow t \cdot g$.	2. Set $t \leftarrow t \cdot g$.	If $t = B$ return 2.
3. Set $t \leftarrow t \cdot g$.	3. Set $t \leftarrow t \cdot g$.	If $t = B$ return 3.
4. Set $t \leftarrow t \cdot g$.	4. Set $t \leftarrow t \cdot g$.	If $t = B$ return 3.
...	...	
$b-2$. Set $t \leftarrow t \cdot g$.	$b-2$. Set $t \leftarrow t \cdot g$.	If $t = B$ return $b-2$.
$b-1$. Set $t \leftarrow t \cdot g$.	$b-1$. Set $t \leftarrow t \cdot g$.	If $t = B$ return $b-1$.
b. Publish $B \leftarrow t \cdot g$.	b. Set $t \leftarrow t \cdot g$.	If $t = B$ return b.
	$b+1$. Set $t \leftarrow t \cdot g$.	If $t = B$ return $b+1$.
	$b+2$. Set $t \leftarrow t \cdot g$.	If $t = B$ return $b+2$.
	...	

Diffie–Hellman: Bob vs. Eve

Bob	**Attacker** Eve	
1. Set $t \leftarrow g$.	1. Set $t \leftarrow g$.	If $t = B$ return 1.
2. Set $t \leftarrow t \cdot g$.	2. Set $t \leftarrow t \cdot g$.	If $t = B$ return 2.
3. Set $t \leftarrow t \cdot g$.	3. Set $t \leftarrow t \cdot g$.	If $t = B$ return 3.
4. Set $t \leftarrow t \cdot g$.	4. Set $t \leftarrow t \cdot g$.	If $t = B$ return 3.
...	...	
$b-2$. Set $t \leftarrow t \cdot g$.	$b-2$. Set $t \leftarrow t \cdot g$.	If $t = B$ return $b-2$.
$b-1$. Set $t \leftarrow t \cdot g$.	$b-1$. Set $t \leftarrow t \cdot g$.	If $t = B$ return $b-1$.
b. Publish $B \leftarrow t \cdot g$.	b. Set $t \leftarrow t \cdot g$.	If $t = B$ return b.
	$b+1$. Set $t \leftarrow t \cdot g$.	If $t = B$ return $b+1$.
	$b+2$. Set $t \leftarrow t \cdot g$.	If $t = B$ return $b+2$.
	...	

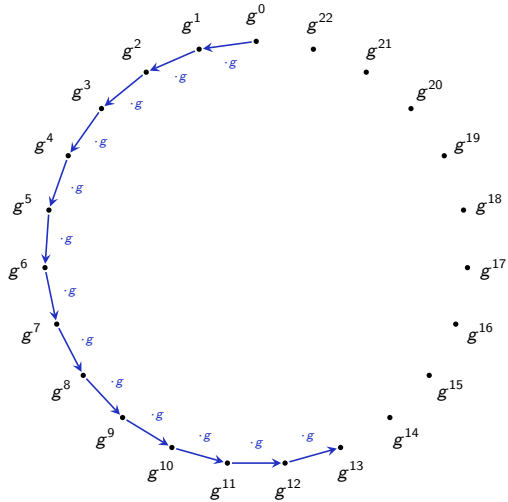
Effort for both: $O(\#G)$. Bob needs to be smarter.

(There also exist better attacks)



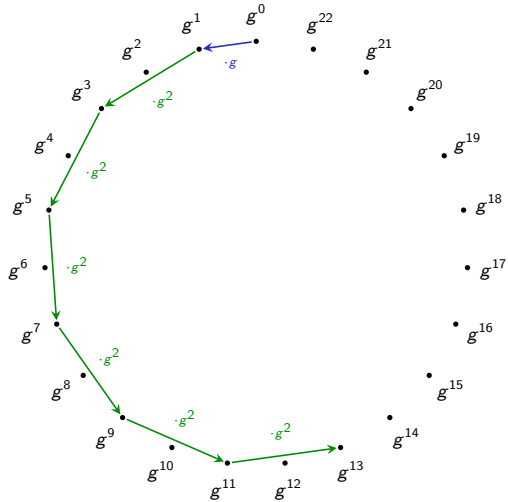
Reminder: DH in group with $\#G = 23$. Bob computes g^{13}.

multiply



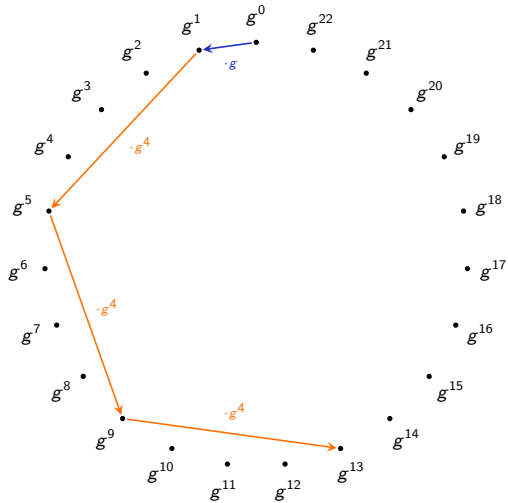
Reminder: DH in group with $\#G = 23$. Bob computes g^{13}.

Square-and-multiply



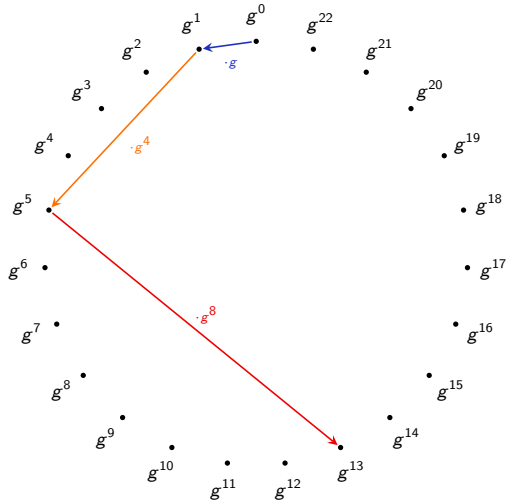
Reminder: DH in group with $\#G = 23$. Bob computes g^{13}.

Square-and-multiply-and-square-and-multiply



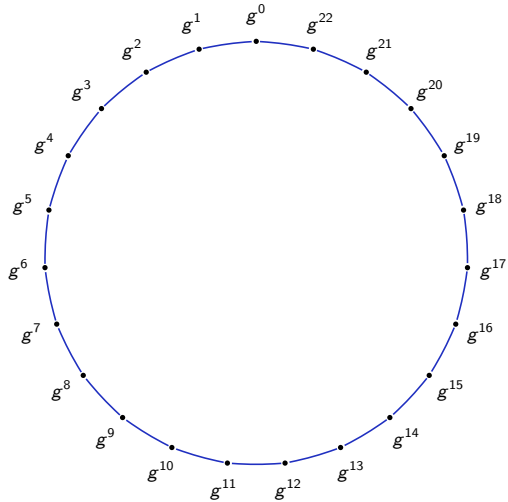
Reminder: DH in group with $\#G = 23$. Bob computes g^{13}.

Square-and-multiply-and-square-and-multiply-and-square-and-multiply



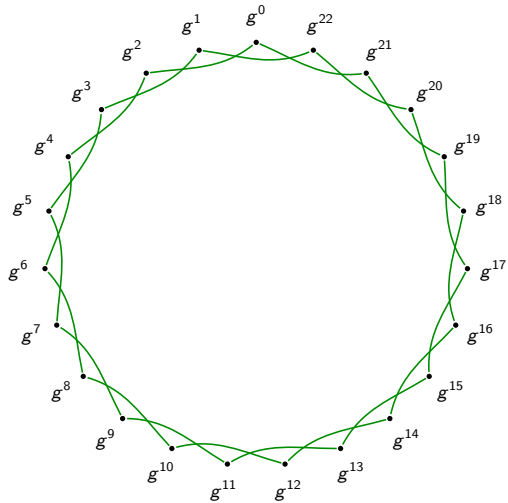
Reminder: DH in group with $\#G = 23$. Bob computes g^{13}.

Square-and-multiply as graphs



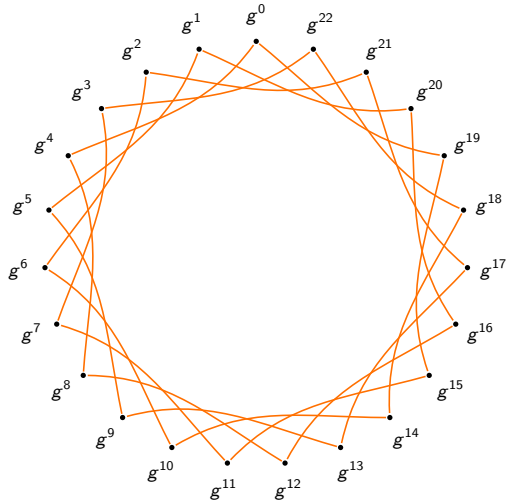
Reminder: DH in group with $\#G = 23$. Bob computes g^{13}.

Square-and-multiply as graphs



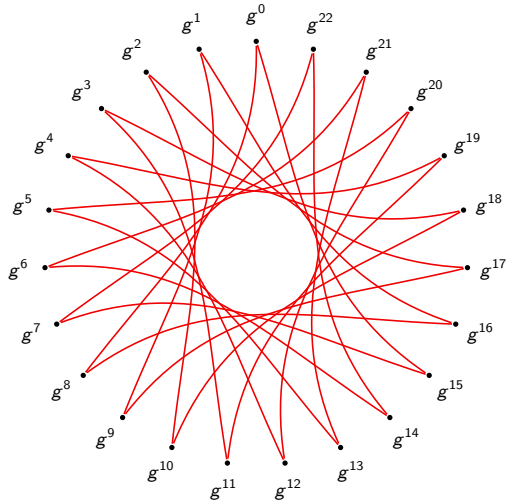
Reminder: DH in group with $\#G = 23$. Bob computes g^{13}.

Square-and-multiply as graphs



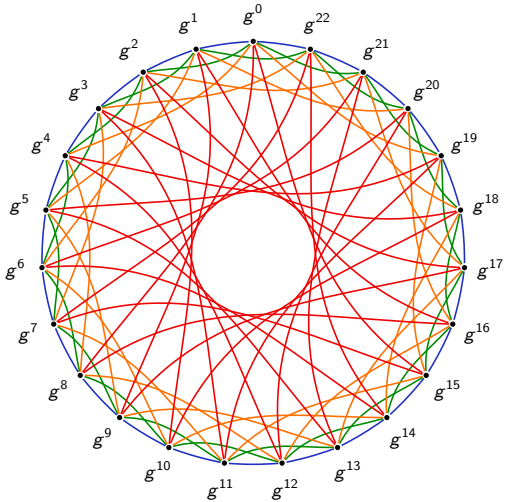
Reminder: DH in group with $\#G = 23$. Bob computes g^{13}.

Square-and-multiply as graphs



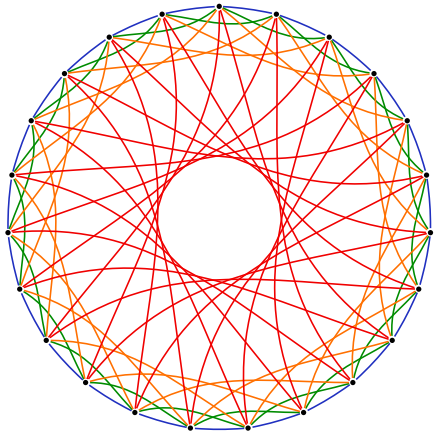
Reminder: DH in group with $\#G = 23$. Bob computes g^{13}.

Square-and-multiply as a graph



Reminder: DH in group with $\#G = 23$. Bob computes g^{13}.

Square-and-multiply as a graph



Fast mixing: paths of length log(# nodes) to everywhere.

Exponential separation

Constructive computation:
With square-and-multiply, applying b takes $\Theta(\log_2 \#G)$.

Attack costs:
For well-chosen groups, recovering b takes $\Theta(\sqrt{\#G})$.

(For less-well chosen groups the attacks are faster.)

As

$$\sqrt{\#G} = 2^{0.5 \log_2 \#G}$$

attacks are exponentially harder.

Exponential separation until quantum computers come

Constructive computation:
With square-and-multiply, applying b takes $\Theta(\log_2 \#G)$.

Attack costs:
For well-chosen groups, recovering b takes $\Theta(\sqrt{\#G})$.

(For less-well chosen groups the attacks are faster.)

As

$$\sqrt{\#G} = 2^{0.5 \log_2 \#G}$$

attacks are exponentially harder.

On a sufficiently large quantum computer, Shor's algorithm quantumly computes b from g^b *in any group* in polynomial time.

Exponential separation until quantum computers come

Constructive computation:
With square-and-multiply, applying b takes $\Theta(\log_2 \#G)$.

Attack costs:
For well-chosen groups, recovering b takes $\Theta(\sqrt{\#G})$.

(For less-well chosen groups the attacks are faster.)

As

$$\sqrt{\#G} = 2^{0.5 \log_2 \#G}$$

attacks are exponentially harder.

On a sufficiently large quantum computer, Shor's algorithm quantumly computes b from g^b *in any group* in polynomial time.
Isogeny graphs to the rescue!

Isogenies

An *isogeny* of elliptic curves is a non-zero map $E \to E'$

- given by *rational functions*
- that is a *group homomorphism*.

The *degree* of a separable isogeny is the size of its *kernel*.

Isogenies

An *isogeny* of elliptic curves is a non-zero map $E \to E'$

- given by *rational functions*
- that is a *group homomorphism*.

The *degree* of a separable isogeny is the size of its *kernel*.

Example #1: For each $m \neq 0$, the *multiplication-by-m map*

$$[m]\colon E \to E$$

is an isogeny from E to itself.

If $m \neq 0$ in the base field, its kernel is

$$E[m] \;\cong\; \mathbb{Z}/m \times \mathbb{Z}/m.$$

Thus $[m]$ is a degree-m^2 isogeny.

Isogenies

An *isogeny* of elliptic curves is a non-zero map $E \to E'$

- given by *rational functions*
- that is a *group homomorphism*.

The *degree* of a separable isogeny is the size of its *kernel*.

Example #2: For any a and b, the map $\iota\colon (x, y) \mapsto (-x, \sqrt{-1} \cdot y)$ defines a degree-1 isogeny of the elliptic curves

$$\{y^2 = x^3 + ax + b\} \longrightarrow \{y^2 = x^3 + ax - b\}\,.$$

It is an *isomorphism*; its kernel is $\{\infty\}$.

Isogenies

An *isogeny* of elliptic curves is a non-zero map $E \to E'$

- given by *rational functions*
- that is a *group homomorphism*.

The *degree* of a separable isogeny is the size of its *kernel*.

Example #3:

$$(x, y) \mapsto \left(\frac{x^3-4x^2+30x-12}{(x-2)^2}, \frac{x^3-6x^2-14x+35}{(x-2)^3} \cdot y \right)$$

defines a degree-3 isogeny of the elliptic curves

$$\{y^2 = x^3 + x\} \longrightarrow \{y^2 = x^3 - 3x + 3\}$$

over $\mathbb{F}_{71}$. Its kernel is $\{(2, 9), (2, -9), \infty\}$.

Big picture

- Isogenies are a source of *exponentially*-sized *graphs*.

Big picture

- Isogenies are a source of *exponentially*-sized *graphs*.
- Isogeny graph: Nodes are isomorphism classes of curves, edges are isogenies.
  (We usually care about *sub*graphs with certain properties.)

Big picture

- Isogenies are a source of *exponentially*-sized *graphs*.
- Isogeny graph: Nodes are isomorphism classes of curves, edges are isogenies.
  (We usually care about *sub*graphs with certain properties.)
- We can *walk efficiently* on these graphs.

Big picture

- Isogenies are a source of *exponentially*-sized *graphs*.
- Isogeny graph: Nodes are isomorphism classes of curves, edges are isogenies.
  (We usually care about *sub*graphs with certain properties.)
- We can *walk efficiently* on these graphs.
- *Fast mixing*: short paths to (almost) all nodes.

Big picture

- Isogenies are a source of *exponentially*-sized *graphs*.
- Isogeny graph: Nodes are isomorphism classes of curves, edges are isogenies.
  (We usually care about *sub*graphs with certain properties.)
- We can *walk efficiently* on these graphs.
- *Fast mixing*: short paths to (almost) all nodes.
- *No efficient* algorithms to *recover paths* from endpoints.
  (*Both* classical and quantum!)

Big picture

- Isogenies are a source of *exponentially*-sized *graphs*.
- Isogeny graph: Nodes are isomorphism classes of curves, edges are isogenies.
  (We usually care about *sub*graphs with certain properties.)
- We can *walk efficiently* on these graphs.
- *Fast mixing*: short paths to (almost) all nodes.
- *No efficient* algorithms to *recover paths* from endpoints.
  (*Both* classical and quantum!)
- *Enough structure* to *navigate* the graph meaningfully.
  That is: some *well-behaved* "directions" to describe paths. More later.

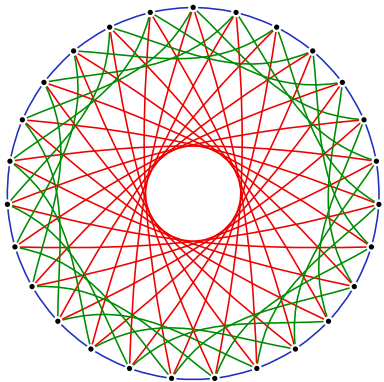
Big picture

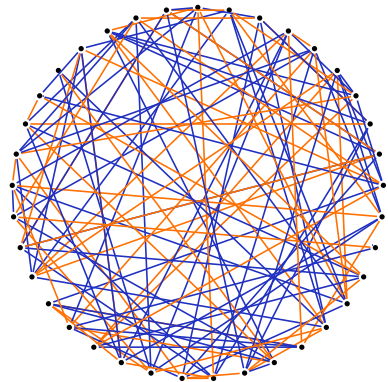
- Isogenies are a source of *exponentially*-sized *graphs*.
- Isogeny graph: Nodes are isomorphism classes of curves, edges are isogenies.
  (We usually care about *sub*graphs with certain properties.)
- We can *walk efficiently* on these graphs.
- *Fast mixing*: short paths to (almost) all nodes.
- *No efficient* algorithms to *recover paths* from endpoints.
  (*Both* classical and quantum!)
- *Enough structure* to *navigate* the graph meaningfully.
  That is: some *well-behaved* "directions" to describe paths. More later.

It is easy to construct graphs that satisfy *almost* all of these
not enough for crypto!

The beauty and the beast

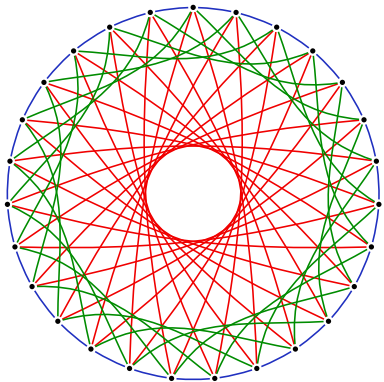
Components of well-chosen isogeny graphs look like this:

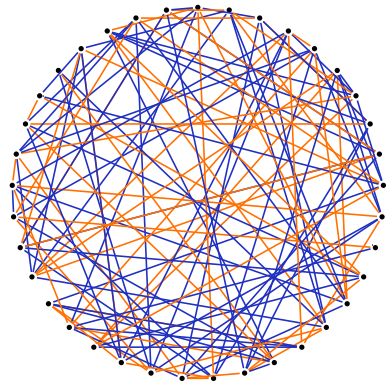




The beauty and the beast

Components of well-chosen isogeny graphs look like this:

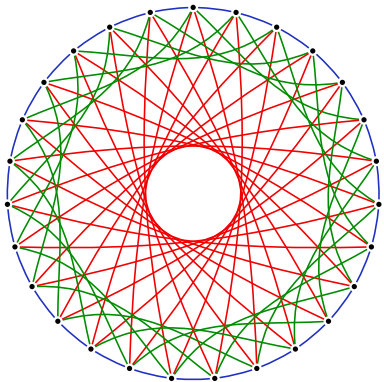


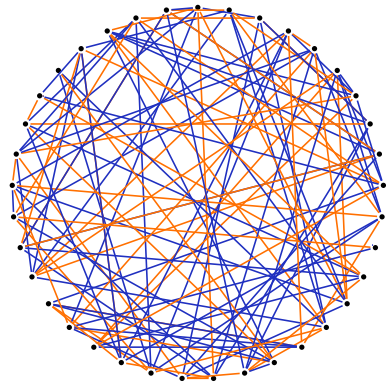


Which of these is good for crypto?

The beauty and the beast

Components of well-chosen isogeny graphs look like this:

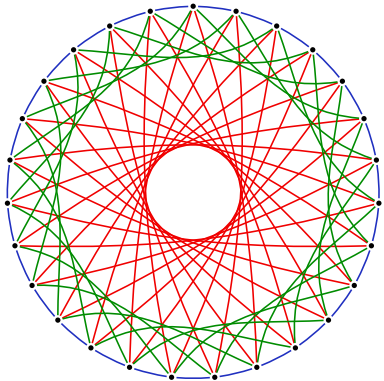




Which of these is good for crypto? **Both.**

The beauty and the beast

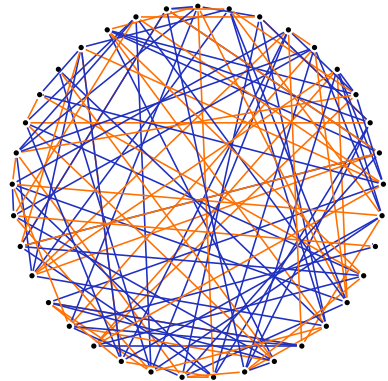
At this time, there are two distinct families of systems:



$q = p$

CSIDH ['si:ˌsaɪd]

https://csidh.isogeny.org



$q = p^2$

SIDH

https://sike.org

(Castryck, Lange, Martindale, Panny, Renes, 2018)

Why CSIDH?

- Closest thing we have in PQC to normal DH key exchange:
  Keys can be reused, blinded; no difference between initiator &responder.
- Public keys are represented by some $A \in \mathbb{F}_p$; p fixed prime.
- Alice computes and distributes her public key A.
  Bob computes and distributes his public key B.
- Alice and Bob do computations on each other's public keys
  to obtain shared secret.
- Fancy math: computations start on some elliptic curve $E_A : y^2 = x^3 + Ax^2 + x$, use *isogenies* to move to a different curve.
- Computations need arithmetic (add, mult, div) modulo p and
  elliptic-curve computations.

CSIDH in one slide

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

CSIDH in one slide

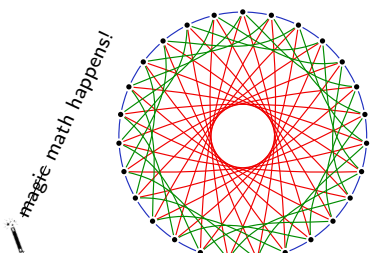
- Choose some small odd primes $\ell_1, ..., \ell_n$.
- Make sure $p = 4 \cdot \ell_1 \cdots \ell_n - 1$ is prime.
- Let $X = \{y^2 = x^3 + Ax^2 + x$ over $\mathbb{F}_p$ with $p+1$ points$\}$.

CSIDH in one slide

- Choose some small odd primes $\ell_1, ..., \ell_n$.
- Make sure $p = 4 \cdot \ell_1 \cdots \ell_n - 1$ is prime.
- Let $X = \{y^2 = x^3 + Ax^2 + x$ over $\mathbb{F}_p$ with $p+1$ points$\}$.
- Look at the ℓ_i-isogenies defined over $\mathbb{F}_p$ within X.

CSIDH in one slide

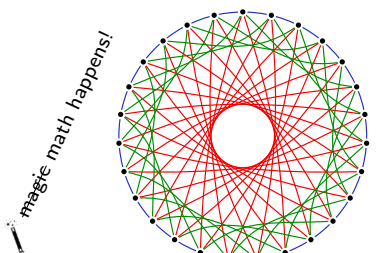
- Choose some small odd primes $\ell_1, ..., \ell_n$.
- Make sure $p = 4 \cdot \ell_1 \cdots \ell_n - 1$ is prime.
- Let $X = \{y^2 = x^3 + Ax^2 + x$ over $\mathbb{F}_p$ with $p+1$ points$\}$.
- Look at the ℓ_i-isogenies defined over $\mathbb{F}_p$ within X.



$p = 419$
$\ell_1 = 3$
$\ell_2 = 5$
$\ell_3 = 7$

CSIDH in one slide

- Choose some small odd primes $\ell_1, ..., \ell_n$.
- Make sure $p = 4 \cdot \ell_1 \cdots \ell_n - 1$ is prime.
- Let $X = \{y^2 = x^3 + Ax^2 + x$ over $\mathbb{F}_p$ with $p+1$ points$\}$.
- Look at the ℓ_i-isogenies defined over $\mathbb{F}_p$ within X.

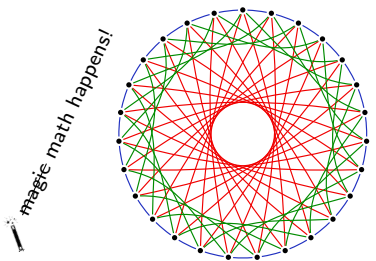


$p = 419$
$\ell_1 = 3$
$\ell_2 = 5$
$\ell_3 = 7$

- Walking "left" and "right" on any ℓ_i-subgraph is efficient.

CSIDH in one slide

- Choose some small odd primes $\ell_1, ..., \ell_n$.
- Make sure $p = 4 \cdot \ell_1 \cdots \ell_n - 1$ is prime.
- Let $X = \{y^2 = x^3 + Ax^2 + x$ over $\mathbb{F}_p$ with $p+1$ points$\}$.
- Look at the ℓ_i-isogenies defined over $\mathbb{F}_p$ within X.

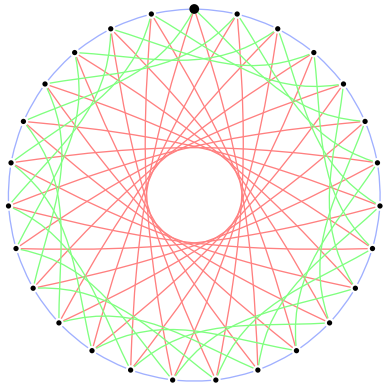


$p = 419$
$\ell_1 = 3$
$\ell_2 = 5$
$\ell_3 = 7$

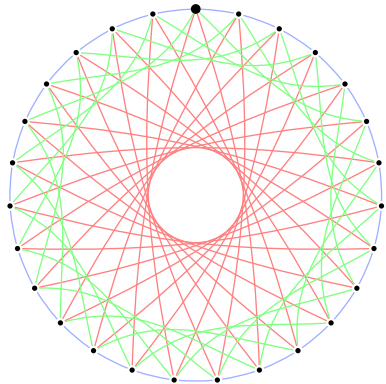
- Walking "left" and "right" on any ℓ_i-subgraph is efficient.
- We can represent $E \in X$ as a single coefficient $A \in \mathbb{F}_p$.

CSIDH key exchange

Alice
$[+,+,-,-]$



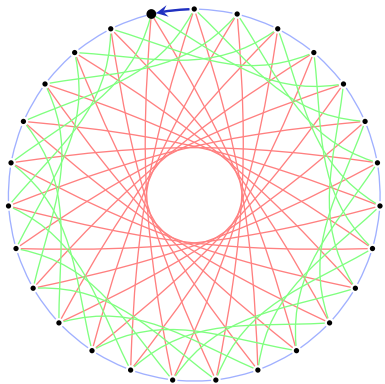
Bob
$[-,+,-,-]$



CSIDH key exchange

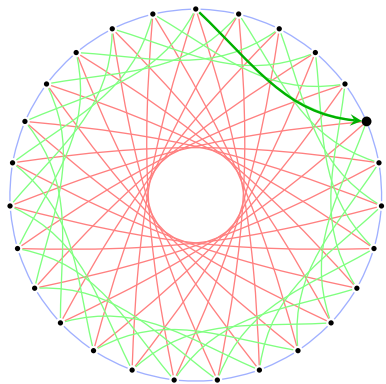
Alice

$[+, +, -, -]$



Bob

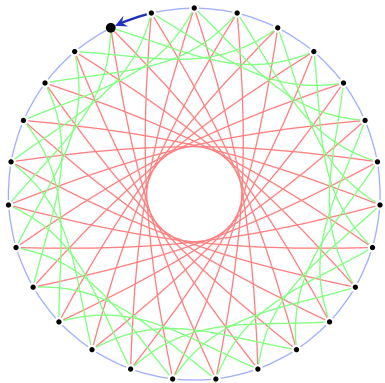
$[-, +, -, -]$



CSIDH key exchange

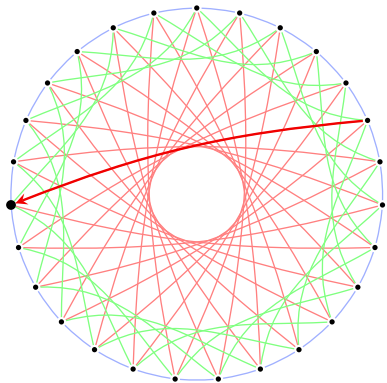
Alice

$[+, +, -, -]$



Bob

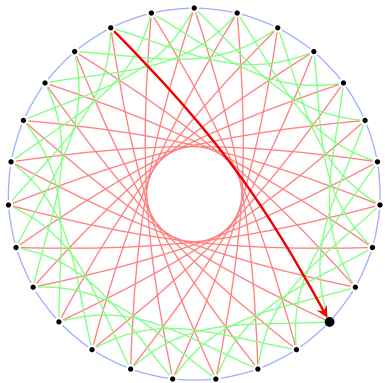
$[-, +, -, -]$



CSIDH key exchange

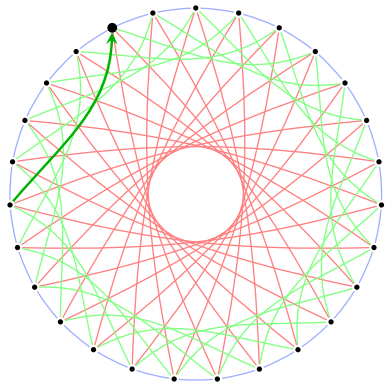
Alice

$[+, +, -, -]$



Bob

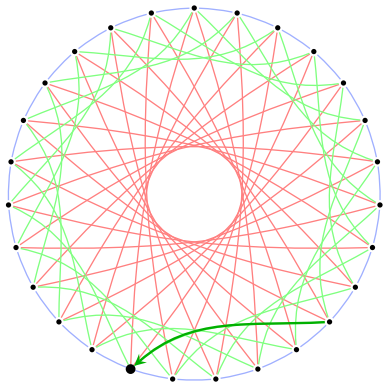
$[-, +, -, -]$



CSIDH key exchange

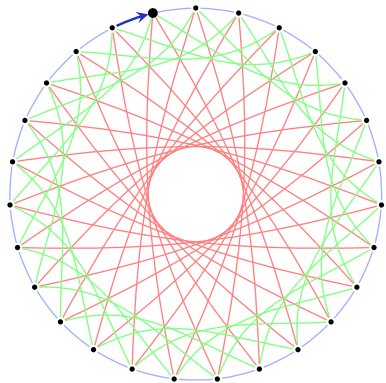
Alice

$[+, +, -, -]$



Bob

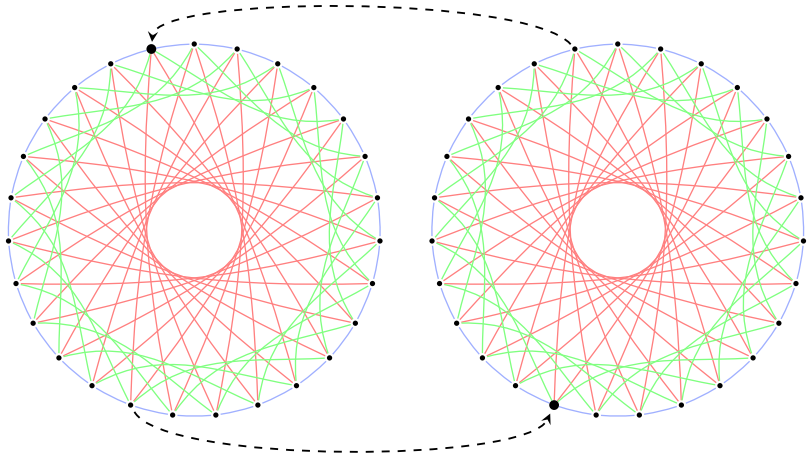
$[-, +, -, -]$



CSIDH key exchange

Alice
$[+,+,-,-]$

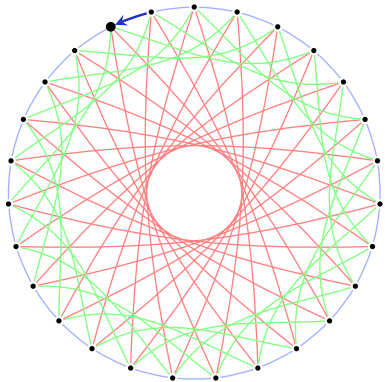
Bob
$[-,+,-,-]$



CSIDH key exchange

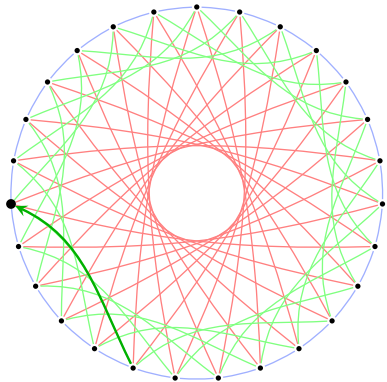
Alice

$[+, +, -, -]$



Bob

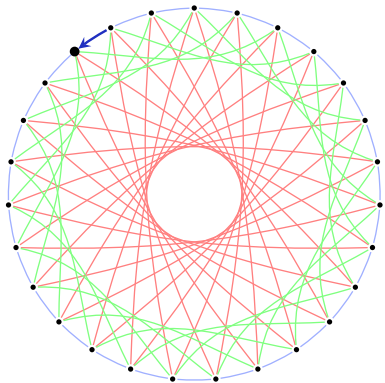
$[-, +, -, -]$



CSIDH key exchange

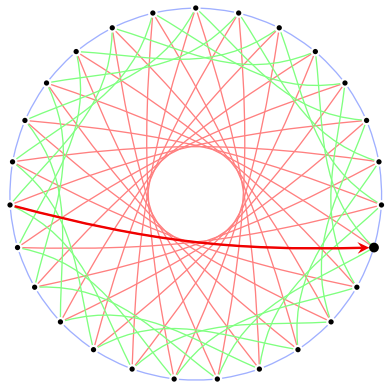
Alice

$[+, +, -, -]$



Bob

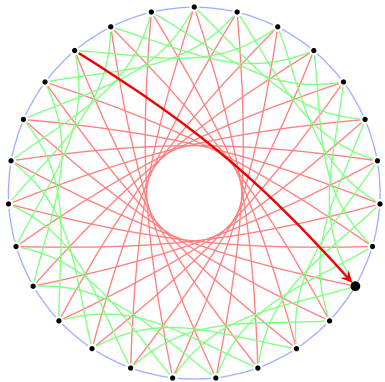
$[-, +, -, -]$



CSIDH key exchange

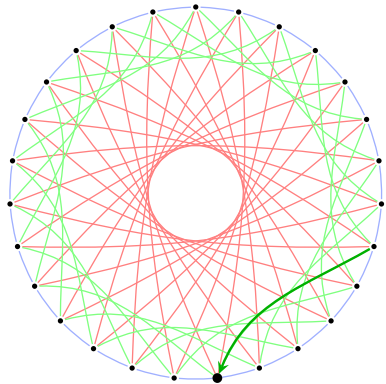
Alice

$[+, +, -, -]$



Bob

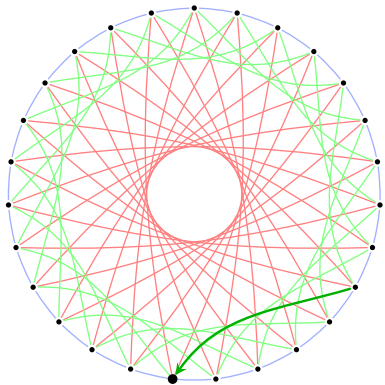
$[-, +, -, -]$



CSIDH key exchange

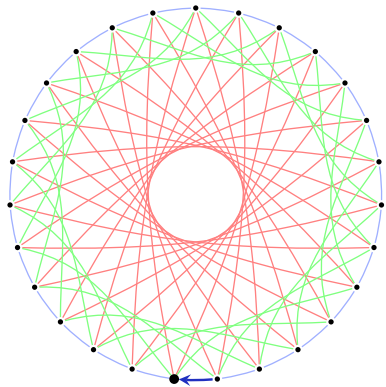
Alice

$[+, +, -, -]$



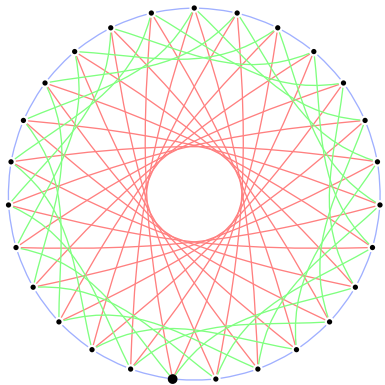
Bob

$[-, +, -, -]$

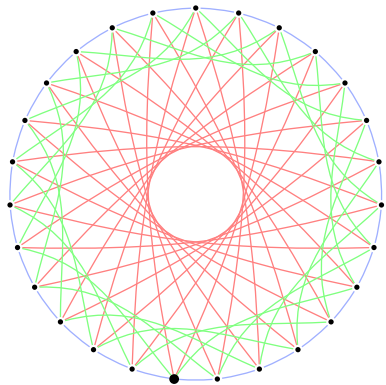


CSIDH key exchange

Alice
$[+,+,-,-]$



Bob
$[-,+,-,-]$



Walking in the CSIDH graph

Taking a *"positive" step* on the ℓ_i*-subgraph*.

1. Find a point $(x, y) \in E$ of *order* ℓ_i with $x, y \in \mathbb{F}_p$.
   The order of any $(x, y) \in E$ divides $p + 1$, so $[(p + 1)/\ell_i](x, y) = \infty$ or a point of order ℓ_i.
   Sample a new point if you get ∞.
2. Compute the *isogeny* with *kernel* $\langle (x, y) \rangle$ using Vélu's formulas.

Walking in the CSIDH graph

Taking a *"positive" step* on the ℓ_i*-subgraph*.

1. Find a point $(x, y) \in E$ of *order* ℓ_i with $x, y \in \mathbb{F}_p$.
   The order of any $(x, y) \in E$ divides $p + 1$, so $[(p + 1)/\ell_i](x, y) = \infty$ or a point of order ℓ_i.
   Sample a new point if you get ∞.
2. Compute the *isogeny* with *kernel* $\langle (x, y) \rangle$ using Vélu's formulas.

Taking a *"negative" step* on the ℓ_i*-subgraph*.

1. Find a point $(x, y) \in E$ of *order* ℓ_i with $x \in \mathbb{F}_p$ but $y \notin \mathbb{F}_p$.
   Same test as above to find such a point.
2. Compute the *isogeny* with *kernel* $\langle (x, y) \rangle$ using Vélu's formulas.

Walking in the CSIDH graph

Taking a *"positive" step* on the ℓ_i*-subgraph*.

1. Find a point $(x, y) \in E$ of *order* ℓ_i with $x, y \in \mathbb{F}_p$.
   The order of any $(x, y) \in E$ divides $p + 1$, so $[(p + 1)/\ell_i](x, y) = \infty$ or a point of order ℓ_i.
   Sample a new point if you get ∞.
2. Compute the *isogeny* with *kernel* $\langle (x, y) \rangle$ using Vélu's formulas.

Taking a *"negative" step* on the ℓ_i*-subgraph*.

1. Find a point $(x, y) \in E$ of *order* ℓ_i with $x \in \mathbb{F}_p$ but $y \notin \mathbb{F}_p$.
   Same test as above to find such a point.
2. Compute the *isogeny* with *kernel* $\langle (x, y) \rangle$ using Vélu's formulas.

Upshot: With "x-only' arithmetic" everything happens *over* $\mathbb{F}_p$.

$\Longrightarrow$ *Efficient* to implement! There are several more speedups, such as pushing points through isogenies.

CSIDH security

Core problem:
Given $E, E' \in X$, *find* and compute *isogeny* $E \to E'$.

Size of key space:

- About $\sqrt{p}$ of all $A \in \mathbb{F}_p$ are valid keys.
  (More precisely $\#\mathrm{cl}(\mathbb{Z}[\sqrt{-p}])$ keys.)

Without quantum computer:

- Meet-in-the-middle variants: Time $O(\sqrt[4]{p})$.
  (2016 Delfs–Galbraith)

CSIDH security

Core problem:
Given $E, E' \in X$, *find* and compute *isogeny* $E \to E'$.

Size of key space:

- About $\sqrt{p}$ of all $A \in \mathbb{F}_p$ are valid keys.
  (More precisely $\#\mathrm{cl}(\mathbb{Z}[\sqrt{-p}])$ keys.)

Without quantum computer:

- Meet-in-the-middle variants: Time $O(\sqrt[4]{p})$.
  (2016 Delfs–Galbraith)

With quantum computer:

- Abelian hidden-shift algorithms apply (2014 Childs–Jao–Soukharev)
  - These have subexponential complexity.
  - Not vulnerable to Shor's attack.

CSIDH security:

- Public-key validation:
  Quickly check that $E_A : y^2 = x^3 + Ax^2 + x$ has $p + 1$ points.

CSIDH-512 https://csidh.isogeny.org/

Definition:

- $p = 4\prod_{i=1}^{74} \ell_i - 1$ with $\ell_1, \ldots, \ell_{73}$ first 73 odd primes. $\ell_{74} = 587$.
- Exponents $-5 \leq e_i \leq 5$ for all $1 \leq i \leq 74$.

Sizes:

- Private keys: 32 bytes. (37 in current software for simplicity.)
- Public keys: 64 bytes (just one $\mathbb{F}_p$ element).

Performance on typical Intel Skylake laptop core:

- Clock cycles: about $12 \cdot 10^7$ per operation.
- Somewhat more for constant-time implementations.

Security:

- Pre-quantum: at least 128 bits.

CSIDH-512 https://csidh.isogeny.org/

Definition:

- $p = 4 \prod_{i=1}^{74} \ell_i - 1$ with $\ell_1, \ldots, \ell_{73}$ first 73 odd primes. $\ell_{74} = 587$.
- Exponents $-5 \leq e_i \leq 5$ for all $1 \leq i \leq 74$.

Sizes:

- Private keys: 32 bytes. (37 in current software for simplicity.)
- Public keys: 64 bytes (just one $\mathbb{F}_p$ element).

Performance on typical Intel Skylake laptop core:

- Clock cycles: about $12 \cdot 10^7$ per operation.
- Somewhat more for constant-time implementations.

Security:

- Pre-quantum: at least 128 bits.
- Post-quantum: complicated.
  Recent work analyzing cost: see https://quantum.isogeny.org.
  Several papers analyzing quantum approaches. (2018 Biasse–Iezzi-Jacobson, 2018-2020 Bonnetain–Schrottenloher, 2020 Peikert)
  https://csidh.isogeny.org/analysis.html

Further information (for whole talk)

- YouTube channel Tanja Lange: Post-quantum cryptography.
- Isogeny-based cryptography school.
- https://2017.pqcrypto.org/school: PQCRYPTO summer school with 21 lectures on video, slides, and exercises.
- https://2017.pqcrypto.org/exec and https://pqcschool.org/index.html: Executive school (less math, more perspective).
- https://pqcrypto.org our overview page.
- ENISA report on PQC, co-authored.
- https://pqcrypto.eu.org: PQCRYPTO EU Project.
  - PQCRYPTO recommendations.
  - Free software libraries (libpqcrypto, pqm4, pqhw).
  - Many reports, scientific articles, (overview) talks.
- Quantum Threat Timeline from Global Risk Institute, 2019; 2021 update.
- Status of quantum computer development (by German BSI).
- NIST PQC competition.
- PQCrypto 2016, PQCrypto 2017, PQCrypto 2018, PQCrypto 2019, PQCrypto 2020, PQCrypto 2021 with many slides and videos online.