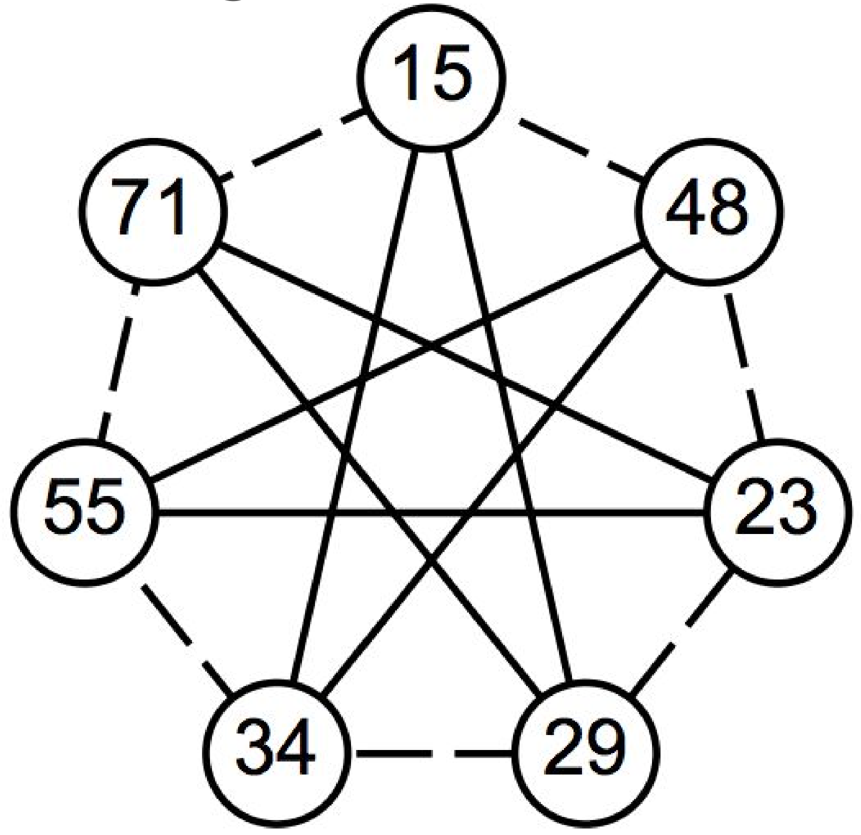
Supersingular Isogeny Diffie-Hellman



Deirdre Connolly
@durumcrustulum

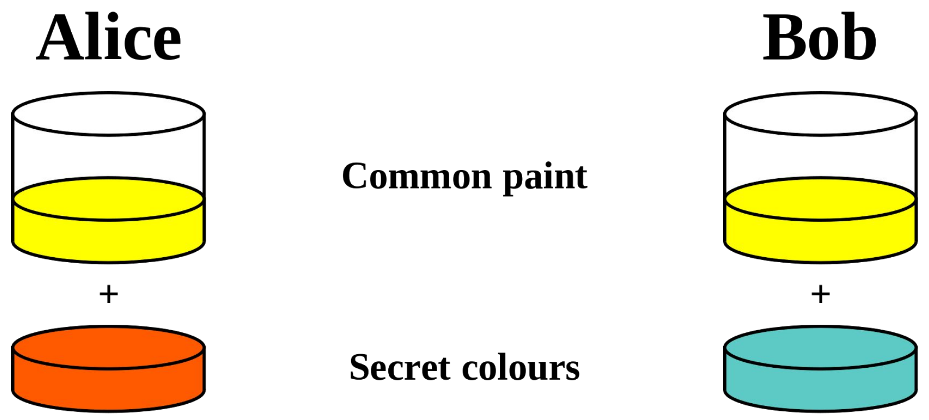
Shor's Algorithm

Diffie-Hellman Key Exchange

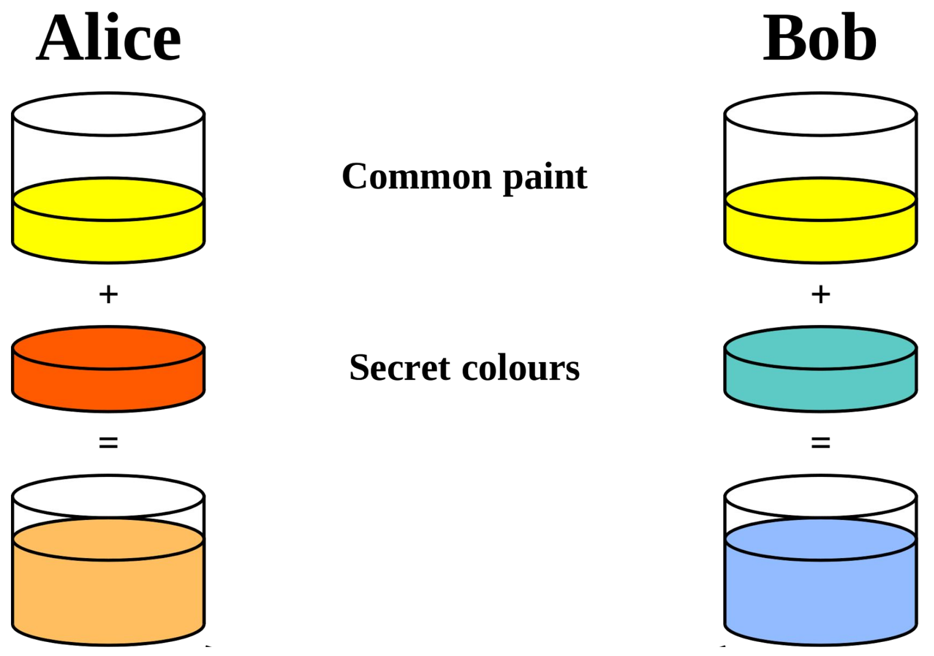
Diffie-Hellman Key Exchange

Alice
Bob
Common paint

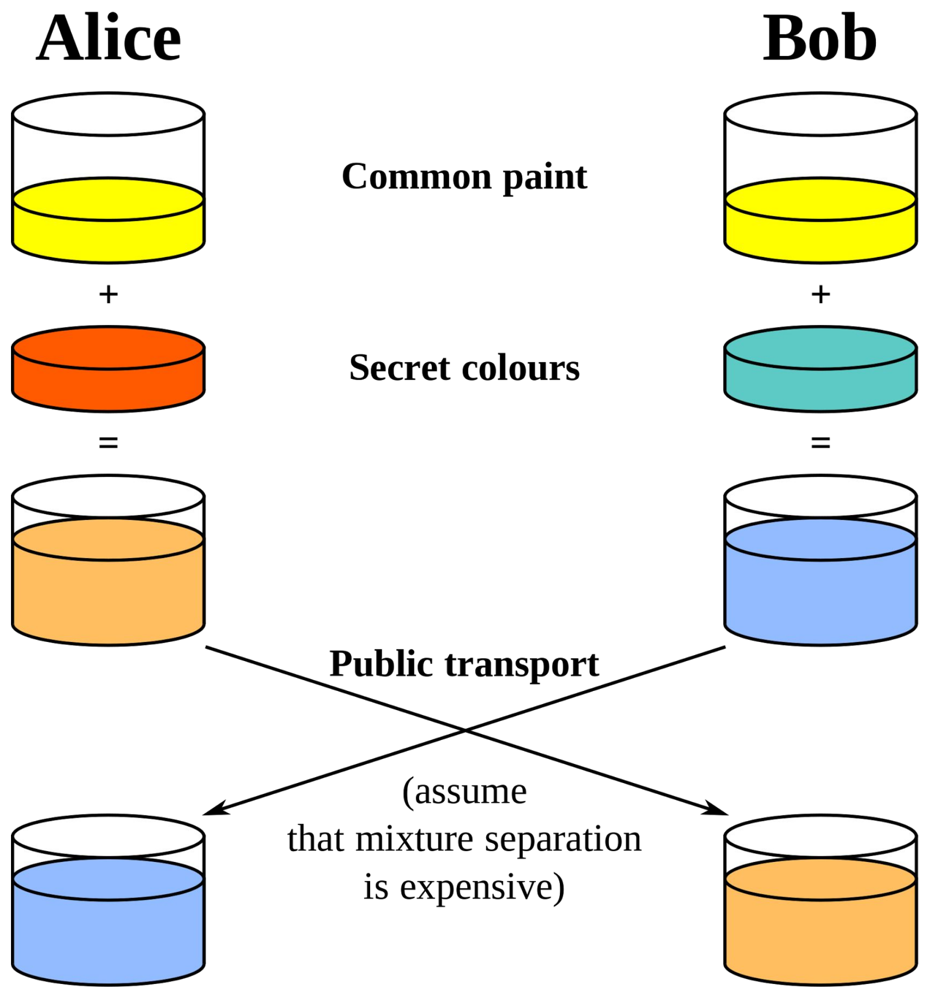
Diffie-Hellman Key Exchange



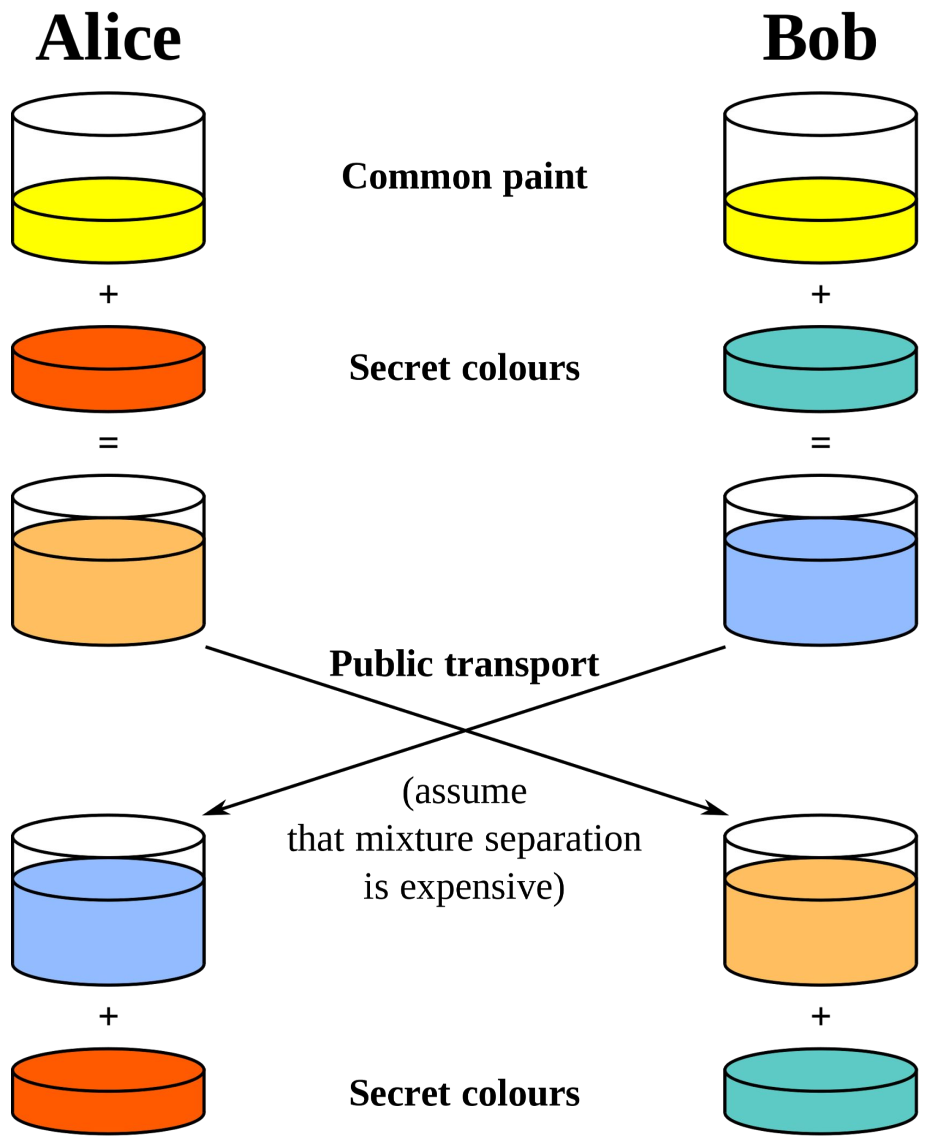
Diffie-Hellman Key Exchange



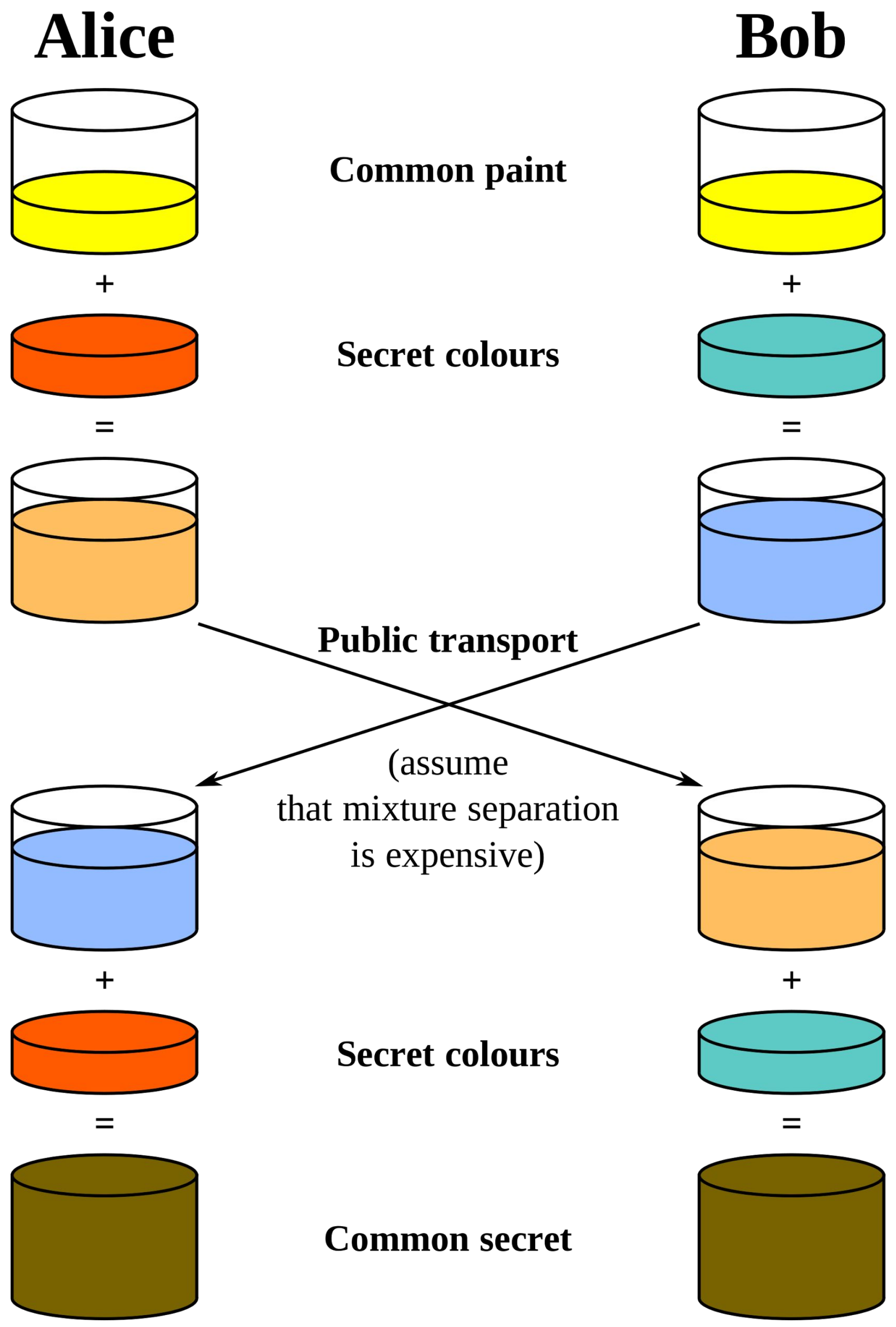
Diffie-Hellman Key Exchange

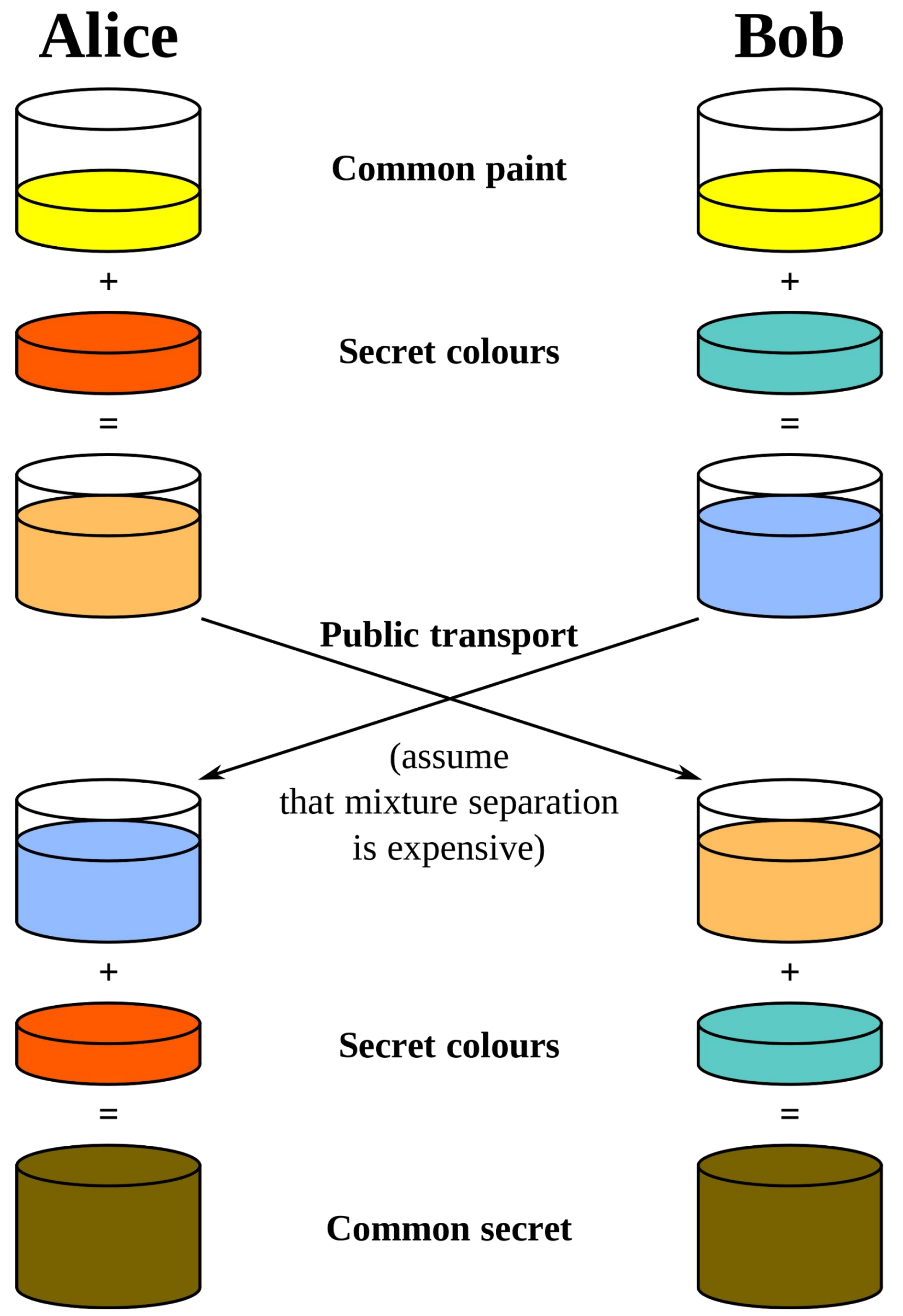


Diffie-Hellman Key Exchange



Diffie-Hellman Key Exchange

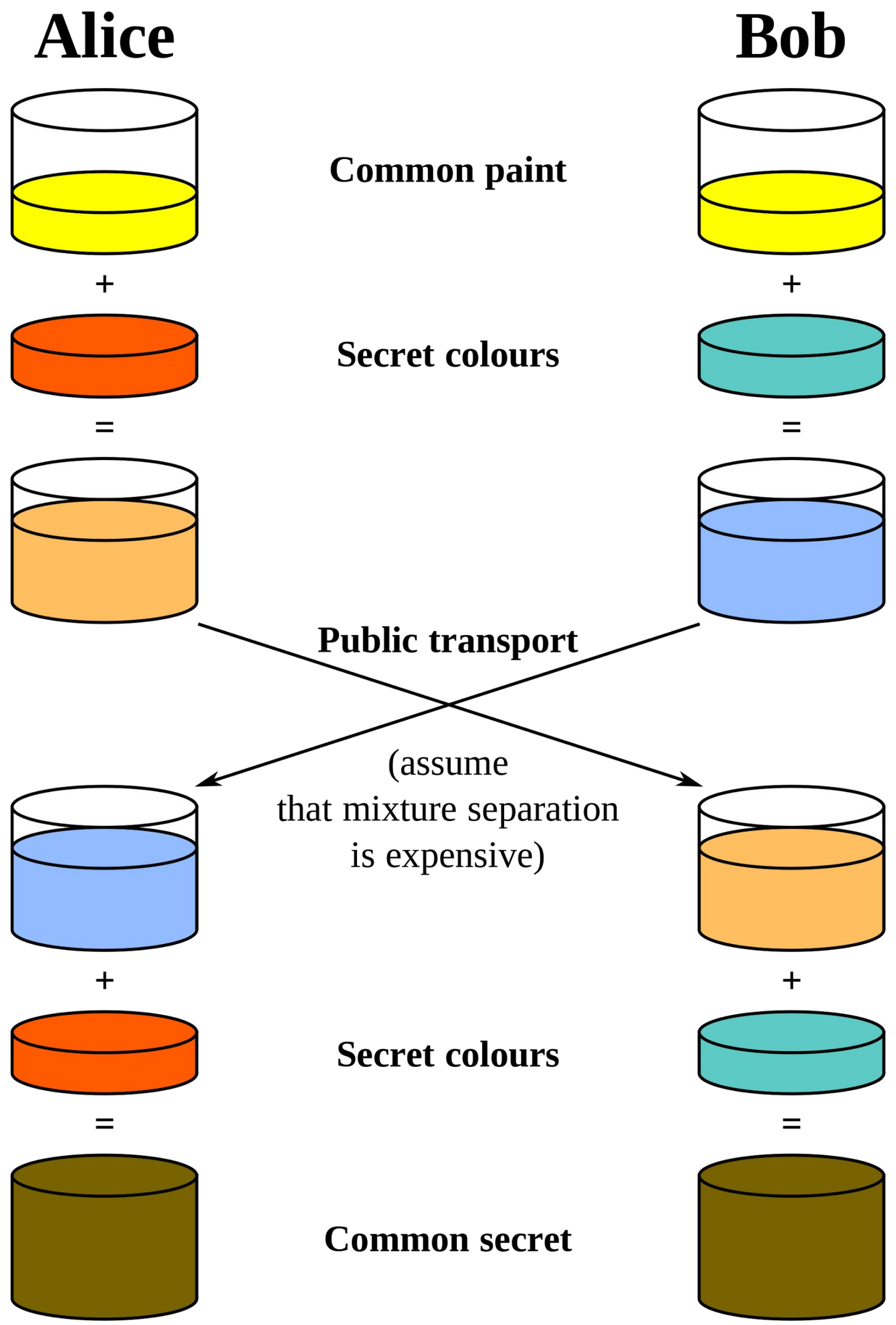




Diffie-Hellman Key Exchange

Originally **FFDH:** $key = g^{ab}\ mod\ p$

Exchange **integers**



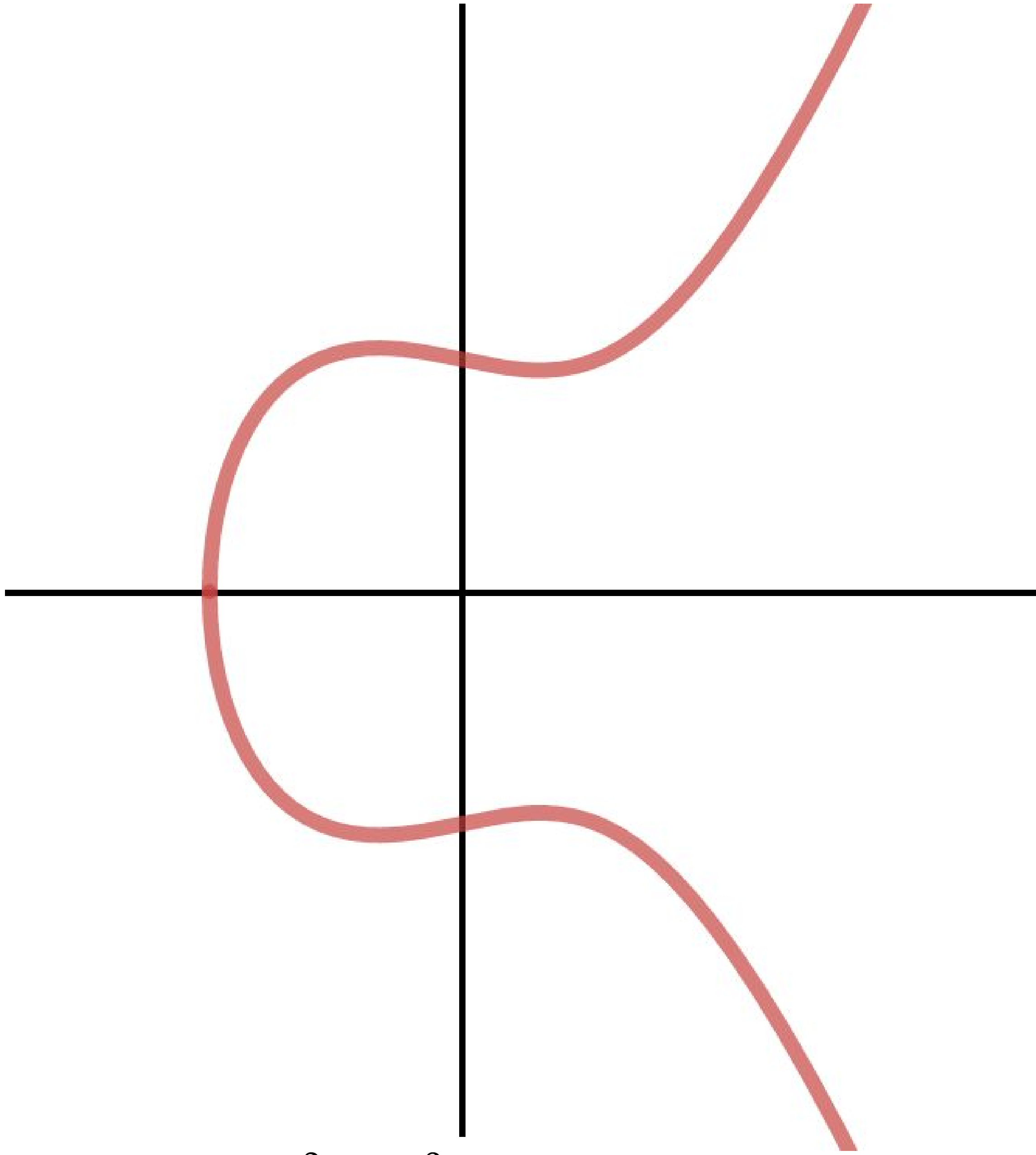
Diffie-Hellman Key Exchange

Originally **FFDH:** $key = g^{ab} \; mod \; p$

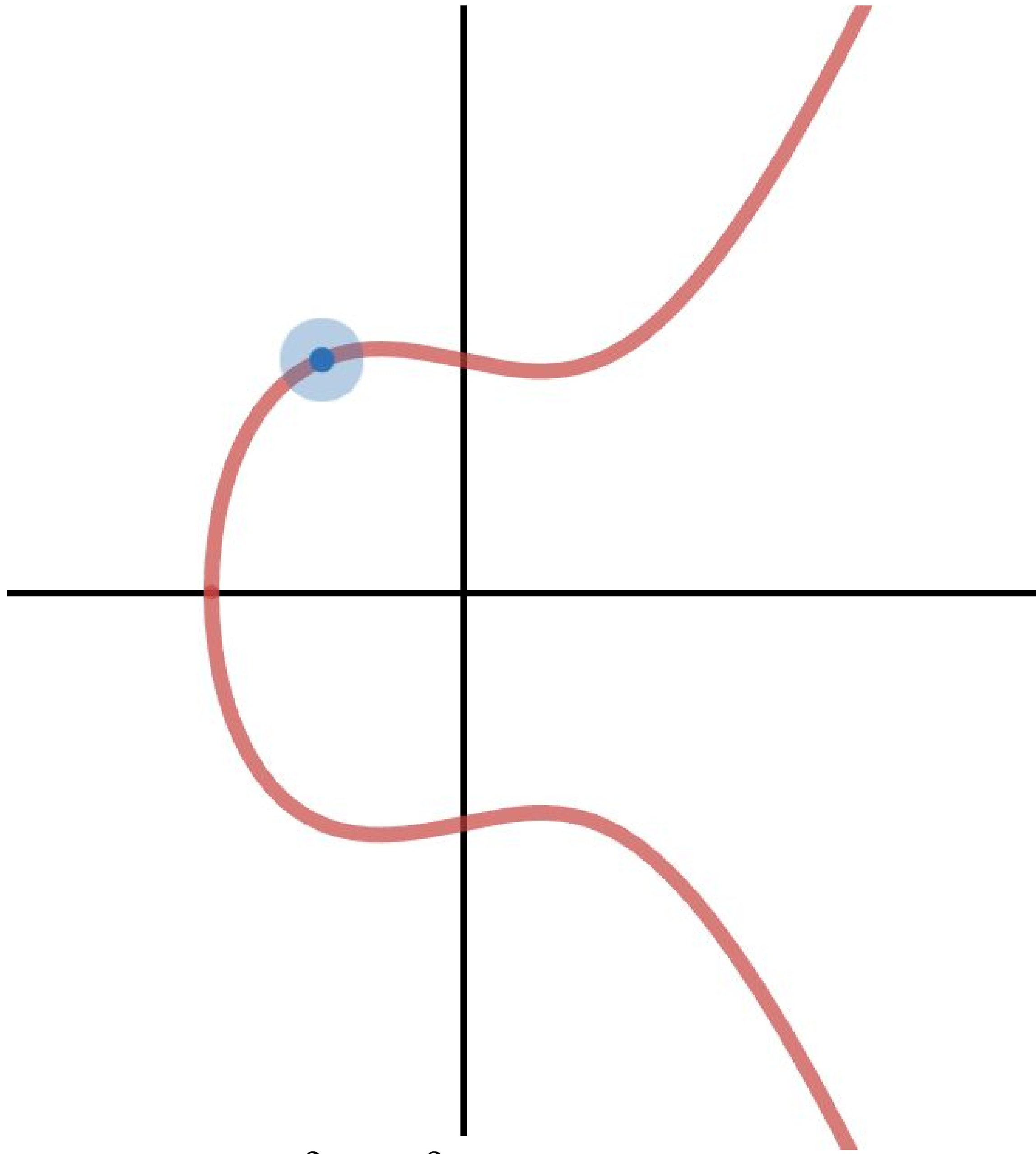
Exchange **integers**

Then **ECDH:** $key = abP \; mod \; p$

Exchange **points** on curve

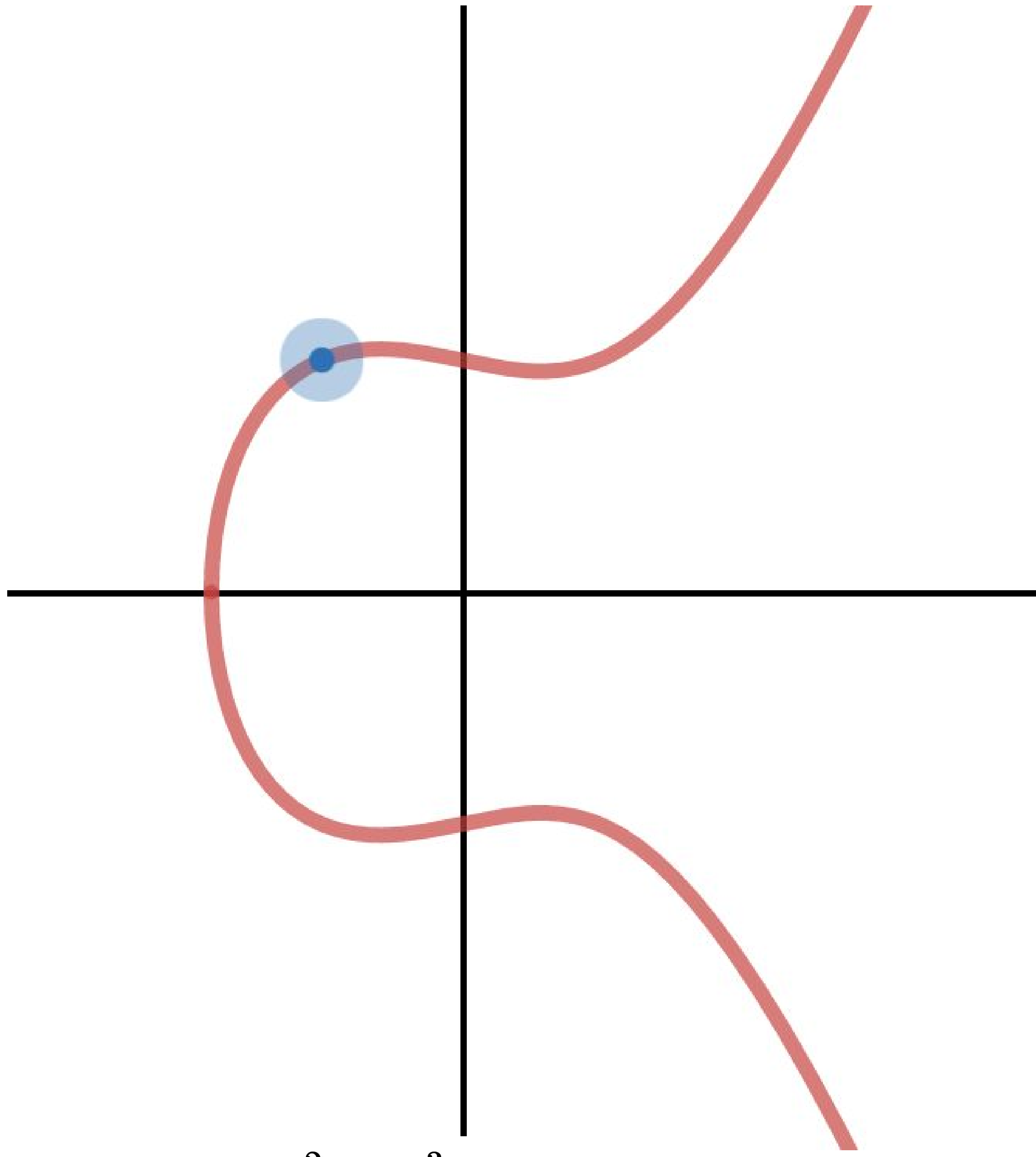


$$y^2 = x^3 + ax + b$$



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

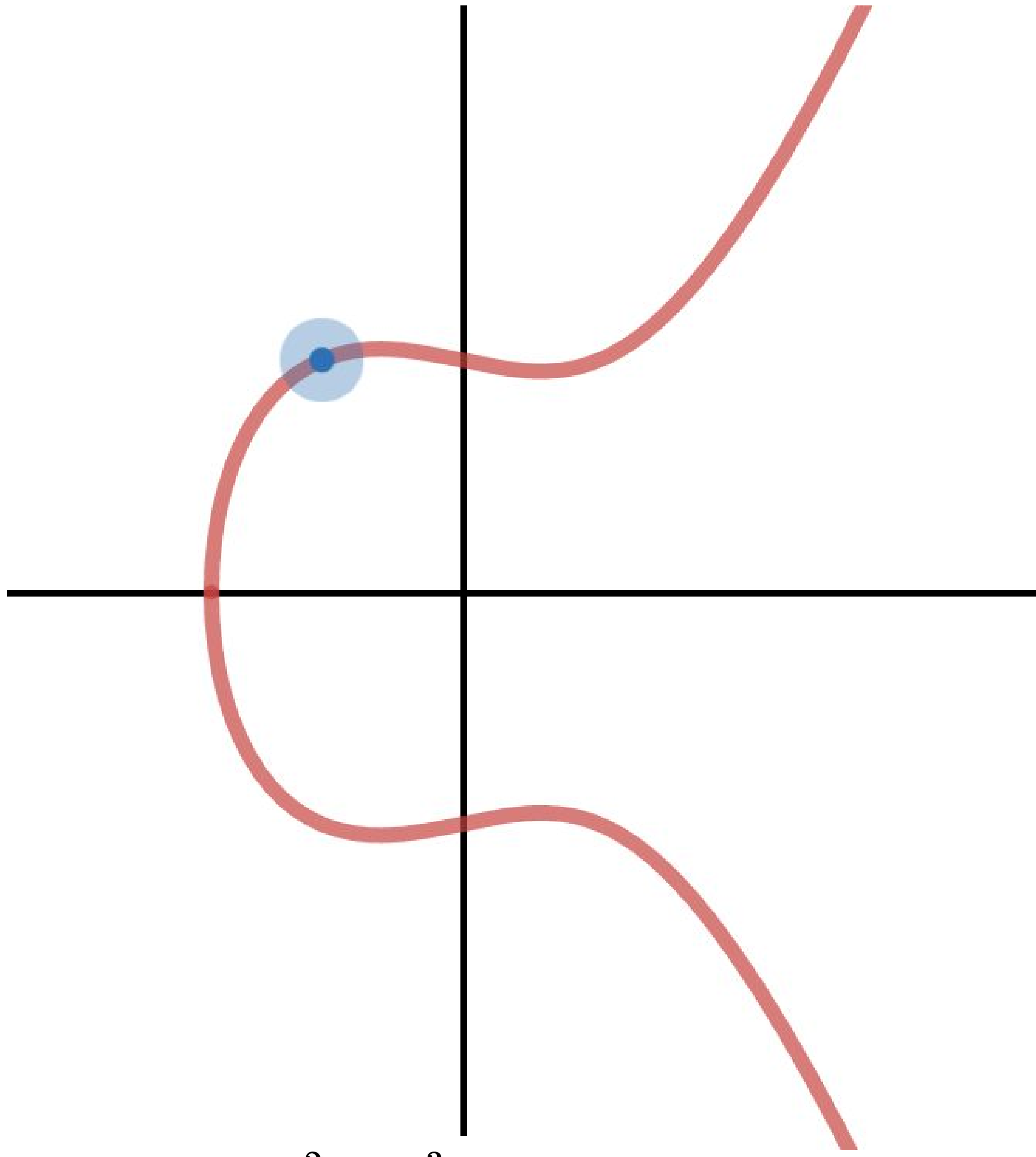
$$P + \mathcal{O} = P$$



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

$$P + \mathcal{O} = P$$

$$P + -P = \mathcal{O}$$

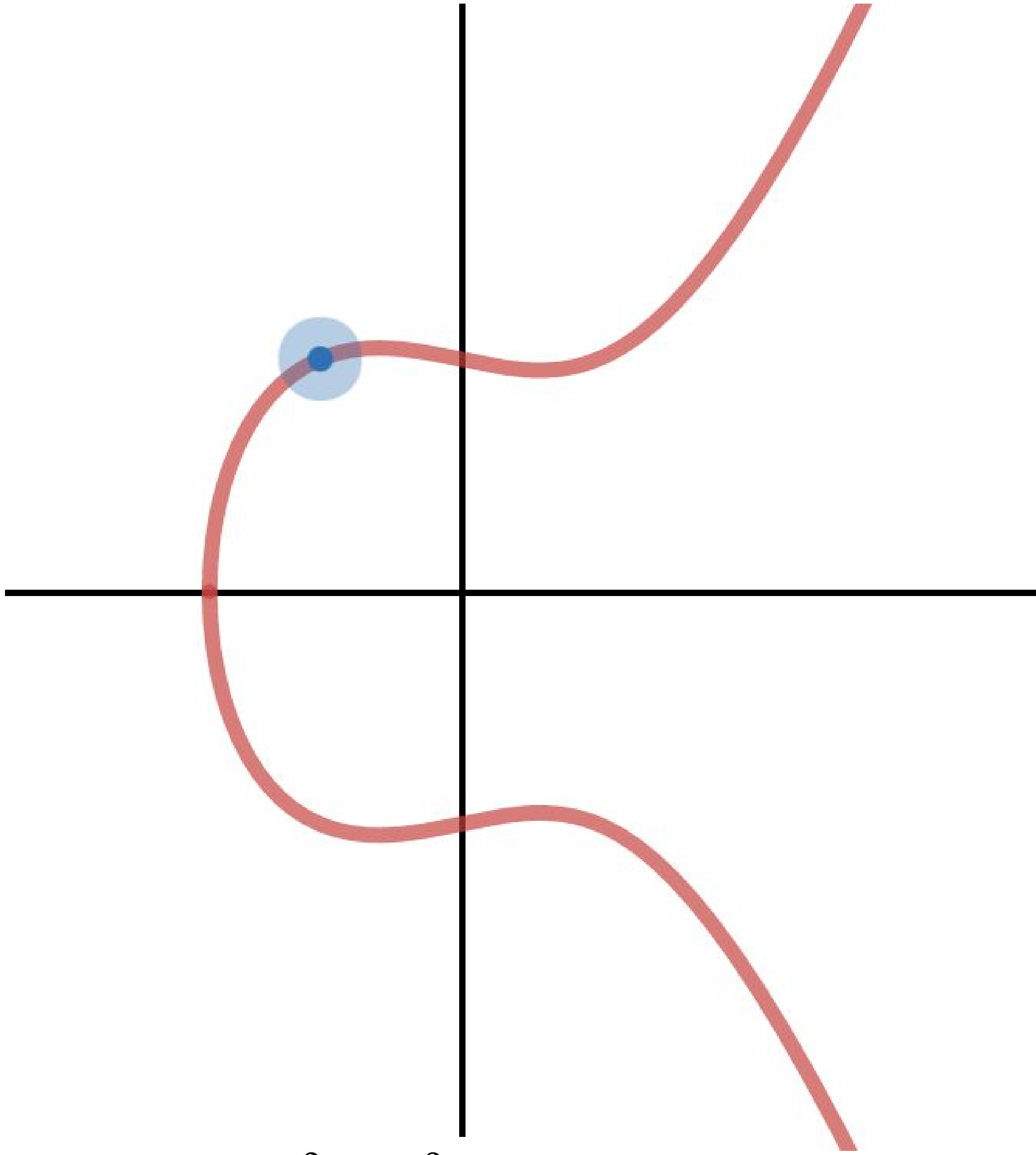


$y^2 = x^3 + ax + b$

$$P + \mathcal{O} = P$$

$$P + -P = \mathcal{O}$$

$$P + P + R = \mathcal{O}$$



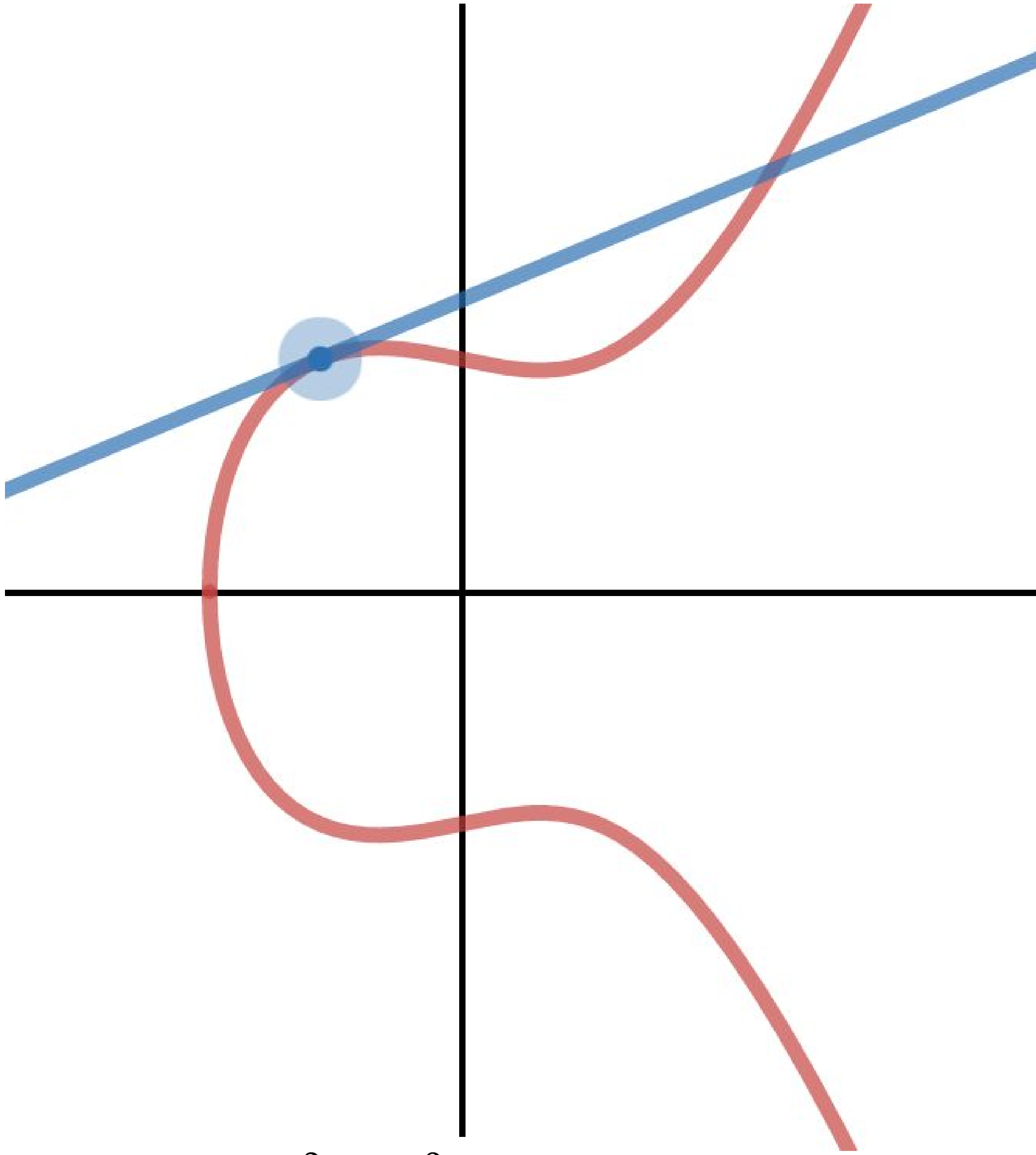
$y^2 = x^3 + ax + b$

$$P + \mathcal{O} = P$$

$$P + -P = \mathcal{O}$$

$$P + P + R = \mathcal{O}$$

$$P + P = -R = 2P$$



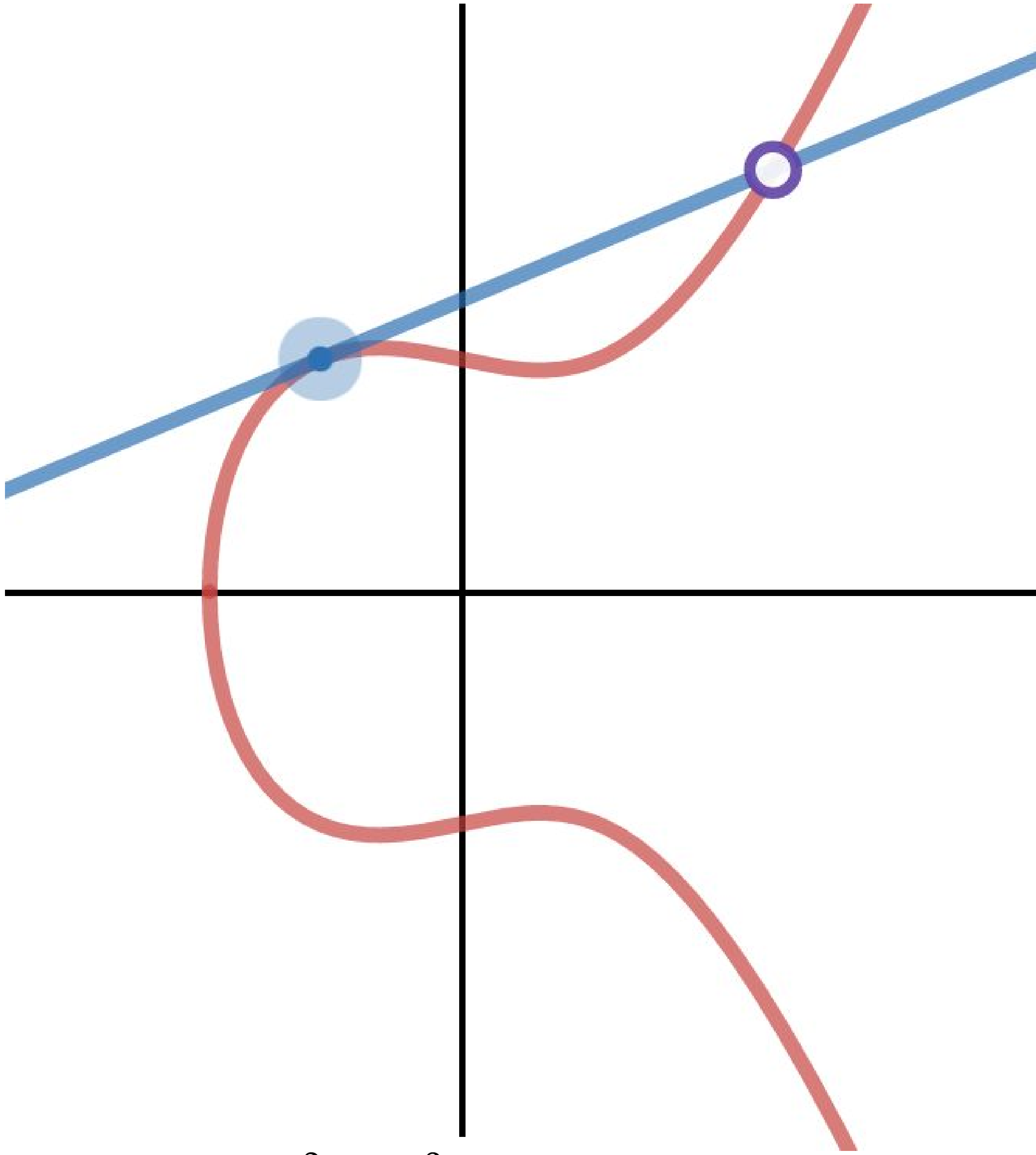
$y^2 = x^3 + ax + b$

$$P + \mathcal{O} = P$$

$$P + -P = \mathcal{O}$$

$$P + P + R = \mathcal{O}$$

$$P + P = -R = 2P$$



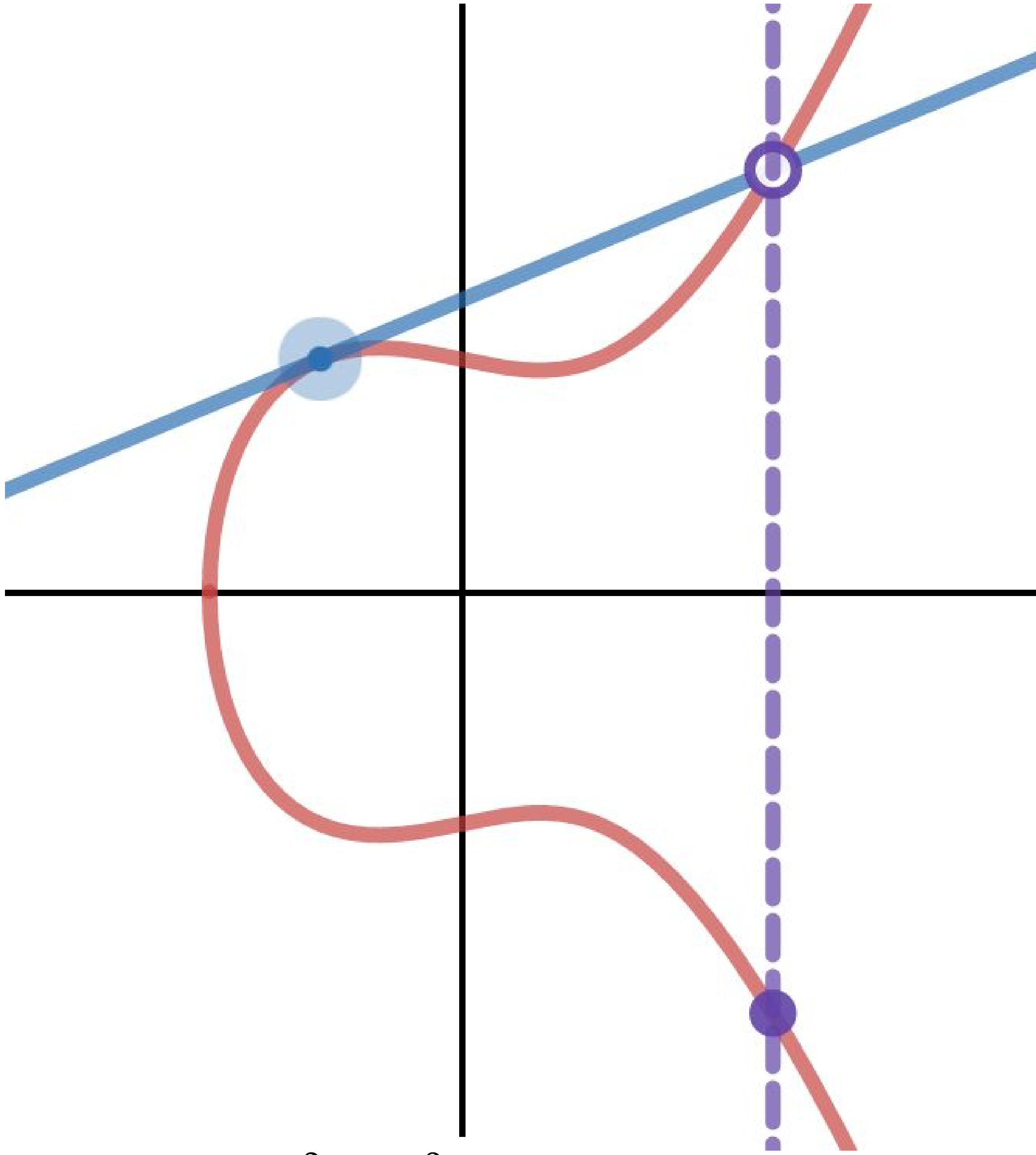
$y^2 = x^3 + ax + b$

$$P + \mathcal{O} = P$$

$$P + -P = \mathcal{O}$$

$$P + P + R = \mathcal{O}$$

$$P + P = -R = 2P$$



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

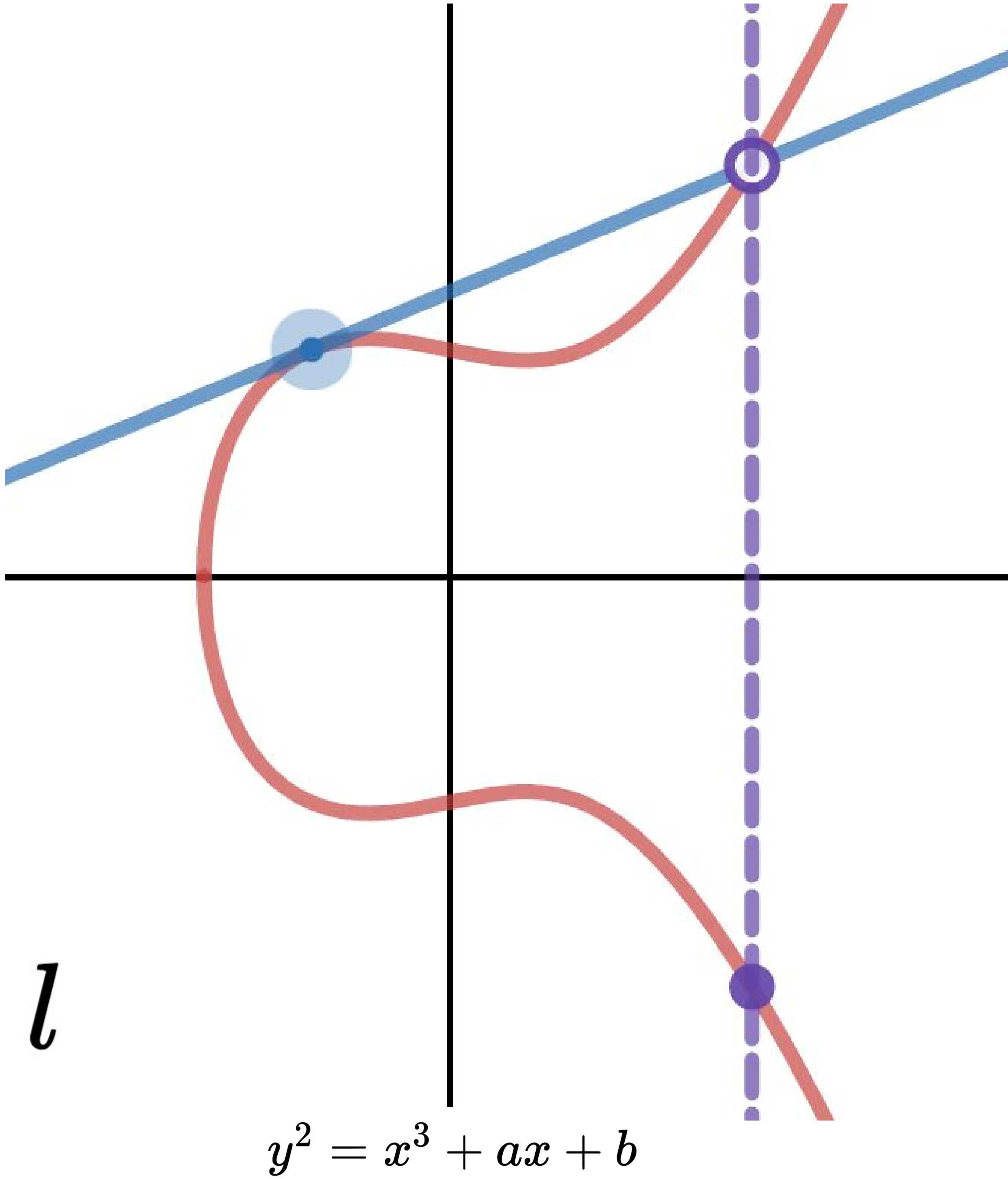
$$P + \mathcal{O} = P$$

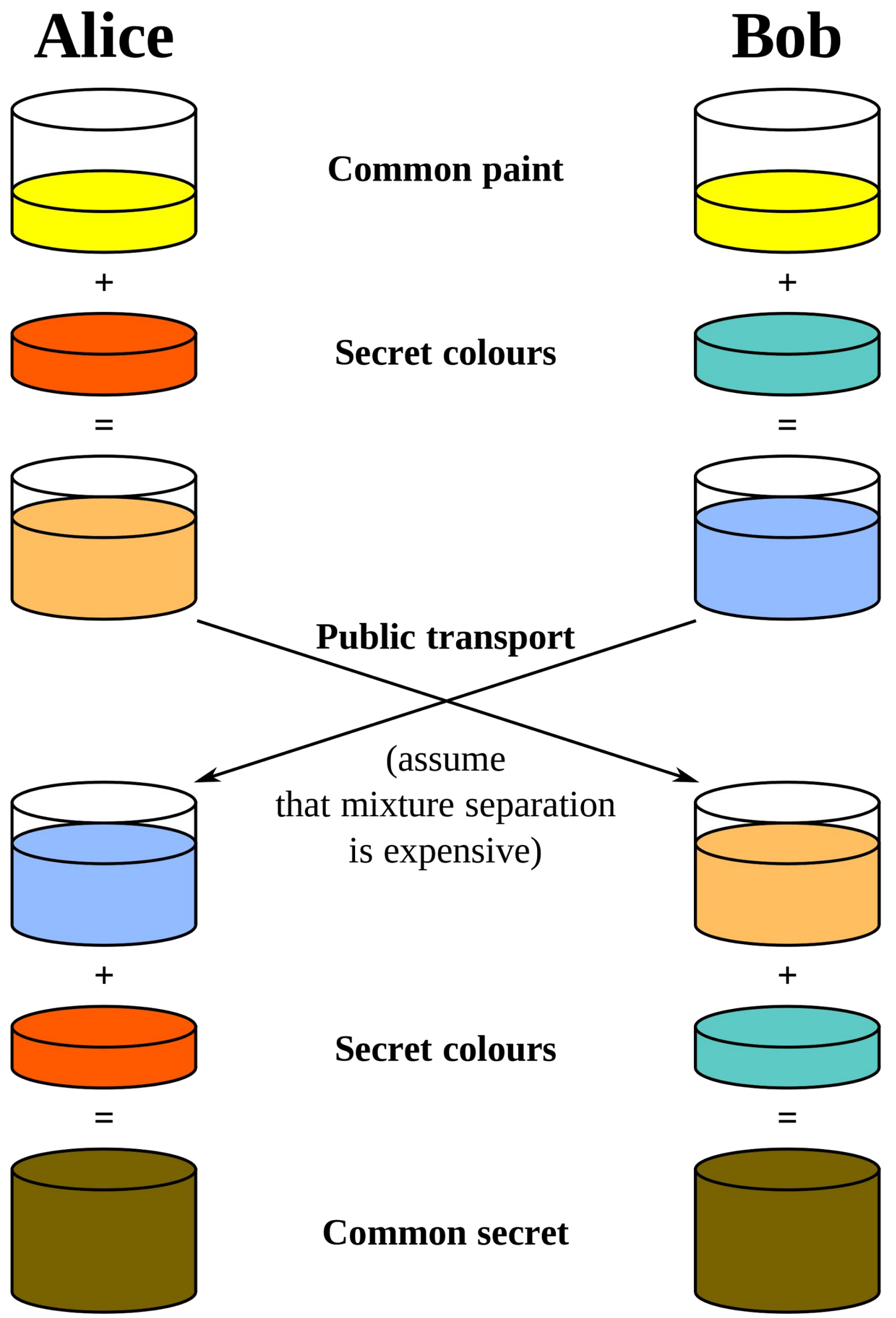
$$P + -P = \mathcal{O}$$

$$P + P + R = \mathcal{O}$$

$$P + P = -R = 2P$$

$$If\, lP = \mathcal{O}, ord(P) = l$$

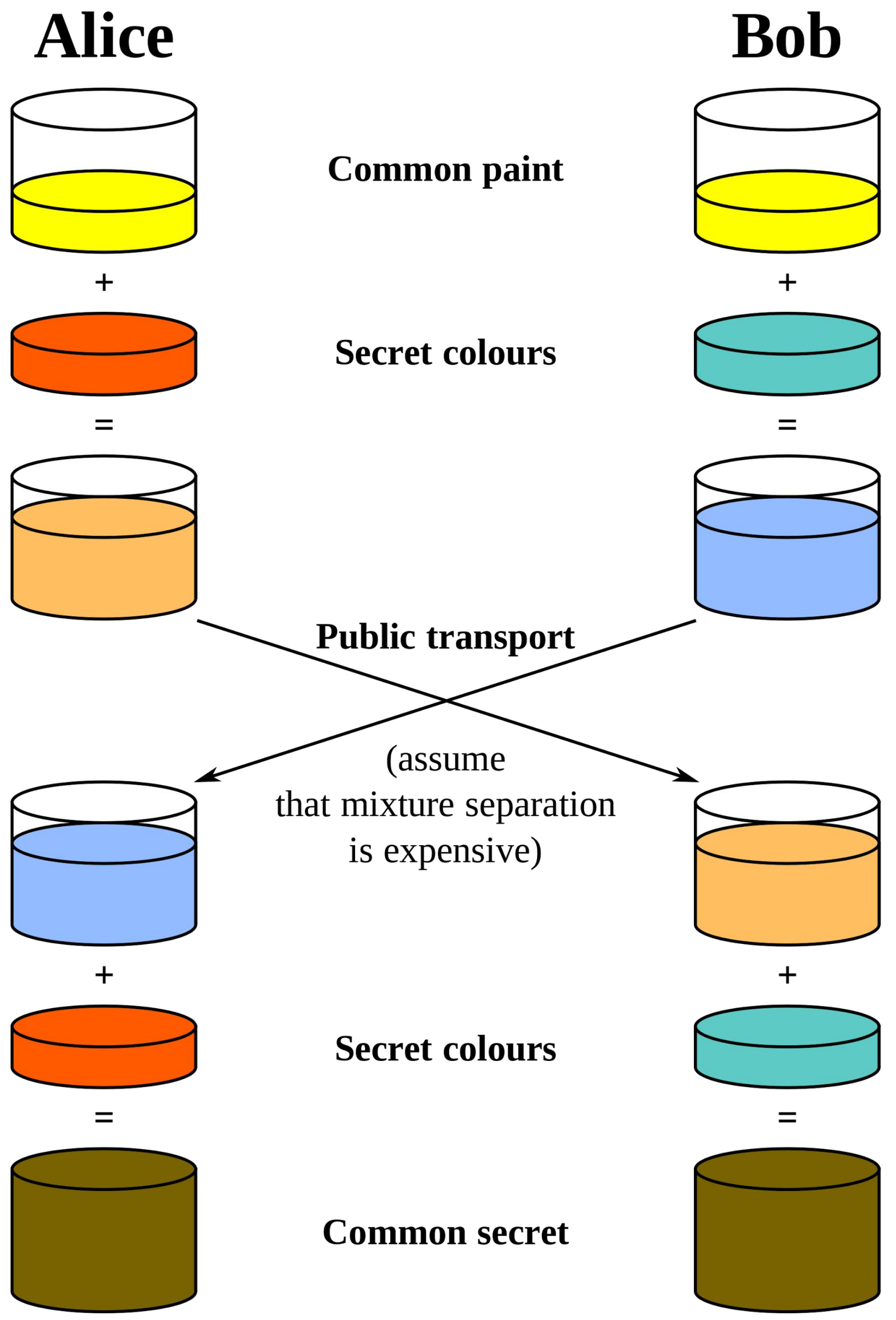




Diffie-Hellman Key Exchange

Originally **FFDH:** $key = g^{ab} \; mod \; p$

Exchange **integers**



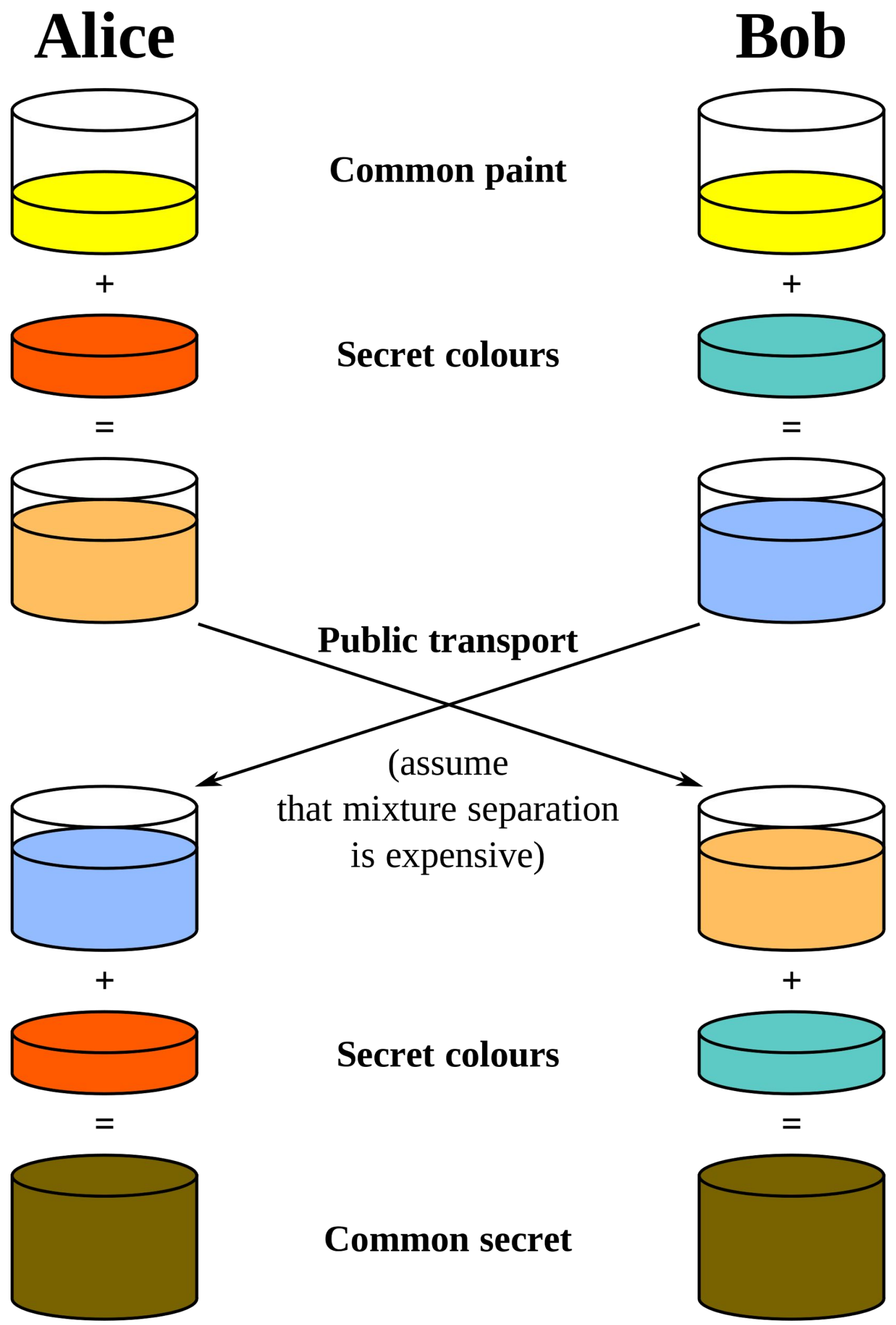
Diffie-Hellman Key Exchange

Originally **FFDH:** $key = g^{ab}\ mod\ p$

Exchange **integers**

Then **ECDH:** $key = abP\ mod\ p$

Exchange **points** on curve



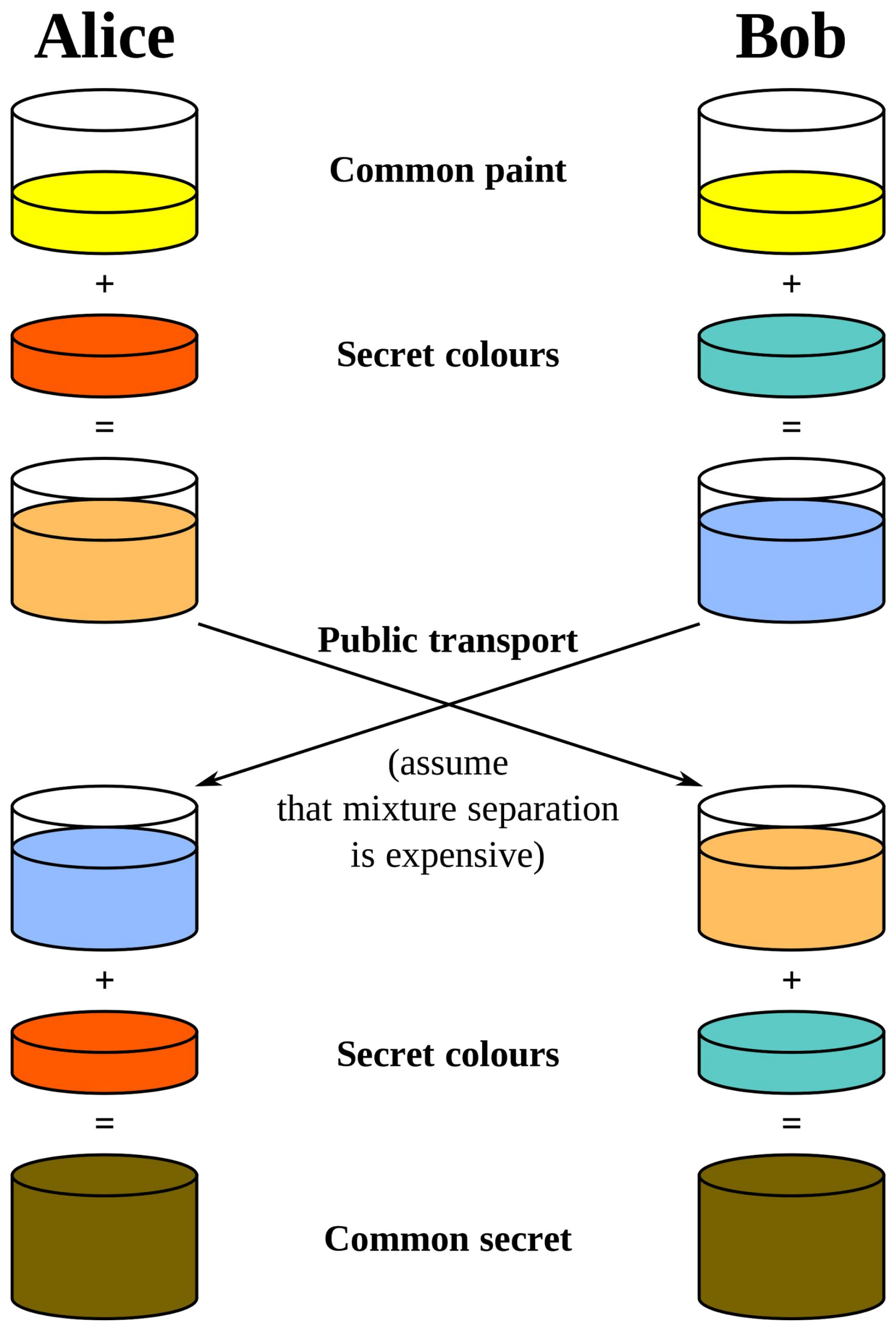
Diffie-Hellman Key Exchange

Originally **FFDH:** $key = g^{ab} \ mod \ p$

Exchange **integers**

Then **ECDH:** $key = abP \ mod \ p$

Exchange **points** on curve



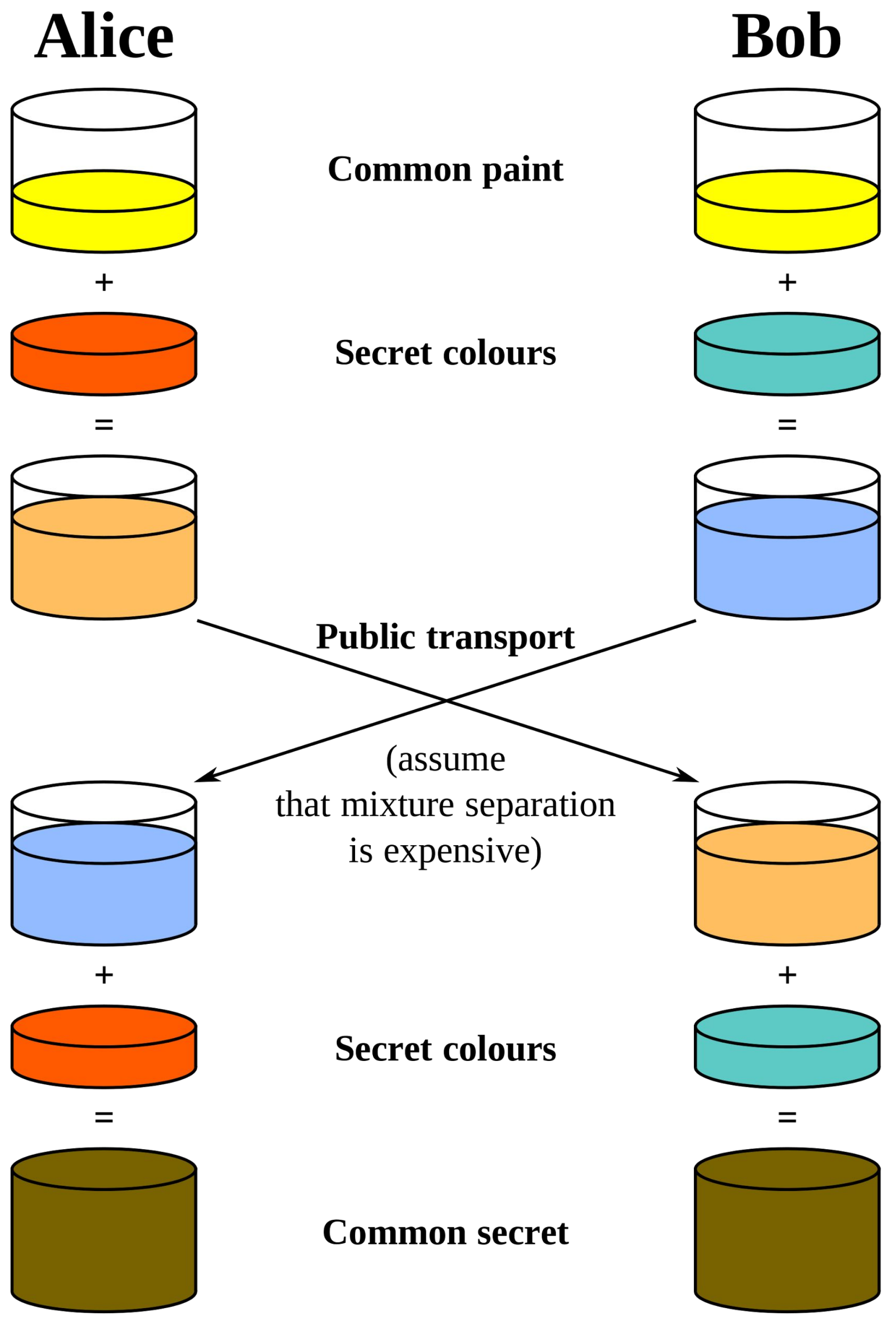
Diffie-Hellman Key Exchange

Originally **FFDH:** $key = g^{ab} \ mod \ p$

Exchange **integers**

Then **ECDH:** $key = abP \ mod \ p$

Exchange **points** on curve



Diffie-Hellman Key Exchange

Originally **FFDH:** $key = g^{ab} \bmod p$

Exchange **integers**

Then **ECDH:** $key = abP \bmod p$

Exchange **points** on curve

Now **SIDH:**

$$key = \phi'_a(\phi_b(E)) = \phi'_B(\phi_A(E))$$

Exchange whole **curves**

$$\phi : E \to E'$$

$$\phi : E \to E'$$

$$\phi(x, y) = (x', y')$$

$$\phi : E \to E'$$

$$\phi(x, y) = (x', y')$$

$$\phi(\mathcal{O}) = \mathcal{O}'$$

$$\phi(x, y) = -(x, y)$$

$$\phi(x, y) = -(x, y)$$

$$\phi(\mathcal{O}) = -(\mathcal{O}) = \mathcal{O}$$

$$\phi(x, y) = -(x, y)$$

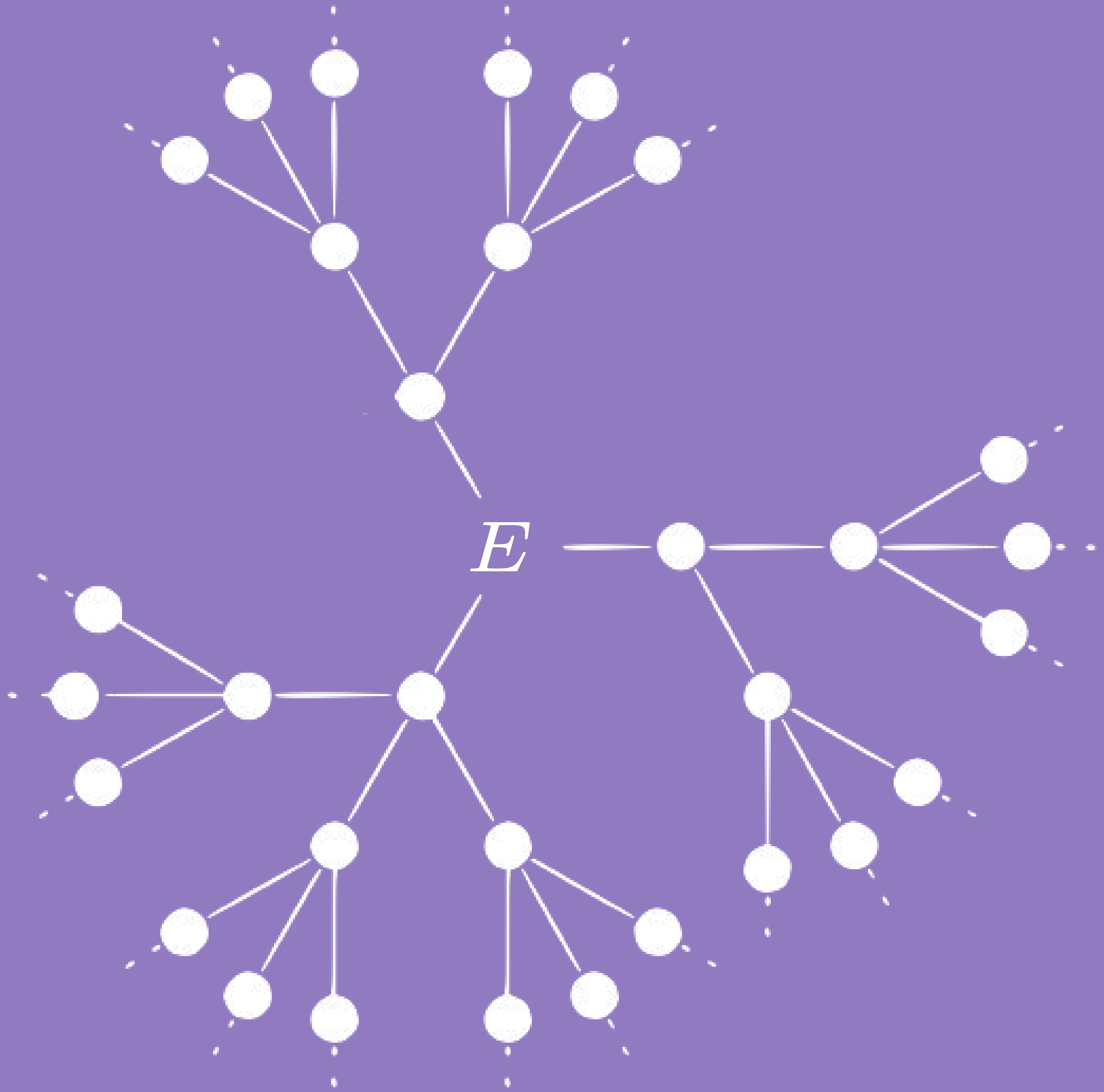
$$\phi(\mathcal{O}) = -(\mathcal{O}) = \mathcal{O}$$

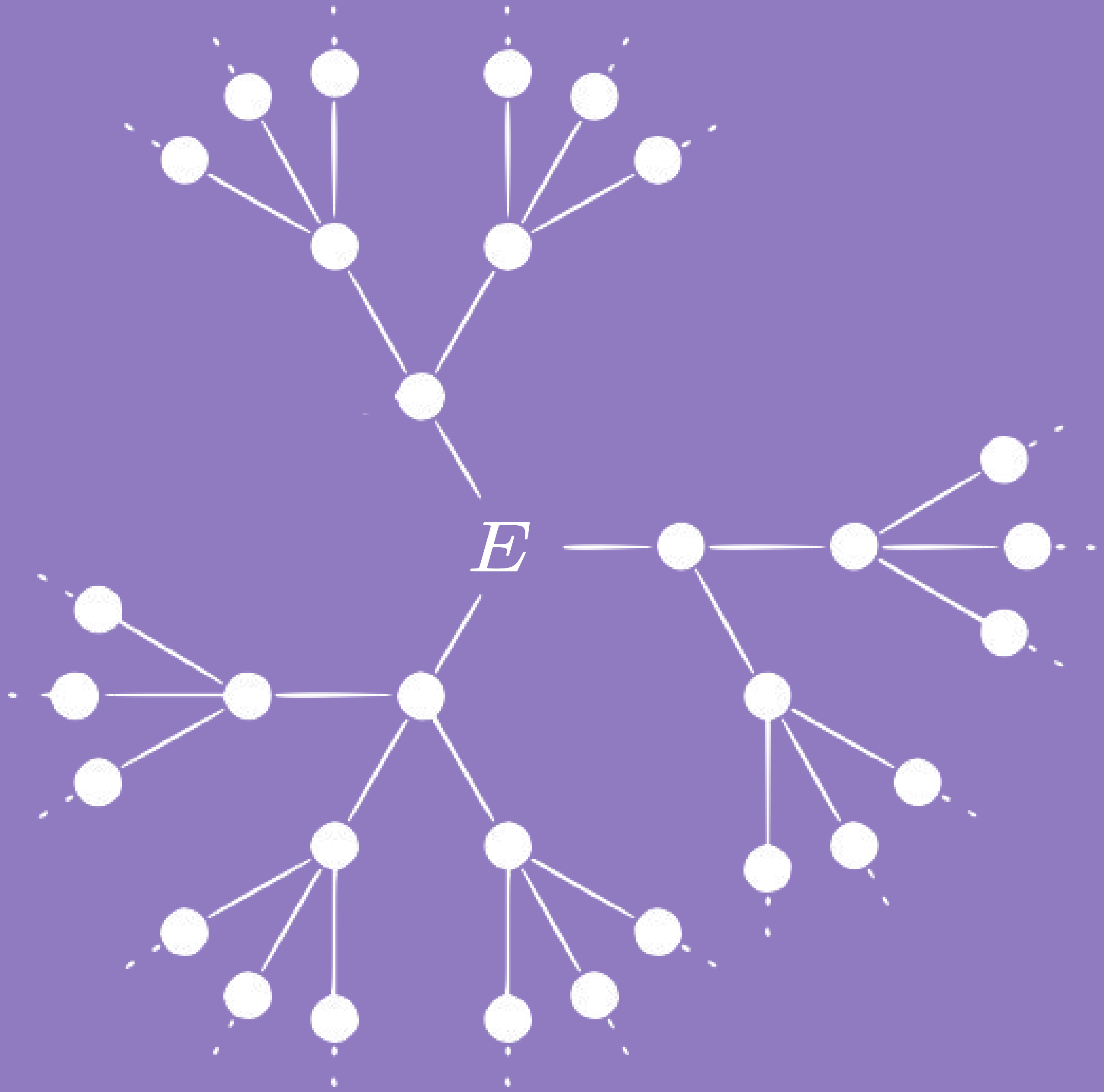
$$n\phi(\mathcal{O}) = \phi(n\mathcal{O}) = \mathcal{O}$$

$$\phi(x, y) = \left(\frac{x^{372} + x \ldots}{x^{279} + \ldots}, y \frac{x^{279} + x \ldots}{x^{372} + x \ldots} \right)$$

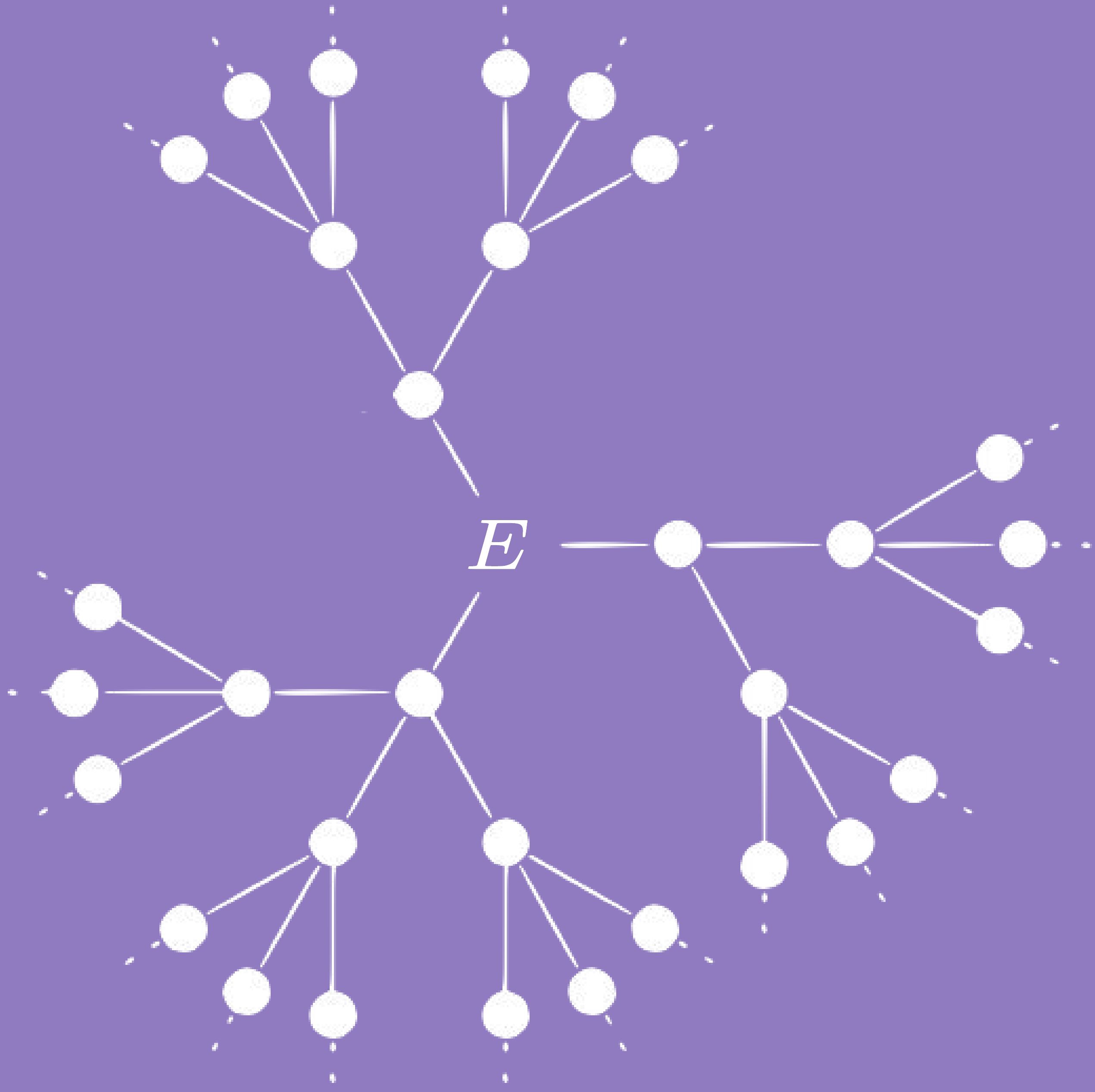
$$\phi_n = \phi_{n-1} \circ \cdots \circ \phi_0$$

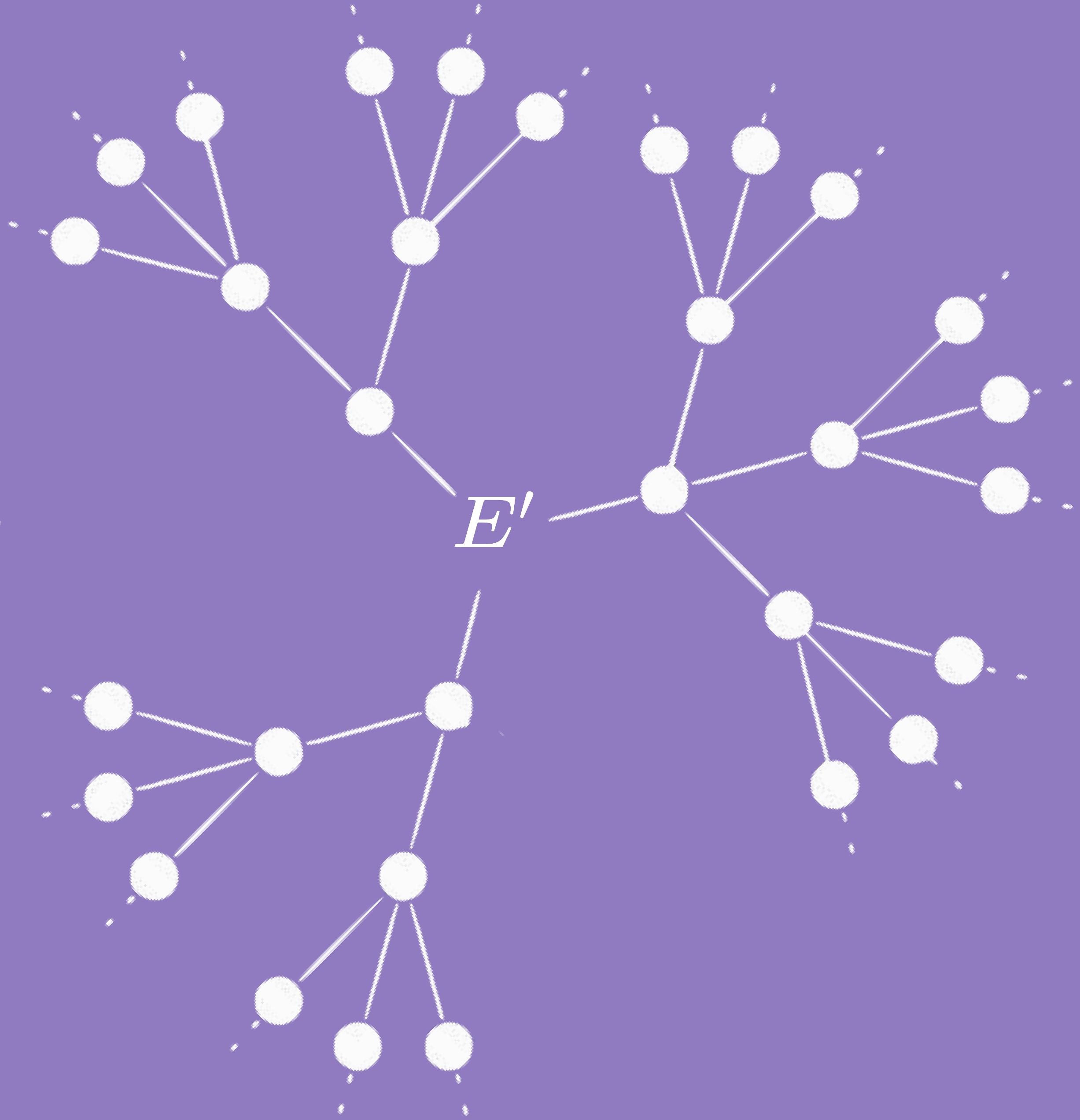
E

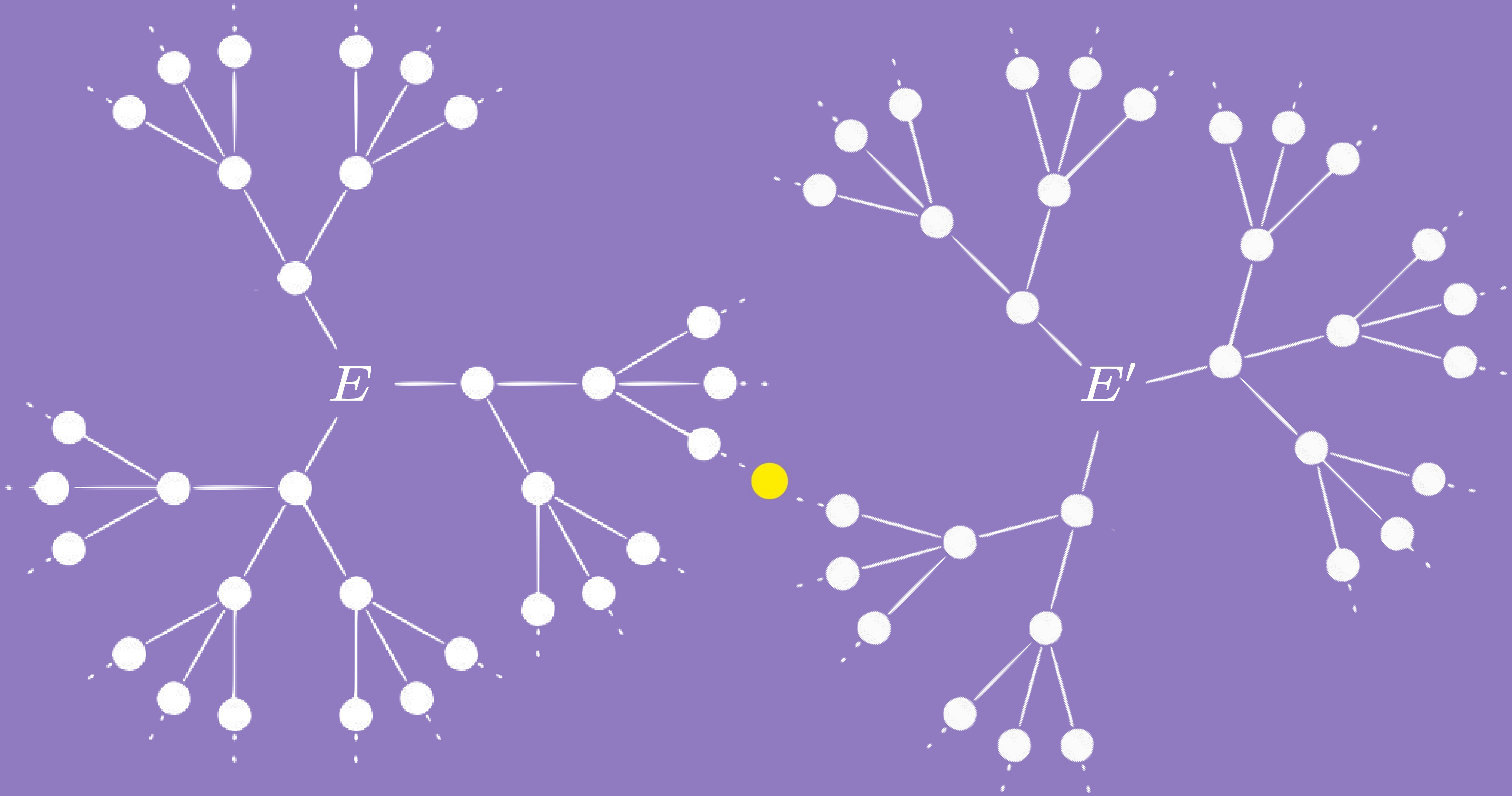

E

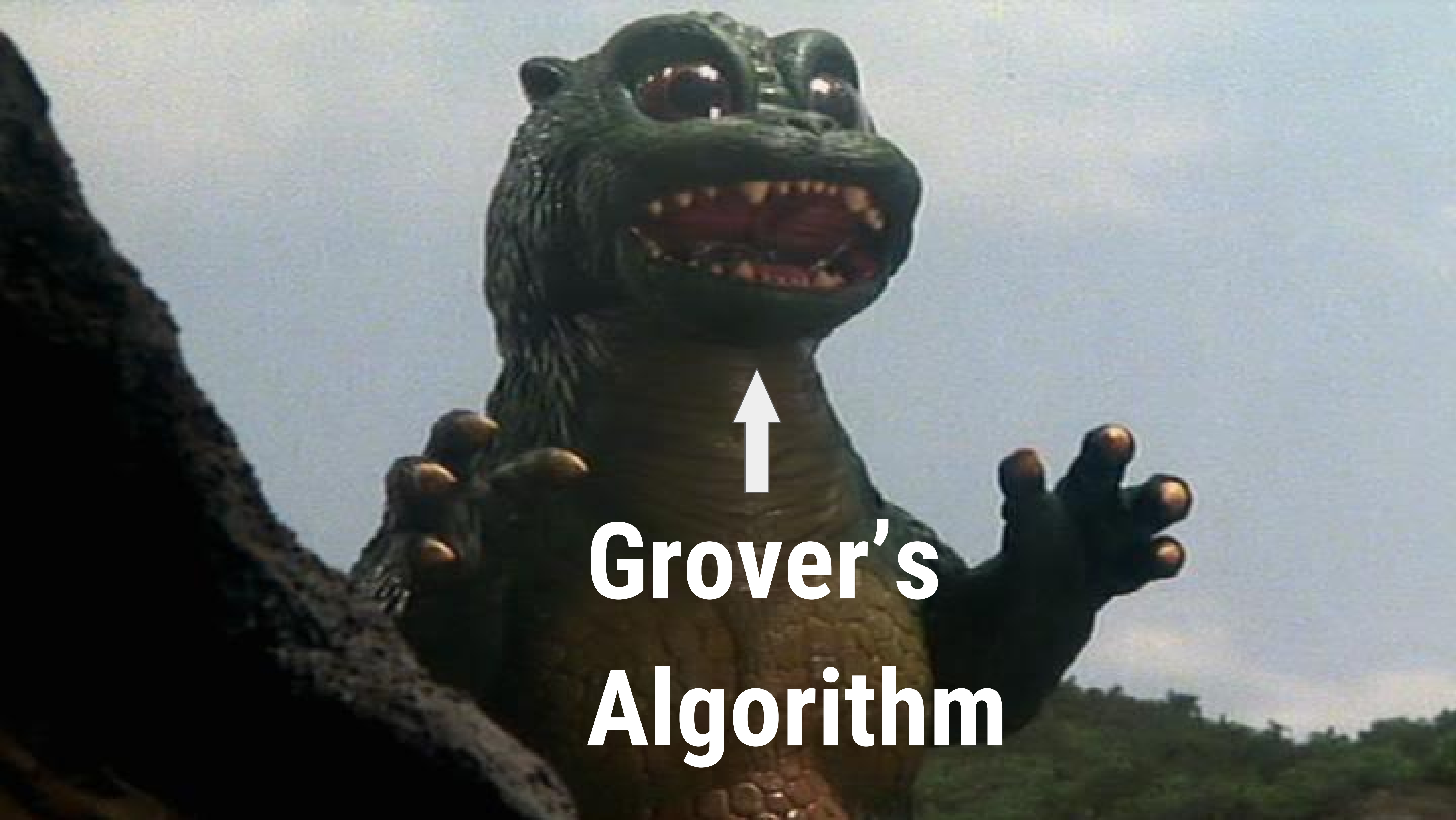


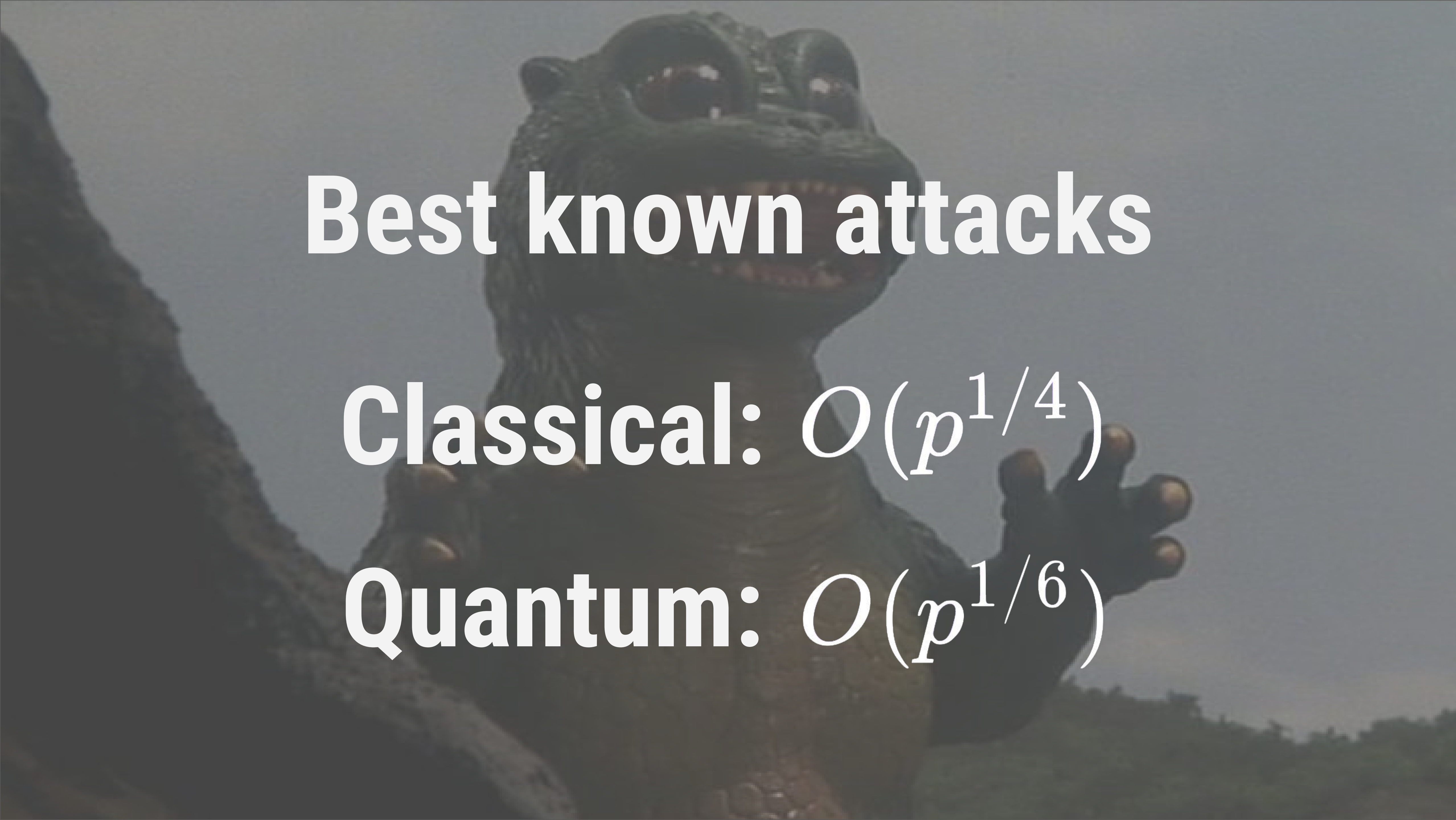
E'


E


E'


E
E'


Grover’s Algorithm



Best known attacks

Classical: $O(p^{1/4})$

Quantum: $O(p^{1/6})$

Alice

Bob

Alice	Bob
E	E

Alice	Bob
E	E
$+$	$+$
$\phi_A := E/\langle R_A \rangle$	$\phi_B := E/\langle R_B \rangle$

Alice	Bob
E	E
$+$	$+$
$\phi_A := E/\langle R_A \rangle$	$\phi_B := E/\langle R_B \rangle$
$=$	$=$
$E_A = \phi_A(E)$	$E_B = \phi_B(E)$

$$\begin{tikzcd}
\textbf{Alice} & \textbf{Bob} \\
E & E \\
+ & + \\
\phi_A := E/\langle R_A \rangle & \phi_B := E/\langle R_B \rangle \\
= & = \\
E_A = \phi_A(E) \arrow[dr] & E_B = \phi_B(E) \arrow[dl] \\
E_B & E_A
\end{tikzcd}$$

$$\begin{tikzcd}
\textbf{Alice} & \textbf{Bob} \\
E & E \\
+ & + \\
\phi_A := E/\langle R_A\rangle & \phi_B := E/\langle R_B\rangle \\
= & = \\
E_A = \phi_A(E) \arrow[dr] & E_B = \phi_B(E) \arrow[dl] \\
E_B & E_A \\
+ & + \\
\phi'_A := E_B/\langle \phi_B(R_A)\rangle & \phi'_B := E_A/\langle \phi_A(R_B)\rangle
\end{tikzcd}$$

$$\begin{tikzcd}
\textbf{Alice} & \textbf{Bob} \\
E & E \\
+ & + \\
\phi_A := E/\langle R_A \rangle & \phi_B := E/\langle R_B \rangle \\
= & = \\
E_A = \phi_A(E) \arrow[dr] & E_B = \phi_B(E) \arrow[dl] \\
E_B & E_A \\
+ & + \\
\phi'_A := E_B/\langle \phi_B(R_A) \rangle & \phi'_B := E_A/\langle \phi_A(R_B) \rangle \\
= & = \\
E_{AB} = \phi'_A(E_B) & E_{BA} = \phi'_B(E_A)
\end{tikzcd}$$

$$\begin{tikzcd}
\textbf{Alice} & \textbf{Bob} \\
E & E \\
+ & + \\
\phi_A := E/\langle R_A \rangle & \phi_B := E/\langle R_B \rangle \\
= & = \\
E_A = \phi_A(E) \arrow[dr] & E_B = \phi_B(E) \arrow[dl] \\
E_B & E_A \\
+ & + \\
\phi'_A := E_B/\langle \phi_B(R_A) \rangle & \phi'_B := E_A/\langle \phi_A(R_B) \rangle \\
= & = \\
E_{AB} = \phi'_A(E_B) & E_{BA} = \phi'_B(E_A)
\end{tikzcd}$$

$$j(E_{AB}) = j(E_{BA})$$

$$E$$

$$+$$

$$\phi_B := E/\langle R_B \rangle$$

$$=$$

$$E_B = \phi_B(E)$$

$$E_A$$

$$+$$

$$\phi'_B := E_A/\langle \phi_A(R_B) \rangle$$

$$=$$

$$E_{BA} = \phi'_B(E_A)$$

$$\boldsymbol{E}$$

$$+$$

$$\phi_B := E/\langle R_B \rangle$$

$$=$$

$$E_B = \phi_B(E)$$

$$E_A$$

$$+$$

$$\phi'_B := E_A/\langle \phi_A(R_B) \rangle$$

$$=$$

$$E_{BA} = \phi'_B(E_A)$$

$$\begin{tikzcd}
\boldsymbol{E} \\
+ \\
\phi_B := E/\langle R_B \rangle \\
= \\
E_B = \phi_B(E) \arrow[d, bend right] \\
E_A \\
+ \\
\phi_B' := E_A/\langle \phi_A(R_B) \rangle \\
= \\
E_{BA} = \phi_B'(E_A)
\end{tikzcd}$$

$$E/\mathbb{F}_{p^2}\colon y^2 = x^3 + x$$

$$E$$

$$+$$

$$\phi_B := E/\langle R_B \rangle$$

$$=$$

$$E_B = \phi_B(E)$$

$$E_A$$

$$+$$

$$\phi'_B := E_A/\langle \phi_A(R_B) \rangle$$

$$=$$

$$E_{BA} = \phi'_B(E_A)$$

$$E/\mathbb{F}_{p^2} \colon y^2 = x^3 + x$$

$$\langle P_A, Q_A \rangle = E[2^{372}]$$

$$\boldsymbol{E}$$

$$+$$

$$\phi_B := E/\langle R_B\rangle$$

$$=$$

$$E_B = \phi_B(E)$$

$$E_A$$

$$+$$

$$\phi_B' := E_A/\langle \phi_A(R_B)\rangle$$

$$=$$

$$E_{BA} = \phi_B'(E_A)$$

$$E/\mathbb{F}_{p^2}\colon y^2 = x^3 + x$$

$$\langle P_A, Q_A\rangle = E[2^{372}]$$

$$\langle P_B, Q_B\rangle = E[3^{239}]$$

$$\begin{tikzcd}
E \\
+ \\
\phi_B := E/\langle R_B \rangle \\
= \\
E_B = \phi_B(E) \arrow[d, bend right] \\
E_A \\
+ \\
\phi'_B := E_A/\langle \phi_A(R_B) \rangle \\
= \\
E_{BA} = \phi'_B(E_A)
\end{tikzcd}$$

$$\begin{tikzcd}
E \\
+ \\
\phi_B := E/\langle R_B \rangle \\
= \\
E_B = \phi_B(E) \arrow[d, bend right] \\
E_A \\
+ \\
\phi_B' := E_A/\langle \phi_A(R_B) \rangle \\
= \\
E_{BA} = \phi_B'(E_A)
\end{tikzcd}$$

$$R_B = mP_B + nQ_B$$

$$E$$

$$+$$

$$\phi_B := E/\langle R_B \rangle$$

$$=$$

$$E_B = \phi_B(E)$$

$$E_A$$

$$+$$

$$\phi'_B := E_A/\langle \phi_A(R_B) \rangle$$

$$=$$

$$E_{BA} = \phi'_B(E_A)$$

$$R_B = mP_B + nQ_B$$

$$\langle R_B \rangle = ker(\phi_B)$$

$$E$$

$$+$$

$$\phi_B := E/\langle R_B \rangle$$

$$=$$

$$E_B = \phi_B(E)$$

$$E_A$$

$$+$$

$$\phi'_B := E_A/\langle \phi_A(R_B) \rangle$$

$$=$$

$$E_{BA} = \phi'_B(E_A)$$

$$R_B = mP_B + nQ_B$$

$$\langle R_B \rangle = \boldsymbol{ker}(\phi_B)$$

$$\phi_B := E/\langle R_B \rangle$$

$$E$$

$$+$$

$$\phi_B := E/\langle R_B \rangle$$

$$=$$

$$\begin{tikzcd}
E_B = \phi_B(E) \arrow[d] \\
E_A
\end{tikzcd}$$

$$+$$

$$\phi'_B := E_A/\langle \phi_A(R_B) \rangle$$

$$=$$

$$E_{BA} = \phi'_B(E_A)$$

$$\begin{tikzcd} E \\ + \\ \phi_B := E/\langle R_B \rangle \\ = \\ E_B = \phi_B(E) \arrow[d, bend right] \\ E_A \\ + \\ \phi_B' := E_A/\langle \phi_A(R_B) \rangle \\ = \\ E_{BA} = \phi_B'(E_A) \end{tikzcd}$$

$$E_B = \phi_B(E)$$

$$\begin{tikzcd}
E \\
+ \\
\phi_B := E/\langle R_B \rangle \\
= \\
E_B = \phi_B(E) \arrow[d] \\
E_A \\
+ \\
\phi'_B := E_A/\langle \phi_A(R_B) \rangle \\
= \\
E_{BA} = \phi'_B(E_A)
\end{tikzcd}$$

$$E_B = \phi_B(E)$$

$$E_B, \phi_B(P_A), \phi_B(Q_A)$$

$$E$$

$$+$$

$$\phi_B := E/\langle R_B \rangle$$

$$=$$

$$\begin{tikzcd} E_B = \phi_B(E) \arrow[d] \\ \boldsymbol{E_A} \end{tikzcd}$$

$$+$$

$$\phi'_B := E_A/\langle \phi_A(R_B) \rangle$$

$$=$$

$$E_{BA} = \phi'_B(E_A)$$

$$E$$

$$+$$

$$\phi_B := E/\langle R_B \rangle$$

$$=$$

$$E_B = \phi_B(E)$$

$$E_A$$

$$+$$

$$\phi_B' := E_A/\langle \phi_A(R_B) \rangle$$

$$=$$

$$E_{BA} = \phi_B'(E_A)$$

$$E_A, \phi_A(P_B), \phi_A(Q_B)$$

$$E$$

$$+$$

$$\phi_B := E/\langle R_B \rangle$$

$$=$$

$$\begin{tikzcd} E_B = \phi_B(E) \arrow[d] \\ E_A \end{tikzcd}$$

$$+$$

$$\phi'_B := E_A/\langle \phi_A(R_B) \rangle$$

$$=$$

$$E_{BA} = \phi'_B(E_A)$$

$$\begin{tikzcd}
E \\
+ \\
\phi_B := E/\langle R_B \rangle \\
= \\
E_B = \phi_B(E) \arrow[d, bend right] \\
E_A \\
+ \\
\phi'_B := E_A/\langle \phi_A(R_B) \rangle \\
= \\
E_{BA} = \phi'_B(E_A)
\end{tikzcd}$$

$$R'_B = m\phi_A(P_B) + n\phi_A(Q_B)$$

$$\begin{tikzcd}
E \\
+ \\
\phi_B := E/\langle R_B\rangle \\
= \\
E_B = \phi_B(E) \arrow[d, bend right] \\
E_A \\
+ \\
\phi'_B := E_A/\langle \phi_A(R_B)\rangle \\
= \\
E_{BA} = \phi'_B(E_A)
\end{tikzcd}$$

$$R'_B = \phi_A(mP_B + nQ_B)$$

$$E$$

$$+$$

$$\phi_B := E/\langle R_B \rangle$$

$$=$$

$$E_B = \phi_B(E)$$

$$E_A$$

$$+$$

$$\phi'_B := E_A/\langle \phi_A(R_B) \rangle$$

$$=$$

$$E_{BA} = \phi'_B(E_A)$$

$$R'_B = \phi_A(R_B)$$

$$\begin{tikzcd}
E \\
+ \\
\phi_B := E/\langle R_B\rangle \\
= \\
E_B = \phi_B(E) \arrow[d, bend right] \\
E_A \\
+ \\
\phi'_B := E_A/\langle \phi_A(R_B)\rangle \\
= \\
E_{BA} = \phi'_B(E_A)
\end{tikzcd}$$

$$R'_B = \phi_A(R_B)$$

$$\langle \phi_A(R_B)\rangle = ker(\phi'_B)$$

$$\begin{tikzcd}
E \\
+ \\
\phi_B := E/\langle R_B \rangle \\
= \\
E_B = \phi_B(E) \arrow[d, bend right] \\
E_A \\
+ \\
\phi'_B := E_A/\langle \phi_A(R_B) \rangle \\
= \\
E_{BA} = \phi'_B(E_A)
\end{tikzcd}$$

$$R'_B = \phi_A(R_B)$$

$$\langle \phi_A(R_B) \rangle = ker(\phi'_B)$$

$$\phi'_B := E_A/\langle \phi_A(R_B) \rangle$$

$$E$$

$$+$$

$$\phi_B := E/\langle R_B \rangle$$

$$=$$

$$E_B = \phi_B(E)$$

$$\downarrow$$

$$E_A$$

$$+$$

$$\phi'_B := E_A/\langle \phi_A(R_B) \rangle$$

$$=$$

$$E_{BA} = \phi'_B(E_A)$$

$$E$$

$$+$$

$$\phi_B := E/\langle R_B \rangle$$

$$=$$

$$\begin{tikzcd}
E_B = \phi_B(E) \arrow[d] \\
E_A
\end{tikzcd}$$

$$+$$

$$\phi'_B := E_A/\langle \phi_A(R_B) \rangle$$

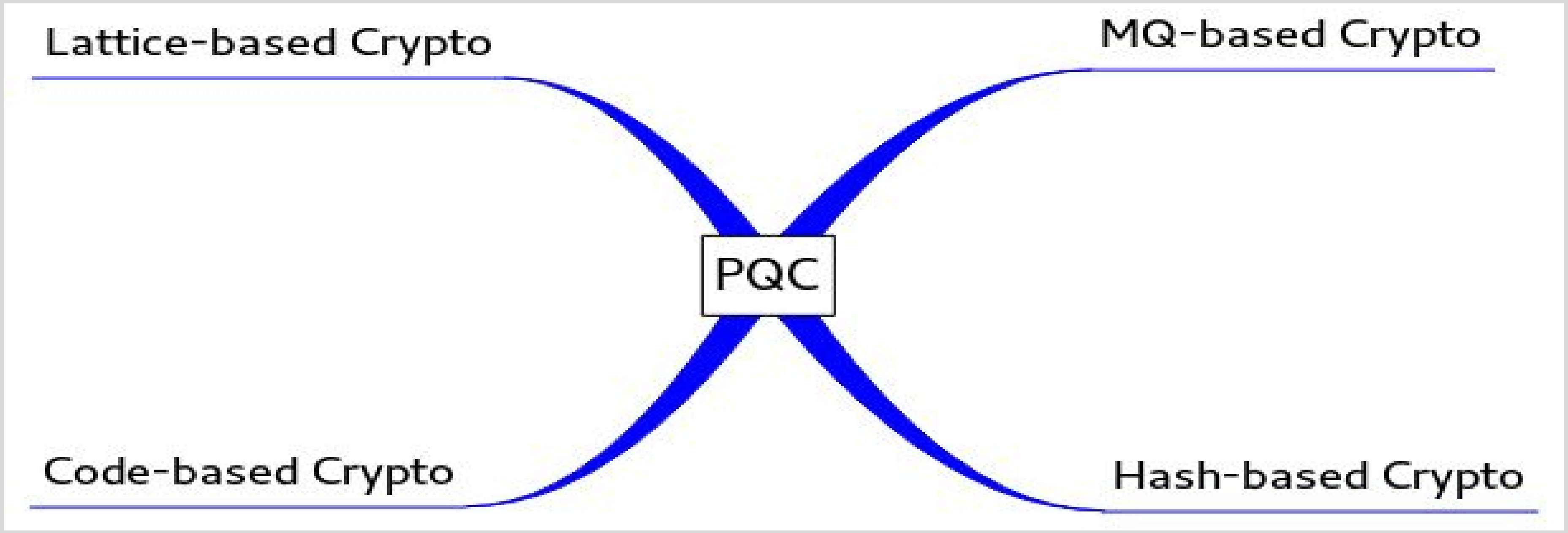
$$=$$

$$E_{BA} = \phi'_B(E_A)$$

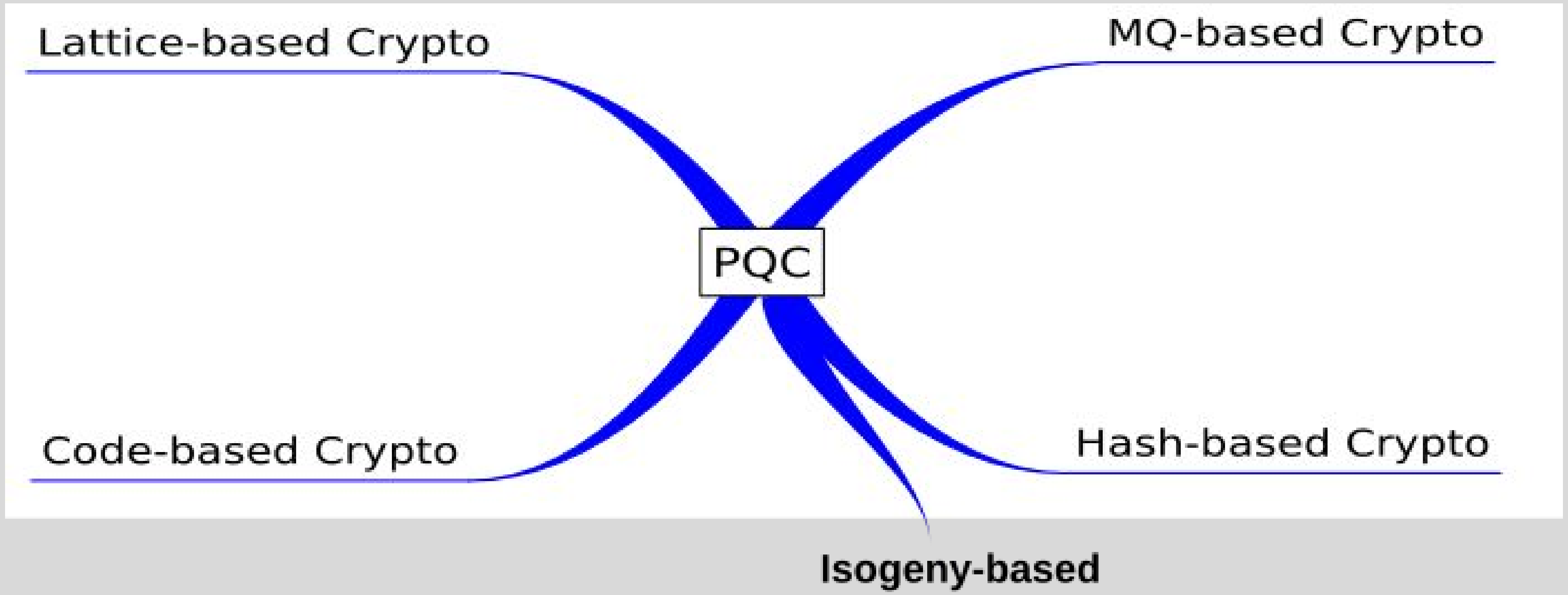
$$E_{BA} = \phi'_B(E_A)$$

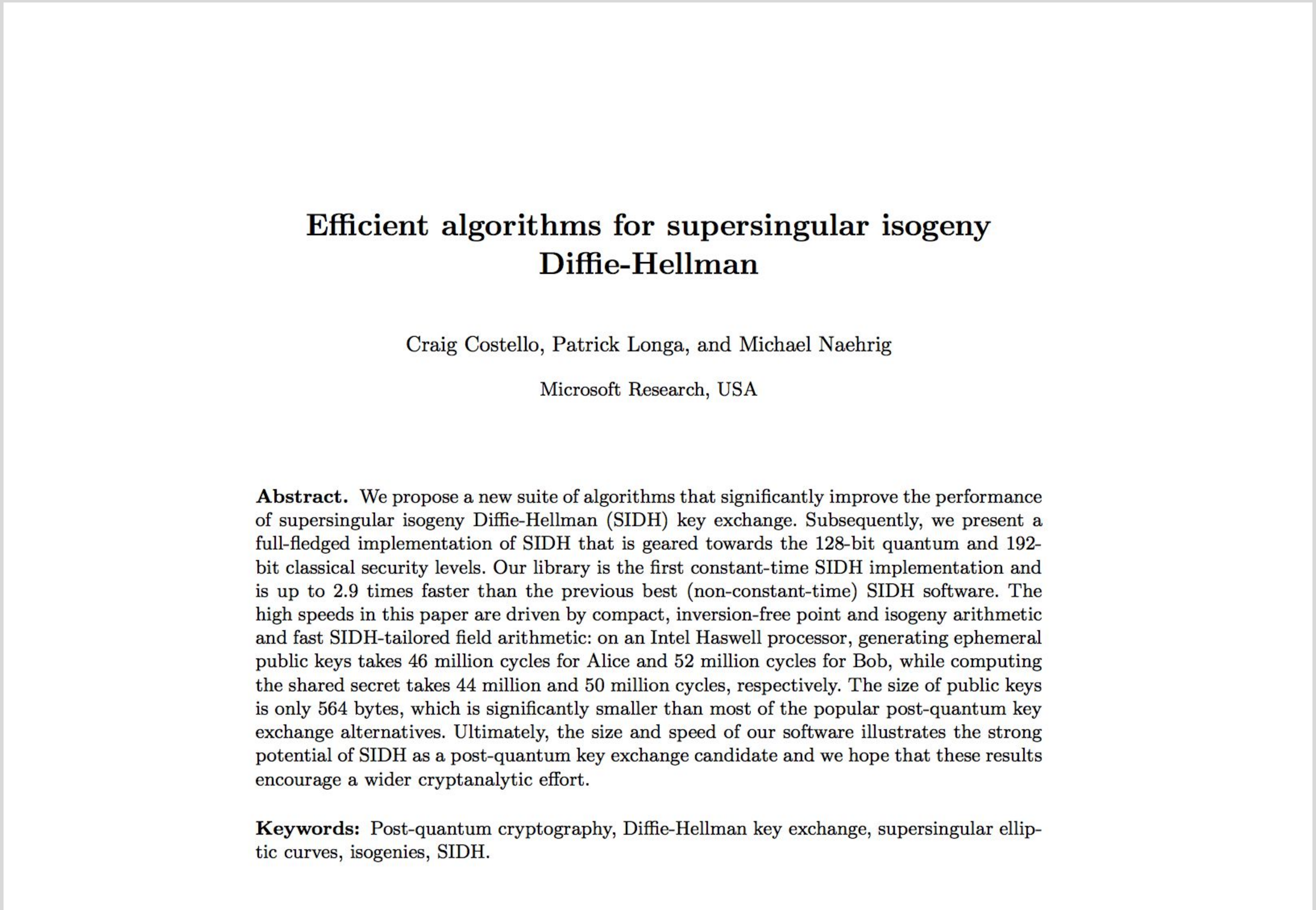
$$=$$

$$E/\langle R_A, R_B \rangle$$


Lattice-based Crypto
MQ-based Crypto
PQC
Code-based Crypto
Hash-based Crypto

ISOGENIES
HD
HISTORY.COM
imgflip.com


Lattice-based Crypto
MQ-based Crypto
PQC
Code-based Crypto
Hash-based Crypto
Isogeny-based



Efficient algorithms for supersingular isogeny Diffie-Hellman

Craig Costello, Patrick Longa, and Michael Naehrig

Microsoft Research, USA

Abstract. We propose a new suite of algorithms that significantly improve the performance of supersingular isogeny Diffie-Hellman (SIDH) key exchange. Subsequently, we present a full-fledged implementation of SIDH that is geared towards the 128-bit quantum and 192-bit classical security levels. Our library is the first constant-time SIDH implementation and is up to 2.9 times faster than the previous best (non-constant-time) SIDH software. The high speeds in this paper are driven by compact, inversion-free point and isogeny arithmetic and fast SIDH-tailored field arithmetic: on an Intel Haswell processor, generating ephemeral public keys takes 46 million cycles for Alice and 52 million cycles for Bob, while computing the shared secret takes 44 million and 50 million cycles, respectively. The size of public keys is only 564 bytes, which is significantly smaller than most of the popular post-quantum key exchange alternatives. Ultimately, the size and speed of our software illustrates the strong potential of SIDH as a post-quantum key exchange candidate and we hope that these results encourage a wider cryptanalytic effort.

Keywords: Post-quantum cryptography, Diffie-Hellman key exchange, supersingular elliptic curves, isogenies, SIDH.

Table 2. Performance results (expressed in millions of clock cycles) of the proposed SIDH implementation in comparison with the implementation by Azarderakhsh et al. [2] on x64 platforms. Benchmark tests were taken with Intel's TurboBoost disabled and the results were rounded to the nearest 10^6 clock cycles. Benchmarks were done on a 3.4GHz Intel Core i7-2600 Sandy Bridge and a 3.4GHz Intel Core i7-4770 Haswell processor running Ubuntu 14.04 LTS.

Operation	This work		Prior work [2]	
	Sandy Bridge	Haswell	Sandy Bridge	Haswell
Alice's keygen	50	46	165	149
Bob's keygen	57	52	172	152
Alice's shared key	47	44	133	118
Bob's shared key	55	50	137	122
Total	207	192	608	540

Remark 7. In Section 4 we discussed several specialized choices that were made for reasons unrelated to performance, e.g., in the name of simplicity and/or compactness. We stress that, should future cryptanalysis reveal that these choices introduce a security vulnerability, the performance of SIDH and the performance improvements in Sections 3 and 5 are unlikely to be affected (in any meaningful way) by reverting back to the more general case(s). In particular, if it turns out that sampling from a fraction of the possible 2- and 3-torsion subgroups gives an attacker some appreciable advantage, then modifying the code to sample from the full set of torsion subgroups is merely an exercise, and the subsequent performance difference would be unnoticeable. Similarly, if any of (i) starting on a subfield curve (see Remark 2), (ii) using of the base-field and trace-zero subgroups, or (iii) using the distortion map, turns out to degrade SIDH security, then the main upshot of reverting to randomized public generators or starting on a curve minimally defined over $\mathbb{F}_{p^2}$ would be the inflated public parameters (see Section 6); the slowdown during key generation would be minor and the shared secret computations would be unchanged.

8 BigMont: a strong ECDH+SIDH hybrid

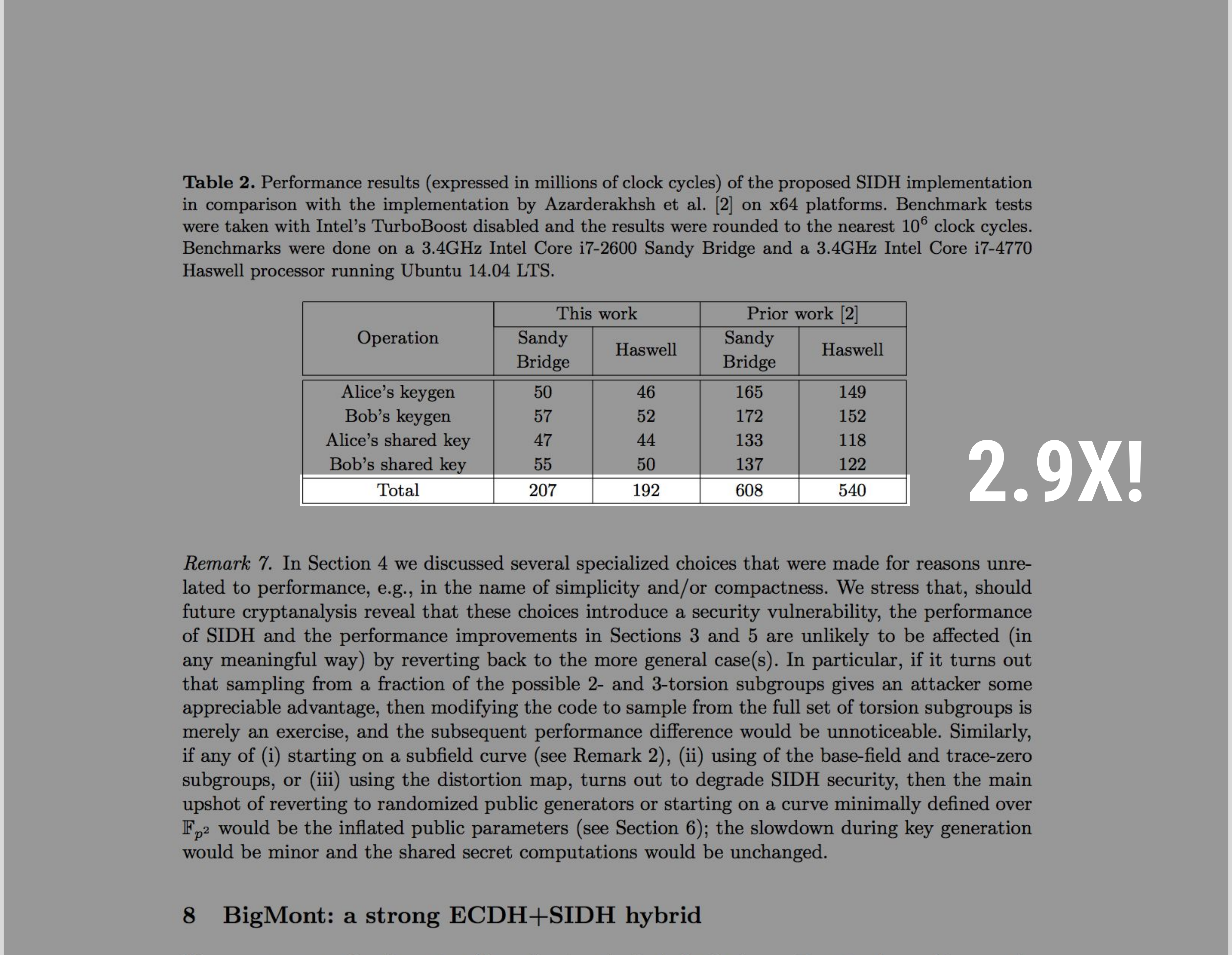


Table 2. Performance results (expressed in millions of clock cycles) of the proposed SIDH implementation in comparison with the implementation by Azarderakhsh et al. [2] on x64 platforms. Benchmark tests were taken with Intel's TurboBoost disabled and the results were rounded to the nearest 10^6 clock cycles. Benchmarks were done on a 3.4GHz Intel Core i7-2600 Sandy Bridge and a 3.4GHz Intel Core i7-4770 Haswell processor running Ubuntu 14.04 LTS.

Operation	This work		Prior work [2]	
	Sandy Bridge	Haswell	Sandy Bridge	Haswell
Alice's keygen	50	46	165	149
Bob's keygen	57	52	172	152
Alice's shared key	47	44	133	118
Bob's shared key	55	50	137	122
Total	207	192	608	540

2.9X!

Remark 7. In Section 4 we discussed several specialized choices that were made for reasons unrelated to performance, e.g., in the name of simplicity and/or compactness. We stress that, should future cryptanalysis reveal that these choices introduce a security vulnerability, the performance of SIDH and the performance improvements in Sections 3 and 5 are unlikely to be affected (in any meaningful way) by reverting back to the more general case(s). In particular, if it turns out that sampling from a fraction of the possible 2- and 3-torsion subgroups gives an attacker some appreciable advantage, then modifying the code to sample from the full set of torsion subgroups is merely an exercise, and the subsequent performance difference would be unnoticeable. Similarly, if any of (i) starting on a subfield curve (see Remark 2), (ii) using of the base-field and trace-zero subgroups, or (iii) using the distortion map, turns out to degrade SIDH security, then the main upshot of reverting to randomized public generators or starting on a curve minimally defined over $\mathbb{F}_{p^2}$ would be the inflated public parameters (see Section 6); the slowdown during key generation would be minor and the shared secret computations would be unchanged.

8 BigMont: a strong ECDH+SIDH hybrid

		SIDH	SIDH+ECDH
≈ bit-security (hard problem)	classical	192 (SSDDH)	384 (ECDHP)
	PQ	128 (SSDDH)	128 (SSDDH)
public key size		564	658
speed ($\text{cc} \times 10^6$)	Alice's keygen	46	52
	Bob's keygen	52	58
	Alice's shared key	44	50
	Bob's shared key	50	57

In Table 3 we compare hybrid SIDH+ECDH versus standalone SIDH. The take-away message is that for a less than 1.17x increase in public key sizes and less than 1.13x increase in the overall computing cost, we can increase the classical security of the key exchange from 192 bits (based on the relatively new SSDDH problem) to 384 bits (based on the long-standing ECDLP).

9 Validating public keys

Recall from Section 2 that De Feo, Jao and Plût [18] prove that SIDH is *session-key secure* (under SSDDH) in the authenticated-links adversarial model [12]. This model assumes perfectly authenticated links which effectively forces adversaries to be passive eavesdroppers; in particular, it assumes that public keys are correctly generated by honest users. While this model can be suitable for key exchange protocols that are instantiated in a truly ephemeral way, in real-world scenarios it is often the case that (static) private keys are reused. This can incentivize malicious users to create faulty public keys that allow them to learn information about the other user's static private key, and in such scenarios validating public keys becomes a mandatory practical requirement.

In traditional elliptic curve Diffie-Hellman (ECDH), validating public keys essentially amounts to checking that points are on the correct and cryptographically secure curve [8]. Such *point validation* is considered trivial in ECDH, since checking that a point satisfies a curve equation requires only a handful of field multiplications and additions, and this is negligible compared to the overall cost (e.g., of a scalar multiplication).

In contexts where SIDH private keys are reused, public key validation is equally as important but is no longer as trivial. In April 2015, a group from the NSA [28] pointed out that "direct public key validation is not always possible for [...] isogeny based schemes" before describing more complicated options that validate public keys *indirectly*. In this section we describe ways to directly validate various properties of our public keys that, in particular, work entirely in our compact framework, i.e., without the need of y-coordinates or of the Montgomery b coefficient that fixes the quadratic twist.

Recall from Section 6 that an honest user generates public keys of the form

		SIDH	SIDH+ECDH
≈ bit-security (hard problem)	classical	192 (SSDDH)	384 (ECDHP)
	PQ	128 (SSDDH)	128 (SSDDH)
public key size		564	658
speed (cc $\times 10^6$)	Alice's keygen	46	52
	Bob's keygen	52	58
	Alice's shared key	44	50
	Bob's shared key	50	57

In Table 3 we compare hybrid SIDH+ECDH versus standalone SIDH. The take-away message is that for a less than 1.17x increase in public key sizes and less than 1.13x increase in the overall computing cost, we can increase the classical security of the key exchange from 192 bits (based on the relatively new SSDDH problem) to 384 bits (based on the long-standing ECDLP).

9 Validating public keys

Recall from Section 2 that De Feo, Jao and Plût [18] prove that SIDH is *session-key secure* (under SSDDH) in the authenticated-links adversarial model [12]. This model assumes perfectly authenticated links which effectively forces adversaries to be passive eavesdroppers; in particular, it assumes that public keys are correctly generated by honest users. While this model can be suitable for key exchange protocols that are instantiated in a truly ephemeral way, in real-world scenarios it is often the case that (static) private keys are reused. This can incentivize malicious users to create faulty public keys that allow them to learn information about the other user's static private key, and in such scenarios validating public keys becomes a mandatory practical requirement.

In traditional elliptic curve Diffie-Hellman (ECDH), validating public keys essentially amounts to checking that points are on the correct and cryptographically secure curve [8]. Such *point validation* is considered trivial in ECDH, since checking that a point satisfies a curve equation requires only a handful of field multiplications and additions, and this is negligible compared to the overall cost (e.g., of a scalar multiplication).

In contexts where SIDH private keys are reused, public key validation is equally as important but is no longer as trivial. In April 2015, a group from the NSA [28] pointed out that "direct public key validation is not always possible for [...] isogeny based schemes" before describing more complicated options that validate public keys *indirectly*. In this section we describe ways to directly validate various properties of our public keys that, in particular, work entirely in our compact framework, i.e., without the need of y-coordinates or of the Montgomery b coefficient that fixes the quadratic twist.

Recall from Section 6 that an honest user generates public keys of the form

ON THE SECURITY OF SUPERSINGULAR ISOGENY CRYPTOSYSTEMS

STEVEN D. GALBRAITH, CHRISTOPHE PETIT, BARAK SHANI, AND YAN BO TI

ABSTRACT. We study cryptosystems based on supersingular isogenies. This is an active area of research in post-quantum cryptography. Our first contribution is to give a very powerful active attack on the supersingular isogeny encryption scheme. This attack can only be prevented by using a (relatively expensive) countermeasure. Our second contribution is to show that the security of all schemes of this type depends on the difficulty of computing the endomorphism ring of a supersingular elliptic curve. This result gives significant insight into the difficulty of the isogeny problem that underlies the security of these schemes. Our third contribution is to give a reduction that uses partial knowledge of shared keys to determine an entire shared key. This can be used to retrieve the secret key, given information leaked from a side-channel attack on the key exchange protocol. A corollary of this work is the first bit security result for the supersingular isogeny key exchange: Computing any component of the j-invariant is as hard as computing the whole j-invariant.

Our paper therefore provides an improved understanding of the security of these cryptosystems. We stress that our work does not imply that these systems are insecure, or that they should not be used. However, it highlights that implementations of these schemes will need to take account of the risks associated with various active and side-channel attacks.

This is the full version of the paper: Steven D. Galbraith, Christophe Petit, Barak Shani and Yan Bo Ti, On the Security of Supersingular Isogeny Cryptosystems, in J. H. Cheon and T. Takagi (eds.), Proceedings of ASIACRYPT 2016, Part I, Springer Lecture Notes in Computer Science 10031 (2016) 63–91.

Keywords: Isogenies, supersingular elliptic curves.

1. INTRODUCTION

In 2011, Jao and De Feo [JF11] introduced the supersingular isogeny Diffie–Hellman key exchange protocol as a candidate for a post-quantum key exchange. The security of this scheme is based on so-called supersingular isogeny problems. Isogeny cryptosystems were first proposed by Couveignes [Cou06] and further developed in [RS06, Sto10]. The supersingular case was first developed in a hash function construction by Charles–Lauter–Goren [CLG09]. Subsequently to the Jao and De Feo construction they have been used to build other cryptographic functions such as public-key encryption, undeniable signatures and designated verifier signatures [FJP14, JS14, XTW12]. As with clas-

Given these roots, one can recover the j-invariant for a curve E_d in this path. Using the modular polynomials, we can "travel back" to find the j-invariant of the root E_s. Indeed, suppose our path is $E_0 = E_s, E_1, \ldots, E_k$. Then as we know $j(E_d)$ for some $d \leq k$, we can use Φ_2 to compute $j(E_{d-1})$ by solving $\Phi_2(j(E_d), y) = 0$. We get at most 3 candidates for $j(E_{d-1})$, and we proceed recursively to find candidates for $j(E_{d-2}), \ldots, j(E_0)$. Since the distance from E_d to the root E_s is short, this results in a small list of candidates for $j(E_s)$.

We remark that in practice the polynomials G_1, G_2 consist of many monomials, and therefore this approach would require knowledge of many bits. However, Coppersmith's method shows how to generate more relations, which help to reduce the number of bits, and as an attack one can also rely on lattice algorithms working better in practice than theoretically guaranteed.

5.2. **Active Attack When Alice Uses a Static Key.** We assume that Alice uses a static key for encryption or key exchange. A legitimate key exchange protocol takes place between Alice and Bob, and an adversary Eve who sees the protocol messages wishes to obtain the resulting shared j-invariant j_{AB}. Hence Eve knows (E, E_A, E_B) and the corresponding points.

We further assume that Eve can (adaptively) engage in protocol sessions with Alice (who always uses the same static secret key) and that, through some side-channel or other means, Eve is able to obtain partial information on the shared key computed by Alice on each protocol session.

Here, Alice acts as the oracle O that provides the partial information. Eve first observes a protocol exchange between Alice and Bob, and so sees $(E_B, \phi_B(P_A), \phi_B(Q_A))$. She learns some partial information on $j(E_{AB})$.

Eve then chooses a small integer r coprime to Alice's prime ℓ, and as described above computes an isogeny ϕ_C, the curve E_C and the corresponding points $\phi_C(P_A)$, $\phi_C(Q_A)$. She sends $(E_C, \phi_C(P_A), \phi_C(Q_A))$ to Alice as part of a key exchange session. Alice then computes $E_{AC} = E_C/\phi_C(G_A)$ and some partial information about this j-invariant $j(E_{AC})$ is leaked. This leads to the scenario described in the isogeny hidden number problem, and using one of the solutions to this problem yields the desired j-invariant $j(E_{AB})$.

Note that this attack can be detected by the countermeasure of Kirkwood et al. [KLM+15], since the query on E_C is not on a correct execution of the protocol. However, the protocol still requires Alice to compute E_{AC} and so in the context of a side-channel attack, an attacker might already have received enough information to determine the desired secret key $j(E_{AB})$.

6. Conclusion

We have given several results on the security of cryptosystems based on the Jao–De Feo concept. Our main conclusion is that it seems very hard to prevent all active attacks using simple methods. Our first active attack seems to be undetectable using pairings or any other tools, as the curves and points appear to be indistinguishable from correct

Given these roots, one can recover the j-invariant for a curve E_d in this path. Using the modular polynomials, we can "travel back" to find the j-invariant of the root E_s. Indeed, suppose our path is $E_0 = E_s, E_1, \dots, E_k$. Then as we know $j(E_d)$ for some $d \leq k$, we can use Φ_2 to compute $j(E_{d-1})$ by solving $\Phi_2(j(E_d), y) = 0$. We get at most 3 candidates for $j(E_{d-1})$, and we proceed recursively to find candidates for $j(E_{d-2}), \dots, j(E_0)$. Since the distance from E_d to the root E_s is short, this results in a small list of candidates for $j(E_s)$.

We remark that in practice the polynomials G_1, G_2 consist of many monomials, and therefore this approach would require knowledge of many bits. However, Coppersmith's method shows how to generate more relations, which help to reduce the number of bits, and as an attack one can also rely on lattice algorithms working better in practice than theoretically guaranteed.

5.2. **Active Attack When Alice Uses a Static Key.** We assume that Alice uses a static key for encryption or key exchange. A legitimate key exchange protocol takes place between Alice and Bob, and an adversary Eve who sees the protocol messages wishes to obtain the resulting shared j-invariant j_{AB}. Hence Eve knows (E, E_A, E_B) and the corresponding points.

We further assume that Eve can (adaptively) engage in protocol sessions with Alice (who always uses the same static secret key) and that, through some side-channel or other means, Eve is able to obtain partial information on the shared key computed by Alice on each protocol session.

Here, Alice acts as the oracle O that provides the partial information. Eve first observes a protocol exchange between Alice and Bob, and so sees $(E_B, \phi_B(P_A), \phi_B(Q_A))$. She learns some partial information on $j(E_{AB})$.

Eve then chooses a small integer r coprime to Alice's prime ℓ, and as described above computes an isogeny ϕ_C, the curve E_C and the corresponding points $\phi_C(P_A)$, $\phi_C(Q_A)$. She sends $(E_C, \phi_C(P_A), \phi_C(Q_A))$ to Alice as part of a key exchange session. Alice then computes $E_{AC} = E_C/\phi_C(G_A)$ and some partial information about this j-invariant $j(E_{AC})$ is leaked. This leads to the scenario described in the isogeny hidden number problem, and using one of the solutions to this problem yields the desired j-invariant $j(E_{AB})$.

Note that this attack can be detected by the countermeasure of Kirkwood et al. [KLM$^+$15], since the query on E_C is not on a correct execution of the protocol. However, the protocol still requires Alice to compute E_{AC} and so in the context of a side-channel attack, an attacker might already have received enough information to determine the desired secret key $j(E_{AB})$.

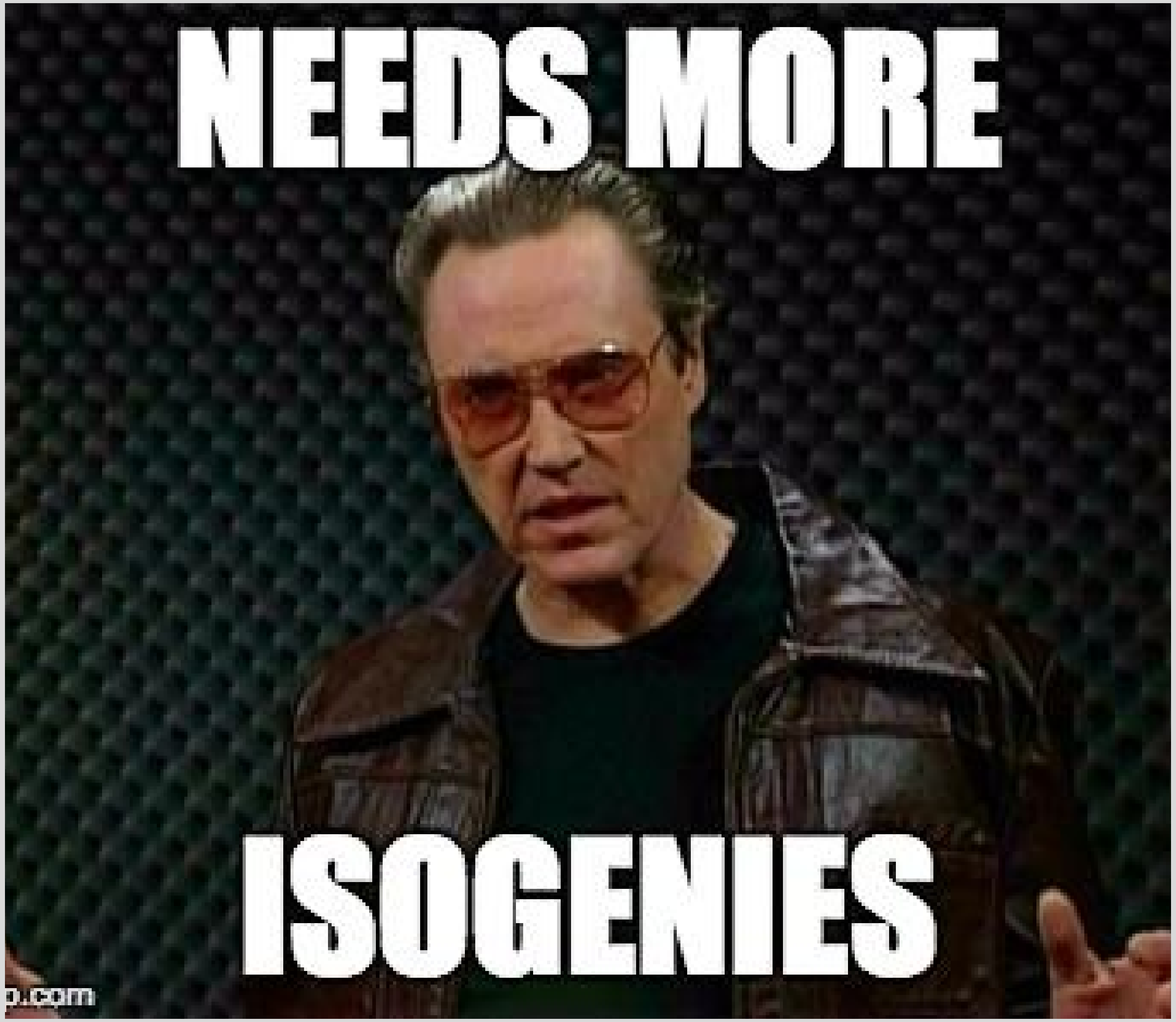
6. Conclusion

We have given several results on the security of cryptosystems based on the Jao–De Feo concept. Our main conclusion is that it seems very hard to prevent all active attacks using simple methods. Our first active attack seems to be undetectable using pairings or any other tools, as the curves and points appear to be indistinguishable from correct

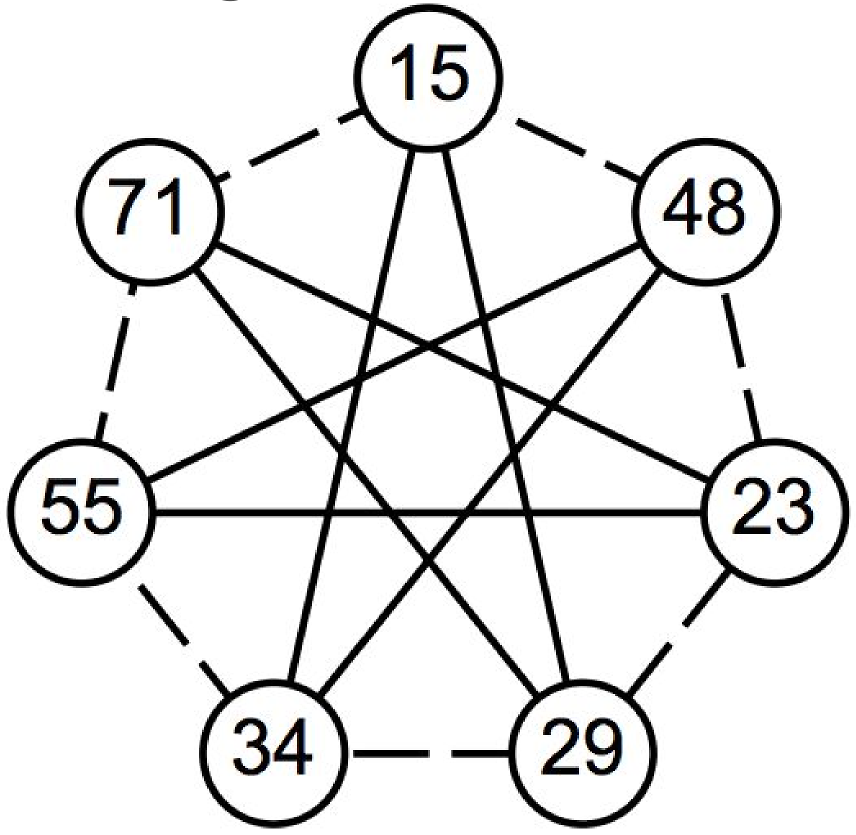
Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies
Luca De Feo and David Jao and Jérôme Plût, 2011

Efficient algorithms for supersingular isogeny Diffie-Hellman
Craig Costello and Patrick Longa and Michael Naehrig, 2016

On the Security of Supersingular Isogeny Cryptosystems
Steven D. Galbraith and Christophe Petit and Barak Shani and Yan Bo Ti, 2016


NEEDS MORE
ISOGENIES
p.com

Supersingular Isogeny Diffie-Hellman



Deirdre Connolly
@durumcrustulum