Efficient algorithms for supersingular isogeny Diffie-Hellman

Craig Costello

Joint work with Patrick Longa and Michael Naehrig

Microsoft® Research

Diffie-Hellman key exchange (circa 1976)

$q = 160693804425899027554196209234116260252220299378279283530130 1$

$g = 123456789$

$g^a \bmod q = 784673745294226535797545963198527025754996929800857779485 93$

$560048104293218128667441021342483133802626271394299410128798 = g^b \bmod q$

$a =$
685408003627063
761059275919665
781694368639459
527871881531452

$b =$
362059131912941
987637880257325
269696682836735
524942246807440

$g^{ab} \bmod q =$ 437452857085801785219961443000845969831329749878767465041215

Diffie-Hellman key exchange (circa 2016)

$q =$

580960599536995806285950253330457437068697517636289523666148615228720373099711022573733604453311840725132615775498051744399052959454004712166288567218703240103211163970644049884404985098905162720024476580704181239472968054002410482797658436938152229236120877904476989274322575173807697956881130957912551133309324351955378481630638158016186020024749256844815024251530444957718760413642873858099017255157393414625583036640591500086964373205321856683254529110790372283163413859958640669032595972518744716905954080501231020963901175074876001709536073423494575741627299485601330861695852995830467763701918159408852834506128586389827176345729488354663887955431161544644633019925438234001629205709075117553388816191898729559153153669870129226768546551743791579082315484463478026010289171803249539607504189948551381112697730747896907485704371071615012131592202455675924123901315291971095646840637944291494161435710791446256732969364 9

$g = 123456789$

$g^a \pmod q =$

197496648183227193286262018614250555971909799762533760654008147994875775445667054218578105133138217497206890599554928429450667899476
854668595594034093493637562451078938296960313488696178848142491351687253054602202966247046105770771577248321682117174246128321195678
537631520278649403464797353691996736993577092687178385602298873558954121056430522899619761453727082217823475746223803790014235051396
799049446508224661850168149957401474638456716624401906701394472447015052569417746372185093302535739383791980070572381421729029651639
304234361268764971707763484300668923972868709121665568669830978657804740157916611563508569886847487772676671207386096152947607114559
706340209059103703018182635521898738094546294558035569752596676346614699327742088471255741184755866117812209895514952436160199336532
605242210147489825669666012419572610049572551002200293281421876806011231076345540456724876139639963334490185787211920851855080379172 4

$g^a \pmod q =$

411604662069593306683228525653441872410777999220572079993574397237156368762038378332742471939666544968793817819321495269833613169937
986164811320795616949957400518206385310292475529284550626247132930124027703140131220968771142788394846592816111078275196955258045178
705254016469773509936925361994895894163065551105161929613139219782198757542984826465893457768888915561514505048091856159412977576049
073563225572809880970058396501719665853110101308432647427786565525121328772587167842037624190143909787938665842005691911997396726455
110758448552553744288464337906540312125397571803103278271979007681841394534114315726120595749993896347981789310754194864577435905673
172970033596584445206671223874399576560291954856168126236657381519414592940237018351232440467191228145585909045861278091800166330876
407323844719948807012687304886027922176162928196104625521958432771481724862624396241361307595677001801738572499949511777914941688218 8

$=$
$g^b \pmod q$

$a =$

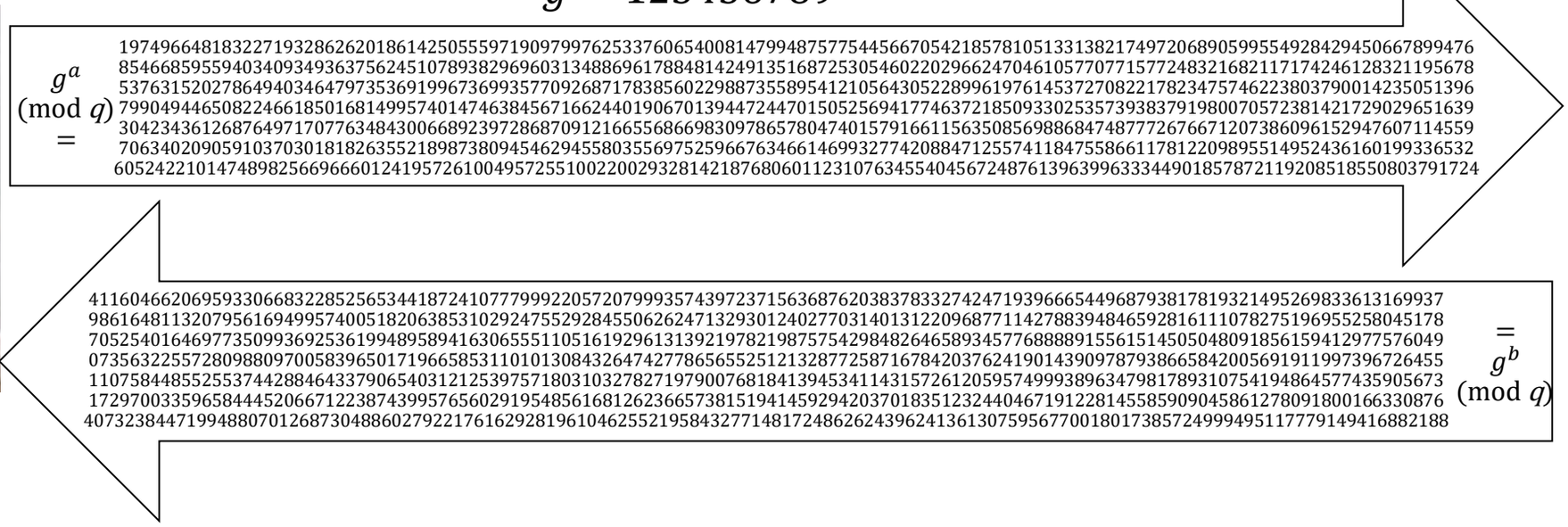
7147687166405; 9571879053605547396582
692405186145916522354912615715297097
100679170037904924330116019497881089
087696131592831386326210951294944584
400497488929803858493191812844757232 1
023987160439062006177648318875457556
233770853912505292364631833219121732 1
464134655845254917228378772756695589
845219962202945089226966507426526912
780244641640 0\902592710400433895826 1
141986237587898819361218794559180286 4
062679\864839578139273043684955597764
130097212218249158109645793763545 56\6
554629883777859568089157882151127357
422042264637917059991767756 7\3042069 8
422392949481690677789617492307207129 7
603455802621072109220\54662739697748
553543758990879608882627763290293452
560094576029847\3913613887675543866
224792652999780598864724145304621 94
52761811989\9746477252908878060493
179541951463829228890455778045929 43
730526 54\1048518026400207941519398 3
851143425084273119820368274789460 58
7100\3049774770692442789896899105 72
120963577252034804024499138445834 48

$g^{ab} =$

330166919524192149323761733598426244691224199958894654036331526394350099088627302979833339501183059198113987880066739
419999231378970715307039317876258453876701124543849520979430233302777503265010724513551209279573183234934359636696506
968325769489511028943698821518689496597758218540767517885836464160289471651364552490713961456608536013301649753975875
610659655755567474438180357958360226708742348175045563437075840969230826767034061119437657466993989389348289599600338
950372251336932673571743428823026014699232071116171392219599691096846714133643382745709376112500514300983651201961186
613464267685926563624589817259637248558104903657371981684417053993082671827345252841433337325420088380059232089174946
086536664984836041334031650438692639106287627157575758383128971053401037407031731509582807639509448704617983930135028
759658938329275199307916131883904312132911893000994819789990758698610895359142027942687477942356022103846 8

$b =$

655456209464694; 93360682685816031704
969423104727624468251177438749706128
879957701\936988268597627904791130 62
308975863428283798589097017957365590
672\835713863895712246676094993008 98
554802446403039544300748002507962036
386619315229886063541005322448463915
897986412102737725583739 65\486539312
854838650709031919742048649235894391
903529930326769610 05\088404319792729
916038927477470940948581926791161465
028635214849 87\086232861934222391717
121545686125300672760188085915004248
49476686\706784051068715397706852664
532638332403983747338379697022624261
377163163204493828299206039808703403
575100467337085017748387148822224875
309641791879395483731754620034884930
540399950519191679471224\05558557093
219350747155777569598163700850920394
705281936392411084\43600686183528465
724969562186437214972625833222544865
996160464558\546299370165894704252 64
445624157899586972652935647856967092
689604\427965012098770368450012467 92
761563917639959736383038665362727158



ECDH key exchange (1999 – nowish)

$$p = 2^{256} - 2^{224} + 2^{192} + 2^{96} - 1$$

$p = 115792089210356248762697446949407573530086143415290314195533631308867097853951$

$$E/\mathbf{F}_p: y^2 = x^3 - 3x + b$$

$\#E = 115792089210356248762697446949407573529996955224135760342422259061068512044369$

$P = (48439561293906451759052585252797914202762949526041747995844080717082404635286,$
$36134250956749795798585127919587881956611106672985015071877198253568414405109)$

$a =$
89130644591246033577639
77064146285502314502849
28352556031837219223173
24614395

[a]$P = (84116208261315898167593067868200525612344221886333785331584793435449501658416,$
$102885655542185598026739250172885300109680266058548048621945393128043427650740)$

[b]$P = (101228882920057626679704131545407930245895491542090988999577542687271695288383,$
$77887418190304022994116595034556257760807185615679689372138134363978498341594)$

[ab]$P = (101228882920057626679704131545407930245895491542090988999577542687271695288383,$
$77887418190304022994116595034556257760807185615679689372138134363978498341594)$

$b =$
10095557463932786418806
93831619070803277191091
90584053916797810821934
05190826

Quantum computers ↔ Cryptopocalypse

- Quantum computers break elliptic curves, finite fields, factoring, everything currently used for PKC

- Aug 2015: NSA announces plans to transition to quantum-resistant algorithms

NIST
National Institute of Standards and Technology

- Feb 2016: NIST calls for quantum-secure submissions

Post-quantum key exchange

What hard problem(s) do we use now???

This talk: isogenies

Popular post-quantum public-key primitives

Lattice-based

Code-based

Hash-based

Multivariate-based

Isogeny-based?

- Current confidence is smaller ☹, but so are current key sizes ☺
- Will curves offer the same advantage in PQ world as they did in the classical world?

SIDH: history

- 2006 (OIDH): Rostovsev and Stolbunov propose isogeny-based Diffie-Hellman, based on ordinary (i.e., non-supersingular) elliptic curves
- 2010 (OIDH quantum break): Childs-Jao-Soukharev give quantum algorithm that computes isogenies between ordinary curves in subexponential time
- 2011 (FIX: SIDH): Jao and De Feo fix by choosing supersingular curves. Crucial difference: endomorphism ring is not commutative (resists above attack)

SIDH: security

- **Setting:** supersingular elliptic curves $E/\mathbb{F}_{p^2}$ where p is a large prime
- **Hard problem**: Given $P, Q \in E$ and $\phi(P), \phi(Q) \in \phi(E)$, compute ϕ (where ϕ has fixed, smooth, public degree)
- **Best (known) attacks:** classical $O(p^{1/4})$ and quantum $O(p^{1/6})$
- **Confidence?:** above complexities are optimal for (above generic) attack

Motivation:

Can we actually securely deploy SIDH?

Isogenies: basic facts

- **Isogeny:** rational map (non-constant) that is a group homomorphism
$$\phi : E_1 \to E_2$$

- Given finite subgroup $\mathbf{G} \in E_1$, there is a unique isogeny $\phi : E_1 \to E_2$ (up to isomorphism) having kernel G

- Degree of (separable) isogeny is number of elements in kernel, same as its degree as a rational map

SIDH: analogues with ECDH

(roughly speaking)

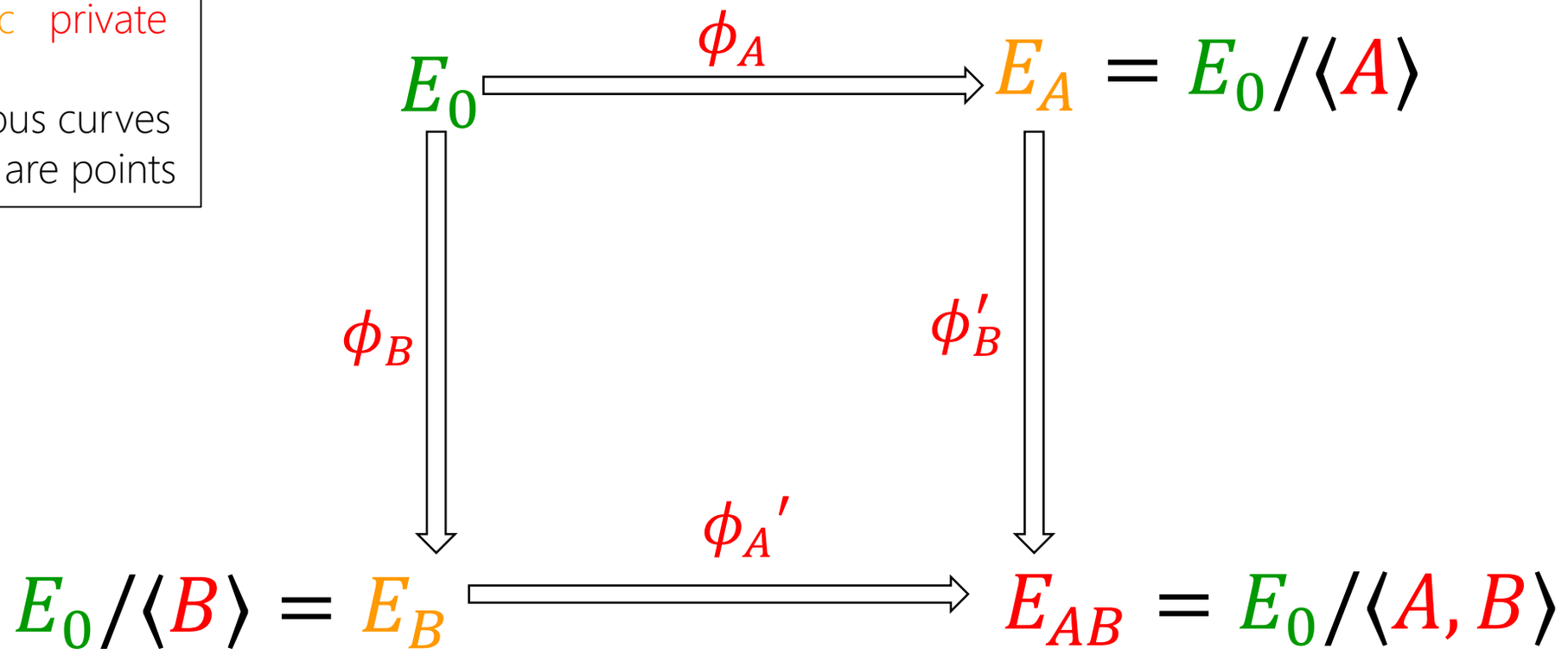
	ECDH	SIDH
elements	points P in curve group	curves E in isogeny class
secrets	scalars k	isogenies ϕ
computations	$k, P \mapsto [k]P$	$\phi, E \mapsto \phi(E)$

- Our (fixed-degree) isogenies will all have cyclic kernels, i.e., $\phi : E \to E/\langle P \rangle$
- Secrets can be thought of as $\langle P \rangle$, or equivalently P (for all P of a fixed order)

SIDH: in a nutshell

params public private

E's are isogenous curves
P's, Q's, R's, S's are points

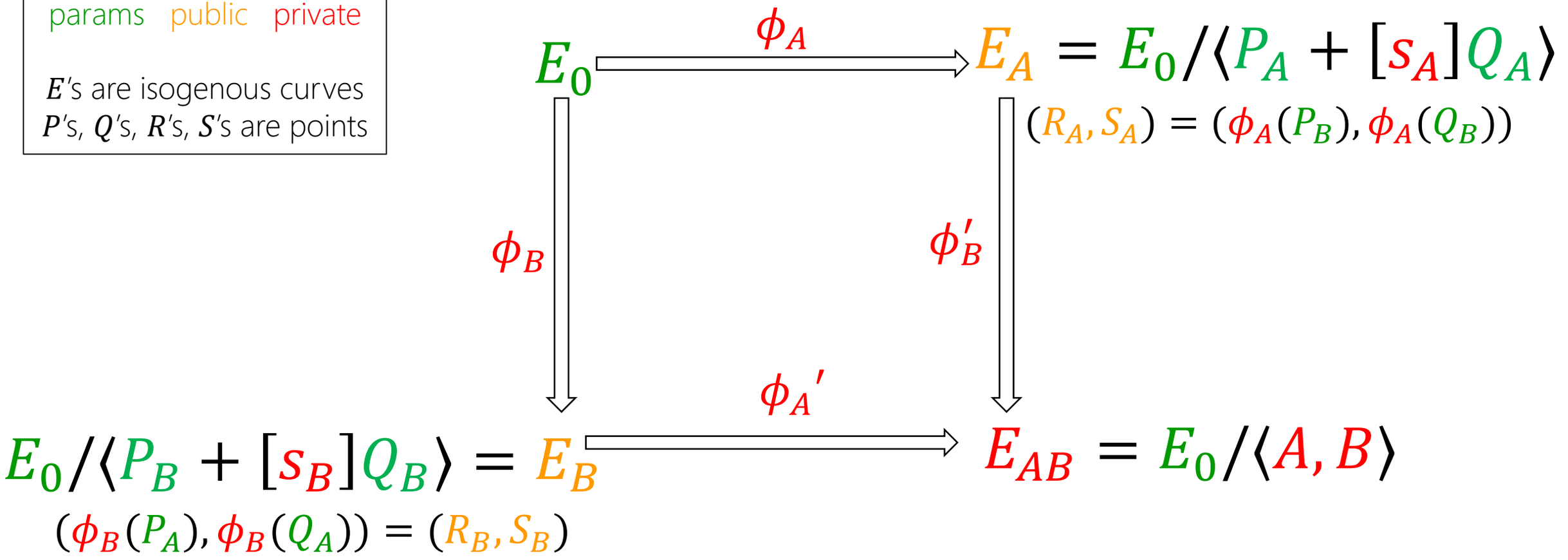


- Non-commutative, so $\phi_B\phi_A \neq \phi_A\phi_B$ (can't even multiply), hence ϕ'_A and ϕ'_B
- Alice can't just take $E_B/\langle A \rangle$, A doesn't lie on E_B

SIDH: in a nutshell

params public private

E's are isogenous curves
P's, Q's, R's, S's are points



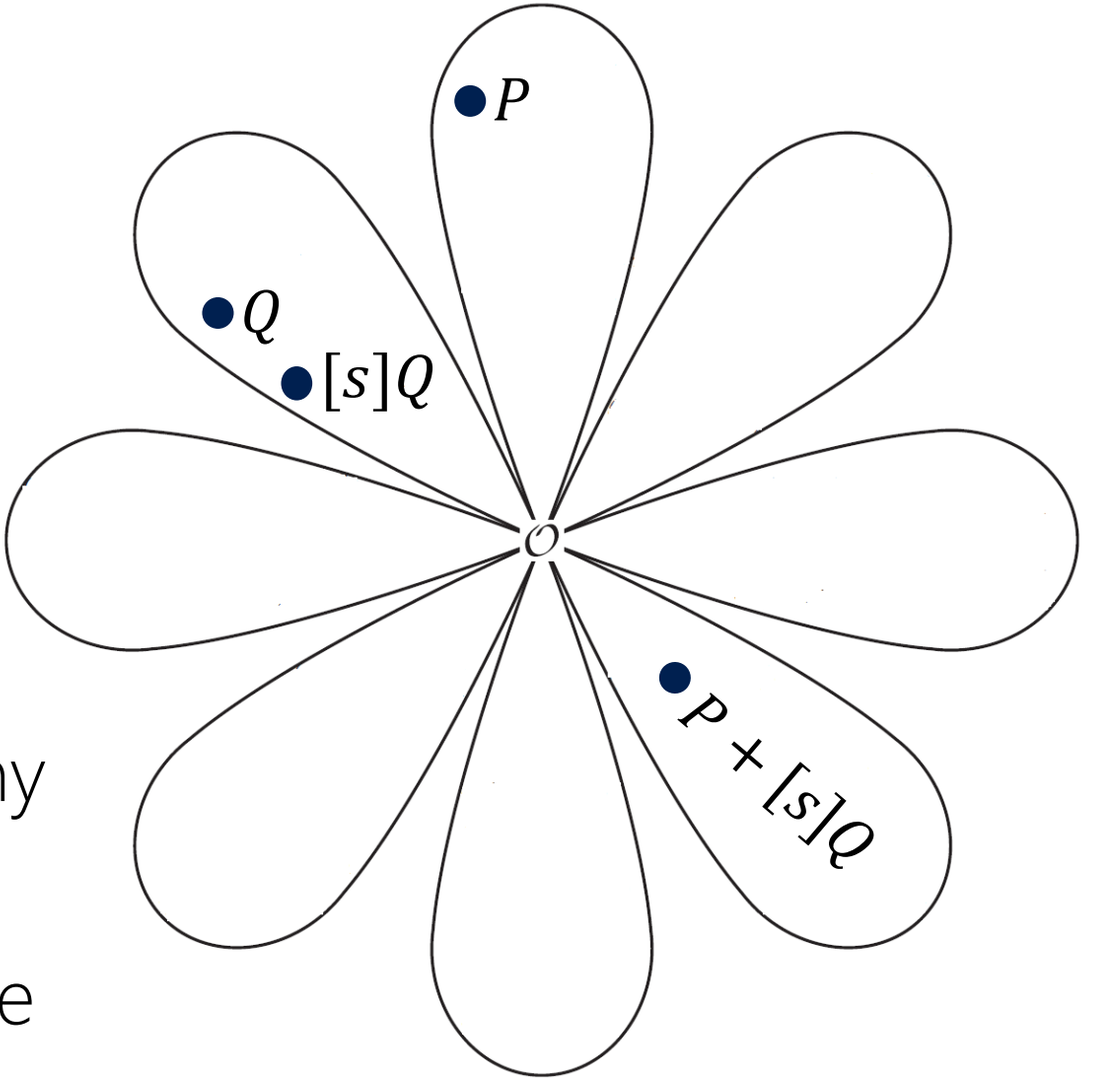
Key: Alice sends her isogeny evaluated at Bob's generators, and vice versa

$$E_A/\langle R_A + [s_B]S_A \rangle \cong E_0/\langle P_A + [s_A]Q_A\,, P_B + [s_B]Q_B \rangle \cong E_B/\langle R_B + [s_A]S_B \rangle$$

- Why $E' = E/\langle P + [s]Q \rangle$, etc?
- Why not just $E' = E/\langle [s]Q \rangle$?...
  because here E' is $\approx$ independent of s
- Need two-dimensional basis to span two-dimensional torsion
- Every different s now gives a different order n subgroup, i.e., kernel, i.e. isogeny
- Composite same thing, just uglier picture

$$E[n] \cong \mathbb{Z}_n \times \mathbb{Z}_n$$

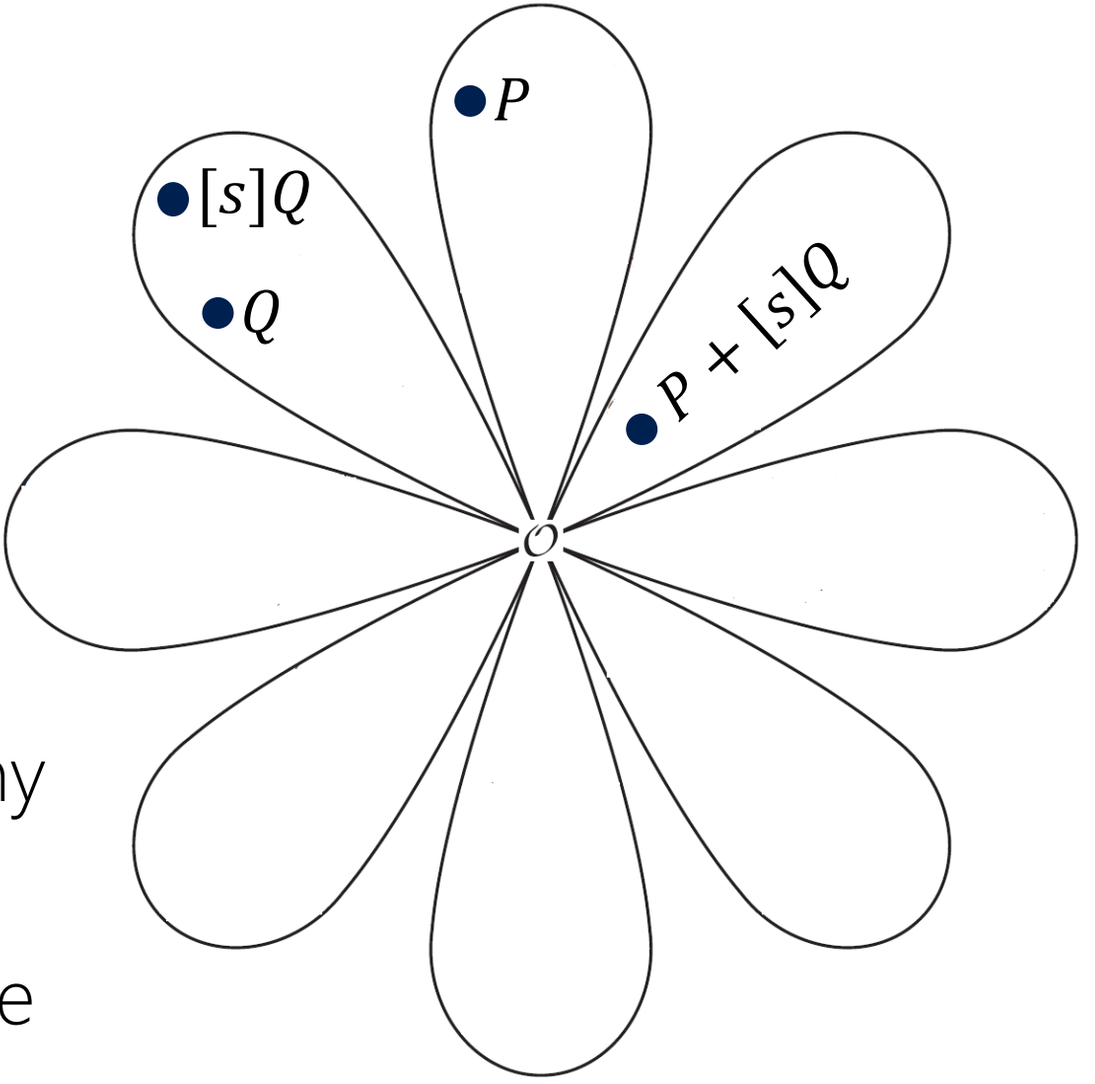
(n prime depicted below)
$n + 1$ cyclic subgroups order n



- Why $E' = E/\langle P + [s]Q \rangle$, etc?
- Why not just $E' = E/\langle [s]Q \rangle$?...
  because here E' is $\approx$ independent of s
- Need two-dimensional basis to span two-dimensional torsion
- Every different s now gives a different order n subgroup, i.e., kernel, i.e. isogeny
- Composite same thing, just uglier picture

$$E[n] \cong \mathbb{Z}_n \times \mathbb{Z}_n$$

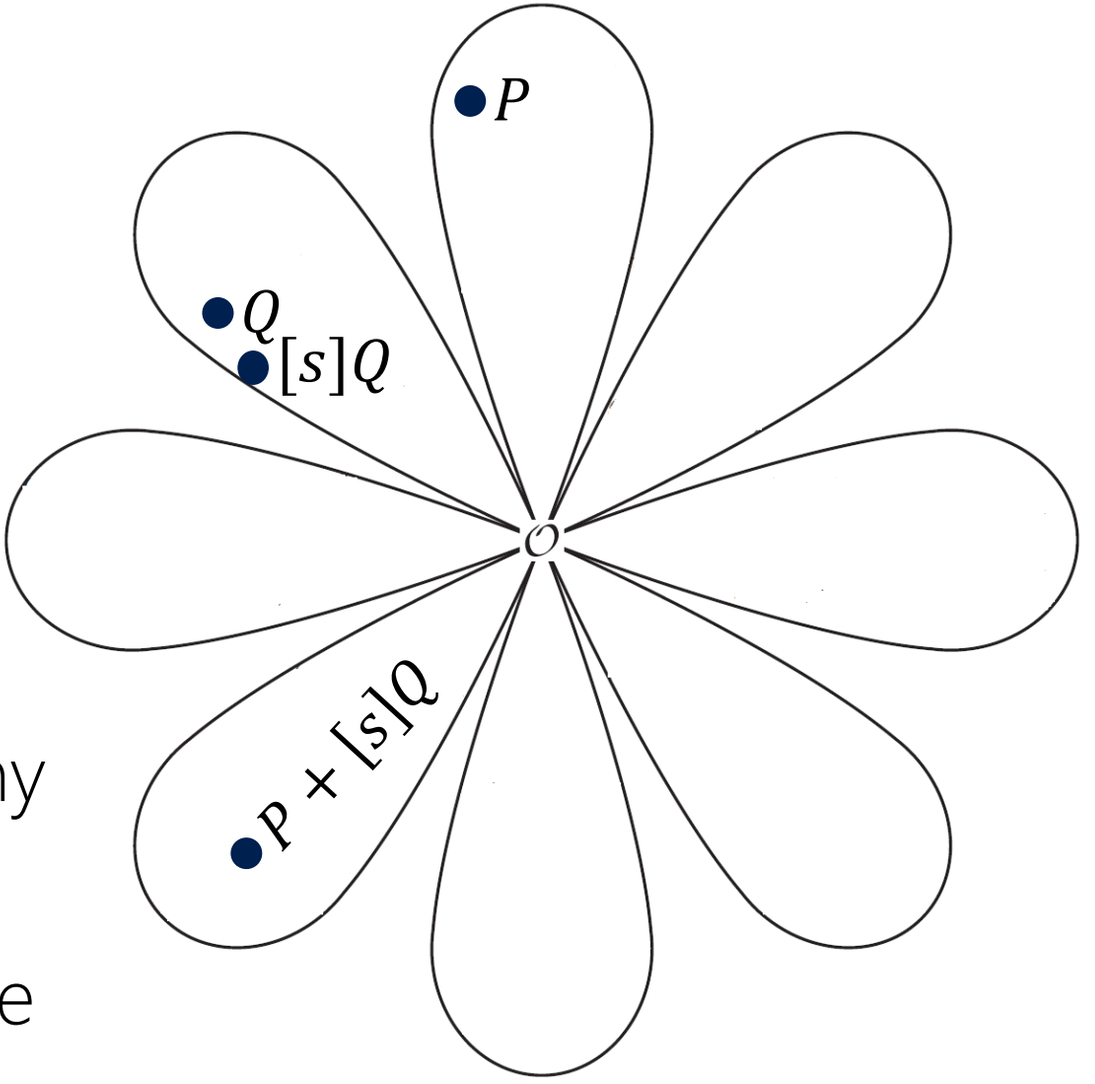
(n prime depicted below)
$n + 1$ cyclic subgroups order n



- Why $E' = E/\langle P + [s]Q \rangle$, etc?
- Why not just $E' = E/\langle [s]Q \rangle$?...
  because here E' is $\approx$ independent of s
- Need two-dimensional basis to span two-dimensional torsion
- Every different s now gives a different order n subgroup, i.e., kernel, i.e. isogeny
- Composite same thing, just uglier picture

$$E[n] \cong \mathbb{Z}_n \times \mathbb{Z}_n$$

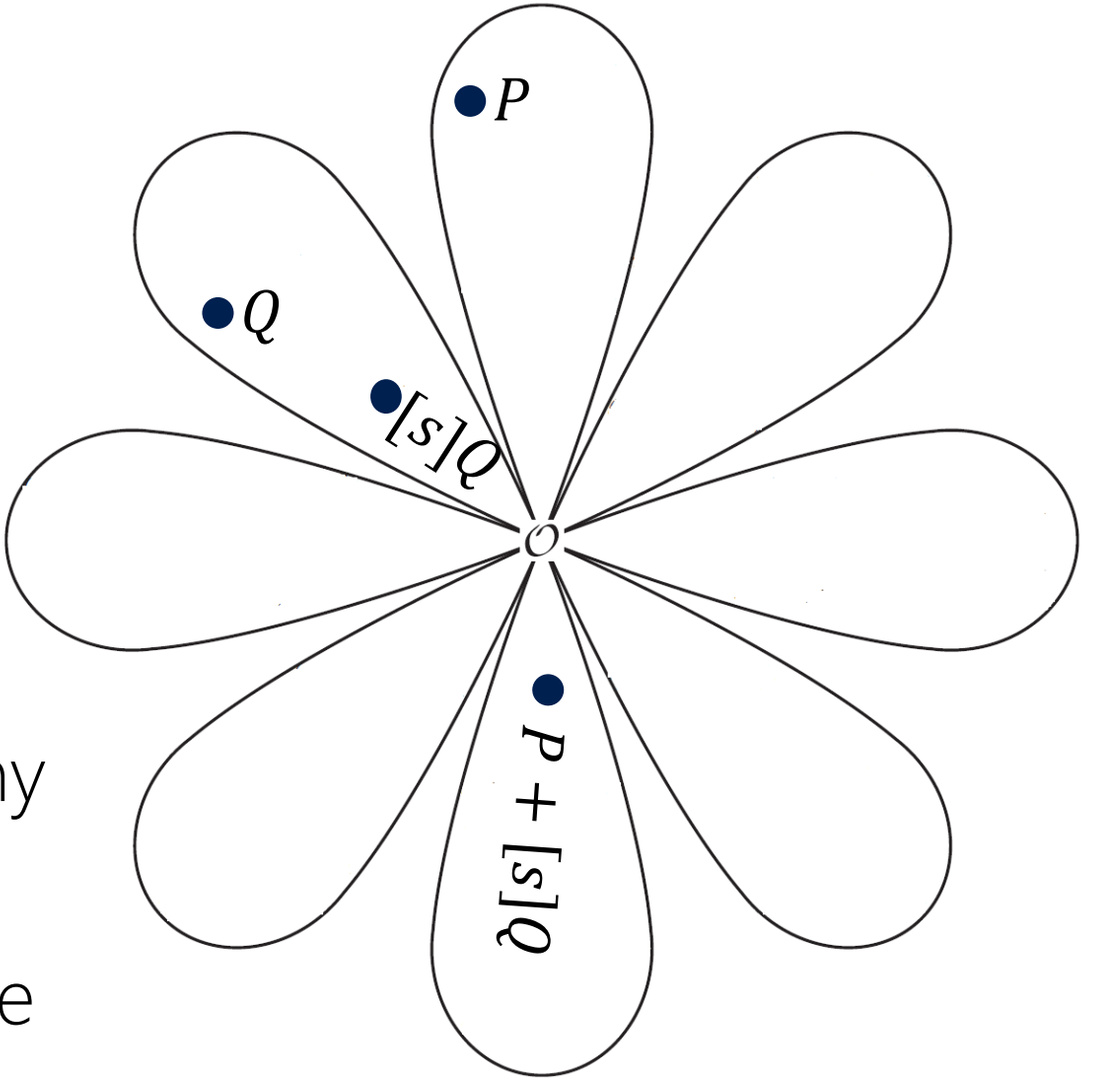
(n prime depicted below)
$n + 1$ cyclic subgroups order n



- Why $E' = E/\langle P + [s]Q \rangle$, etc?
- Why not just $E' = E/\langle [s]Q \rangle$?...
  because here E' is $\approx$ independent of s
- Need two-dimensional basis to span two-dimensional torsion
- Every different s now gives a different order n subgroup, i.e., kernel, i.e. isogeny
- Composite same thing, just uglier picture

$$E[n] \cong \mathbb{Z}_n \times \mathbb{Z}_n$$

(n prime depicted below)
$n + 1$ cyclic subgroups order n



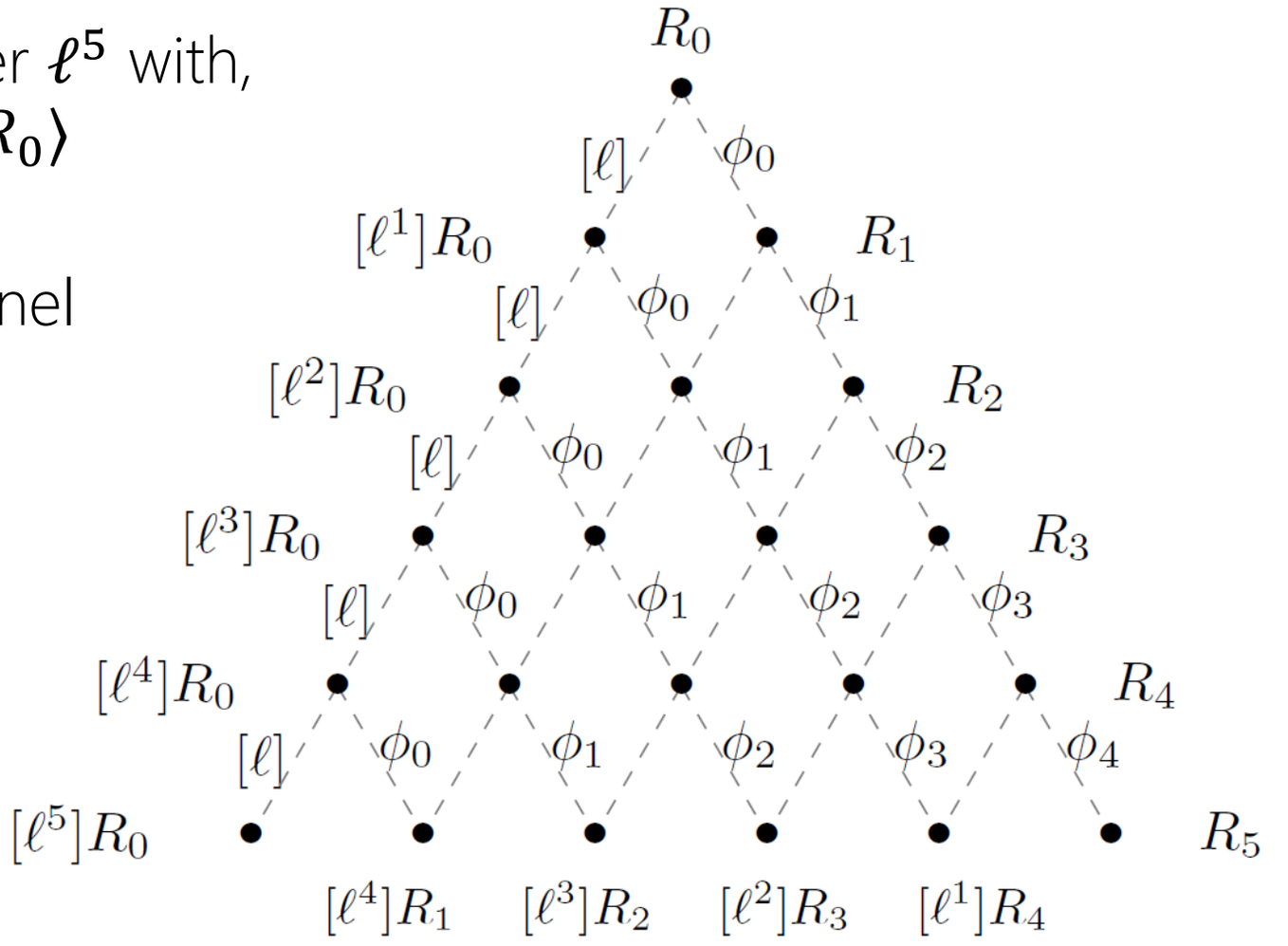
Exploiting smooth degree isogenies

- Computing isogenies of prime degree ℓ at least $O(\ell)$, e.g., Velu's formulas need the whole kernel specified
- We (obviously) need exp. set of kernels, meaning exp. sized isogenies, which we can't compute unless they're smooth
- Here (for efficiency/ease) we will only use isogenies of degree ℓ^e for $\ell \in \{2,3\}$

Exploiting smooth degree isogenies

- Suppose our secret point R_0 has order ℓ^5 with, e.g., $\ell \in \{2,3\}$, we need $\phi : E \to E/\langle R_0 \rangle$
- Could compute all ℓ^5 elements in kernel (but only because exp is 5)
- Better to factor $\phi = \phi_4\phi_3\phi_2\phi_1\phi_0$, where all ϕ_i have degree ℓ, and

$\phi_0 = E_0 \to E_0/\langle [\ell^4]R_0 \rangle$, $R_1 = \phi_0(R_0)$;
$\phi_1 = E_1 \to E_1/\langle [\ell^3]R_1 \rangle$, $R_2 = \phi_1(R_1)$;
$\phi_2 = E_2 \to E_2/\langle [\ell^2]R_2 \rangle$, $R_3 = \phi_2(R_2)$;
$\phi_3 = E_3 \to E_3/\langle [\ell^1]R_3 \rangle$, $R_4 = \phi_3(R_3)$;
$\phi_4 = E_4 \to E_4/\langle R_4 \rangle$.



(credit DJP'14 for picture, and for a much better way to traverse the tree)

Our performance improvements

1. Tighter parameter choices

2. Projective isogenies → $\mathbb{P}^1$ everywhere

3. Faster $\mathbb{F}_{p^2}$ arithmetic

(just 1 and 2 today…)

params public private

Parameters

$$p = 2^{372}3^{239} - 1$$

($p \approx 2^{768}$ gives ≈ 192 bits classical and 128 bits quantum security against best known attacks)

Weierstrass & Mont.

$$E_0 / \mathbb{F}_{p^2} : y^2 = x^3 + x$$

$$\#E_0 = (p+1)^2 = (2^{372}3^{239})^2$$

Easy ECDLP

Alice's generators are P_A and Q_A, not same subgroup, both order 2^{372}
She chooses $s_A \in \{1,2, \ldots 2^{371} - 1\}$, computes $P_A + [2s_A]Q_A$ and
sends $E_A = E_0/\langle P_A + [2s_A]Q_A \rangle$ together with $(R_A, S_A) = (\phi_A(P_B), \phi_A(Q_B))$

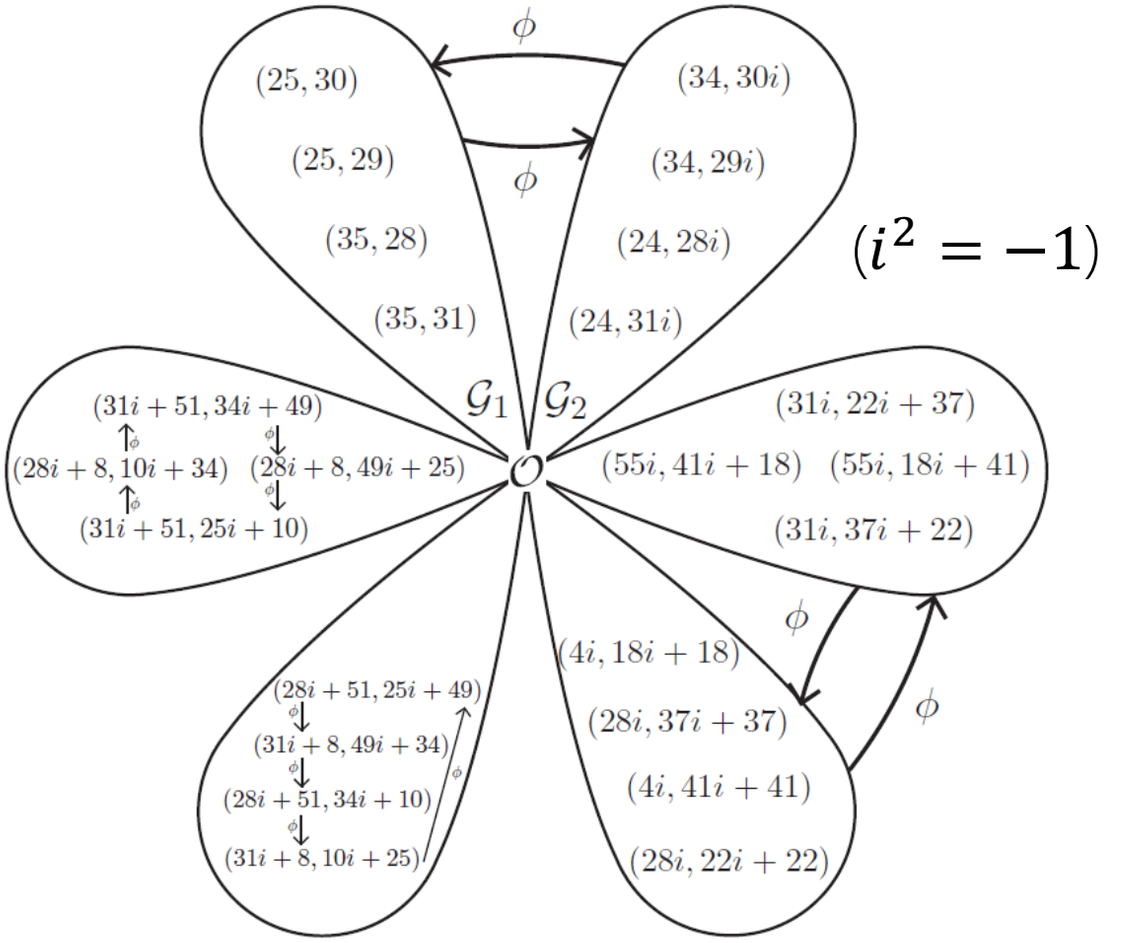
Bob's generators are P_B and Q_B, not same subgroup, both order 3^{239}
He chooses $s_B \in \{1,2, \ldots 3^{238} - 1\}$, computes $P_B + [3s_B]Q_B$ and
sends $E_B = E_0/\langle P_B + [3s_B]Q_B \rangle$ together with $(R_B, S_B) = (\phi_B(P_A), \phi_B(Q_A))$

The base field and trace-zero subgroups

toy curve $E/\mathbb{F}_{59^2} : y^2 = x^3 + x$

distortion map $\phi\colon (x, y) \mapsto (-x, iy)$

$E[5] \cong \mathbb{Z}_5 \times \mathbb{Z}_5$



$(i^2 = -1)$

our choices:

P_A and P_B from base field group $\mathcal{G}_1$

$Q_A = \phi(P_A)$ and $Q_B = \phi(P_B)$ in $\mathcal{G}_2$

benefits:

3004 bit public params (vs. 12016)

faster key-gen scalar multiplication

Montgomery/Kummer adv. is two-fold

- Montgomery showed how use x-only or $(X : Z) \in \mathbb{P}^1$ to do much faster/simpler *point* arithmetic on

$$\mathrm{E_{a,b}} : by^2 = x^3 + ax^2 + x$$

- In SIDH, point arithmetic is only (<) half the story…

- Typical isogeny takes $E_{a,b} \to E_{a',b'}$ with $a' = \frac{f_1(a,b)}{g_1(a,b)}$ and $b' = \frac{f_2(a,b)}{g_2(a,b)}$

- We show that isogeny arithmetic can mimic point arithmetic

Point *and* isogeny arithmetic in $\mathbb{P}^1$

ECDH: move around different points on a fixed curve.

SIDH: move around different points and different curves

$$E_{\mathrm{a,b}} : \quad by^2 = x^3 + ax^2 + x$$

$(x, y) \leftrightarrow (X : Y : Z)$ $\qquad$ $(a, b) \leftrightarrow (A : B : C)$

$$E_{(\mathrm{A:B:C})} : \quad BY^2Z = CX^3 + AX^2Z + CXZ^2$$

Point arithmetic: $(X : Z) \mapsto (X' : Z')$

Isogeny arithmetic: $(A : C) \mapsto (A' : C')$

what was…

$$G : \frac{B}{2-A}y^2 = x^3 - 2\frac{A+6}{2-A}x^2 + x,$$
$$\psi : F \to G,$$
$$(x,y) \mapsto \left(\frac{1}{2-A}\frac{(x+4)(x+(A+2))}{x}, \frac{y}{2-A}\left(1 - \frac{4(2+A)}{x^2}\right)\right)$$

```
void iso2_comp(iso2* iso, GF* iA, GF* iB, GF* iA24,
               const GF A, const GF B,
               const GF x, const GF z) {
  GF* tmp = x.parent->GFtmp;

  sub_GF(&tmp[0], x, z);
  sqr_GF(&tmp[1], tmp[0]);
  inv_GF(&tmp[0], tmp[1]);
  mul_GF(&tmp[1], tmp[0], z);
  mul_GF(iso, tmp[1], x); // iA2 = x z / (x-z)^2
  add_GF_ui(&tmp[0], A, 6);
  mul_GF(iB, B, *iso); // iB = B iA2
  mul_GF(iA, tmp[0], *iso); // iA = (A+6) iA2
  a24(iA24, *iA);
}
```

Division in $\mathbb{F}_p$

… is now (division-free):

$$(A' : C') = \left(2(2X_4^4 - Z_4^4) : Z_4^4\right),$$

$$(X' : Z') = \left(X\left(2X_4Z_4Z - X(X_4^2 + Z_4^2)\right)(X_4X - Z_4Z)^2 : Z\left(2X_4Z_4X - Z(X_4^2 + Z_4^2)\right)(Z_4X - X_4Z)^2\right).$$

```
void get_4_isog(point_proj_t P, f2elm_t A, f2elm_t C, f2elm_t* coeff)
{ // Computes the corresponding 4-isogeny of a projective Montgomery point (X4:Z4) of order 4.
  // Input:  projective point of order four P = (X4:Z4).
  // Output: the 4-isogenous Montgomery curve with projective coefficient A/C and the 5 coefficients
  //         that are used to evaluate the isogeny at a point in eval_4_isog().

    fp2add751(P->X, P->Z, coeff[0]);                    // coeff[0] = X4+Z4
    fp2sqr751_mont(P->X, coeff[3]);                     // coeff[3] = X4^2
    fp2sqr751_mont(P->Z, coeff[4]);                     // coeff[4] = Z4^2
    fp2sqr751_mont(coeff[0], coeff[0]);                 // coeff[0] = (X4+Z4)^2
    fp2add751(coeff[3], coeff[4], coeff[1]);            // coeff[1] = X4^2+Z4^2
    fp2sub751(coeff[3], coeff[4], coeff[2]);            // coeff[2] = X4^2-Z4^2
    fp2sqr751_mont(coeff[3], coeff[3]);                 // coeff[3] = X4^4
    fp2sqr751_mont(coeff[4], coeff[4]);                 // coeff[4] = Z4^4
    fp2add751(coeff[3], coeff[3], A);                   // A = 2*X4^4
    fp2sub751(coeff[0], coeff[1], coeff[0]);            // coeff[0] = 2*X4*Z4 = (X4+Z4)^2 - (X4^2+Z4^2)
    fp2sub751(A, coeff[4], A);                          // A = 2*X4^4-Z4^4
    fp2copy751(coeff[4], C);                            // C = Z4^4
    fp2add751(A, A, A);                                 // A = 2(2*X4^4-Z4^4)
}
```

params public private

Implementation summary (Alice)

parameters: curve $E_0/\mathbb{F}_{p^2} : y^2 = x^3 + x$ over $p = 2^{372}3^{239} - 1$
Alice generators $P_A = [3^{239}](11, \sqrt{11^3 + 11})$ and $Q_A = \tau(P_A)$
Bob generators $P_B = [2^{372}](6, \sqrt{6^3 + 6})$ and $Q_B = \tau(P_B)$

key generation: computes $R_A = P_A + [2s_A]Q_A$
computes $E_A = E_0/\langle R_A \rangle$ where $\phi_A : E_0 \to E_A$
sends public key $[E_A, x(\phi_A(P_B)), x(\phi_A(Q_B)), x(\phi_A(Q_B - P_B))] \in \left(\mathbb{F}_{p^2}\right)^4$

cost:

638 mul-by-[4]
1330 $\boldsymbol{\phi_4}$ eval

shared: input $\mathrm{PK}_{\mathrm{Bob}} = [E_B, x(\phi_B(P_A)), x(\phi_B(Q_A)), x(\phi_B(Q_A - P_A))]$
secret: computes $R_A{}' = x(\phi_B(P_A) + [2s_A]\phi_B(Q_A)))$
computes $E_{AB} = E_B/\langle R_A' \rangle$
shared secret is $j(E_{AB}) \in \mathbb{F}_{p^2}$

cost:

638 mul-by-[4]
772 $\boldsymbol{\phi_4}$ eval

Performance benchmarks

Operation	This work		Prior work [2]	
	Sandy Bridge	Haswell	Sandy Bridge	Haswell
Alice's keygen	54	51	165	149
Bob's keygen	64	59	172	152
Alice's shared key	51	47	133	118
Bob's shared key	61	57	137	122
Total	230	214	608	540

Table: millions of clock cycles for DH operations on Intel core i-7 (3.4GHz) platforms

- SIDH currently (orders of mag.) slower than other PQ stuff, but keys **751 byte**
- actually, with square root [no visible perf. diff] **compressed** keys are **563 bytes**

Non-performance stuff

1. BigMont: a strong ECDH+SIDH hybrid

2. Validating public keys

BigMont: a strong SIDH+ECDH hybrid

- No clear frontrunner for PQ key exchange (in terms of confidence and suitability), so use hybrids in the meantime…
- Hybrid particularly good idea for (relatively young) SIDH
- Hybrid particularly easy for SIDH

There are exponentially many A such that $E_A / \mathbb{F}_{p^2} : y^2 = x^3 + Ax^2 + x$ is in the supersingular isogeny class (the ones we walk around on).
These are all unsuitable for ECDH since $\#E_A = 2^i 3^j$

TRUE…BUT…

There are also exponentially many A such that $E_A / \mathbb{F}_{p^2} : y^2 = x^3 + Ax^2 + x$ is twist-secure and suitable for ECDH.

BigMont

$$p = 2^{372}3^{239} - 1$$

$$E_A / \mathbb{F}_{p^2} : y^2 = x^3 + Ax^2 + x$$

$$A = 624450$$

Then $\#E_A = 4r$ and $\#E'_A = 4r'$ where r and r' are 749-bit primes

comparison		standalone SIDH	hybrid SIDH+ECDH
≈ bit-security (hard problem)	classical	192 (SSDDH)	384 (ECDHP)
	PQ	128 (SSDDH)	128 (SSDDH)
public key size (bytes)	uncompressed	751	845
	compressed	564	658
speed (cc $\times 10^6$)	Alice's keygen	51	58
	Bob's keygen	59	66
	Alice's shared key	47	54
	Bob's shared key	57	64

Colossal amount of classical security almost-for-free (≈ no more code)

Validating public keys

- Validation not needed in truly ephemeral key exchange, but necessary in static DH
- NSA (at NIST workshop 2015) said that public key validation is often not possible for isogeny-based. Proposed “indirect” validation.
- Validation definitely non-trivial in SIDH, but we show how full validation can be achieved in our compact framework

Validating public keys

$$\mathrm{PK} = [A, x(P), x(Q), x(R)]$$

- **Need:** to assert that E_A is in supersingular isogeny class.
  How: have to show that $(p+1)$ kills a random point on E_A (Sutherland).

- **Need:** to assert that $x(R) = x(Q - P)$.
  How: easy.

- **Need:** $x(P)$ and $x(Q)$ full order and independent subgroups
  How: x-only Weil pairing computation ☹ and order check.

- **Cost**: validation between 1.17x and 1.34x key generation and shared secret operations. **Upshot:** No performance reason for static (maybe others though).

Summary: can we securely *deploy* SIDH?

(i.e., 192-bit classical security and 128-bit quantum security)

- Only known implementation had no side-channel resistance

  - we give full-fledged, constant-time, more efficient SIDH
  - speed: 60-78 million cycles per round
  - size: public keys 4507 bits (563 bytes) ≈ RSA

- NSA: “public key validation not always possible” for isogeny-based

  - we present full validation of public keys (validation is possible)

- Hedge your bets (almost-for-free):

  - use same prime to do hybrid ECDH (negligible speed/code overhead)
  - colossal 384-bit classical security for an extra 96 bytes in public key

Questions?