Supersingular isogeny based cryptography gets practical

Patrick Longa

Microsoft Research

https://www.microsoft.com/en-us/research/people/plonga/

Real World Crypto 2018

Zurich, Switzerland – Jan 10-12, 2018

Quick motivation recap

- Quantum computers break public-key cryptography currently in use: cryptosystems based on factoring and (elliptic curve) discrete logarithms

- NIST launches the post-quantum cryptography standardization project:

https://csrc.nist.gov/CSRC/media/Projects/Post-Quantum-Cryptography/documents/call-for-proposals-final-dec-2016.pdf

"The goal of this process is to select a number of acceptable candidate cryptosystems for standardization."
(This includes: digital signatures, encryption and key encapsulation).

Post-quantum candidates

Code-based	McEliece
Lattice-based	NTRU, LWE-based
Hash-based	Merkle’s hash-tree signatures
Multivariate	HFE^{v-} signature scheme
Isogeny-based	SIDH, SIKE

Post-quantum candidates: *in this talk…*

Code-based	McEliece
Lattice-based	NTRU, LWE-based
Hash-based	Merkle's hash-tree signatures
Multivariate	HFE^{v-} signature scheme
Isogeny-based	SIDH, SIKE

(A brief) Timeline of isogeny-based crypto, part I

1996 Couveignes describes first isogeny-based (key exchange) scheme.

2006 Rostovtsev and Stolbunov, and later Stolbunov (2010), propose key exchange using *ordinary* isogenies.

- These schemes are impractical, and
- Can be broken in (quantum) subexponential time (Childs, Jao and Soukharev 2010).

2010 Jao and De Feo propose key exchange using *supersingular* isogenies (SIDH).

- Much better performance.
- Best quantum and classical attack complexity is, as of today, exponential.

Supersingular Isogeny Diffie-Hellman (SIDH)

private Alice   private Bob
public   params

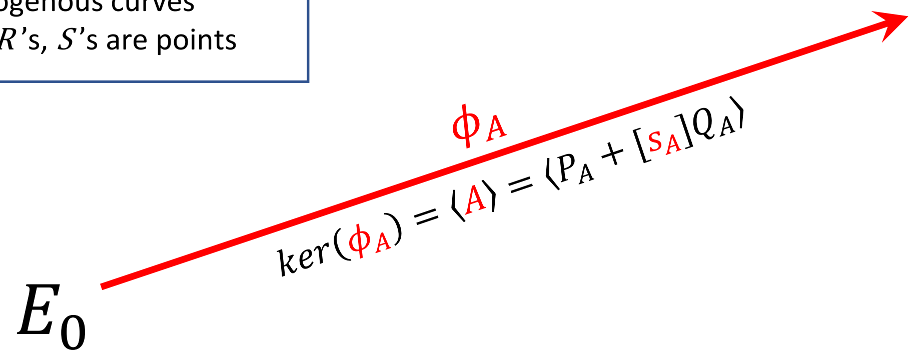
E's are isogenous curves
P's, Q's, R's, S's are points

E_0

Supersingular Isogeny Diffie-Hellman (SIDH)

private Alice | private Bob
public | params

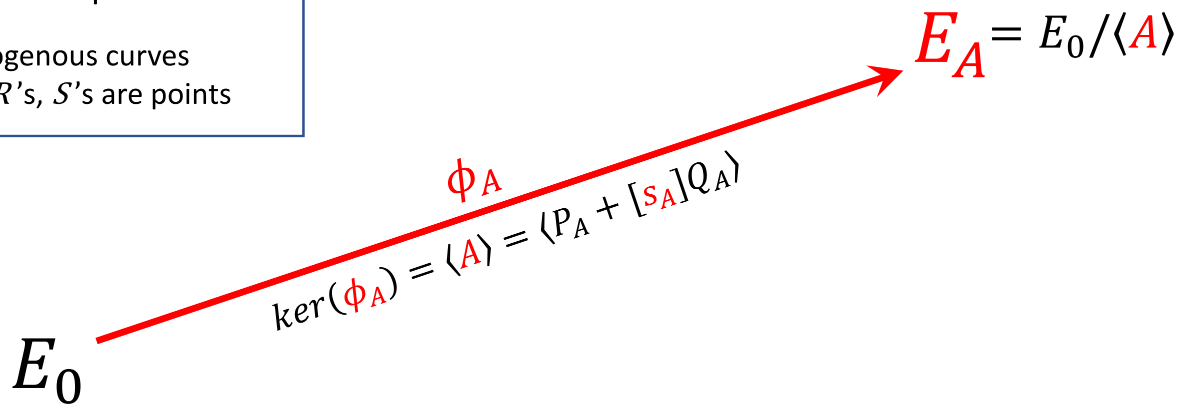
E's are isogenous curves
P's, Q's, R's, S's are points



Supersingular Isogeny Diffie-Hellman (SIDH)

private Alice   private Bob
public   params

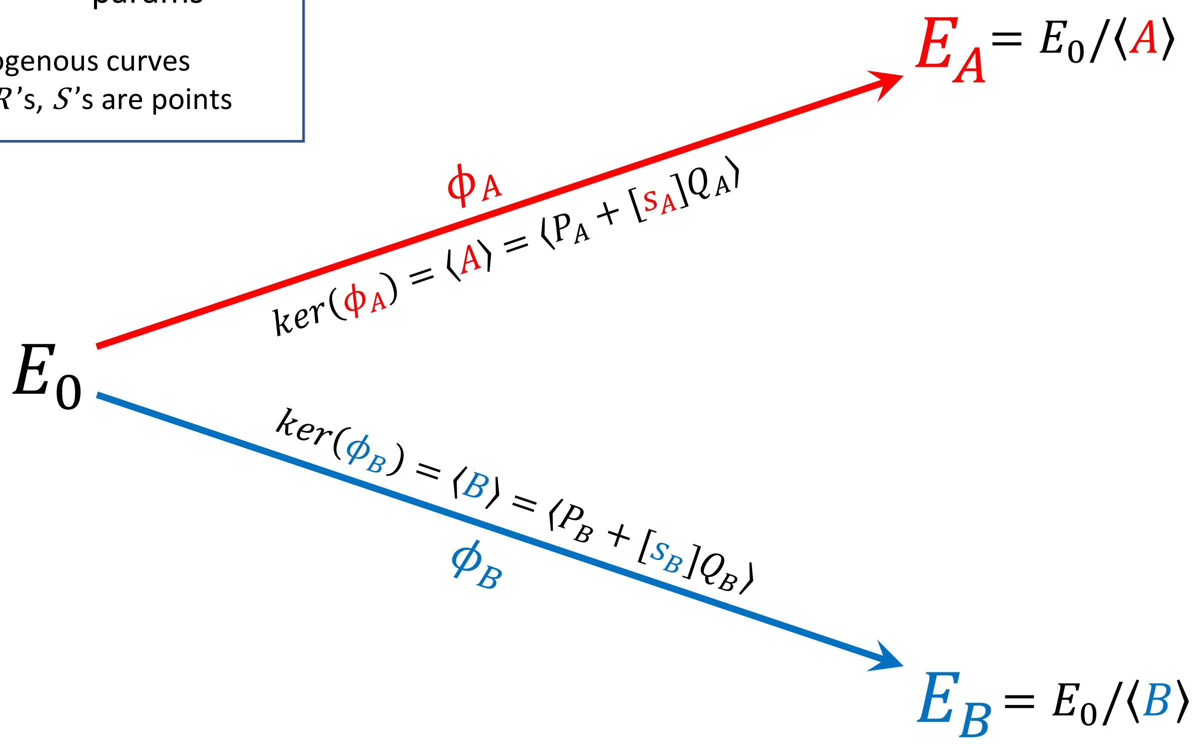
E's are isogenous curves
P's, Q's, R's, S's are points



Supersingular Isogeny Diffie-Hellman (SIDH)

private Alice private Bob
public params

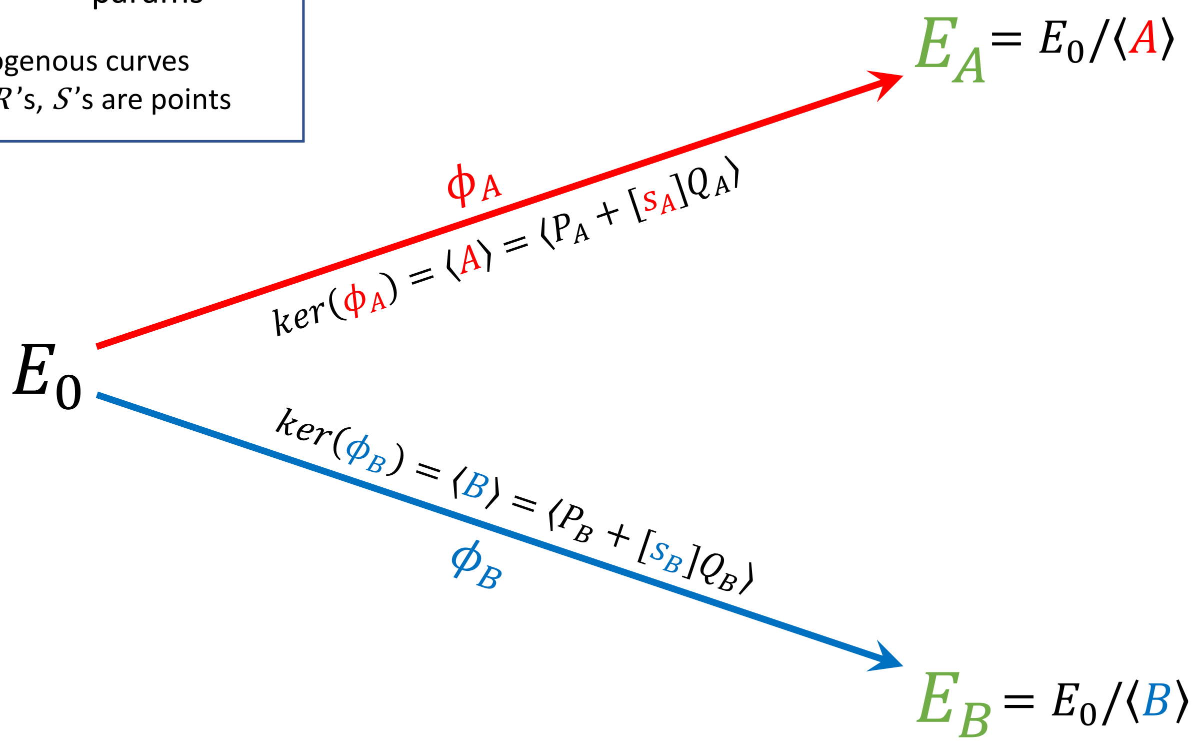
E's are isogenous curves
P's, Q's, R's, S's are points



Supersingular Isogeny Diffie-Hellman (SIDH)

private Alice   private Bob
public   params

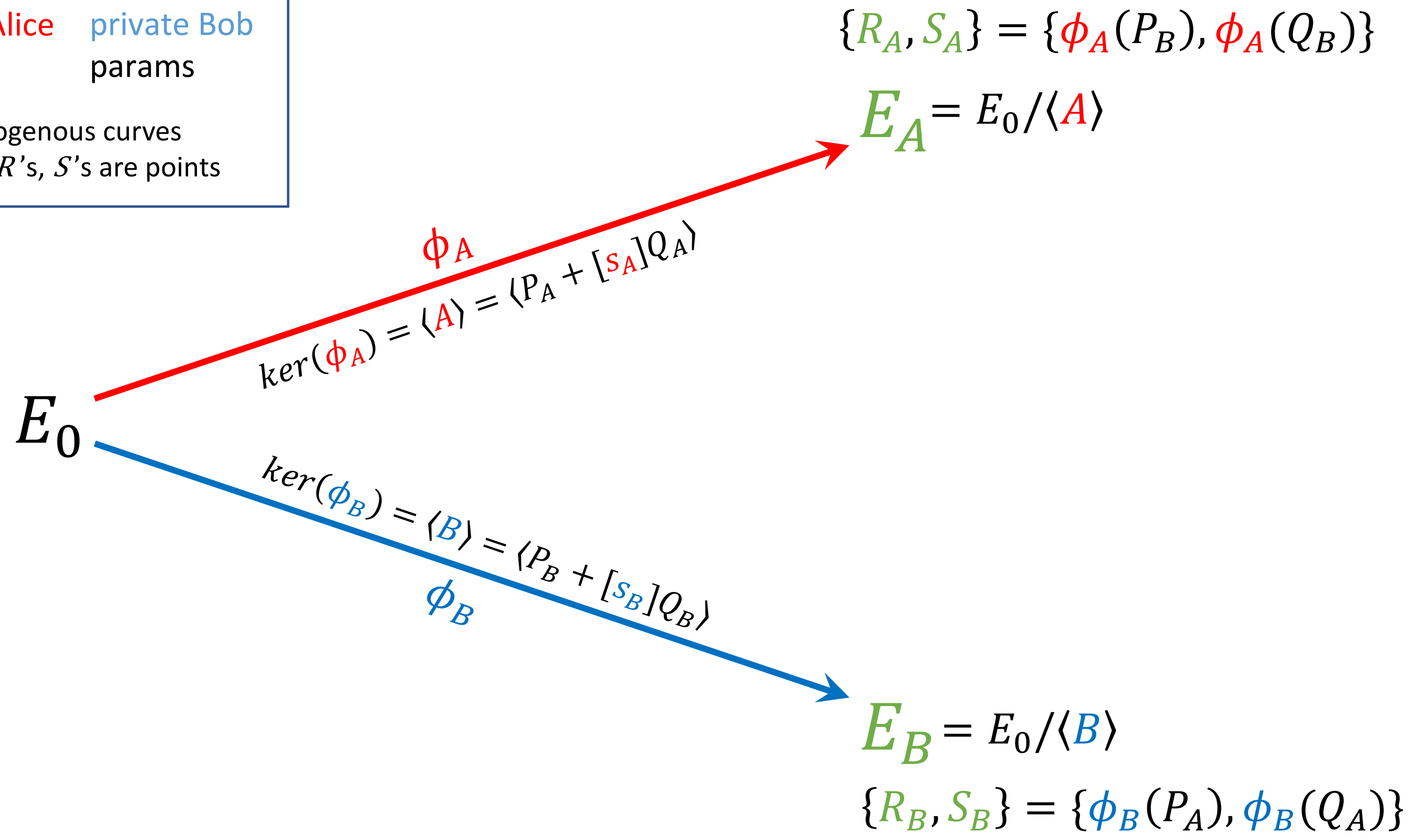
E's are isogenous curves
P's, Q's, R's, S's are points



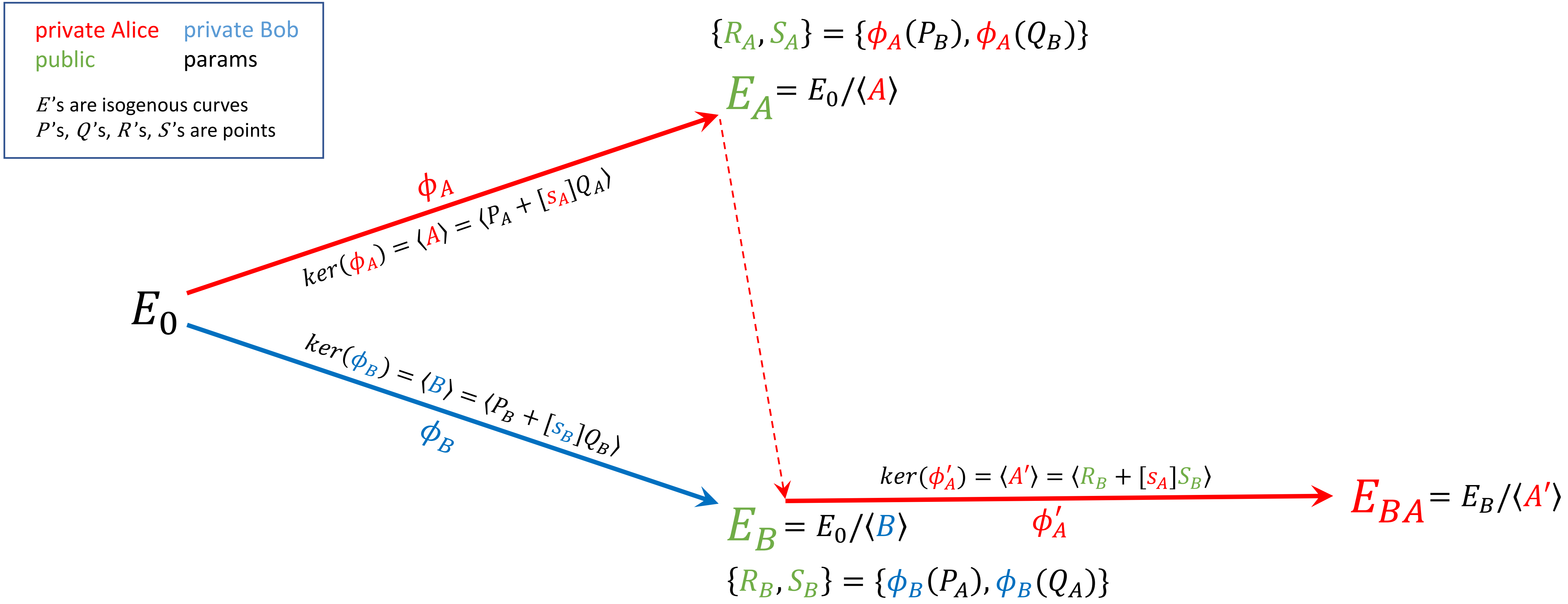
Supersingular Isogeny Diffie-Hellman (SIDH)

private Alice private Bob
public params

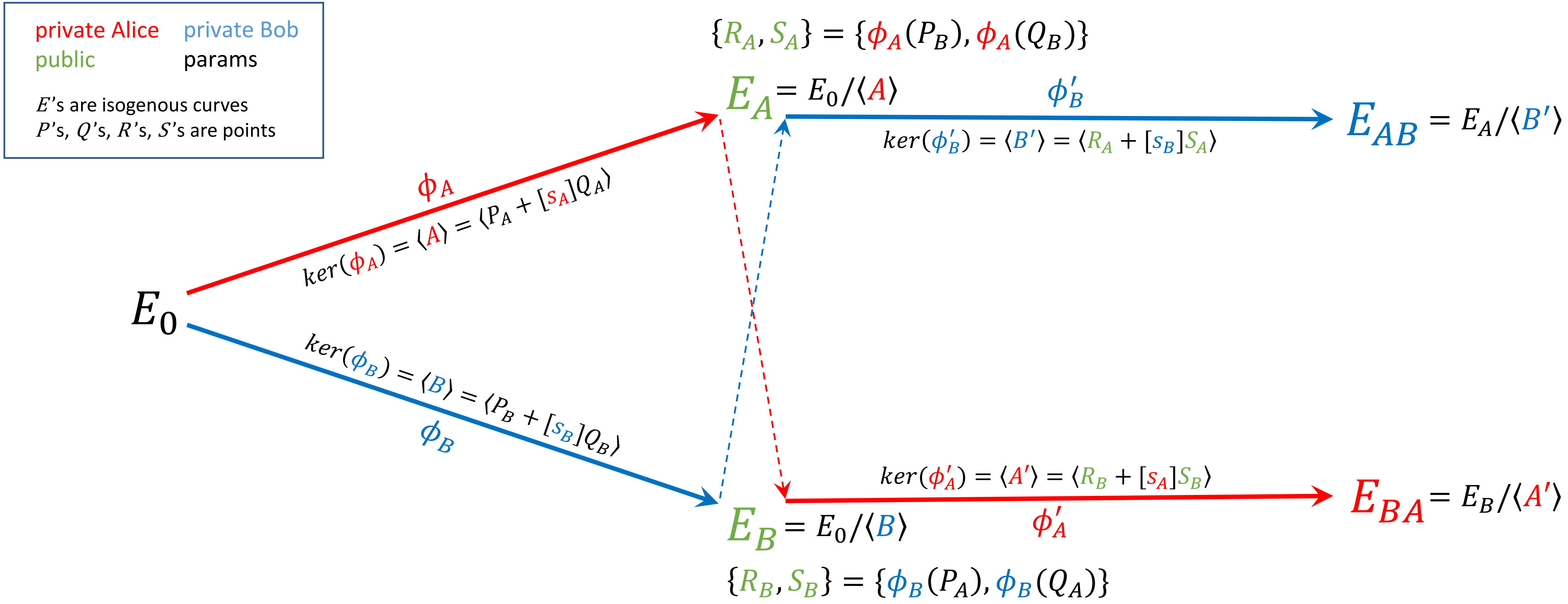
E's are isogenous curves
P's, Q's, R's, S's are points



Supersingular Isogeny Diffie-Hellman (SIDH)



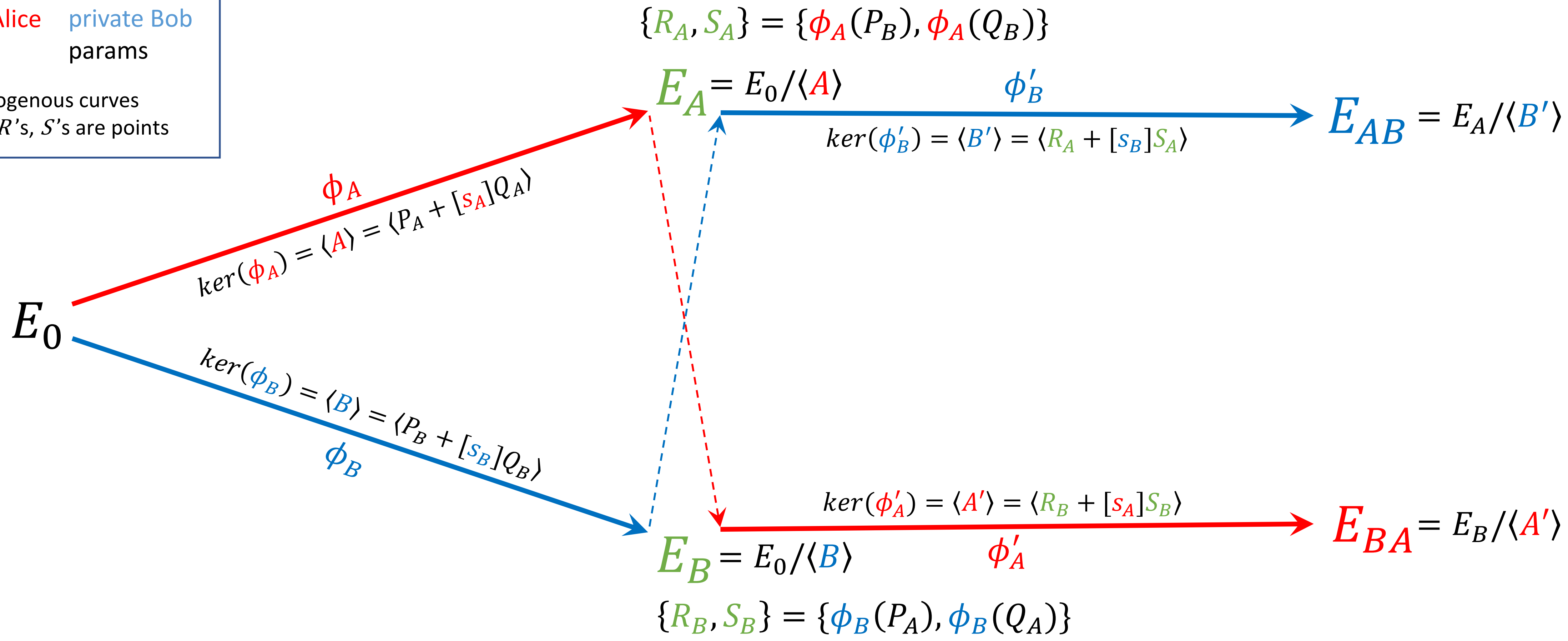
Supersingular Isogeny Diffie-Hellman (SIDH)



Supersingular Isogeny Diffie-Hellman (SIDH)

private Alice — private Bob
public — params

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

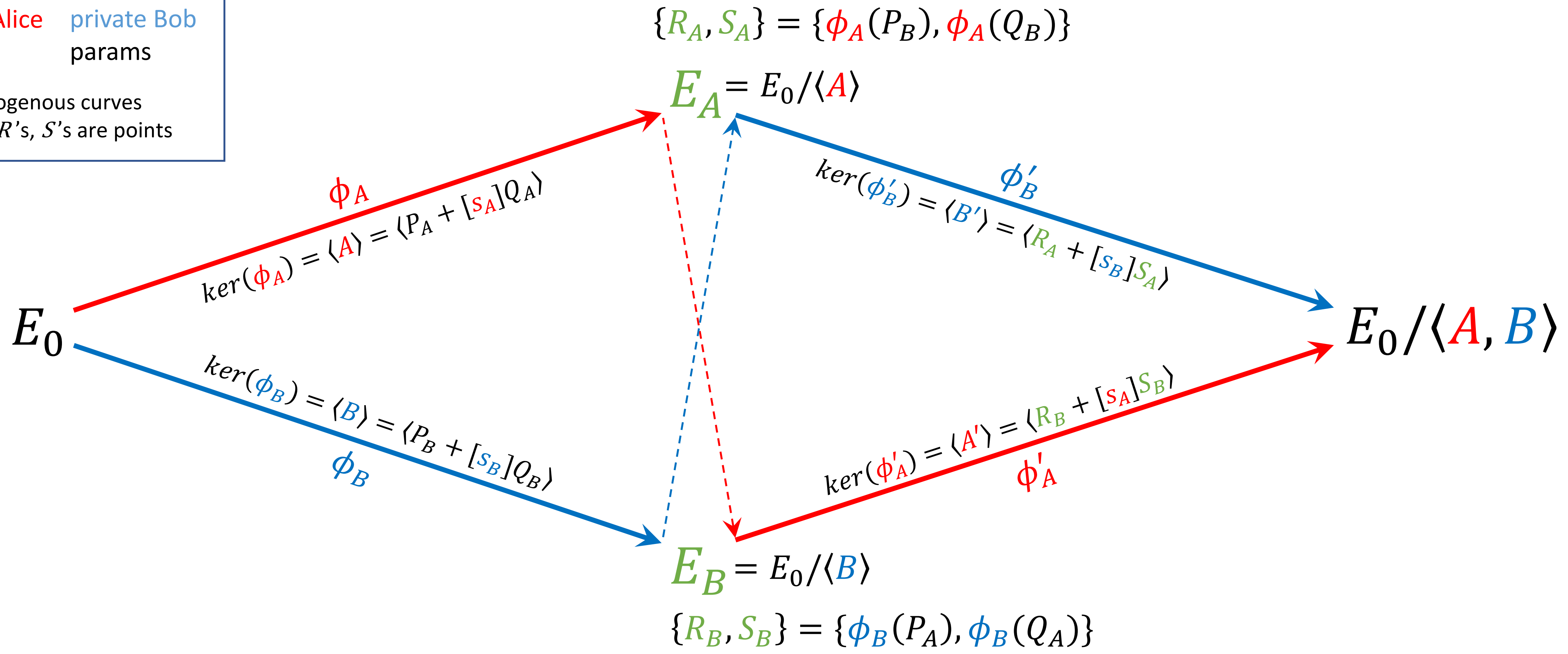


$$E_{AB} = \phi'_B(\phi_A(E_0)) \cong E_0/\langle P_A + [s_A]Q_A, P_B + [s_B]Q_B \rangle \cong E_{BA} = \phi'_A(\phi_B(E_0))$$

Supersingular Isogeny Diffie-Hellman (SIDH)

private Alice | private Bob
public | params

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



$$E_{AB} = \phi'_B(\phi_A(E_0)) \cong E_0/\langle P_A + [s_A]Q_A, P_B + [s_B]Q_B\rangle \cong E_{BA} = \phi'_A(\phi_B(E_0))$$

SIDH security

Setting: supersingular curves $E_1/\mathbb{F}_{p^2}$ and $E_2/\mathbb{F}_{p^2}$, a large prime p, and isogeny $\phi: E_1 \to E_2$ with fixed, smooth, public degree.

Supersingular isogeny problem: given $P, Q \in E_1$ and $\phi(P_1), \phi(P_2) \in E_2$, compute ϕ.

- **Best known attacks:** classical $O(p^{1/4})$ and quantum $O(p^{1/6})$ via **generic claw finding algorithms**

Supersingular Isogeny Diffie-Hellman (SIDH)

(Until recently) two problems remained:

- Existing realizations were still slow (running in the hundreds of milliseconds) and unprotected against side-channel attacks
- SIDH is not secure when keys are reused (Galbraith-Petit-Shani-Ti 2016)
  - Only recommended in **ephemeral mode**

(A brief) Timeline of isogeny-based crypto, part II

2016 SIDH gets closer to practical use (Costello-Longa-Naehrig 2016).

- New parameter set (SIDHp751) for the 128-bit quantum security level.
- Several optimization techniques push performance below 60 milliseconds (in "constant-time").

But still not fast enough for some applications,
and not secure with static keys.

2017 …

Supersingular isogeny key encapsulation (SIKE)

Costello–De Feo–Jao–Longa–Naehrig–Renes, 2017

- IND-CCA secure key encapsulation: no problem reusing keys!
- Uses a variant of Hofheinz–Hövelmanns–Kiltz (HHK) transform: **IND-CPA PKE → IND-CCA KEM**
- HHK transform is secure in **both the classical and quantum ROM models**

Supersingular isogeny key encapsulation (SIKE)

Costello–De Feo–Jao–Longa–Naehrig–Renes, 2017

- IND-CCA secure key encapsulation: no problem reusing keys!
- Uses a variant of Hofheinz–Hövelmanns–Kiltz (HHK) transform: **IND-CPA PKE → IND-CCA KEM**
- HHK transform is secure in **both the classical and quantum ROM models**
- Offline key generation gives performance boost (no perf loss SIDH → SIKE)

Supersingular isogeny key encapsulation (SIKE)

Costello–De Feo–Jao–Longa–Naehrig–Renes, 2017

- IND-CCA secure key encapsulation: no problem reusing keys!
- Uses a variant of Hofheinz–Hövelmanns–Kiltz (HHK) transform: **IND-CPA PKE → IND-CCA KEM**
- HHK transform is secure in **both the classical and quantum ROM models**
- Offline key generation gives performance boost (no perf loss SIDH → SIKE)
- *Three* parameter sets matching security of AES-128, 192 and 256.

For a starting curve $E_0/\mathbb{F}_{p^2}$: $y^2 = x^3 + x$, where $p = 2^{e_A}3^{e_B} - 1$

Scheme (SIKEp + $\lceil \log_2 p \rceil$)	(e_A, e_B)	classical sec.	quantum sec.	Security level
SIKEp503	(250,159)	126 bits	84 bits	AES-128 (NIST level 1)
SIKEp751	(372,239)	188 bits	125 bits	AES-192 (NIST level 3)
SIKEp964	(486,301)	241 bits	161 bits	AES-256 (NIST level 5)

Supersingular isogeny key encapsulation (SIKE)

Costello–De Feo–Jao–Longa–Naehrig–Renes, 2017

KeyGen

1. $s_B \in_R [0, 2^{\lfloor \log_2 3^{e_B} \rfloor})$
2. Set $ker(\phi_B) = \langle P_B + [s_B]Q_B \rangle$
3. $\mathrm{pk}_B = \{\phi_B(E_0), \phi_B(P_A), \phi_B(Q_A)\}$
4. $s \in_R \{0,1\}^n$
5. **keypair:** $\{\mathrm{sk}_B = (s, s_B), \mathrm{pk}_B\}$

Supersingular isogeny key encapsulation (SIKE)

Costello–De Feo–Jao–Longa–Naehrig–Renes, 2017

KeyGen

1. $s_B \in_R [0, 2^{\lfloor \log_2 3^{e_B} \rfloor})$
2. Set $ker(\phi_B) = \langle P_B + [s_B]Q_B \rangle$
3. $\mathrm{pk}_B = \{\phi_B(E_0), \phi_B(P_A), \phi_B(Q_A)\}$
4. $s \in_R \{0,1\}^n$
5. **keypair:** $\{\mathrm{sk}_B = (s, s_B), \mathrm{pk}_B\}$

pk_B →

Encaps

1. message $m \in_R \{0,1\}^n$
2. $r = G(m, \mathrm{pk}_B) \bmod 2^{e_A}$
3. Set $ker(\phi_A) = \langle P_A + [r]Q_A \rangle$
4. $\mathrm{pk}_A = \{\phi_A(E_0), \phi_A(P_B), \phi_A(Q_B)\}$
5. $j = j(E_{AB}) = j(\phi'_A(\phi_B(E_0)))$
6. **Shared key:** $ss = H(m, c)$

Supersingular isogeny key encapsulation (SIKE)

Costello–De Feo–Jao–Longa–Naehrig–Renes, 2017

KeyGen

1. $s_B \in_R [0, 2^{\lfloor \log_2 3^{e_B} \rfloor})$
2. Set $ker(\phi_B) = \langle P_B + [s_B]Q_B \rangle$
3. $\mathrm{pk}_B = \{\phi_B(E_0), \phi_B(P_A), \phi_B(Q_A)\}$
4. $s \in_R \{0,1\}^n$
5. **keypair:** $\{\mathrm{sk}_B = (s, s_B), \mathrm{pk}_B\}$

pk_B →

Encaps

1. message $m \in_R \{0,1\}^n$
2. $r = G(m, \mathrm{pk}_B) \bmod 2^{e_A}$

encryption

3. Set $ker(\phi_A) = \langle P_A + [r]Q_A \rangle$
4. $\mathrm{pk}_A = \{\phi_A(E_0), \phi_A(P_B), \phi_A(Q_B)\}$
5. $j = j(E_{AB}) = j(\phi_A'(\phi_B(E_0)))$
6. **Shared key:** $ss = H(m, c)$

Supersingular isogeny key encapsulation (SIKE)

Costello–De Feo–Jao–Longa–Naehrig–Renes, 2017

KeyGen

1. $s_B \in_R [0, 2^{\lfloor \log_2 3^{e_B} \rfloor})$
2. Set $ker(\phi_B) = \langle P_B + [s_B]Q_B \rangle$
3. $\text{pk}_B = \{\phi_B(E_0), \phi_B(P_A), \phi_B(Q_A)\}$
4. $s \in_R \{0,1\}^n$
5. **keypair:** $\{\text{sk}_B = (s, s_B), \text{pk}_B\}$

pk_B →

← $c = (\text{pk}_A, F(j) \oplus m)$

Encaps

1. message $m \in_R \{0,1\}^n$
2. $r = G(m, \text{pk}_B) \bmod 2^{e_A}$

encryption

3. Set $ker(\phi_A) = \langle P_A + [r]Q_A \rangle$
4. $\text{pk}_A = \{\phi_A(E_0), \phi_A(P_B), \phi_A(Q_B)\}$
5. $j = j(E_{AB}) = j(\phi_A'(\phi_B(E_0)))$
6. **Shared key:** $ss = H(m, c)$

Decaps

1. $j' = j(E_{BA}) = j(\phi_B'(\phi_A(E_0)))$
2. $m' = F(j') \oplus c[2]$
3. $r' = G(m', \text{pk}_B) \bmod 2^{e_A}$
4. Set $ker(\phi_A) = \langle P_A + [r']Q_A \rangle$
5. $\text{pk}_A' = \{\phi_A(E_0), \phi_A(P_B), \phi_A(Q_B)\}$
6. If $\text{pk}_A' = c[1]$ then

   Shared key: $ss = H(m', c)$
7. Else $ss = H(s, c)$

Supersingular isogeny key encapsulation (SIKE)

Costello–De Feo–Jao–Longa–Naehrig–Renes, 2017

KeyGen

1. $s_B \in_R [0, 2^{\lfloor \log_2 3^{e_B} \rfloor})$
2. Set $ker(\phi_B) = \langle P_B + [s_B]Q_B \rangle$
3. $\mathrm{pk}_B = \{\phi_B(E_0), \phi_B(P_A), \phi_B(Q_A)\}$
4. $s \in_R \{0,1\}^n$
5. **keypair:** $\{\mathrm{sk}_B = (s, s_B), \mathrm{pk}_B\}$

pk_B →

← $c = (\mathrm{pk}_A, F(j) \oplus m)$

Encaps

1. message $m \in_R \{0,1\}^n$
2. $r = G(m, \mathrm{pk}_B) \bmod 2^{e_A}$

encryption

3. Set $ker(\phi_A) = \langle P_A + [r]Q_A \rangle$
4. $\mathrm{pk}_A = \{\phi_A(E_0), \phi_A(P_B), \phi_A(Q_B)\}$
5. $j = j(E_{AB}) = j(\phi_A'(\phi_B(E_0)))$
6. **Shared key:** $ss = H(m, c)$

Decaps

1. $j' = j(E_{BA}) = j(\phi_B'(\phi_A(E_0)))$
2. $m' = F(j') \oplus c[2]$

decryption

3. $r' = G(m', \mathrm{pk}_B) \bmod 2^{e_A}$
4. Set $ker(\phi_A) = \langle P_A + [r']Q_A \rangle$
5. $\mathrm{pk}_A' = \{\phi_A(E_0), \phi_A(P_B), \phi_A(Q_B)\}$
6. If $\mathrm{pk}_A' = c[1]$ then
   Shared key: $ss = H(m', c)$
7. Else $ss = H(s, c)$

Supersingular isogeny key encapsulation (SIKE)

Costello–De Feo–Jao–Longa–Naehrig–Renes, 2017

KeyGen

1. $s_B \in_R [0, 2^{\lfloor \log_2 3^{e_B} \rfloor})$
2. Set $ker(\phi_B) = \langle P_B + [s_B]Q_B \rangle$
3. $\mathrm{pk}_B = \{\phi_B(E_0), \phi_B(P_A), \phi_B(Q_A)\}$
4. $s \in_R \{0,1\}^n$
5. **keypair:** $\{\mathrm{sk}_B = (s, s_B), \mathrm{pk}_B\}$

pk_B →

← $c = (\mathrm{pk}_A, F(j) \oplus m)$

Encaps

1. message $m \in_R \{0,1\}^n$
2. $r = G(m, \mathrm{pk}_B) \bmod 2^{e_A}$

encryption

3. Set $ker(\phi_A) = \langle P_A + [r]Q_A \rangle$
4. $\mathrm{pk}_A = \{\phi_A(E_0), \phi_A(P_B), \phi_A(Q_B)\}$
5. $j = j(E_{AB}) = j(\phi'_A(\phi_B(E_0)))$
6. **Shared key:** $ss = H(m, c)$

Decaps

1. $j' = j(E_{BA}) = j(\phi'_B(\phi_A(E_0)))$
2. $m' = F(j') \oplus c[2]$

decryption

3. $r' = G(m', \mathrm{pk}_B) \bmod 2^{e_A}$
4. Set $ker(\phi_A) = \langle P_A + [r']Q_A \rangle$
5. $\mathrm{pk}'_A = \{\phi_A(E_0), \phi_A(P_B), \phi_A(Q_B)\}$

partial re-encryption

6. If $\mathrm{pk}'_A = c[1]$ then
   Shared key: $ss = H(m', c)$
7. Else $ss = H(s, c)$

F, G, H instantiated with cSHAKE256.

SIDH library

- Version 3.0 recently released:

  https://github.com/Microsoft/PQCrypto-SIDH

- Implements **SIDH** and **SIKE**
- Covers *two* security levels: SIDH/SIKEp503 (AES-128) and SIDH/SIKEp751 (AES-192)

SIDH library

- Version 3.0 recently released:

  https://github.com/Microsoft/PQCrypto-SIDH

- Implements **SIDH** and **SIKE**
- Covers *two* security levels: SIDH/SIKEp503 (AES-128) and SIDH/SIKEp751 (AES-192)
- With the following implementations:
  - A portable C implementation
  - A 64-bit optimized implementation
    - With high-speed x64 assembly code for the field arithmetic (Linux only)
    - With high-speed ARMv8 assembly code for the field arithmetic (SIDH/SIKEp751 only)

SIDH library

- Version 3.0 recently released:

  https://github.com/Microsoft/PQCrypto-SIDH

- Implements **SIDH** and **SIKE**
- Covers *two* security levels: SIDH/SIKEp503 (AES-128) and SIDH/SIKEp751 (AES-192)
- With the following implementations:
  - A portable C implementation
  - A 64-bit optimized implementation
    - With high-speed x64 assembly code for the field arithmetic (Linux only)
    - With high-speed ARMv8 assembly code for the field arithmetic (SIDH/SIKEp751 only)

- No secret branches, no secret memory accesses

SIDH library

- Version 3.0 recently released:

  https://github.com/Microsoft/PQCrypto-SIDH

- Implements **SIDH** and **SIKE**
- Covers *two* security levels: SIDH/SIKEp503 (AES-128) and SIDH/SIKEp751 (AES-192)
- With the following implementations:
  - A portable C implementation
  - A 64-bit optimized implementation
    - With high-speed x64 assembly code for the field arithmetic (Linux only)
    - With high-speed ARMv8 assembly code for the field arithmetic (SIDH/SIKEp751 only)

- No secret branches, no secret memory accesses: **code protected against cache and timing attacks!**

SIDH library

- Version 3.0 recently released:

  https://github.com/Microsoft/PQCrypto-SIDH

- Implements **SIDH** and **SIKE**
- Covers *two* security levels: SIDH/SIKEp503 (AES-128) and SIDH/SIKEp751 (AES-192)
- With the following implementations:
  - A portable C implementation
  - A 64-bit optimized implementation
    - With high-speed x64 assembly code for the field arithmetic (Linux only)
    - With high-speed ARMv8 assembly code for the field arithmetic (SIDH/SIKEp751 only)

- No secret branches, no secret memory accesses: ~~**code protected against cache and timing attacks!**~~

SIDH library

- Version 3.0 recently released:

  https://github.com/Microsoft/PQCrypto-SIDH

- Implements **SIDH** and **SIKE**
- Covers *two* security levels: SIDH/SIKEp503 (AES-128) and SIDH/SIKEp751 (AES-192)
- With the following implementations:
  - A portable C implementation
  - A 64-bit optimized implementation
    - With high-speed x64 assembly code for the field arithmetic (Linux only)
    - With high-speed ARMv8 assembly code for the field arithmetic (SIDH/SIKEp751 only)

- No secret branches, no secret memory accesses: ~~**code protected against cache and timing attacks!**~~
  - Assembly code **is not vulnerable** to recent branch target injection attacks (no branches)
  - For the C code: make sure to use a compiler that has been patched!

Performance on x64

Primitive	Quantum sec.	Problem	Speed	Comm.
Classical				
RSA 3072	~0 bits	factoring	4.6 ms	0.8 KB
ECDH NIST P-256	~0 bits	EC dlog	1.4 ms	0.1 KB
Passively secure key-exchange				
SIDHp503	84 bits	isogenies	10.3 ms	0.7 KB
SIDHp751	125 bits	isogenies	31.5 ms	1.1 KB
IND-CCA secure KEMs				
Kyber	161 bits	M-LWE	0.07 ms	1.2 KB
FrodoKEM	103–150 bits	LWE	1.2–2.3 ms	9.5–15.4 KB
SIKEp503	84 bits	isogenies	10.1 ms	0.4 KB
SIKEp751	125 bits	isogenies	30.5 ms	0.6 KB

very fast ■ ■ ■ slow   very small ■ ■ ■ large

(*) Obtained on 3.4GHz Intel Haswell (Kyber) or Skylake (FrodoKEM and SIKE).

Performance on 64-bit ARM

- Implementation by Matthew Campagna (Amazon)
- Timings obtained on 1.992GHz 64-bit ARM Cortex-A72 processor

Primitive	Speed
SIKEp503	53.4 ms
SIKEp751	171.6 ms

SIKE in the NIST post-quantum “competition”

- Package (protocol specifications and implementations) submitted to NIST:

  https://csrc.nist.gov/CSRC/media/Projects/Post-Quantum-Cryptography/documents/round-1/submissions/SIKE.zip

The full SIKE team

Reza Azarderakhsh, Matthew Campagna, Craig Costello, Luca De Feo, Basil Hess, David Jao, Brian Koziel, Brian LaMacchia, Patrick Longa, Michael Naehrig, Joost Renes, Vladimir Soukharev

Other relevant work in 2017

- **Faster compression:** Zanon *et al*. https://eprint.iacr.org/2017/1143
- **Optimized algorithms:** Faz-Hernández *et al*. https://eprint.iacr.org/2017/1015
- **Signatures:** Yoo *et al*. https://eprint.iacr.org/2017/186, and Galbraith et al. https://eprint.iacr.org/2016/1154

References

- J.-M. Couveignes. **Computing l-isogenies using the p-torsion**, in ANTS-II, 1996.
- J.-M. Couveignes. **Hard homogeneous spaces**, 1997. https://eprint.iacr.org/2006/291
- A. Childs, D. Jao, V. Soukharev. **Constructing elliptic curve isogenies in quantum subexponential time**, Journal of Math. Cryptology, 2014. http://arxiv.org/abs/1012.4019 (2010)
- C. Costello, P. Longa, M. Naehrig. **Efficient Algorithms for supersingular isogeny Diffie-Hellman**, in Advances in Cryptology–CRYPTO 2016. https://eprint.iacr.org/2016/413
- S.D. Galbraith, C. Petit, B. Shani, Y.B. Ti. **On the security of supersingular isogeny cryptosystems**, in ASIACRYPT 2016.
- D. Jao, L. De Feo. **Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies**, in PQCrypto 2011.
- A. Rostovtsev and A. Stolbunov. **Public-key cryptosystem based on isogenies**, 2006. https://eprint.iacr.org/2006/145
- A. Stolbunov, **Constructing public-key cryptographic schemes based on class group action on a set of isogenous elliptic curves**, in Adv. Math. Commun., 2010.

Supersingular isogeny based cryptography gets practical

Patrick Longa
Microsoft Research
https://www.microsoft.com/en-us/research/people/plonga/

Real World Crypto 2018
Zurich, Switzerland – Jan 10-12, 2018