A quick journey on what SI[DH/KE] is

ASCRYPTO 2021 - cryptography summer school

Jesús-Javier Chi-Domínguez [1]

[1] Cryptography Research Centre, Technology Innovation Institute, Abu Dhabi, UAE

jesus.dominguez@tii.ae

October 4, 2021

Technology Innovation Institute

Cryptography Research Centre

Finite Fields

- $\mathbb{F}_q$: finite field with q elements
- $\mathbb{F}_q^*$: multiplicative group (invertible elements)
- exponentiation: $g^k = \underbrace{g \times \cdots \times g}_{k \text{ times}}$

Elliptic curves

- $E(\mathbb{F}_q)$: group of $\mathbb{F}_q$-rational points on the curve E
- point at infinity $\mathcal{O}$: neutral element in $E(\mathbb{F}_q)$
- scalar point multiplication: $[k]P = \underbrace{P + \cdots + P}_{k \text{ times}}$
- order-d point P: $[d]P = \mathcal{O}$
- $x(P)$: x-coordinate of a point P

Common notation

- p: prime number
- q: a power of p (either p or p^2)
- $[\![a \mathinner{..} b]\!]$: integers in the interval $[a, b]$
- $\xleftarrow{\$}$: random selection from a given set

Framework: Notation

Finite Fields

- $\mathbb{F}_q$: finite field with q elements
- $\mathbb{F}_q^*$: multiplicative group (invertible elements)
- exponentiation: $g^k = \underbrace{g \times \cdots \times g}_{k \text{ times}}$

Elliptic curves

- $E(\mathbb{F}_q)$: group of $\mathbb{F}_q$-rational points on the curve E
- point at infinity $\mathcal{O}$: neutral element in $E(\mathbb{F}_q)$
- scalar point multiplication: $[k]P = \underbrace{P + \cdots + P}_{k \text{ times}}$
- order-d point P: $[d]P = \mathcal{O}$
- $x(P)$: x-coordinate of a point P

Common notation

- p: prime number
- q: a power of p (either p or p^2)
- $[\![a \mathinner{..} b]\!]$: integers in the interval $[a, b]$
- $\xleftarrow{\$}$: random selection from a given set

Finite Fields

- $\mathbb{F}_q$: finite field with q elements
- $\mathbb{F}_q^*$: multiplicative group (invertible elements)
- exponentiation: $g^k = \underbrace{g \times \cdots \times g}_{k \text{ times}}$

Elliptic curves

- $E(\mathbb{F}_q)$: group of $\mathbb{F}_q$-rational points on the curve E
- point at infinity $\mathcal{O}$: neutral element in $E(\mathbb{F}_q)$
- scalar point multiplication: $[k]P = \underbrace{P + \cdots + P}_{k \text{ times}}$
- order-d point P: $[d]P = \mathcal{O}$
- $x(P)$: x-coordinate of a point P

Common notation

- p: prime number
- q: a power of p (either p or p^2)
- $[\![a \,..\, b]\!]$: integers in the interval $[a, b]$
- $\xleftarrow{\$}$: random selection from a given set

Overview to the protocol: DH and ECDH

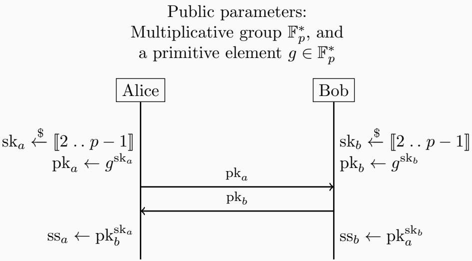


Figure: DH protocol assumes p is a prime number.
Notice, public keys are integers.

Remarks

1. Alice and Bob perform the same computations
2. Private keys are integers

Overview to the protocol: DH and ECDH

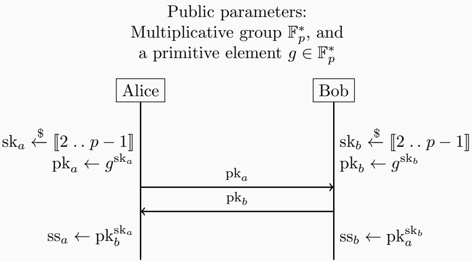


Figure: DH protocol assumes p is a prime number. Notice, public keys are integers.

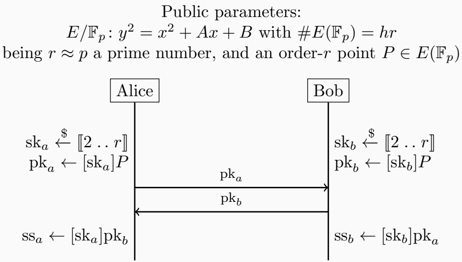


Figure: ECDH protocol assumes p is a prime number. Notice, public keys are points.

Remarks

1. Alice and Bob perform the same computations
2. Private keys are integers

Overview to the protocol: DH and ECDH

DH setup

1. $G = \mathbb{F}_p^*$, and public keys pk's are integers;
2. `keygen()` performs modular exponentiations with fixed primitive element $g \in G$;
3. `derive()` performs modular exponentiations with variable element pk.

ECDH setup

1. $G = E(\mathbb{F}_p)$, and public keys pk's are points;
2. `keygen()` performs scalar point multiplications with fixed order-r point $g \in G$;
3. `derive()` performs scalar point multiplications with variable order-r point pk.

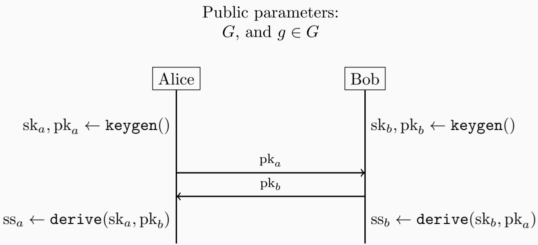


Figure: DH protocol on G by using `keygen` and `derive` procedures.

SIDH setup

1. Montgomery curves with $\#E(\mathbb{F}_{p^2}) = (p+1)^2$;
2. Alice's public key involves P_b and Q_b;
3. Bob's public key involves P_a and Q_a;
4. Alice and Bob perform different computations.

Our goals are

1. Sum up Kummer line arithmetic and isogenies;
2. Describe $\texttt{keygen}_{\texttt{A}}$ and $\texttt{keygen}_{\texttt{B}}$ blocks;
3. Describe $\texttt{derive}_{\texttt{A}}$ and $\texttt{derive}_{\texttt{B}}$ blocks;
4. Illustrate the hard problem on SI[DH/KE]

Public parameters:

$$E/\mathbb{F}_{p^2}: y^2 = x^3 + 6x^2 + x \text{ with } p = 2^{e_2}3^{e_3} - 1,$$
$$x(P_a),\ x(Q_a),\ x(P_a - Q_a),\ x(P_b),\ x(Q_b), \text{ and } x(P_b - Q_b)$$

Alice		Bob
$\mathrm{sk}_a, \mathrm{pk}_a \leftarrow \texttt{keygen}_{\texttt{A}}()$		$\mathrm{sk}_b, \mathrm{pk}_b \leftarrow \texttt{keygen}_{\texttt{B}}()$
	$\xrightarrow{\mathrm{pk}_a}$	
	$\xleftarrow{\mathrm{pk}_b}$	
$\mathrm{ss}_a \leftarrow \texttt{derive}_{\texttt{A}}(\mathrm{sk}_a, \mathrm{pk}_b)$		$\mathrm{ss}_b \leftarrow \texttt{derive}_{\texttt{B}}(\mathrm{sk}_b, \mathrm{pk}_a)$

Figure: SIDH protocol. Alice's and Bob's secret computations involves $\{P_a, Q_a\}$ and $\{P_b, Q_b\}$, respectively.

SIDH setup

1. Montgomery curves with $\#E(\mathbb{F}_{p^2}) = (p+1)^2$;
2. Alice's public key involves P_b and Q_b;
3. Bob's public key involves P_a and Q_a;
4. Alice and Bob perform different computations.

Our goals are

1. Sum up Kummer line arithmetic and isogenies;
2. Describe $\texttt{keygen}_\texttt{A}$ and $\texttt{keygen}_\texttt{B}$ blocks;
3. Describe $\texttt{derive}_\texttt{A}$ and $\texttt{derive}_\texttt{B}$ blocks;
4. Illustrate the hard problem on SI[DH/KE]

Public parameters:
$$E/\mathbb{F}_{p^2}: y^2 = x^3 + 6x^2 + x \text{ with } p = 2^{e_2}3^{e_3} - 1,$$
$$x(P_a),\ x(Q_a),\ x(P_a - Q_a),\ x(P_b),\ x(Q_b), \text{ and } x(P_b - Q_b)$$

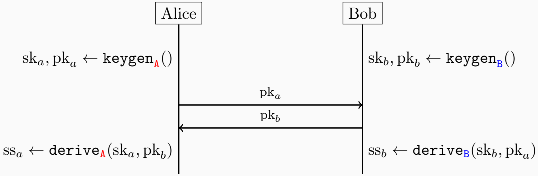


Figure: SIDH protocol. Alice's and Bob's secret computations involves $\{P_a, Q_a\}$ and $\{P_b, Q_b\}$, respectively.

- Only x-coordinates are required;
- Supersingular curves with $\#E(\mathbb{F}_{p^2}) = (p+1)^2$, and j-invariant $j(E) = \frac{256(A^2-2)^3}{A^2-4}$;
- Three-point ladder: $x(P + [k]Q)$.

$$\underbrace{x(P+Q) = \frac{\left(x(P)x(Q)-1\right)^2}{\left(x(P)-x(Q)\right)^2 x(P-Q)}}_{\text{Point addition}}$$

$$\underbrace{x([2]P) = \frac{\left(x(P)^2-1\right)^2}{4x(P)(x(P)^2+Ax(P)+1)}}_{\text{Point doubling}}$$

$$\underbrace{x([3]P) = \frac{\left(x(P)^4-4Ax(P)-6x(P)^2-3\right)^2 x(P)}{\left(4Ax(P)^3+3x(P)^4+6x(P)^2-1\right)^2}}_{\text{Point tripling}}$$

What is a d-isogeny $\phi\colon E \to E'$?

- A rational map between two curves with finite kernel;
- A group homomorphism with $\# \ker \phi = d$;

Remarks

- $\ker \phi = \langle P \rangle$ for some order-d point P, that is, $\phi([k]P) = \mathcal{O}$;
- $\phi(S + T) = \phi(S) + \phi(T)$;
- ϕ preserves curve size: $\#E(\mathbb{F}_{p^2}) = \#E'(\mathbb{F}_{p^2})$.

$$\underbrace{A' = 2\left(1 - 2x(P_2)^2\right)}_{\text{2-isogeny}}$$

$$\underbrace{A' = \left(Ax(P_3) - 6x(P_3)^2 + 6\right)x(P_3)}_{\text{3-isogeny}}$$

$$\underbrace{A' = 4x(P_4)^2 - 2}_{\text{4-isogeny}}$$

Why $(0, 0)$ cannot live in $\ker \phi$?

Kummer line: isogenies (codomain curves)

What is a d-isogeny $\phi\colon E \to E'$?

- A rational map between two curves with finite kernel;
- A group homomorphism with $\#\ker\phi = d$;

Remarks

- $\ker\phi = \langle P\rangle$ for some order-d point P, that is, $\phi([k]P) = \mathcal{O}$;
- $\phi(S+T) = \phi(S) + \phi(T)$;
- ϕ preserves curve size: $\#E(\mathbb{F}_{p^2}) = \#E'(\mathbb{F}_{p^2})$.

$$\underbrace{A' = 2\left(1 - 2x(P_2)^2\right)}_{\text{2-isogeny}}$$

$$\underbrace{A' = \left(Ax(P_3) - 6x(P_3)^2 + 6\right)x(P_3)}_{\text{3-isogeny}}$$

$$\underbrace{A' = 4x(P_4)^2 - 2}_{\text{4-isogeny}}$$

Why $(0,0)$ cannot live in $\ker\phi$?

Kummer line: isogenies (codomain curves)

What is a d-isogeny $\phi \colon E \to E'$?

- A rational map between two curves with finite kernel;
- A group homomorphism with $\# \ker \phi = d$;

Remarks

- $\ker \phi = \langle P \rangle$ for some order-d point P, that is, $\phi([k]P) = \mathcal{O}$;
- $\phi(S + T) = \phi(S) + \phi(T)$;
- ϕ preserves curve size: $\#E(\mathbb{F}_{p^2}) = \#E'(\mathbb{F}_{p^2})$.

empty line

$$\underbrace{A' = 2\left(1 - 2x(P_2)^2\right)}_{\text{2-isogeny}}$$

$$\underbrace{A' = \left(Ax(P_3) - 6x(P_3)^2 + 6\right)x(P_3)}_{\text{3-isogeny}}$$

$$\underbrace{A' = 4x(P_4)^2 - 2}_{\text{4-isogeny}}$$

Why $(0, 0)$ cannot live in $\ker \phi$? It gives $A = 2$, and then $j(E')$ is undetermined (there is a division by zero)

Kummer line: pushing points through isogenies

- SI[DH/KE] performs large 2^{e_2}-isogenies and 3^{e_3}-isogenies;
- So we need an efficient way to map points to codomain curves;

$$\underbrace{x(\phi(Q)) = \frac{x(Q)^2 x(P_2) - x(Q)}{x(Q) - x(P_2)}}_{\text{2-isogeny}}$$

$$\underbrace{x(\phi(Q)) = \frac{x(Q)\big(x(Q)x(P_3) - 1\big)^2}{\big(x(Q) - x(P_3)\big)^2}}_{\text{3-isogeny}}$$

$$\underbrace{x(\phi(Q)) = \frac{-\big(x(Q)x(P_4)^2 + x(Q) - 2x(P_4)\big)x(Q)\big(x(Q)x(P_4) - 1\big)^2}{\big(x(Q) - x(P_4)\big)^2\big(2x(Q)x(P_4) - x(P_4)^2 - 1\big)}}_{\text{4-isogeny}}$$

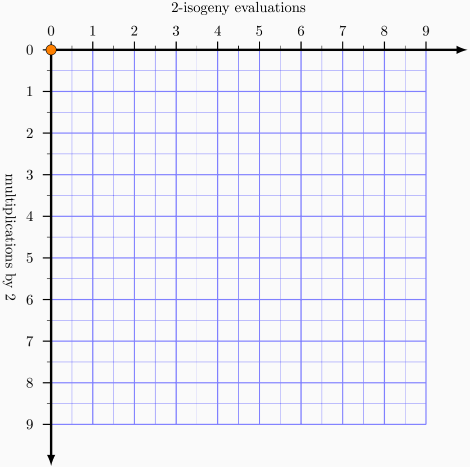


Figure: Strategy evaluation for isogenies chains

Computing a 2^{10}-isogeny by using an order-2^{10} point P:

1. Split the task into 10 2-isogenies;
2. Compute $K_1 = [2^9]P$, and the 2-isogeny $\phi_1 : E \to E_1$ with kernel $\langle K_1 \rangle$;
3. Get $P_1 = \phi_1(P)$. Which is the order of $\phi_1(P)$?
4. Compute $K_2 = [2^8]P_1$, and the 2-isogeny $\phi_1 : E_1 \to E_2$ with kernel $\langle K_2 \rangle$;
5. The i-th 2-isogeny has kernel generator $K_i = \phi_1 \circ \cdots \circ \phi_{i-1}([2^{10-i}]P)$.
6. Different strategies!
7. Which is the running time of this strategy?
8. Dynamic programming gives $O(e_2 \log_2(e_2))$

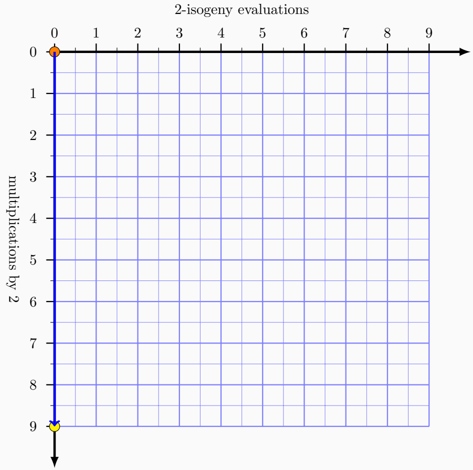


Figure: Strategy evaluation for isogenies chains

Computing a 2^{10}-isogeny by using an order-2^{10} point P:

1. Split the task into 10 2-isogenies;
2. Compute $K_1 = [2^9]P$, and the 2-isogeny $\phi_1 : E \to E_1$ with kernel $\langle K_1 \rangle$;
3. Get $P_1 = \phi_1(P)$. Which is the order of $\phi_1(P)$?
4. Compute $K_2 = [2^8]P_1$, and the 2-isogeny $\phi_1 : E_1 \to E_2$ with kernel $\langle K_2 \rangle$;
5. The i-th 2-isogeny has kernel generator $K_i = \phi_1 \circ \cdots \circ \phi_{i-1}([2^{10-i}]P)$.
6. Different strategies!
7. Which is the running time of this strategy?
8. Dynamic programming gives $O(e_2 \log_2(e_2))$

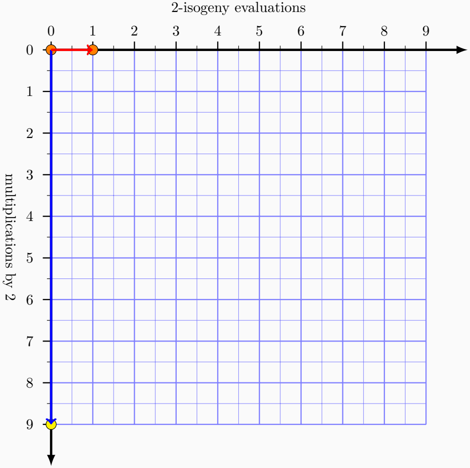


Figure: Strategy evaluation for isogenies chains

Computing a 2^{10}-isogeny by using an order-2^{10} point P:

1. Split the task into 10 2-isogenies;
2. Compute $K_1 = [2^9]P$, and the 2-isogeny $\phi_1 : E \to E_1$ with kernel $\langle K_1 \rangle$;
3. Get $P_1 = \phi_1(P)$. Which is the order of $\phi_1(P)$?
4. Compute $K_2 = [2^8]P_1$, and the 2-isogeny $\phi_1 : E_1 \to E_2$ with kernel $\langle K_2 \rangle$;
5. The i-th 2-isogeny has kernel generator $K_i = \phi_1 \circ \cdots \circ \phi_{i-1}([2^{10-i}]P)$.
6. Different strategies!
7. Which is the running time of this strategy?
8. Dynamic programming gives $O(e_2 \log_2(e_2))$

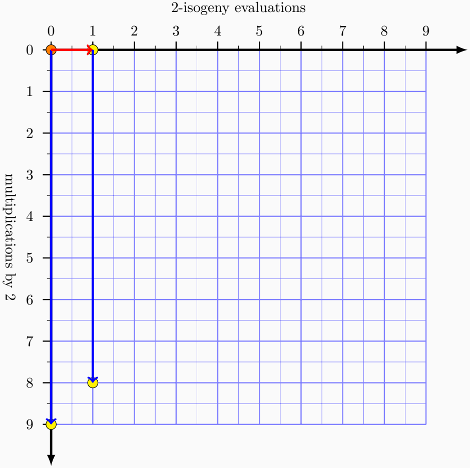


Figure: Strategy evaluation for isogenies chains

Computing a 2^{10}-isogeny by using an order-2^{10} point P:

1. Split the task into 10 2-isogenies;
2. Compute $K_1 = [2^9]P$, and the 2-isogeny $\phi_1 : E \to E_1$ with kernel $\langle K_1 \rangle$;
3. Get $P_1 = \phi_1(P)$. Which is the order of $\phi_1(P)$?
4. Compute $K_2 = [2^8]P_1$, and the 2-isogeny $\phi_1 : E_1 \to E_2$ with kernel $\langle K_2 \rangle$;
5. The i-th 2-isogeny has kernel generator $K_i = \phi_1 \circ \cdots \circ \phi_{i-1}([2^{10-i}]P)$.
6. Different strategies!
7. Which is the running time of this strategy?
8. Dynamic programming gives $O(e_2 \log_2(e_2))$

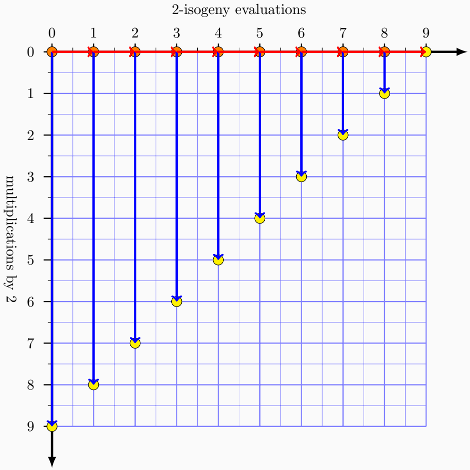


Figure: Strategy evaluation for isogenies chains

Computing a 2^{10}-isogeny by using an order-2^{10} point P:

1. Split the task into 10 2-isogenies;
2. Compute $K_1 = [2^9]P$, and the 2-isogeny $\phi_1 : E \to E_1$ with kernel $\langle K_1 \rangle$;
3. Get $P_1 = \phi_1(P)$. Which is the order of $\phi_1(P)$?
4. Compute $K_2 = [2^8]P_1$, and the 2-isogeny $\phi_1 : E_1 \to E_2$ with kernel $\langle K_2 \rangle$;
5. The i-th 2-isogeny has kernel generator $K_i = \phi_1 \circ \cdots \circ \phi_{i-1}([2^{10-i}]P)$.
6. Different strategies!
7. Which is the running time of this strategy?
8. Dynamic programming gives $O(e_2 \log_2(e_2))$

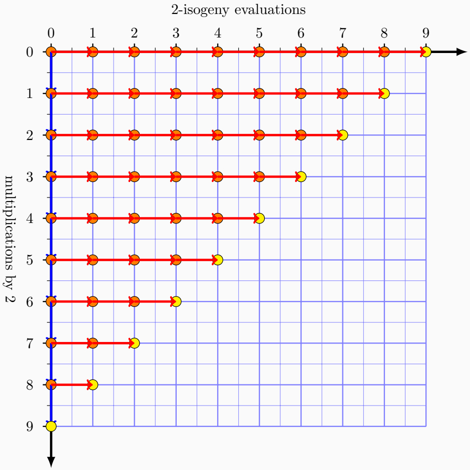


Figure: Strategy evaluation for isogenies chains

Computing a 2^{10}-isogeny by using an order-2^{10} point P:

1. Split the task into 10 2-isogenies;
2. Compute $K_1 = [2^9]P$, and the 2-isogeny $\phi_1 : E \to E_1$ with kernel $\langle K_1 \rangle$;
3. Get $P_1 = \phi_1(P)$. Which is the order of $\phi_1(P)$?
4. Compute $K_2 = [2^8]P_1$, and the 2-isogeny $\phi_1 : E_1 \to E_2$ with kernel $\langle K_2 \rangle$;
5. The i-th 2-isogeny has kernel generator $K_i = \phi_1 \circ \cdots \circ \phi_{i-1}([2^{10-i}]P)$.
6. Different strategies!
7. Which is the running time of this strategy?
8. Dynamic programming gives $O(e_2 \log_2(e_2))$

Kummer line: isogenies chain (example)

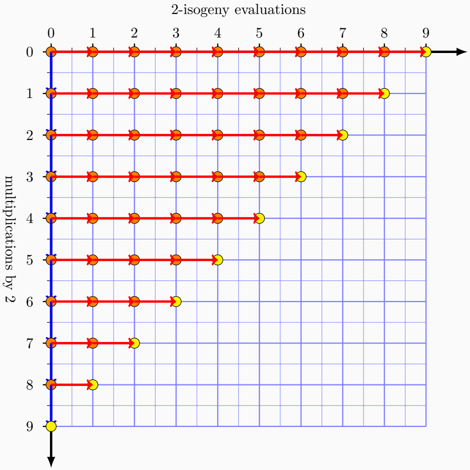


Figure: Strategy evaluation for isogenies chains

Computing a 2^{10}-isogeny by using an order-2^{10} point P:

1. Split the task into 10 2-isogenies;
2. Compute $K_1 = [2^9]P$, and the 2-isogeny $\phi_1 : E \to E_1$ with kernel $\langle K_1 \rangle$;
3. Get $P_1 = \phi_1(P)$. Which is the order of $\phi_1(P)$?
4. Compute $K_2 = [2^8]P_1$, and the 2-isogeny $\phi_1 : E_1 \to E_2$ with kernel $\langle K_2 \rangle$;
5. The i-th 2-isogeny has kernel generator $K_i = \phi_1 \circ \cdots \circ \phi_{i-1}([2^{10-i}]P)$.
6. Different strategies! Any idea to find an optimal one?
7. Which is the running time of this strategy? Quadratic complexity!
8. Dynamic programming gives $O(e_2 \log_2(e_2))$

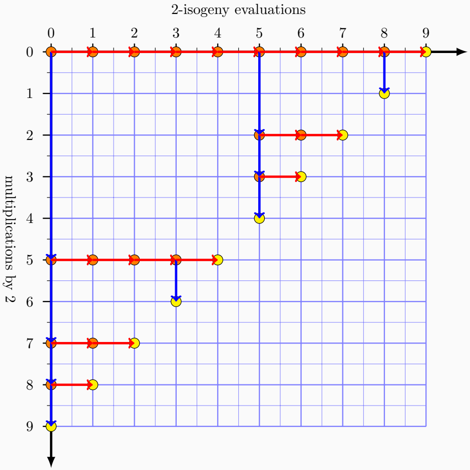


Figure: Strategy evaluation for isogenies chains

Computing a 2^{e_2}-isogeny by using an order-2^{e_2} point P:

1. Split the task into e_2 2-isogenies;
2. Compute $K_1 = [2^9]P$, and the 2-isogeny $\phi_1 : E \to E_1$ with kernel $\langle K_1 \rangle$;
3. Get $P_1 = \phi_1(P)$. Which is the order of $\phi_1(P)$?
4. Compute $K_2 = [2^8]P_1$, and the 2-isogeny $\phi_1 : E_1 \to E_2$ with kernel $\langle K_2 \rangle$;
5. The i-th 2-isogeny has kernel generator $K_i = \phi_1 \circ \cdots \circ \phi_{i-1}([2^{10-i}]P)$.
6. Different strategies! Any idea to find an optimal one?
7. Which is the running time of this strategy? Quadratic complexity!
8. Dynamic programming gives $O(e_2 \log_2(e_2))$

1. Staring curve $E: y^2 = x^3 + 6x^2 + x$;
2. Pushing public order-3^{e_3}; points through a secret 2^{e_2}-isogeny.

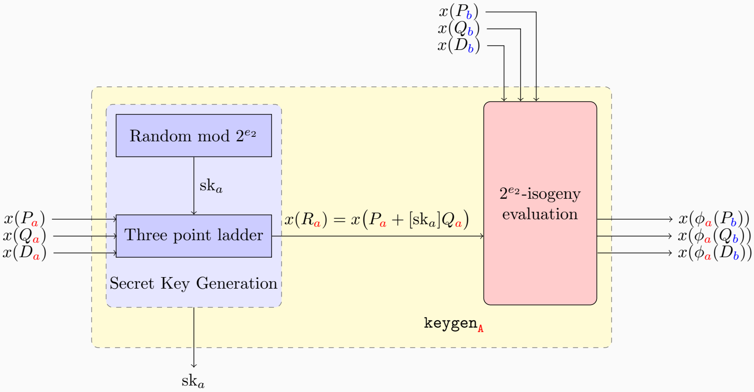


Figure: Public key generation.

SIDH main block: `keygen`

1. Staring curve $E: y^2 = x^3 + 6x^2 + x$;
2. Pushing public order-2^{e_2}; points through a secret 3^{e_3}-isogeny.

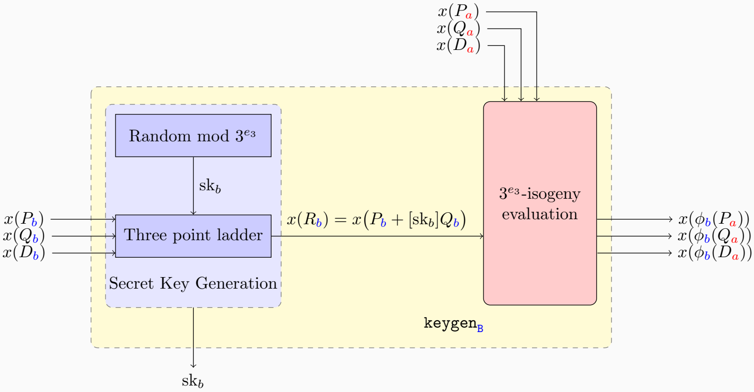


Figure: Public key generation.

SIDH main block: `derive`

1. Secret 2^{e_2}-isogeny;
2. No extra points required;
3. $E/\langle R\rangle$ denotes the codomain curve of the isogeny with kernel $\langle R\rangle$.

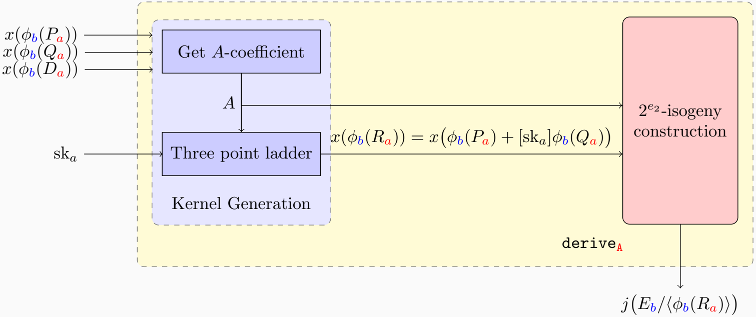


Figure: Shared secret derivation

SIDH main block: `derive`

1. Secret 3^{e_3}-isogeny;
2. No extra points required;
3. $E/\langle R\rangle$ denotes the codomain curve of the isogeny with kernel $\langle R\rangle$.

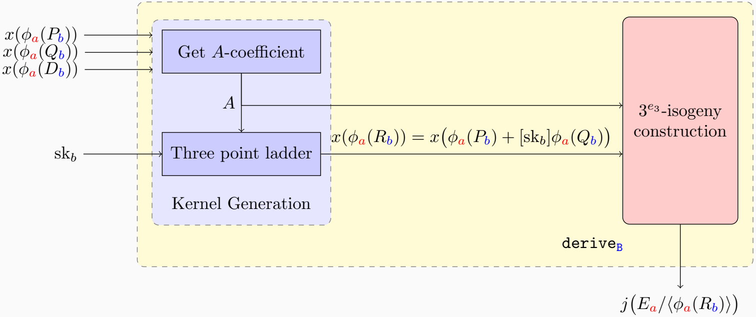


Figure: Shared secret derivation

SIDH protocol: diagram

$$\begin{tikzcd}
E \arrow[r, "\phi_a"] \arrow[d, "\phi_b"'] & E_a \arrow[d, "\psi_b"] \\
E_b \arrow[r, "\psi_a"'] & E'
\end{tikzcd}$$

Figure: SIDH diagram.

- Alice gets the codomain curve of $\phi_b \circ \psi_a$;
- Bob obtains the codomain curve of $\phi_a \circ \psi_b$;
- What are the kernels of $\phi_b \circ \psi_a$ and $\phi_a \circ \psi_b$?

$$
\begin{tikzcd}
E \arrow[r, "\phi_a"] \arrow[d, "\phi_b"] & E_a \arrow[d, "\psi_b"] \\
E_b \arrow[r, "\psi_a"'] & E'
\end{tikzcd}
$$

Figure: SIDH diagram.

- Alice gets the codomain curve of $\phi_b \circ \psi_a$;
- Bob obtains the codomain curve of $\phi_a \circ \psi_b$;
- What are the kernels of $\phi_b \circ \psi_a$ and $\phi_a \circ \psi_b$?

SIDH protocol: diagram

$$\begin{tikzcd} E \arrow[r, "\phi_a"] \arrow[d, "\phi_b"] & E_a \arrow[d, "\psi_b"] \\ E_b \arrow[r, "\psi_a"'] & E' \end{tikzcd}$$

Figure: SIDH diagram.

- Alice gets the codomain curve of $\phi_b \circ \psi_a$;
- Bob obtains the codomain curve of $\phi_a \circ \psi_b$;
- What are the kernels of $\phi_b \circ \psi_a$ and $\phi_a \circ \psi_b$?

SIDH protocol: diagram

$$\begin{tikzcd}[column sep=huge, row sep=huge]
E \arrow[r, "\phi_a"] \arrow[d, "\phi_b"'] & E_a \arrow[d, "\psi_b"] \\
E_b \arrow[r, "\psi_a"'] & E'
\end{tikzcd}$$

Figure: SIDH diagram.

- Alice gets the codomain curve of $\phi_b \circ \psi_a$;
- Bob obtains the codomain curve of $\phi_a \circ \psi_b$;
- What are the kernels of $\phi_b \circ \psi_a$ and $\phi_a \circ \psi_b$? $\langle R_a, R_b \rangle$
- Different isogeny composition ordering that gives isomorphic curves!

Technology Innovation Institute

Public parameters:

$$E/\mathbb{F}_{p^2} : y^2 = x^3 + 6x^2 + x \text{ with } p = 2^{e_2}3^{e_3} - 1,$$
$$x(P_a),\ x(Q_a),\ x(P_a - Q_a),\ x(P_b),\ x(Q_b), \text{ and } x(P_b - Q_b)$$

Alice		Bob
$\mathrm{sk}_a, \mathrm{pk}_a \leftarrow \texttt{keygen}_{\texttt{A}}()$		$\mathrm{sk}_b, \mathrm{pk}_b \leftarrow \texttt{keygen}_{\texttt{B}}()$
	$\xrightarrow{\mathrm{pk}_a}$	
	$\xleftarrow{\mathrm{pk}_b}$	
$\mathrm{ss}_a \leftarrow \texttt{derive}_{\texttt{A}}(\mathrm{sk}_a, \mathrm{pk}_b)$		$\mathrm{ss}_b \leftarrow \texttt{derive}_{\texttt{B}}(\mathrm{sk}_b, \mathrm{pk}_a)$

Figure: SIDH protocol.

Overview to the protocol: SIKE

Public parameter:

$$E/\mathbb{F}_{p^2}: y^2 = x^3 + 6x^2 + x \text{ with } p = 2^{e_2}3^{e_3} - 1,$$
$$x(P_a),\ x(Q_a),\ x(P_a - Q_a),\ x(P_b),\ x(Q_b), \text{ and } x(P_b - Q_b)$$

KeyGen

$$\mathrm{sk}_3, \mathrm{pk}_3 \leftarrow \mathtt{keygen}_{\mathtt{B}}()$$
$$s \xleftarrow{\$} \{0,1\}^n$$

Encaps

$$m \xleftarrow{\$} \{0,1\}^n$$
$$\mathrm{sk}_2 = \mathtt{SHAKE256}(m||\mathrm{pk}_3, e_2)$$
$$c_0 \leftarrow \mathtt{keygen}^*_{\mathtt{A}}(\mathrm{sk}_2)$$
$$j \leftarrow \mathtt{derive}_{\mathtt{A}}(\mathrm{sk}_2, \mathrm{pk}_3)$$
$$c_1 = \mathtt{SHAKE256}(j, n) \oplus m$$
$$\xrightarrow{c_0, c_1}$$
$$K = \mathtt{SHAKE256}(m||c_0||c_1, k)$$

Decaps

$$j' \leftarrow \mathtt{derive}_{\mathtt{B}}(\mathrm{sk}_3, \mathrm{pk}_2)$$
$$m' = \mathtt{SHAKE256}(j', n) \oplus c_1$$
$$\mathrm{sk}_2' = \mathtt{SHAKE256}(m'||\mathrm{pk}_3, e_2)$$
$$c_0' \leftarrow \mathtt{keygen}^*_{\mathtt{A}}(\mathrm{sk}_2')$$

if $c_0 = c_0'$ **then**

$$K = \mathtt{SHAKE256}(m'||c_0||c_1, k)$$

else

$$K = \mathtt{SHAKE256}(s||c_0||c_1, k)$$

Figure: SIKE protocol. The $\mathtt{keygen}^*_{\mathtt{A}}()$ procedure is $\mathtt{keygen}_{\mathtt{A}}()$ but taking as input sk_a instead of computing it.

Overview to the protocol: SIKE

Public parameter:

$$E/\mathbb{F}_{p^2} : y^2 = x^3 + 6x^2 + x \text{ with } p = 2^{e_2}3^{e_3} - 1,$$
$$x(P_a),\ x(Q_a),\ x(P_a - Q_a),\ x(P_b),\ x(Q_b),\ \text{and } x(P_b - Q_b)$$

KeyGen	Encaps	Decaps
$\mathrm{sk}_3, \mathrm{pk}_3 \leftarrow \texttt{keygen}_{\texttt{B}}()$	$m \xleftarrow{\$} \{0,1\}^n$	$j' \leftarrow \texttt{derive}_{\texttt{B}}(\mathrm{sk}_3, \mathrm{pk}_2)$
$s \xleftarrow{\$} \{0,1\}^n$	$\mathrm{sk}_2 = \texttt{SHAKE256}(m \\| \mathrm{pk}_3, e_2)$	$m' = \texttt{SHAKE256}(j', n) \oplus c_1$
	$c_0 \leftarrow \texttt{keygen}^*_{\texttt{A}}(\mathrm{sk}_2)$	$\mathrm{sk}'_2 = \texttt{SHAKE256}(m' \\| \mathrm{pk}_3, e_2)$
	$j \leftarrow \texttt{derive}_{\texttt{A}}(\mathrm{sk}_2, \mathrm{pk}_3)$	$c'_0 \leftarrow \texttt{keygen}^*_{\texttt{A}}(\mathrm{sk}'_2)$
	$c_1 = \texttt{SHAKE256}(j, n) \oplus m$	**if** $c_0 = c'_0$ **then**
	$\xrightarrow{c_0, c_1}$	$K = \texttt{SHAKE256}(m' \\| c_0 \\| c_1, k)$
	$K = \texttt{SHAKE256}(m \\| c_0 \\| c_1, k)$	**else**
		$K = \texttt{SHAKE256}(s \\| c_0 \\| c_1, k)$

Figure: SIKE protocol. The $\texttt{keygen}^*_{\texttt{A}}()$ procedure is $\texttt{keygen}_{\texttt{A}}()$ but taking as input sk_a instead of computing it.

Public parameter:

$$E/\mathbb{F}_{p^2} : y^2 = x^3 + 6x^2 + x \text{ with } p = 2^{e_2}3^{e_3} - 1,$$
$$x(P_a),\ x(Q_a),\ x(P_a - Q_a),\ x(P_b),\ x(Q_b), \text{ and } x(P_b - Q_b)$$

KeyGen

$$\mathrm{sk}_3, \mathrm{pk}_3 \leftarrow \texttt{keygen}_{\texttt{B}}()$$
$$s \xleftarrow{\$} \{0,1\}^n$$

Encaps

$$m \xleftarrow{\$} \{0,1\}^n$$
$$\mathrm{sk}_2 = \texttt{SHAKE256}(m||\mathrm{pk}_3, e_2)$$
$$c_0 \leftarrow \texttt{keygen}^*_{\texttt{A}}(\mathrm{sk}_2)$$
$$j \leftarrow \texttt{derive}_{\texttt{A}}(\mathrm{sk}_2, \mathrm{pk}_3)$$
$$c_1 = \texttt{SHAKE256}(j, n) \oplus m$$
$$\xrightarrow{c_0,\, c_1}$$
$$K = \texttt{SHAKE256}(m||c_0||c_1, k)$$

Decaps

$$j' \leftarrow \texttt{derive}_{\texttt{B}}(\mathrm{sk}_3, \mathrm{pk}_2)$$
$$m' = \texttt{SHAKE256}(j', n) \oplus c_1$$
$$\mathrm{sk}'_2 = \texttt{SHAKE256}(m'||\mathrm{pk}_3, e_2)$$
$$c'_0 \leftarrow \texttt{keygen}^*_{\texttt{A}}(\mathrm{sk}'_2)$$

if $c_0 = c'_0$ **then**

$$K = \texttt{SHAKE256}(m'||c_0||c_1, k)$$

else

$$K = \texttt{SHAKE256}(s||c_0||c_1, k)$$

Figure: SIKE protocol. The $\texttt{keygen}^*_{\texttt{A}}()$ procedure is $\texttt{keygen}_{\texttt{A}}()$ but taking as input sk_a instead of computing it.

1. What are the private keys?
2. What are the public keys?
3. What is the shared secret?
4. There are ways to reduce the public-key sizes (not presented in this talk)
5. Let's see a demo using the sibc python-library
6. Hard problem
   - Alice side: Given E and $E/\langle P_a + [\mathrm{sk}_a]Q_a \rangle$, to find the 2^{e_2}-isogeny with kernel $\langle P_a + [\mathrm{sk}_a]Q_a \rangle$
   - Bob side: Given E and $E/\langle P_b + [\mathrm{sk}_b]Q_b \rangle$, to find the 3^{e_3}-isogeny with kernel $\langle P_b + [\mathrm{sk}_b]Q_b \rangle$

SI[DH/KE] key space (remarks)

1. What are the private keys? integers of $\frac{\log_2(p)}{2}$ bits
2. What are the public keys?
3. What is the shared secret?
4. There are ways to reduce the public-key sizes (not presented in this talk)
5. Let's see a demo using the sibc python-library
6. Hard problem
   - Alice side: Given E and $E/\langle P_a + [\mathrm{sk}_a]Q_a \rangle$, to find the 2^{e_2}-isogeny with kernel $\langle P_a + [\mathrm{sk}_a]Q_a \rangle$
   - Bob side: Given E and $E/\langle P_b + [\mathrm{sk}_b]Q_b \rangle$, to find the 3^{e_3}-isogeny with kernel $\langle P_b + [\mathrm{sk}_b]Q_b \rangle$

1. What are the private keys? integers of $\frac{\log_2(p)}{2}$ bits
2. What are the public keys?
3. What is the shared secret?
4. There are ways to reduce the public-key sizes (not presented in this talk)
5. Let's see a demo using the sibc python-library
6. Hard problem
   - Alice side: Given E and $E/\langle P_a + [\mathrm{sk}_a]Q_a \rangle$, to find the 2^{e_2}-isogeny with kernel $\langle P_a + [\mathrm{sk}_a]Q_a \rangle$
   - Bob side: Given E and $E/\langle P_b + [\mathrm{sk}_b]Q_b \rangle$, to find the 3^{e_3}-isogeny with kernel $\langle P_b + [\mathrm{sk}_b]Q_b \rangle$

SI[DH/KE] key space (remarks)

1. What are the private keys? integers of $\frac{\log_2(p)}{2}$ bits
2. What are the public keys? three x-coordinates of $2\log_2(p)$ bits: total of $6\log_2(p)$
3. What is the shared secret?
4. There are ways to reduce the public-key sizes (not presented in this talk)
5. Let's see a demo using the sibc python-library
6. Hard problem
   - Alice side: Given E and $E/\langle P_a + [\mathrm{sk}_a]Q_a\rangle$, to find the 2^{e_2}-isogeny with kernel $\langle P_a + [\mathrm{sk}_a]Q_a\rangle$
   - Bob side: Given E and $E/\langle P_b + [\mathrm{sk}_b]Q_b\rangle$, to find the 3^{e_3}-isogeny with kernel $\langle P_b + [\mathrm{sk}_b]Q_b\rangle$

1. What are the private keys? integers of $\frac{\log_2(p)}{2}$ bits
2. What are the public keys? three x-coordinates of $2\log_2(p)$ bits: total of $6\log_2(p)$
3. What is the shared secret?
4. There are ways to reduce the public-key sizes (not presented in this talk)
5. Let's see a demo using the sibc python-library
6. Hard problem
   - Alice side: Given E and $E/\langle P_a + [\mathrm{sk}_a]Q_a \rangle$, to find the 2^{e_2}-isogeny with kernel $\langle P_a + [\mathrm{sk}_a]Q_a \rangle$
   - Bob side: Given E and $E/\langle P_b + [\mathrm{sk}_b]Q_b \rangle$, to find the 3^{e_3}-isogeny with kernel $\langle P_b + [\mathrm{sk}_b]Q_b \rangle$

SI[DH/KE] key space (remarks)

1. What are the private keys? integers of $\frac{\log_2(p)}{2}$ bits
2. What are the public keys? three x-coordinates of $2\log_2(p)$ bits: total of $6\log_2(p)$
3. What is the shared secret? an $\mathbb{F}_{p^2}$-element of $2\log_2(p)$ bits
4. There are ways to reduce the public-key sizes (not presented in this talk)
5. Let's see a demo using the sibc python-library
6. Hard problem
   - Alice side: Given E and $E/\langle P_a + [\mathrm{sk}_a]Q_a\rangle$, to find the 2^{e_2}-isogeny with kernel $\langle P_a + [\mathrm{sk}_a]Q_a\rangle$
   - Bob side: Given E and $E/\langle P_b + [\mathrm{sk}_b]Q_b\rangle$, to find the 3^{e_3}-isogeny with kernel $\langle P_b + [\mathrm{sk}_b]Q_b\rangle$

SI[DH/KE] key space (remarks)

1. What are the private keys? integers of $\frac{\log_2(p)}{2}$ bits
2. What are the public keys? three x-coordinates of $2\log_2(p)$ bits: total of $6\log_2(p)$
3. What is the shared secret? an $\mathbb{F}_{p^2}$-element of $2\log_2(p)$ bits
4. There are ways to reduce the public-key sizes (not presented in this talk)
5. Let's see a demo using the sibc python-library
6. Hard problem
   - Alice side: Given E and $E/\langle P_a + [\mathrm{sk}_a]Q_a\rangle$, to find the 2^{e_2}-isogeny with kernel $\langle P_a + [\mathrm{sk}_a]Q_a\rangle$
   - Bob side: Given E and $E/\langle P_b + [\mathrm{sk}_b]Q_b\rangle$, to find the 3^{e_3}-isogeny with kernel $\langle P_b + [\mathrm{sk}_b]Q_b\rangle$

SI[DH/KE] key space (remarks)

1. What are the private keys? integers of $\frac{\log_2(p)}{2}$ bits
2. What are the public keys? three x-coordinates of $2\log_2(p)$ bits: total of $6\log_2(p)$
3. What is the shared secret? an $\mathbb{F}_{p^2}$-element of $2\log_2(p)$ bits
4. There are ways to reduce the public-key sizes (not presented in this talk)
5. Let's see a demo using the sibc python-library
6. Hard problem
   - Alice side: Given E and $E/\langle P_a + [\mathrm{sk}_a]Q_a\rangle$, to find the 2^{e_2}-isogeny with kernel $\langle P_a + [\mathrm{sk}_a]Q_a\rangle$
   - Bob side: Given E and $E/\langle P_b + [\mathrm{sk}_b]Q_b\rangle$, to find the 3^{e_3}-isogeny with kernel $\langle P_b + [\mathrm{sk}_b]Q_b\rangle$

1. What are the private keys? integers of $\frac{\log_2(p)}{2}$ bits
2. What are the public keys? three x-coordinates of $2\log_2(p)$ bits: total of $6\log_2(p)$
3. What is the shared secret? an $\mathbb{F}_{p^2}$-element of $2\log_2(p)$ bits
4. There are ways to reduce the public-key sizes (not presented in this talk)
5. Let's see a demo using the sibc python-library
6. Hard problem
   - Alice side: Given E and $E/\langle P_a + [\mathrm{sk}_a]Q_a\rangle$, to find the 2^{e_2}-isogeny with kernel $\langle P_a + [\mathrm{sk}_a]Q_a\rangle$
   - Bob side: Given E and $E/\langle P_b + [\mathrm{sk}_b]Q_b\rangle$, to find the 3^{e_3}-isogeny with kernel $\langle P_b + [\mathrm{sk}_b]Q_b\rangle$

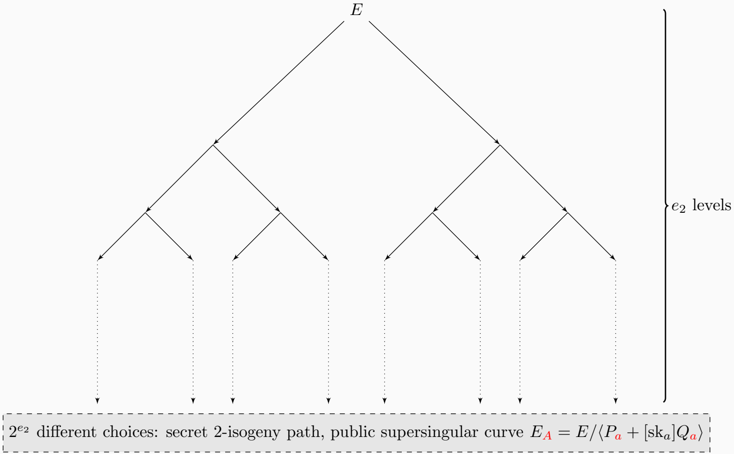
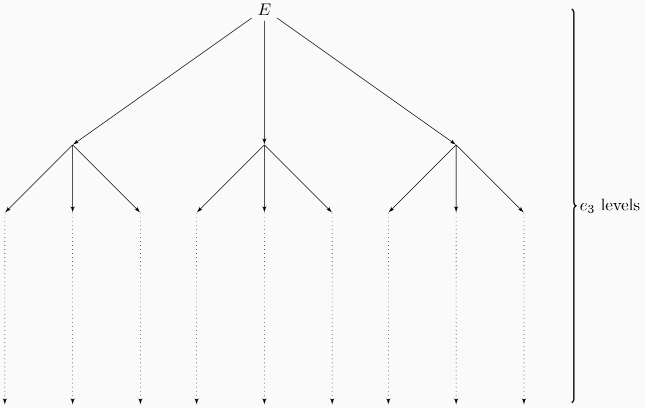


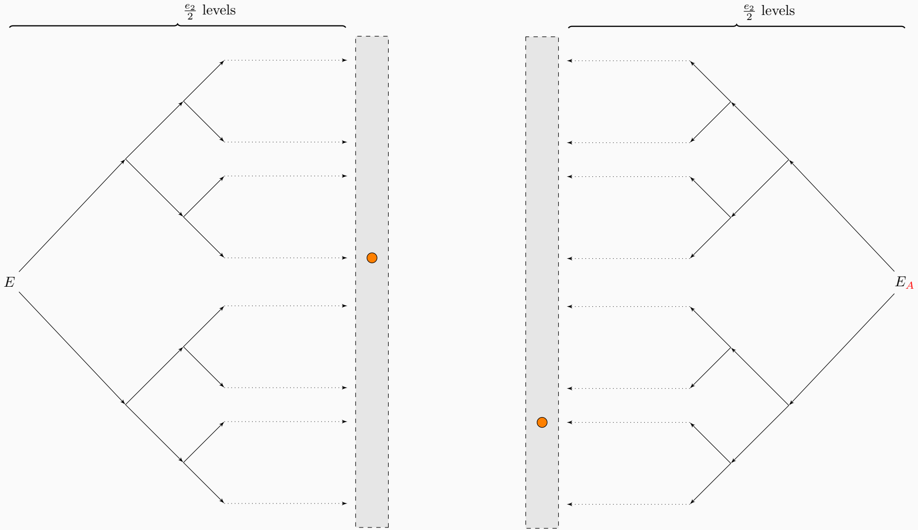
Figure: 2^{e_2}-isogeny tree with root $E/\mathbb{F}_{p^2} : y^2 = x^3 + Ax^2 + x$ having $E[2^{e_2}] = \langle P_a, Q_a \rangle$. Edges describe 2-isogenies.



3^{e_3} different choices: secret 3-isogeny path, public supersingular curve $E_B = E/\langle P_b + [\mathrm{sk}_b]Q_b \rangle$

Figure: 3^{e_3}-isogeny tree with root $E/\mathbb{F}_{p^2} : y^2 = x^3 + Ax^2 + x$ having $E[3^{e_3}] = \langle P_b, Q_b \rangle$. Edges describe 3-isogenies.

SI[DH/KE] key space Alice side (MITM-based attack)




$\frac{e_3}{2}$ levels
$\frac{e_3}{2}$ levels
E
E_B

1. How much memory does MITM require?
2. What is the MITM running time?
3. Can you implement MITM using the sibc python-library?

1. How much memory does MITM require? $2^{\frac{e_2}{2}} \approx p^{1/4}$ cells of memory
2. What is the MITM running time?
3. Can you implement MITM using the sibc python-library?

1. How much memory does MITM require? $2^{\frac{e_2}{2}} \approx p^{1/4}$ cells of memory
2. What is the MITM running time?
3. Can you implement MITM using the sibc python-library?

1. How much memory does MITM require? $2^{\frac{e_2}{2}} \approx p^{1/4}$ cells of memory
2. What is the MITM running time? $1.5 \times 2^{\frac{e_2}{2}} \approx 1.5 \times p^{1/4}$ (in average)
3. Can you implement MITM using the sibc python-library?

1. How much memory does MITM require? $2^{\frac{e_2}{2}} \approx p^{1/4}$ cells of memory
2. What is the MITM running time? $1.5 \times 2^{\frac{e_2}{2}} \approx 1.5 \times p^{1/4}$ (in average)
3. Can you implement MITM using the sibc python-library?

Any questions?

Thanks for attending!

For further questions, contact me by email: *jesus.dominguez@tii.ae*

References I

[1] Craig Costello and Benjamin Smith.
Montgomery curves and their arithmetic - the case of large characteristic fields.
J. Cryptogr. Eng., 8(3):227–240, 2018.

[2] Gora Adj, Jesús-Javier Chi-Domínguez, and Francisco Rodríguez-Henríquez.
SIBC python library.
`https://github.com/JJChiDguez/sibc/`, 2021.

[3] Reza Azarderakhsh, Matthew Campagna, Craig Costello, Luca De Feo, Basil Hess, Aaron Hutchinson, Amir Jalali, David Jao, Koray Karabina, Brian Koziel, Brian LaMacchia, Patrick Longa, Michael Naehrig, Geovandro Pereira, Joost Renes, Vladimir Soukharev, and David Urbanik.
SIDH v3.4 (C Edition).
`https://github.com/microsoft/PQCrypto-SIDH`, 2021.
Online; accessed 9 June 2021.

[4] Reza Azarderakhsh, Matthew Campagna, Craig Costello, Luca De Feo, Basil Hess, Aaron Hutchinson, Amir Jalali, David Jao, Koray Karabina, Brian Koziel, Brian LaMacchia, Patrick Longa, Michael Naehrig, Geovandro Pereira, Joost Renes, Vladimir Soukharev, and David Urbanik.
Supersingular Isogeny Key Encapsulation.
`https://sike.org/files/SIDH-spec.pdf`, 2021.
Online; accessed 9 June 2021.