

Protocols: continued

Chloe Martindale

Technische Universiteit Eindhoven

Birmingham, UK, 16 September 2019

Slides at www.martindale.info/talks

Signatures 1/4

Application 1 of (C)SIDH: Digital signatures.

Signer

Verifier

msg

Signatures 1/4

Application 1 of (C)SIDH: Digital signatures.

Signer

Verifier

msg

$(\text{sk}, \text{pk}) \leftarrow \text{KeyGen}$

Signatures 1/4

Application 1 of (C)SIDH: Digital signatures.

Signer

Verifier

msg

$(\text{sk}, \text{pk}) \leftarrow \text{KeyGen}$

$\sigma = \text{Sign}(\text{sk}, \text{msg}) \xrightarrow{\sigma, \text{pk}} \text{Verify}(\text{pk}, \text{msg}, \sigma)$

Signatures 2/4

One way to build signatures: **Identification scheme** (simplified here)

Prover

Verifier

$\xrightarrow{\text{KeyGen}} (\text{sk}, \text{pk})$

$\text{sk}, \text{ran}_1 \xrightarrow{\text{magic}} \text{ID}$

$\text{sk}, \text{ID}, c, \text{ran}_2 \xrightarrow{\text{magic}} \text{ID-c}$

c

compute challenge c

$(\text{pk}, \text{ID}, \text{ID-c})$

$\text{Verify}(\text{pk}, \text{ID}, c, \text{ID-c})$

Signatures 2/4

One way to build signatures: **Identification scheme** (simplified here)

Signer

Verifier

msg, H

$\xrightarrow{\text{KeyGen}} (\text{sk}, \text{pk})$

$\text{sk}, \text{ran}_1 \xrightarrow{\text{magic}} \text{ID}$

$c := H(\text{ID} || \text{msg})$

$\text{sk}, \text{ID}, c, \text{ran}_2 \xrightarrow{\text{magic}} \text{ID-c}$

$(\text{pk}, \text{ID}, \text{ID-c}) \rightarrow \text{Verify}(\text{pk}, \text{ID}, H(\text{ID} || \text{msg}), \text{ID-c})$

Signatures 2/4

One way to build signatures: **Identification scheme** (simplified here)

Prover

Verifier

$\xrightarrow{\text{KeyGen}} (\text{sk}, \text{pk})$

$\text{sk}, \text{ran}_1 \xrightarrow{\text{magic}} \text{ID}$

$\text{sk}, \text{ID}, c, \text{ran}_2 \xrightarrow{\text{magic}} \text{ID-c}$

c

compute challenge c

$(\text{pk}, \text{ID}, \text{ID-c})$

$\text{Verify}(\text{pk}, \text{ID}, c, \text{ID-c})$

Signatures 3/4

One way to build signatures: **Isogeny identification scheme**

(Stolbunov; SeaSign: De Feo, Galbraith).

Prover

Public

Verifier

$E, \mathcal{O} = \text{End}(E),$

ideals $\mathfrak{l}_1, \dots, \mathfrak{l}_n \in \text{cl}(\mathcal{O})$

Signatures 3/4

One way to build signatures: **Isogeny identification scheme**

(Stolbunov; SeaSign: De Feo, Galbraith).

Prover

Public

Verifier

$$E, \mathcal{O} = \text{End}(E),$$

ideals $\mathfrak{l}_1, \dots, \mathfrak{l}_n \in \text{cl}(\mathcal{O})$

$$e_1, \dots, e_n \in \mathbb{Z}_{[-B, B]}$$

$$\xrightarrow{\text{KeyGen}} ([\mathbf{a}] = [\prod \mathfrak{l}_i^{e_i}], [\mathbf{a}] * E)$$

Signatures 3/4

One way to build signatures: **Isogeny identification scheme**
(Stolbunov; SeaSign: De Feo, Galbraith).

Prover

Public

Verifier

$$E, \mathcal{O} = \text{End}(E),$$

$$\text{ideals } \mathfrak{l}_1, \dots, \mathfrak{l}_n \in \text{cl}(\mathcal{O})$$

$$e_1, \dots, e_n \in \mathbb{Z}_{[-B, B]}$$

$$\xrightarrow{\text{KeyGen}} ([\mathbf{a}] = [\prod \mathfrak{l}_i^{e_i}], [\mathbf{a}] * E)$$

$$\text{random } f_1, \dots, f_n \in \mathbb{Z}_{[-B, B]}$$

$$\xrightarrow{\text{KeyGen}} \begin{cases} [\mathbf{b}] = [\prod \mathfrak{l}_i^{f_i}], \\ \text{ID} = [\mathbf{b}] * E \end{cases}$$

$$\text{ID-c} = \mathbf{ba}^{-c}$$

c

random $c \in \{0, 1\}$

Signatures 3/4

One way to build signatures: **Isogeny identification scheme**

(Stolbunov; SeaSign: De Feo, Galbraith).

Prover

Public

Verifier

$$E, \mathcal{O} = \text{End}(E),$$

$$\text{ideals } \mathfrak{l}_1, \dots, \mathfrak{l}_n \in \text{cl}(\mathcal{O})$$

$$e_1, \dots, e_n \in \mathbb{Z}_{[-B, B]}$$

$$\xrightarrow{\text{KeyGen}} ([\mathbf{a}] = [\prod \mathfrak{l}_i^{e_i}], [\mathbf{a}] * E)$$

$$\text{random } f_1, \dots, f_n \in \mathbb{Z}_{[-B, B]}$$

$$\xrightarrow{\text{KeyGen}} \begin{cases} [\mathbf{b}] = [\prod \mathfrak{l}_i^{f_i}], \\ \text{ID} = [\mathbf{b}] * E \end{cases}$$

$$\text{ID-c} = \mathbf{ba}^{-c}$$

$$([\mathbf{a}] * E, \text{ID}, \text{ID-c})$$

random $c \in \{0, 1\}$

check that

$$\text{ID} \cong \text{ID-c} * ([\mathbf{a}^c] * E)$$

Signatures 3/4

One way to build signatures: **Isogeny identification scheme**

(Stolbunov; SeaSign: De Feo, Galbraith).

Prover

Public

Verifier

$$E, \mathcal{O} = \text{End}(E),$$

$$\text{ideals } \mathfrak{l}_1, \dots, \mathfrak{l}_n \in \text{cl}(\mathcal{O})$$

$$e_1, \dots, e_n \in \mathbb{Z}_{[-B, B]}$$

$$\xrightarrow{\text{KeyGen}} ([\mathbf{a}] = [\prod \mathfrak{l}_i^{e_i}], [\mathbf{a}] * E)$$

$$\text{random } f_1, \dots, f_n \in \mathbb{Z}_{[-B, B]}$$

$$\xrightarrow{\text{KeyGen}} \begin{cases} [\mathbf{b}] = [\prod \mathfrak{l}_i^{f_i}], \\ \text{ID} = [\mathbf{b}] * E \end{cases}$$

$$\text{ID-c} = \mathbf{ba}^{-c}$$

$$([\mathbf{a}] * E, \text{ID}, \text{ID-c})$$

check that

$$\text{ID} \cong \text{ID-c} * ([\mathbf{a}^c] * E)$$

... after k challenges c , an imposter prover succeeds with probability 2^{-k} .

Signatures 4/4

- ▶ Big question: how do you communicate $ID-c = \mathbf{ba}^{-1}$ without leaking \mathbf{a} ?

Signatures 4/4

- ▶ Big question: how do you communicate $\text{ID-c} = \mathbf{ba}^{-1}$ without leaking \mathbf{a} ?
- ▶ Big answer: in terms of generators of class group $\text{cl}(\mathcal{O})$.

Signatures 4/4

- ▶ Big question: how do you communicate $ID-c = \mathbf{ba}^{-1}$ without leaking \mathbf{a} ?
- ▶ Big answer: in terms of generators of class group $cl(\mathcal{O})$.
- ▶ CSI-FiSh (Beullens, Kleinjung, Vercauteren): computed the class group $cl(\mathcal{O})$ for CSIDH-512:

Signatures 4/4

- ▶ Big question: how do you communicate $ID-c = \mathbf{ba}^{-1}$ without leaking \mathbf{a} ?
- ▶ Big answer: in terms of generators of class group $\text{cl}(\mathcal{O})$.
- ▶ CSI-FiSh (Beullens, Kleinjung, Vercauteren): computed the class group $\text{cl}(\mathcal{O})$ for CSIDH-512:
 - ▶ $\text{cl}(\mathcal{O})$ is cyclic.

Signatures 4/4

- ▶ Big question: how do you communicate $ID-c = \mathbf{ba}^{-1}$ without leaking \mathbf{a} ?
- ▶ Big answer: in terms of generators of class group $cl(\mathcal{O})$.
- ▶ CSI-FiSh (Beullens, Kleinjung, Vercauteren): computed the class group $cl(\mathcal{O})$ for CSIDH-512:
 - ▶ $cl(\mathcal{O})$ is cyclic.
 - ▶ $cl(\mathcal{O})$ is generated by an ideal of norm 3.

Signatures 4/4

- ▶ Big question: how do you communicate $ID-c = \mathbf{ba}^{-1}$ without leaking \mathbf{a} ?
- ▶ Big answer: in terms of generators of class group $cl(\mathcal{O})$.
- ▶ CSI-FiSh (Beullens, Kleinjung, Vercauteren): computed the class group $cl(\mathcal{O})$ for CSIDH-512:
 - ▶ $cl(\mathcal{O})$ is cyclic.
 - ▶ $cl(\mathcal{O})$ is generated by an ideal of norm 3.
 - ▶ Elements of $cl(\mathcal{O})$ easy to represent.

Signatures 4/4

- ▶ Big question: how do you communicate $ID-c = \mathbf{ba}^{-1}$ without leaking \mathbf{a} ?
- ▶ Big answer: in terms of generators of class group $cl(\mathcal{O})$.
- ▶ CSI-FiSh (Beullens, Kleinjung, Vercauteren): computed the class group $cl(\mathcal{O})$ for CSIDH-512:
 - ▶ $cl(\mathcal{O})$ is cyclic.
 - ▶ $cl(\mathcal{O})$ is generated by an ideal of norm 3.
 - ▶ Elements of $cl(\mathcal{O})$ easy to represent.
 - ▶ CSI-FiSh signatures take $\approx 390\text{ms}/263\text{B}$.

Signatures 4/4

- ▶ Big question: how do you communicate $ID-c = \mathbf{ba}^{-1}$ without leaking \mathbf{a} ?
- ▶ Big answer: in terms of generators of class group $cl(\mathcal{O})$.
- ▶ CSI-FiSh (Beullens, Kleinjung, Vercauteren): computed the class group $cl(\mathcal{O})$ for CSIDH-512:
 - ▶ $cl(\mathcal{O})$ is cyclic.
 - ▶ $cl(\mathcal{O})$ is generated by an ideal of norm 3.
 - ▶ Elements of $cl(\mathcal{O})$ easy to represent.
 - ▶ CSI-FiSh signatures take $\approx 390\text{ms}/263\text{B}$.
- ▶ For higher security levels (NIST 3, 5), computing the entire class group become impractical.

Signatures 4/4

- ▶ Big question: how do you communicate $ID-c = \mathbf{ba}^{-1}$ without leaking \mathbf{a} ?
- ▶ Big answer: in terms of generators of class group $cl(\mathcal{O})$.
- ▶ CSI-FiSh (Beullens, Kleinjung, Vercauteren): computed the class group $cl(\mathcal{O})$ for CSIDH-512:
 - ▶ $cl(\mathcal{O})$ is cyclic.
 - ▶ $cl(\mathcal{O})$ is generated by an ideal of norm 3.
 - ▶ Elements of $cl(\mathcal{O})$ easy to represent.
 - ▶ CSI-FiSh signatures take $\approx 390\text{ms}/263\text{B}$.
- ▶ For higher security levels (NIST 3, 5), computing the entire class group become impractical.
- ▶ For SIDH, more complicated as keys cannot be reused; class group computation much harder
 \rightsquigarrow signatures take $\approx 3.7\text{s}/141\text{KB}$.

Verifiable Delay Functions: Slide 1/5

Application 2 of (C)SIDH: VDFs.

A **Verifiable Delay Function** (Boneh, Bonneau, Bünz, Fisch) is a function $f : X \rightarrow Y$ that:

Verifiable Delay Functions: Slide 1/5

Application 2 of (C)SIDH: VDFs.

A **Verifiable Delay Function** (Boneh, Bonneau, Bünz, Fisch) is a function $f : X \rightarrow Y$ that:

- ▶ Is computed in n **sequential steps**, each of which takes time t . The total time $T = nt$ is the **delay factor**.

Verifiable Delay Functions: Slide 1 / 5

Application 2 of (C)SIDH: VDFs.

A **Verifiable Delay Function** (Boneh, Bonneau, Bünz, Fisch) is a function $f : X \rightarrow Y$ that:

- ▶ Is computed in n **sequential steps**, each of which takes time t . The total time $T = nt$ is the **delay factor**.
- ▶ Example: repeated hashing
 $s \rightarrow H(s) \rightarrow H(H(s)) \rightarrow \dots \rightarrow H^{(n)}(s)$.

Verifiable Delay Functions: Slide 1 / 5

Application 2 of (C)SIDH: VDFs.

A **Verifiable Delay Function** (Boneh, Bonneau, Bünz, Fisch) is a function $f : X \rightarrow Y$ that:

- ▶ Is computed in n **sequential steps**, each of which takes time t . The total time $T = nt$ is the **delay factor**.
 - ▶ Example: repeated hashing
 $s \rightarrow H(s) \rightarrow H(H(s)) \rightarrow \dots \rightarrow H^{(n)}(s)$.
- ▶ Cannot be computed in time faster than T , even given unlimited resources.

Verifiable Delay Functions: Slide 1 / 5

Application 2 of (C)SIDH: VDFs.

A **Verifiable Delay Function** (Boneh, Bonneau, Bünz, Fisch) is a function $f : X \rightarrow Y$ that:

- ▶ Is computed in n **sequential steps**, each of which takes time t . The total time $T = nt$ is the **delay factor**.
 - ▶ Example: repeated hashing
 $s \rightarrow H(s) \rightarrow H(H(s)) \rightarrow \dots \rightarrow H^{(n)}(s)$.
- ▶ Cannot be computed in time faster than T , even given unlimited resources.
- ▶ The correctness of the output can be **quickly verified**.

Verifiable Delay Functions: Slide 1 / 5

Application 2 of (C)SIDH: VDFs.

A **Verifiable Delay Function** (Boneh, Bonneau, Bünz, Fisch) is a function $f : X \rightarrow Y$ that:

- ▶ Is computed in n **sequential steps**, each of which takes time t . The total time $T = nt$ is the **delay factor**.
 - ▶ Example: repeated hashing
 $s \rightarrow H(s) \rightarrow H(H(s)) \rightarrow \dots \rightarrow H^{(n)}(s)$.
- ▶ Cannot be computed in time faster than T , even given unlimited resources.
- ▶ The correctness of the output can be **quickly verified**.
 - ▶ Non-example: repeated hashing.

Verifiable Delay Functions: Slide 2/5

One way to build VDFs:

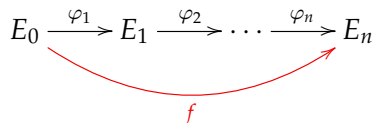
Verifiable Delay Functions: Slide 2/5

One way to build VDFs: **Isogenies!** (De Feo, Masson, Petit, Sanso)

Verifiable Delay Functions: Slide 2/5

One way to build VDFs: **Isogenies!** (De Feo, Masson, Petit, Sanso)

- ▶ Natural sequential function f : compose ℓ -isogenies φ_i

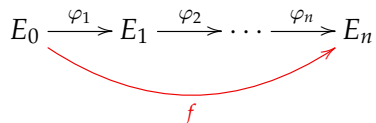
$$E_0 \xrightarrow{\varphi_1} E_1 \xrightarrow{\varphi_2} \dots \xrightarrow{\varphi_n} E_n$$


The diagram illustrates a sequence of elliptic curves E_0, E_1, \dots, E_n connected by isogenies $\varphi_1, \varphi_2, \dots, \varphi_n$. A red curved arrow labeled f points from E_0 to E_n , representing the composition of all isogenies.

Verifiable Delay Functions: Slide 2/5

One way to build VDFs: **Isogenies!** (De Feo, Masson, Petit, Sanso)

- ▶ Natural sequential function f : compose ℓ -isogenies φ_i

$$E_0 \xrightarrow{\varphi_1} E_1 \xrightarrow{\varphi_2} \dots \xrightarrow{\varphi_n} E_n$$


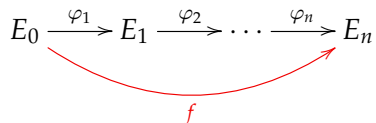
The diagram illustrates a sequence of elliptic curves E_0, E_1, \dots, E_n connected by isogenies $\varphi_1, \varphi_2, \dots, \varphi_n$. A red curved arrow labeled f points from E_0 to E_n , representing the composition of all isogenies.

- ▶ How to **quickly** verify correctness of the output?

Verifiable Delay Functions: Slide 2/5

One way to build VDFs: **Isogenies!** (De Feo, Masson, Petit, Sanso)

- ▶ Natural sequential function f : compose ℓ -isogenies φ_i

$$E_0 \xrightarrow{\varphi_1} E_1 \xrightarrow{\varphi_2} \dots \xrightarrow{\varphi_n} E_n$$


The diagram illustrates a sequence of elliptic curves E_0, E_1, \dots, E_n connected by isogenies $\varphi_1, \varphi_2, \dots, \varphi_n$. A red curved arrow labeled f points from E_0 to E_n , representing the composition of all isogenies.

- ▶ How to **quickly** verify correctness of the output? **Pairings**.

Interlude: Pairings 1/4

Let $(N, p) = 1$, fix any basis $E[N] = \langle R, S \rangle$.

(Slide stolen shamelessly from Luca De Feo)

Interlude: Pairings 1/4

Let $(N, p) = 1$, fix any basis $E[N] = \langle R, S \rangle$.

For any points $P, Q \in E[N]$ there exist $a, b, c, d \in \mathbb{Z}/N\mathbb{Z}$ such that

$$P = aR + bS$$

$$Q = cR + dS.$$

(Slide stolen shamelessly from Luca De Feo)

Interlude: Pairings 1/4

Let $(N, p) = 1$, fix any basis $E[N] = \langle R, S \rangle$.

For any points $P, Q \in E[N]$ there exist $a, b, c, d \in \mathbb{Z}/N\mathbb{Z}$ such that

$$P = aR + bS$$

$$Q = cR + dS.$$

The form

$$\det_N(P, Q) = \det \begin{pmatrix} a & b \\ c & d \end{pmatrix} = ad - bc \in \mathbb{Z}/n\mathbb{Z}$$

is **bilinear, non-degenerate, and independent from the choice of basis.**

(Slide stolen shamelessly from Luca De Feo)

Interlude: Pairings 2/4

Theorem

Let E/\mathbb{F}_q be a curve. There exists a Galois invariant bilinear map

$$e : E[N] \times E[N] \rightarrow \mu_N \subseteq \overline{\mathbb{F}_q},$$

called the *Weil pairing of order N* , and a primitive N -th root of unity $\zeta \in \overline{\mathbb{F}_q}$ such that

$$e(P, Q) = \zeta^{\det_N(P, Q)}.$$

(Slide stolen shamelessly from Luca De Feo)

Interlude: Pairings 2/4

Theorem

Let E/\mathbb{F}_q be a curve. There exists a Galois invariant bilinear map

$$e : E[N] \times E[N] \rightarrow \mu_N \subseteq \overline{\mathbb{F}_q},$$

called the *Weil pairing of order N* , and a primitive N -th root of unity $\zeta \in \overline{\mathbb{F}_q}$ such that

$$e(P, Q) = \zeta^{\text{det}_N(P, Q)}.$$

The degree k of the smallest extension such that $\zeta \in \mathbb{F}_{q^k}$ is called the *embedding degree of the pairing*.

(Slide stolen shamelessly from Luca De Feo)

Interlude: Pairings 3/4

For any elliptic curve E/\mathbb{F}_q , we have the Weil pairing

$$e : E[N] \times E[N] \rightarrow \mu_N \subseteq \overline{\mathbb{F}_q}.$$

Interlude: Pairings 3/4

For any elliptic curve E/\mathbb{F}_q , we have the Weil pairing

$$e : E[N] \times E[N] \rightarrow \mu_N \subseteq \overline{\mathbb{F}_q}.$$

Can think of it as a **map of groups**

$$e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_3.$$

Interlude: Pairings 3/4

For any elliptic curve E/\mathbb{F}_q , we have the Weil pairing

$$e : E[N] \times E[N] \rightarrow \mu_N \subseteq \overline{\mathbb{F}_q}.$$

Can think of it as a **map of groups**

$$e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_3.$$

There exist **efficiently computable** pairings with:

- ▶ $\mathbb{G}_1 \subseteq E(\mathbb{F}_q)$ of prime order.

Interlude: Pairings 3/4

For any elliptic curve E/\mathbb{F}_q , we have the Weil pairing

$$e : E[N] \times E[N] \rightarrow \mu_N \subseteq \overline{\mathbb{F}_q}.$$

Can think of it as a **map of groups**

$$e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_3.$$

There exist **efficiently computable** pairings with:

- ▶ $\mathbb{G}_1 \subseteq E(\mathbb{F}_q)$ of prime order.
- ▶ $\mathbb{G}_2 \subseteq E(\mathbb{F}_{q^k})$ of prime order
(remember k is the embedding degree).

Interlude: Pairings 3/4

For any elliptic curve E/\mathbb{F}_q , we have the Weil pairing

$$e : E[N] \times E[N] \rightarrow \mu_N \subseteq \overline{\mathbb{F}_q}.$$

Can think of it as a **map of groups**

$$e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_3.$$

There exist **efficiently computable** pairings with:

- ▶ $\mathbb{G}_1 \subseteq E(\mathbb{F}_q)$ of prime order.
- ▶ $\mathbb{G}_2 \subseteq E(\mathbb{F}_{q^k})$ of prime order
(remember k is the embedding degree).
- ▶ $\mathbb{G}_3 \subseteq \mathbb{F}_{q^k}^*$ of prime order.

Interlude: Pairings 4/4

- ▶ Let E/\mathbb{F}_q and E'/\mathbb{F}_q be elliptic curves with the same embedding degree k .

Interlude: Pairings 4/4

- ▶ Let E/\mathbb{F}_q and E'/\mathbb{F}_q be elliptic curves with the same embedding degree k .
- ▶ Let $f : E \rightarrow E'$ be an isogeny with dual $\hat{f} : E' \rightarrow E$.

Interlude: Pairings 4/4

- ▶ Let E/\mathbb{F}_q and E'/\mathbb{F}_q be elliptic curves with the same embedding degree k .
- ▶ Let $f : E \rightarrow E'$ be an isogeny with dual $\hat{f} : E' \rightarrow E$.
- ▶ Let $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_3$ and $e' : \mathbb{G}'_1 \times \mathbb{G}'_2 \rightarrow \mathbb{G}_3$ be pairings on E and E' respectively.

Interlude: Pairings 4/4

- ▶ Let E/\mathbb{F}_q and E'/\mathbb{F}_q be elliptic curves with the same embedding degree k .
- ▶ Let $f : E \rightarrow E'$ be an isogeny with dual $\hat{f} : E' \rightarrow E$.
- ▶ Let $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_3$ and $e' : \mathbb{G}'_1 \times \mathbb{G}'_2 \rightarrow \mathbb{G}_3$ be pairings on E and E' respectively.
- ▶ Then we get a commutative diagram:

$$\begin{array}{ccc} \mathbb{G}_1 \times \mathbb{G}'_2 & \xrightarrow{f \times 1} & \mathbb{G}'_1 \times \mathbb{G}'_2 \\ 1 \times \hat{f} \downarrow & & \downarrow e' \\ \mathbb{G}_1 \times \mathbb{G}_2 & \xrightarrow{e} & \mathbb{G}_3 \end{array}$$

Interlude: Pairings 4/4

- ▶ Let E/\mathbb{F}_q and E'/\mathbb{F}_q be elliptic curves with the same embedding degree k .
- ▶ Let $f : E \rightarrow E'$ be an isogeny with dual $\hat{f} : E' \rightarrow E$.
- ▶ Let $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_3$ and $e' : \mathbb{G}'_1 \times \mathbb{G}'_2 \rightarrow \mathbb{G}_3$ be pairings on E and E' respectively.
- ▶ Then we get a commutative diagram:

$$\begin{array}{ccc} \mathbb{G}_1 \times \mathbb{G}'_2 & \xrightarrow{f \times 1} & \mathbb{G}'_1 \times \mathbb{G}'_2 \\ 1 \times \hat{f} \downarrow & & \downarrow e' \\ \mathbb{G}_1 \times \mathbb{G}_2 & \xrightarrow{e} & \mathbb{G}_3 \end{array}$$

- ▶ For $P \in \mathbb{G}_1$ and $Q \in \mathbb{G}'_2$:

$$e(P, \hat{f}(Q)) = e'(f(P), Q).$$

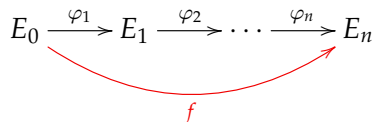
Verifiable Delay Functions: Slide 3/5

Protocol - setup and evaluation:

Verifiable Delay Functions: Slide 3/5

Protocol - setup and evaluation:

- ▶ Compute the composition of ℓ -isogenies

$$E_0 \xrightarrow{\varphi_1} E_1 \xrightarrow{\varphi_2} \dots \xrightarrow{\varphi_n} E_n$$


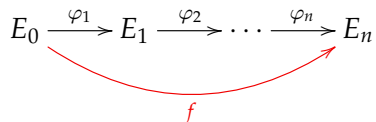
The diagram illustrates a sequence of elliptic curves E_0, E_1, \dots, E_n connected by isogenies $\varphi_1, \varphi_2, \dots, \varphi_n$. A red curved arrow labeled f connects E_0 and E_n , representing the composition of all isogenies.

and the dual \hat{f} .

Verifiable Delay Functions: Slide 3/5

Protocol - setup and evaluation:

- ▶ Compute the composition of ℓ -isogenies

$$E_0 \xrightarrow{\varphi_1} E_1 \xrightarrow{\varphi_2} \dots \xrightarrow{\varphi_n} E_n$$


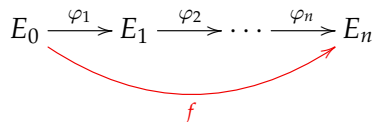
and the dual \hat{f} .

- ▶ Publish f, \hat{f} , groups $\mathbb{G}_1, \mathbb{G}_2 \subseteq E_0$, groups $\mathbb{G}'_1, \mathbb{G}'_2 \subseteq E_n$, pairings e and e' , a generator P of \mathbb{G}_1 , and $f(P)$.

Verifiable Delay Functions: Slide 3/5

Protocol - setup and evaluation:

- ▶ Compute the composition of ℓ -isogenies

$$E_0 \xrightarrow{\varphi_1} E_1 \xrightarrow{\varphi_2} \dots \xrightarrow{\varphi_n} E_n$$


and the dual \hat{f} .

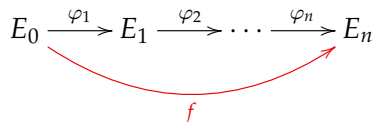
- ▶ Publish f, \hat{f} , groups $\mathbb{G}_1, \mathbb{G}_2 \subseteq E_0$, groups $\mathbb{G}'_1, \mathbb{G}'_2 \subseteq E_n$, pairings e and e' , a generator P of \mathbb{G}_1 , and $f(P)$.

Protocol - verify:

Verifiable Delay Functions: Slide 3/5

Protocol - setup and evaluation:

- ▶ Compute the composition of ℓ -isogenies

$$E_0 \xrightarrow{\varphi_1} E_1 \xrightarrow{\varphi_2} \dots \xrightarrow{\varphi_n} E_n$$


and the dual \hat{f} .

- ▶ Publish f, \hat{f} , groups $\mathbb{G}_1, \mathbb{G}_2 \subseteq E_0$, groups $\mathbb{G}'_1, \mathbb{G}'_2 \subseteq E_n$, pairings e and e' , a generator P of \mathbb{G}_1 , and $f(P)$.

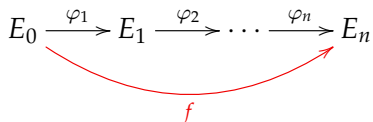
Protocol - verify:

- ▶ Choose $Q \in \mathbb{G}'_2$.

Verifiable Delay Functions: Slide 3/5

Protocol - setup and evaluation:

- ▶ Compute the composition of ℓ -isogenies

$$E_0 \xrightarrow{\varphi_1} E_1 \xrightarrow{\varphi_2} \dots \xrightarrow{\varphi_n} E_n$$


and the dual \hat{f} .

- ▶ Publish f, \hat{f} , groups $\mathbb{G}_1, \mathbb{G}_2 \subseteq E_0$, groups $\mathbb{G}'_1, \mathbb{G}'_2 \subseteq E_n$, pairings e and e' , a generator P of \mathbb{G}_1 , and $f(P)$.

Protocol - verify:

- ▶ Choose $Q \in \mathbb{G}'_2$.
- ▶ Check that $e(P, \hat{f}(Q)) = e'(f(P), Q)$.

Verifiable Delay Functions: Slide 4/5

- ▶ Proposal uses **2-isogenies** of **supersingular** elliptic curves defined over \mathbb{F}_p or \mathbb{F}_{p^2} ; p is a well-chosen 1503-bit prime (for 128-bit security).

Verifiable Delay Functions: Slide 4/5

- ▶ Proposal uses **2-isogenies** of **supersingular** elliptic curves defined over \mathbb{F}_p or \mathbb{F}_{p^2} ; p is a well-chosen 1503-bit prime (for 128-bit security).
- ▶ Over \mathbb{F}_p : Setup takes **238 KB / 1416s**, evaluation takes **2056s**, verification takes **7s**.

Verifiable Delay Functions: Slide 4/5

- ▶ Proposal uses **2-isogenies** of **supersingular** elliptic curves defined over \mathbb{F}_p or \mathbb{F}_{p^2} ; p is a well-chosen 1503-bit prime (for 128-bit security).
- ▶ Over \mathbb{F}_p : Setup takes **238 KB / 1416s**, evaluation takes **2056s**, verification takes **7s**.
- ▶ Over \mathbb{F}_{p^2} : Setup takes **491KB / 2727s**, evaluation takes **2817s**, verification takes **7s**.

Verifiable Delay Functions: Slide 5/5

De Feo, Masson, Petit, Sanso give the following comparison of their isogeny VDF with the literature:

VDF	Sequential Eval	Parallel Eval	Verify	Setup	Proof size
Modular square root	T	$T^{2/3}$	$T^{2/3}$	T	—
Univariate permutation polynomials ⁶	T^2	$> T - o(T)$	$\log(T)$	$\log(T)$	—
Wesolowski's VDF	$(1 + \frac{2}{\log(T)})T$	$(1 + \frac{2}{s \log(T)})T$	λ^4	λ^3	λ^3
Pietrzak's VDF	$(1 + \frac{2}{\sqrt{T}})T$	$(1 + \frac{2}{s\sqrt{T}})T$	$\log(T)$	λ^3	$\log(T)$
This work	T	T	λ^4	$T\lambda^3$	—
This work (optimized)	T	T	λ^4	$T \log(\lambda)$	—

Table 1. VDF comparison—Asymptotic VDF comparison: T represents the delay factor, λ the security parameter, s the number of processors. For simplicity, we assume that T is super-polynomial in λ . All times are to be understood up to a (global across a line) constant factor.

Multiparty key exchange: Slide 1/5

Natural question: how efficiently can we do n -party key exchange?

Multiparty key exchange: Slide 1/5

Natural question: how efficiently can we do n -party key exchange?

- ▶ Easy CSIDH-style algorithm with $n = 3$:

Multiparty key exchange: Slide 1/5

Natural question: how efficiently can we do n -party key exchange?

- ▶ Easy CSIDH-style algorithm with $n = 3$:
 - ▶ Alice, Bob, and Chloe compute $([\mathbf{a}], [\mathbf{a}] * E)$, $([\mathbf{b}], [\mathbf{b}] * E)$, and $([\mathbf{c}], [\mathbf{c}] * E)$.

Multiparty key exchange: Slide 1 / 5

Natural question: how efficiently can we do n -party key exchange?

- ▶ Easy CSIDH-style algorithm with $n = 3$:
 - ▶ Alice, Bob, and Chloe compute $([\mathbf{a}], [\mathbf{a}] * E)$, $([\mathbf{b}], [\mathbf{b}] * E)$, and $([\mathbf{c}], [\mathbf{c}] * E)$.
 - ▶ Alice and Bob do a pairwise key exchange and find $AB = [\mathbf{ab}] * E$.

Multiparty key exchange: Slide 1 / 5

Natural question: how efficiently can we do n -party key exchange?

- ▶ Easy CSIDH-style algorithm with $n = 3$:
 - ▶ Alice, Bob, and Chloe compute $([\mathbf{a}], [\mathbf{a}] * E)$, $([\mathbf{b}], [\mathbf{b}] * E)$, and $([\mathbf{c}], [\mathbf{c}] * E)$.
 - ▶ Alice and Bob do a pairwise key exchange and find $AB = [\mathbf{ab}] * E$.
 - ▶ They exchange AB with Chloe and find $[\mathbf{abc}] * E$.

Multiparty key exchange: Slide 1 / 5

Natural question: how efficiently can we do n -party key exchange?

- ▶ Easy CSIDH-style algorithm with $n = 3$:
 - ▶ Alice, Bob, and Chloe compute $([\mathbf{a}], [\mathbf{a}] * E)$, $([\mathbf{b}], [\mathbf{b}] * E)$, and $([\mathbf{c}], [\mathbf{c}] * E)$.
 - ▶ Alice and Bob do a pairwise key exchange and find $AB = [\mathbf{ab}] * E$.
 - ▶ They exchange AB with Chloe and find $[\mathbf{abc}] * E$.
- ▶ This requires 2 key exchanges.

Multiparty key exchange: Slide 1 / 5

Natural question: how efficiently can we do n -party key exchange?

- ▶ Easy CSIDH-style algorithm with $n = 3$:
 - ▶ Alice, Bob, and Chloe compute $([\mathbf{a}], [\mathbf{a}] * E)$, $([\mathbf{b}], [\mathbf{b}] * E)$, and $([\mathbf{c}], [\mathbf{c}] * E)$.
 - ▶ Alice and Bob do a pairwise key exchange and find $AB = [\mathbf{ab}] * E$.
 - ▶ They exchange AB with Chloe and find $[\mathbf{abc}] * E$.
- ▶ This requires 2 key exchanges.
- ▶ The time and memory needed for this basic interactive n -party key exchange **grows with n** .

Multiparty key exchange: Slide 1 / 5

Natural question: how efficiently can we do n -party key exchange?

- ▶ Easy CSIDH-style algorithm with $n = 3$:
 - ▶ Alice, Bob, and Chloe compute $([\mathbf{a}], [\mathbf{a}] * E)$, $([\mathbf{b}], [\mathbf{b}] * E)$, and $([\mathbf{c}], [\mathbf{c}] * E)$.
 - ▶ Alice and Bob do a pairwise key exchange and find $AB = [\mathbf{ab}] * E$.
 - ▶ They exchange AB with Chloe and find $[\mathbf{abc}] * E$.
- ▶ This requires 2 key exchanges.
- ▶ The time and memory needed for this basic interactive n -party key exchange **grows with n** .

Open question: is there something much better for large n ?

Multiparty key exchange: Slide 2/5

Natural question: how efficiently can we do n -party key exchange?

Multiparty key exchange: Slide 2/5

Natural question: how efficiently can we do n -party key exchange?

- ▶ For the same construction with SIKE (Azarderashkhsh, Jalali, Jao, Soukharev), need to change parameter choices.

Multiparty key exchange: Slide 2/5

Natural question: how efficiently can we do n -party key exchange?

- ▶ For the same construction with SIKE (Azarderashkhsh, Jalali, Jao, Soukharev), need to change parameter choices.
- ▶ With three parties, Alice computes a chain of 2-isogenies, Bob 3-isogenies, and Chloe 5-isogenies.

Multiparty key exchange: Slide 2/5

Natural question: how efficiently can we do n -party key exchange?

- ▶ For the same construction with SIKE (Azarderashkhsh, Jalali, Jao, Soukharev), need to change parameter choices.
- ▶ With three parties, Alice computes a chain of 2-isogenies, Bob 3-isogenies, and Chloe 5-isogenies.
- ▶ For base field, take $p = 2^a 3^b 5^c \cdot f \pm 1$.

Multiparty key exchange: Slide 2/5

Natural question: how efficiently can we do n -party key exchange?

- ▶ For the same construction with SIKE (Azarderashkhsh, Jalali, Jao, Soukharev), need to change parameter choices.
- ▶ With three parties, Alice computes a chain of 2-isogenies, Bob 3-isogenies, and Chloe 5-isogenies.
- ▶ For base field, take $p = 2^a 3^b 5^c \cdot f \pm 1$.
- ▶ As n increases, isogeny computations become slower (higher degree) – but not a big problem...

Multiparty key exchange: Slide 3/5

For a **large number** of parties with the SIKE version, there is an **attack** due to Petit (see tomorrow).

Multiparty key exchange: Slide 3/5

For a **large number** of parties with the SIKE version, there is an **attack** due to Petit (see tomorrow).

Open question: what is the largest number of parties for which this is still secure?

Multiparty key exchange: Slide 3/5

For a **large number** of parties with the SIKE version, there is an **attack** due to Petit (see tomorrow).

Open question: what is the largest number of parties for which this is still secure?

Open question: is there something much better for medium-large n ?

Multiparty key exchange: Slide 4/5

The Dream (Boneh, Glass, Krashen, Lauter, Sharif, Silverberg, Tibouchi, Zhandry):

Multiparty key exchange: Slide 4/5

The Dream (Boneh, Glass, Krashen, Lauter, Sharif, Silverberg, Tibouchi, Zhandry):

- ▶ **Awesome fact:** There is an isomorphism of abelian varieties

$$[\mathbf{a}_1] * E \times \cdots \times [\mathbf{a}_n] * E \rightarrow [\mathbf{a}_1 \cdots \mathbf{a}_n] * E \times E^{n-1}.$$

Multiparty key exchange: Slide 4/5

The Dream (Boneh, Glass, Krashen, Lauter, Sharif, Silverberg, Tibouchi, Zhandry):

- ▶ **Awesome fact:** There is an isomorphism of abelian varieties

$$[\mathbf{a}_1] * E \times \cdots \times [\mathbf{a}_n] * E \rightarrow [\mathbf{a}_1 \cdots \mathbf{a}_n] * E \times E^{n-1}.$$

Hard open problem: find an efficiently computable isomorphism invariant for abelian varieties of this form.

Multiparty key exchange: Slide 4/5

The Dream (Boneh, Glass, Krashen, Lauter, Sharif, Silverberg, Tibouchi, Zhandry):

- ▶ **Awesome fact:** There is an isomorphism of abelian varieties

$$[\mathbf{a}_1] * E \times \cdots \times [\mathbf{a}_n] * E \rightarrow [\mathbf{a}_1 \cdots \mathbf{a}_n] * E \times E^{n-1}.$$

Hard open problem: find an efficiently computable isomorphism invariant for abelian varieties of this form.

Consequences:

Multiparty key exchange: Slide 4/5

The Dream (Boneh, Glass, Krashen, Lauter, Sharif, Silverberg, Tibouchi, Zhandry):

- ▶ **Awesome fact:** There is an isomorphism of abelian varieties

$$[\mathbf{a}_1] * E \times \cdots \times [\mathbf{a}_n] * E \rightarrow [\mathbf{a}_1 \cdots \mathbf{a}_n] * E \times E^{n-1}.$$

Hard open problem: find an efficiently computable isomorphism invariant for abelian varieties of this form.

Consequences:

- ▶ Efficient multiparty non-interactive key exchange.
- ▶ Verifiable random functions.
- ▶ World peace.
- ▶ etc.

Dreaming from isogenies: n -way NIKE

- ▶ Choose an elliptic curve E/\mathbb{F}_q .

Dreaming from isogenies: n -way NIKE

- ▶ Choose an elliptic curve E/\mathbb{F}_q .
- ▶ Each party chooses $[\mathbf{a}_i] \in \text{cl}(\text{End}(E))$ and publishes $E_i = [\mathbf{a}_i] * E$.

Dreaming from isogenies: n -way NIKE

- ▶ Choose an elliptic curve E/\mathbb{F}_q .
- ▶ Each party chooses $[\mathbf{a}_i] \in \text{cl}(\text{End}(E))$ and publishes $E_i = [\mathbf{a}_i] * E$.
- ▶ Choose $j \neq i$. The shared secret is the isomorphism invariant of

$$E_1 \times \cdots \times E_{j-1} \times [\mathbf{a}_i] * E_j \times E_{j+1} \times \cdots \times E_n,$$

where E_i is omitted from the product.

Dreaming from isogenies: n -way NIKE

- ▶ Choose an elliptic curve E/\mathbb{F}_q .
- ▶ Each party chooses $[\mathbf{a}_i] \in \text{cl}(\text{End}(E))$ and publishes $E_i = [\mathbf{a}_i] * E$.
- ▶ Choose $j \neq i$. The shared secret is the isomorphism invariant of

$$E_1 \times \cdots \times E_{j-1} \times [\mathbf{a}_i] * E_j \times E_{j+1} \times \cdots \times E_n,$$

where E_i is omitted from the product.

- ▶ Note that by [Awesome Fact](#) this is the isomorphism invariant of $[\prod_{i=1}^n \mathbf{a}_i] * E^{n-1}$.

Thank you!