Highly Vectorized SIKE for AVX-512

Hao Cheng, Georgios Fotiadis, Johann Großschädl, Peter Y. A. Ryan

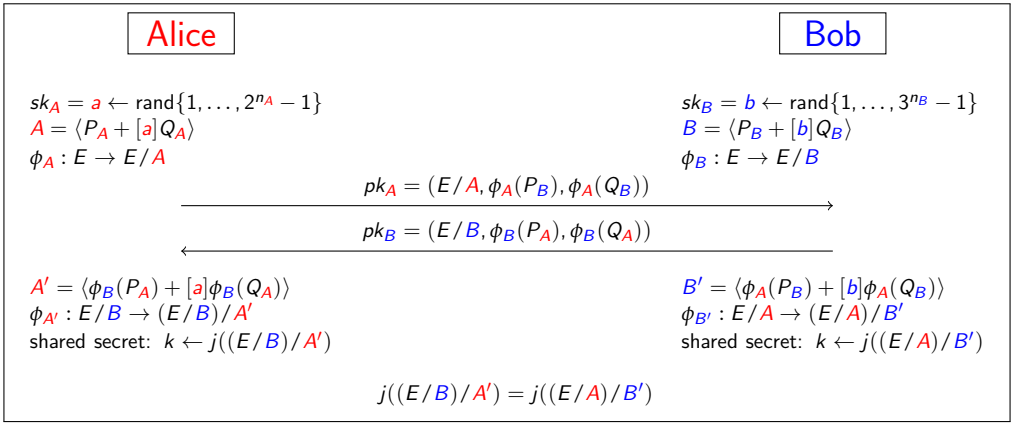
DCS and SnT, University of Luxembourg

CHES 2022

Prologue

- SIDH/SIKE is currently broken due to Castryck-Decru attack [CD22]
- more information regarding security `https://issikebrokenyet.github.io`
- implementation/vectorization techniques here still useful for ECC and other isogeny-based crypto schemes

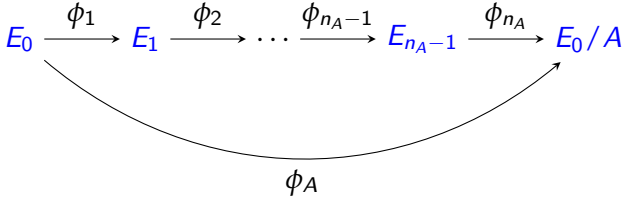
SIDH - **S**upersingular **I**sogeny **D**iffie-**H**ellman



Isogenies in SIDH

Alice: computes isogenies of degree 2^{n_A}, by composing n_A isogenies of degree 2.

- each $\phi_i : E_{i-1} \to E_i$ has $\deg(\phi_i) = 2$, for $i = 1, \ldots, n_A$.
- the isogeny $\phi_A : E_0 \to E_A$ is $\phi_A = \phi_1 \circ \phi_2 \circ \ldots \circ \phi_{n_A}$ and has $\deg(\phi_A) = 2^{n_A}$.



- further speed-up by composing $n_A/2$ isogenies of degree 4.

Bob: computes isogenies of degree 3^{n_B}, by composing n_B isogenies of degree 3.

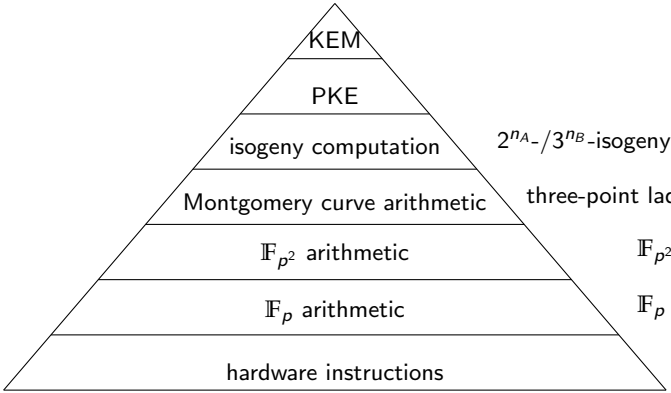
SIKE - **S**upersingular **I**sogeny **K**ey **E**ncapsulation

- SIDH + HHK/FO transformation
- security based on SIDH problem
- short key lengths and high computing costs
- (uncompressed) SIKE and compressed SIKE

AvxSike

- Intel AVX-512 extension, particularly AVX-512IFMA
- Low-Latency implementation AvxSike-LL
- High-Throughput (batched) implementation AvxSike-HT
- supports 4 (uncompressed) parameter sets SIKEp434, SIKEp503, SIKEp610, SIKEp751
- no secret-dependent conditional statements or memory accesses;
  fixed operation/instruction sequence
- source code at `https://gitlab.uni.lu/APSIA/AVXSIKE`
- this talk focuses on low-latency version

SIKE hierarchy



2^{n_A}-/3^{n_B}-isogeny computation and evaluation (optimal strategies)

three-point ladder; xDBL, xTPL; 4-/3-isogeny gen and eval

$\mathbb{F}_{p^2}$ multiply, square, add, subtract ...

$\mathbb{F}_p$ multiply, square, add, subtract ...

AVX-512, particularly IFMA

Hardware layer - AVX-512 and AVX-512IFMA

Single Instruction Multiple Data – SIMD

- 512-bit vectors
- 64-bit element $\times$ 8 lanes

Integer Fused Multiply-Add

- $a \times b \rightarrow t$ $\quad (52 \times 52)$-bit $\rightarrow$ 104-bit
- $t_L / t_H + c \rightarrow r$ $\quad$ 52-bit + 64-bit $\rightarrow$ 64-bit

Intel hardware support: Cannon Lake, Ice Lake, Tiger Lake, Rocket Lake

We target the Intel Ice Lake processor (Sunny Cove microarchitecture)

- Intel Core i3-1005G1 CPU clocked at 1.2 GHz.

We use 51-bit-per-limb representation

- 1-bit headroom is more friendly for Karatsuba

$\mathbb{F}_p$ layer - multiplication

$(y \times z)$-way vectorization

- y $\mathbb{F}_p$ operations
- z lanes used by each $\mathbb{F}_p$ operation

$\mathbb{F}_p$ layer - multiplication

$(y \times z)$-way vectorization

- y $\mathbb{F}_p$ operations
- z lanes used by each $\mathbb{F}_p$ operation

Our implementations (example SIKEp503)

- (8×1)-way Karatsuba $\mathcal{O}(n^{\log_2 3})$

$$\left\{ \begin{matrix} [a_0, b_0, c_0, d_0, e_0, f_0, g_0, h_0] \\ [a_1, b_1, c_1, d_1, e_1, f_1, g_1, h_1] \\ \vdots \\ [a_9, b_9, c_9, d_9, e_9, f_9, g_9, h_9] \end{matrix} \right\}$$
$$< a, b, c, d, e, f, g, h >$$

$\mathbb{F}_p$ layer - multiplication

$(y \times z)$-way vectorization

- y $\mathbb{F}_p$ operations
- z lanes used by each $\mathbb{F}_p$ operation

Our implementations (example SIKEp503)

- (8×1)-way Karatsuba $\mathcal{O}(n^{\log_2 3})$

$$\left\{\begin{array}{c}[a_0, b_0, c_0, d_0, e_0, f_0, g_0, h_0] \\ [a_1, b_1, c_1, d_1, e_1, f_1, g_1, h_1] \\ \vdots \\ [a_9, b_9, c_9, d_9, e_9, f_9, g_9, h_9]\end{array}\right\}$$

$$< a, b, c, d, e, f, g, h >$$

- (4×2)-way schoolbook $\mathcal{O}(n^2)$

$$\left\{\begin{array}{c}[a_0, a_5, b_0, b_5, c_0, c_5, d_0, d_5] \\ [a_1, a_6, b_1, b_6, c_1, c_6, d_1, d_6] \\ \vdots \\ [a_4, a_9, b_4, b_9, c_4, c_9, d_4, d_9]\end{array}\right\}$$

$$< a, b, c, d >$$

$\mathbb{F}_p$ layer - multiplication

$(y \times z)$-way vectorization

- y $\mathbb{F}_p$ operations
- z lanes used by each $\mathbb{F}_p$ operation

Our implementations (example SIKEp503)

- (8×1)-way Karatsuba $\mathcal{O}(n^{\log_2 3})$

$$\left\{\begin{matrix}[a_0, b_0, c_0, d_0, e_0, f_0, g_0, h_0] \\ [a_1, b_1, c_1, d_1, e_1, f_1, g_1, h_1] \\ \vdots \\ [a_9, b_9, c_9, d_9, e_9, f_9, g_9, h_9]\end{matrix}\right\}$$
$$< a, b, c, d, e, f, g, h >$$

- (4×2)-way schoolbook $\mathcal{O}(n^2)$

$$\left\{\begin{matrix}[a_0, a_5, b_0, b_5, c_0, c_5, d_0, d_5] \\ [a_1, a_6, b_1, b_6, c_1, c_6, d_1, d_6] \\ \vdots \\ [a_4, a_9, b_4, b_9, c_4, c_9, d_4, d_9]\end{matrix}\right\}$$
$$< a, b, c, d >$$

- ~~(2×4)-way, (1×8)-way~~

$\mathbb{F}_{p^2}$ layer - multiplication

$$r = r_0 + r_1 i = a \times b = (a_0 + a_1 i) \times (b_0 + b_1 i)$$
$$r = (a_0 b_0 - a_1 b_1) + (a_0 b_1 + a_1 b_0) i \quad \text{schoolbook}$$
$$r = (a_0 b_0 - a_1 b_1) + [(a_0 + a_1)(b_0 + b_1) - a_0 b_0 - a_1 b_1] i \quad \text{Karatsuba}$$

$\mathbb{F}_{p^2}$ layer - multiplication

$$r = r_0 + r_1 i = a \times b = (a_0 + a_1 i) \times (b_0 + b_1 i)$$
$$r = (a_0 b_0 - a_1 b_1) + (a_0 b_1 + a_1 b_0) i \quad \text{schoolbook}$$
$$r = (a_0 b_0 - a_1 b_1) + [(a_0 + a_1)(b_0 + b_1) - a_0 b_0 - a_1 b_1] i \quad \text{Karatsuba}$$

1-way at $\mathbb{F}_p$-level (Karatsuba)

01 $t_1 \leftarrow a_0 + a_1$
02 $t_2 \leftarrow b_0 + b_1$
03 $tt_1 \leftarrow a_0 \times b_0$
04 $tt_2 \leftarrow a_1 \times b_1$
05 $tt_3 \leftarrow t_1 \times t_2$
06 $tt_3 \leftarrow tt_3 - tt_1 - tt_2$
07 $tt_1 \leftarrow tt_1 - tt_2$
08 $r_0 \leftarrow tt_1 \bmod p$
09 $r_1 \leftarrow tt_3 \bmod p$

$\mathbb{F}_{p^2}$ layer - multiplication

$$r = r_0 + r_1 i = a \times b = (a_0 + a_1 i) \times (b_0 + b_1 i)$$
$$r = (a_0 b_0 - a_1 b_1) + (a_0 b_1 + a_1 b_0) i \quad \text{schoolbook}$$
$$r = (a_0 b_0 - a_1 b_1) + [(a_0 + a_1)(b_0 + b_1) - a_0 b_0 - a_1 b_1] i \quad \text{Karatsuba}$$

1-way at $\mathbb{F}_p$-level — Karatsuba

01 $t_1 \leftarrow a_0 + a_1$
02 $t_2 \leftarrow b_0 + b_1$
03 $tt_1 \leftarrow a_0 \times b_0$
04 $tt_2 \leftarrow a_1 \times b_1$
05 $tt_3 \leftarrow t_1 \times t_2$
06 $tt_3 \leftarrow tt_3 - tt_1 - tt_2$
07 $tt_1 \leftarrow tt_1 - tt_2$
08 $r_0 \leftarrow tt_1 \bmod p$
09 $r_1 \leftarrow tt_3 \bmod p$

2-way at $\mathbb{F}_p$-level — schoolbook

01 $tt_1 \leftarrow a_0 \times b_0$
02 $tt_2 \leftarrow a_1 \times b_0$
03 $tt_3 \leftarrow tt_1 - ss_2$
04 $r_0 \leftarrow tt_3 \bmod p$
$ss_1 \leftarrow a_0 \times b_1$
$ss_2 \leftarrow a_1 \times b_1$
$ss_3 \leftarrow tt_2 + ss_1$
$r_1 \leftarrow ss_3 \bmod p$

$\mathbb{F}_{p^2}$ layer - multiplication

$$r = r_0 + r_1 i = a \times b = (a_0 + a_1 i) \times (b_0 + b_1 i)$$
$$r = (a_0 b_0 - a_1 b_1) + (a_0 b_1 + a_1 b_0) i \quad \text{schoolbook}$$
$$r = (a_0 b_0 - a_1 b_1) + [(a_0 + a_1)(b_0 + b_1) - a_0 b_0 - a_1 b_1] i \quad \text{Karatsuba}$$

1-way at $\mathbb{F}_p$-level — Karatsuba

$$\begin{array}{lll} 01\ t_1 \leftarrow a_0 + a_1 & 04\ tt_2 \leftarrow a_1 \times b_1 & 07\ tt_1 \leftarrow tt_1 - tt_2 \\ 02\ t_2 \leftarrow b_0 + b_1 & 05\ tt_3 \leftarrow t_1 \times t_2 & 08\ r_0 \leftarrow tt_1 \bmod p \\ 03\ tt_1 \leftarrow a_0 \times b_0 & 06\ tt_3 \leftarrow tt_3 - tt_1 - tt_2 & 09\ r_1 \leftarrow tt_3 \bmod p \end{array}$$

2-way at $\mathbb{F}_p$-level — schoolbook

$$\begin{array}{ll} 01\ tt_1 \leftarrow a_0 \times b_0 & ss_1 \leftarrow a_0 \times b_1 \\ 02\ tt_2 \leftarrow a_1 \times b_0 & ss_2 \leftarrow a_1 \times b_1 \\ 03\ tt_3 \leftarrow tt_1 - ss_2 & ss_3 \leftarrow tt_2 + ss_1 \\ 04\ r_0 \leftarrow tt_3 \bmod p & r_1 \leftarrow ss_3 \bmod p \end{array}$$

4-way at $\mathbb{F}_p$-level — schoolbook

$$\begin{array}{llll} 01\ tt_1 \leftarrow a_0 \times b_0 & ss_1 \leftarrow a_0 \times b_1 & tt_2 \leftarrow a_1 \times b_0 & ss_2 \leftarrow a_1 \times b_1 \\ 02\ t_1 \leftarrow tt_1 \bmod p & s_1 \leftarrow ss_1 \bmod p & t_2 \leftarrow tt_2 \bmod p & s_2 \leftarrow ss_2 \bmod p \\ 03\ r_0 \leftarrow t_1 - s_2 & r_1 \leftarrow t_2 + s_1 & & \end{array}$$

$\mathbb{F}_{p^2}$ layer - multiplication

$(x \times y \times z)$-way vectorization

- x $\mathbb{F}_{p^2}$ operations
- y $\mathbb{F}_p$ operations
- z lanes used by each $\mathbb{F}_p$ operation

Our implementations

- $(8 \times 1 \times 1)$-way $\Leftarrow$ (8×1)-way $\mathbb{F}_p$ Karatsuba $\Leftarrow$ Karatsuba ✓
- $(4 \times 2 \times 1)$-way $\Leftarrow$ (8×1)-way $\mathbb{F}_p$ schoolbook $\Leftarrow$ Karatsuba ✓
- $(4 \times 1 \times 2)$-way $\Leftarrow$ (4×2)-way $\mathbb{F}_p$ Karatsuba $\Leftarrow$ schoolbook
- $(2 \times 4 \times 1)$-way $\Leftarrow$ (8×1)-way $\mathbb{F}_p$ schoolbook $\Leftarrow$ Karatsuba
- $(2 \times 2 \times 2)$-way $\Leftarrow$ (4×2)-way $\mathbb{F}_p$ schoolbook $\Leftarrow$ schoolbook ✓

About $(2 \times 4 \times 1)$-way: no corresponding addition/subtraction/squaring implementations

Curve layer - three-point ladder [FLOR18]

- standard way to compute $R \leftarrow P + [k]Q$
- based on Montgomery ladder step `xDBLADD` (point doubling + differential point addition)
- some papers about vectorizing Montgomery ladder step on Curve25519 in 2-way [FL15, Cho16, FLD19] or 4-way [CS09, HEY20, NS20]
- choose the 4-way method of Hisil et al. [HEY20] (no special multiplication with curve coefficient)
- $(1 \times 4 \times 2 \times 1)$-way $\Leftarrow$ $(4 \times 2 \times 1)$-way $\mathbb{F}_{p^2}$

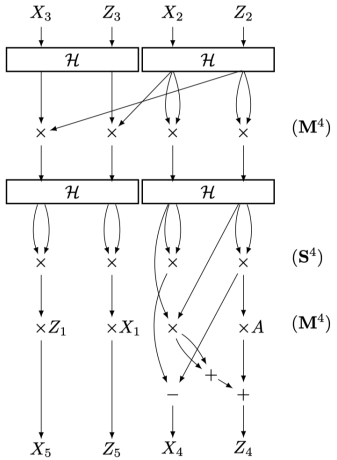


Figure 1: `DBLADD`: 4 way vectorized ladder step for the curve $By^2 = x^3 + Ax^2 + x$.

Curve layer - point tripling

We use C_{24} instead of A_{24}^- ($6M^2 + 6A^2 \Leftarrow 12M + 12A$ vs $6M^2 + 6A^2 \Leftarrow 12M + 10A$)

$$X_{[3]P} = X_P \left[C_{24} \left(X_P^2 - Z_P^2 \right)^2 - 4Z_P^2 \left(4A_{24}^+ X_P Z_P + C_{24}(X_P - Z_P)^2 \right) \right]^2$$

$$Z_{[3]P} = Z_P \left[C_{24} \left(X_P^2 - Z_P^2 \right)^2 - 4X_P^2 \left(4A_{24}^+ X_P Z_P + C_{24}(X_P - Z_P)^2 \right) \right]^2$$

Algorithm 1: XZ-coordinate point tripling with 2-way parallelization at $\mathbb{F}_{p^2}$-level.

Input: $P = (X_P : Z_P)$, $(A_{24}^+ : C_{24})$.
Output: $Q = [3]P = (X_Q : Z_Q)$.

$t_0 \leftarrow X_P + Z_P$	$s_0 \leftarrow X_P - Z_P$
$t_1 \leftarrow t_0^2$	$s_1 \leftarrow s_0^2$
$t_0 \leftarrow X_P + X_P$	$s_0 \leftarrow t_1 - s_1$
$t_2 \leftarrow t_1 + t_1$	$s_2 \leftarrow s_0 + s_0$
$t_3 \leftarrow C_{24} \times s_1$	$s_3 \leftarrow A_{24}^+ \times s_0$
$t_0 \leftarrow t_3 \times t_1$	$s_0 \leftarrow t_0 \times t_0$
$t_1 \leftarrow t_2 + t_2$	$s_1 \leftarrow s_0 + s_2$
$t_3 \leftarrow t_3 + s_3$	$s_3 \leftarrow t_1 - s_1$
$t_4 \leftarrow s_3 \times t_3$	$s_4 \leftarrow s_0 \times t_3$
$t_4 \leftarrow t_0 - t_4$	$s_4 \leftarrow t_0 - s_4$
$t_4 \leftarrow t_4^2$	$s_4 \leftarrow s_4^2$
$X_Q \leftarrow X_P \times t_4$	$Z_Q \leftarrow Z_P \times s_4$
return $Q = (X_Q : Z_Q)$	

Curve layer - 3-isogeny generation

We use C_{24} instead of A_{24}^{-} ($3M^2 + 5A^2 \Leftarrow 6M + 10A$ vs $3M^2 + 7A^2 \Leftarrow 5M + 13A$)

$$
\begin{aligned}
K_1 &= X_3 - Z_3 \\
K_2 &= X_3 + Z_3 \\
A_{24}^{+} &= (3X_3^2 - 2X_3Z_3 - Z_3^2)(3X_3 + Z_3)^2 \\
C_{24} &= -16X_3Z_3^3
\end{aligned}
$$

Algorithm 2: 3-isogeny computation with 2-way parallelization at $\mathbb{F}_{p^2}$-level.

Input: $P_3 = (X_3 : Z_3)$.
Output: $(A_{24}^{+} : C_{24})$, $(K_1 : K_2)$.

$K_2 \leftarrow X_3 + Z_3$	$K_1 \leftarrow X_3 - Z_3$
$t_0 \leftarrow K_2^2$	$s_0 \leftarrow K_1^2$
$t_1 \leftarrow X_3 + X_3$	$s_1 \leftarrow s_0 - t_0$
$t_2 \leftarrow t_1^2$	$s_2 \leftarrow Z_3^2$
$t_1 \leftarrow t_2 + t_2$	$s_1 \leftarrow s_1 + s_1$
$t_3 \leftarrow t_1 - s_1$	$s_3 \leftarrow s_2 + s_2$
$t_4 \leftarrow t_3 + s_0$	$s_4 \leftarrow t_2 - t_0$
$C_{24} \leftarrow s_1 \times s_3$	$A_{24}^{+} \leftarrow t_4 \times s_4$
return $(A_{24}^{+} : C_{24}), (K_1 : K_2)$	

Curve layer - point doubling and 4-isogeny generation

$3M^2 + 3A^2 \Leftarrow 6M + 4A$

Algorithm 3: XZ-coordinate point doubling with 2-way parallelization at $\mathbb{F}_{p^2}$-level.

Input: $P = (X_P : Z_P)$, $(A_{24}^+ : C_{24})$.
Output: $Q = [2]P = (X_Q : Z_Q)$.

1	$t_0 \leftarrow X_P + Z_P$	$s_0 \leftarrow X_P - Z_P$
2	$t_1 \leftarrow t_0^2$	$s_1 \leftarrow s_0^2$
3		$s_0 \leftarrow t_1 - s_1$
4	$t_2 \leftarrow C_{24} \times s_1$	$s_2 \leftarrow A_{24}^+ \times s_0$
5	$t_3 \leftarrow t_2 + s_2$	
6	$X_Q \leftarrow t_2 \times t_1$	$Z_Q \leftarrow t_3 \times s_0$
7	**return** $Q = (X_Q : Z_Q)$	

$2M^2 + 3A^2 \Leftarrow 4M + 5A$

Algorithm 4: 4-isogeny computation with 2-way parallelization at $\mathbb{F}_{p^2}$-level.

Input: $P_4 = (X_4 : Z_4)$.
Output: $(A_{24}^+ : C_{24})$, $(K_0 : K_1 : K_2)$.

1	$K_2 \leftarrow X_4 + Z_4$	$K_1 \leftarrow X_4 - Z_4$
2	$t_0 \leftarrow Z_4^2$	$s_0 \leftarrow X_4^2$
3	$t_0 \leftarrow t_0 + t_0$	$s_0 \leftarrow s_0 + s_0$
4	$C_{24} \leftarrow t_0^2$	$A_{24}^+ \leftarrow s_0^2$
5	$K_0 \leftarrow t_0 + t_0$	
6	**return** $(A_{24}^+ : C_{24}), (K_0 : K_1 : K_2)$	

Curve layer - 4/3-isogeny evaluation

$4M^2 + 3A^2 \Leftarrow 8M + 6A$

Algorithm 5: 4-isogeny evaluation with 2-way parallelization at $\mathbb{F}_{p^2}$-level.

Input: $P = (X_P : Z_P)$, $(K_0 : K_1 : K_2)$.
Output: $P' = \phi_4(P) = (X_{P'} : Z_{P'})$.

1	$t_0 \leftarrow X_P + Z_P$	$s_0 \leftarrow X_P - Z_P$
2	$t_1 \leftarrow K_1 \times t_0$	$s_1 \leftarrow t_0 \times s_0$
3	$t_2 \leftarrow K_2 \times s_0$	$s_2 \leftarrow K_0 \times s_1$
4	$t_0 \leftarrow t_2 + t_1$	$s_0 \leftarrow t_2 - t_1$
5	$t_0 \leftarrow t_0^2$	$s_0 \leftarrow s_0^2$
6	$t_1 \leftarrow t_0 + s_2$	$s_1 \leftarrow s_0 - s_2$
7	$X_{P'} \leftarrow t_1 \times t_0$	$Z_{P'} \leftarrow s_1 \times s_0$
8	**return** $(X_{P'} : Z_{P'})$	

$3M^2 + 2A^2 \Leftarrow 6M + 4A$

Algorithm 6: 3-isogeny evaluation with 2-way parallelization at $\mathbb{F}_{p^2}$-level.

Input: $P = (X_P : Z_P)$, $(K_1 : K_2)$.
Output: $P' = \phi_3(P) = (X_{P'} : Z_{P'})$.

1	$t_0 \leftarrow X_P + Z_P$	$s_0 \leftarrow X_P - Z_P$
2	$t_0 \leftarrow K_1 \times t_0$	$s_0 \leftarrow K_2 \times s_0$
3	$t_1 \leftarrow t_0 + s_0$	$s_1 \leftarrow t_0 - s_0$
4	$t_1 \leftarrow t_1^2$	$s_1 \leftarrow s_1^2$
5	$X_{P'} \leftarrow X_P \times t_1$	$Z_{P'} \leftarrow Z_P \times s_1$
6	**return** $(X_{P'} : Z_{P'})$	

Curve layer - implementations (for AvxSike-LL)

- three-point ladder
  $(1 \times 4 \times 2 \times 1)$-way
- point doubling
  $(1 \times 2 \times 2 \times 2)$-way
- point tripling
  $(1 \times 2 \times 2 \times 2)$-way
- 4-isogeny generation
  $(1 \times 2 \times 2 \times 2)$-way
- 3-isogeny generation
  $(1 \times 2 \times 2 \times 2)$-way
- 4-isogeny evaluation
  $(8 \times 1 \times 1 \times 1)$-way, $(4 \times 2 \times 1 \times 1)$-way, $(2 \times 2 \times 2 \times 1)$-way, $(1 \times 2 \times 2 \times 2)$-way
- 3-isogeny evaluation
  $(8 \times 1 \times 1 \times 1)$-way, $(4 \times 2 \times 1 \times 1)$-way, $(2 \times 2 \times 2 \times 1)$-way, $(1 \times 2 \times 2 \times 2)$-way

Curve layer - implementations (for AvxSike-LL)

- three-point ladder
  $(1 \times 4 \times 2 \times 1)$-way, $(2 \times 4 \times 1 \times 1)$-way
- point doubling
  $(1 \times 2 \times 2 \times 2)$-way, $(2 \times 2 \times 2 \times 1)$-way
- point tripling
  $(1 \times 2 \times 2 \times 2)$-way
- 4-isogeny generation
  $(1 \times 2 \times 2 \times 2)$-way, $(2 \times 2 \times 2 \times 1)$-way
- 3-isogeny generation
  $(1 \times 2 \times 2 \times 2)$-way
- 4-isogeny evaluation
  $(8 \times 1 \times 1 \times 1)$-way, $(4 \times 2 \times 1 \times 1)$-way, $(2 \times 2 \times 2 \times 1)$-way, $(1 \times 2 \times 2 \times 2)$-way
- 3-isogeny evaluation
  $(8 \times 1 \times 1 \times 1)$-way, $(4 \times 2 \times 1 \times 1)$-way, $(2 \times 2 \times 2 \times 1)$-way, $(1 \times 2 \times 2 \times 2)$-way

Isogeny computation layer - 2^{n_A}-isogeny computation

Algorithm 7: Vectorized 2^{n_A}-isogeny computation in SIKEp503 using optimal strategies.

Input: Curve E_A, public parameter n_A, point (X_R, Z_R), and a *strategy* $(s_1, \ldots, s_{n_A/2-1})$.
Output: Curve E_B such that $\phi_2 : E_A \to E_B$ with $\deg \phi_2 = 2^{n_A}$, $\ker \phi_2 = \langle (X_R : Z_R) \rangle$.

$pts \leftarrow [\,]$, $i \leftarrow 0$, $E_B \leftarrow E_A$, $k \leftarrow 1$
for j **from** 1 **to** $n_A/2 - 1$ **by** 1 **do**
  while $i < n_A/2 - j$ **do**
    push(X_R, Z_R, i) to pts // Append it to the end of pts
    $e \leftarrow s_k$
    $(X_R, Z_R) \leftarrow$ xDBLe_1x2x2x2w$(X_R, Z_R, E_B, 2e)$ // $[2^{2e}]R$
    $i \leftarrow i + e$, $k \leftarrow k + 1$
  $E_B, \phi \leftarrow$ get_4_isog_1x2x2x2w(X_R, Z_R) // 4-isogeny generation
  $pts \leftarrow$ eval_4_isog_parallel(ϕ, pts) // Parallel 4-isogeny evaluation
  pop(X_S, Z_S, i_S) from pts // Remove it from the end of pts
  $X_R \leftarrow X_S$, $Z_R \leftarrow Z_S$, $i \leftarrow i_S$
$E_B, \phi \leftarrow$ get_4_isog_1x2x2x2w(X_R, Z_R) // 4-isogeny generation
return E_B

PKE layer - optimized encryption

Algorithm 8: The optimized SIKE.PKE encryption operation.

function Enc($\mathsf{pk}_3, m \in \mathcal{M}, \mathsf{sk}_2 \in \mathcal{K}_2$)
$x_{R_2} \leftarrow x_{P_2} + [\mathsf{sk}_2]x_{Q_2}$
$(\phi_2, E_2) \leftarrow \mathtt{isogeny}_2(E_0, x_{R_2})$
$c_1 \leftarrow (\phi_2(x_{P_3}), \phi_2(x_{Q_3}), \phi_2(x_{PQ_3}))$
$x'_{R_2} \leftarrow \phi_3(x_{P_2}) + [\mathsf{sk}_2]\phi_3(x_{Q_2})$
$(\phi'_2, E_{32}) \leftarrow \mathtt{isogeny}_2(E_3, x'_{R_2})$
$h \leftarrow \mathtt{SHAKE256}(j(E_{32}))$
$c_2 \leftarrow h \oplus m$
return (c_1, c_2)

function EncOpt($\mathsf{pk}_3, m \in \mathcal{M}, \mathsf{sk}_2 \in \mathcal{K}_2$)
$x_{R_2} \leftarrow x_{P_2} + [\mathsf{sk}_2]x_{Q_2}$ | $x'_{R_2} \leftarrow \phi_3(x_{P_2}) + [\mathsf{sk}_2]\phi_3(x_{Q_2})$
$(\phi_2, E_2) \leftarrow \mathtt{isogeny}_2(E_0, x_{R_2})$ | $(\phi'_2, E_{32}) \leftarrow \mathtt{isogeny}_2(E_3, x'_{R_2})$
$c_1 \leftarrow (\phi_2(x_{P_3}), \phi_2(x_{Q_3}), \phi_2(x_{PQ_3}))$
$h \leftarrow \mathtt{SHAKE256}(j(E_{32}))$
$c_2 \leftarrow h \oplus m$
return (c_1, c_2)

Results

Table: Execution times (in cycles) of implementations of SIKEp503 on an Intel Core i3-1005G1 processor (compiler gcc 9.3.0; turbo-boost disabled).

Operation	SIDHv3.4 (1 instance)	Kostic [KG19] (1 instance)		AvxSike-LL (1 instance)		AvxSike-HT (8 instances)	
	Cycles	Cycles	Speed-up	Cycles	Speed-up	Cycles	Throughput
KeyGen	8,078,669	4,842,909	1.67×	3,215,375	2.51×	14,179,026	4.56×
Encaps	13,188,788	7,923,514	1.66×	4,111,650	3.21×	22,992,807	4.59×
Decaps	14,026,750	8,513,409	1.65×	5,715,005	2.45×	24,619,263	4.56×

Results

Table: Execution times (in cycles) of implementations of SIKEp434, SIKEp610, and SIKEp751 on an Intel Core i3-1005G1 processor (compiler gcc 9.3.0; turbo-boost disabled).

Scheme	Operation	SIDHv3.4 (1 instance)	AvxSike-LL (1 instance)		AvxSike-HT (8 instances)	
		Cycles	Cycles	Speed-up	Cycles	Throughput
SIKEp434	KeyGen	5,976,700	2,474,187	2.42×	10,442,609	4.58×
	Encaps	9,690,764	3,062,491	3.16×	16,801,041	4.61×
	Decaps	10,357,218	4,341,099	2.39×	18,053,398	4.59×
SIKEp610	KeyGen	14,096,085	6,918,618	2.04×	32,172,538	3.51×
	Encaps	25,875,968	10,001,282	2.59×	58,747,976	3.52×
	Decaps	26,040,095	13,124,052	1.98×	59,103,361	3.52×
SIKEp751	KeyGen	23,843,419	10,212,410	2.33×	46,662,723	4.09×
	Encaps	38,446,643	12,804,923	3.00×	74,885,499	4.11×
	Decaps	41,368,995	17,834,974	2.32×	80,684,214	4.10×

Epilogue

- vector engines like AVX-512 offer great potential to optimize large integer arithmetic and isogeny-based crypto
- thorough analysis for SIKE arithmetic
- less dependency then much better performance in the vectorized implementation

Thanks for your attention!

References I

Wouter Castryck and Thomas Decru.

An efficient key recovery attack on sidh (preliminary version).

Cryptology ePrint Archive, Paper 2022/975, 2022.

https://eprint.iacr.org/2022/975.

Tung Chou.

Sandy2x: New Curve25519 speed records.

In Orr Dunkelman and Liam Keliher, editors, *Selected Areas in Cryptography — SAC 2015*, volume 9566 of *Lecture Notes in Computer Science*, pages 145–160. Springer Verlag, 2016.

Neil Costigan and Peter Schwabe.

Fast elliptic-curve cryptography on the Cell Broadband Engine.

In Bart Preneel, editor, *Progress in Cryptology — AFRICACRYPT 2009*, volume 5580 of *Lecture Notes in Computer Science*. Springer Verlag, 2009.

References II

Armando Faz-Hernández and Julio López.

Fast implementation of Curve25519 using AVX2.

In Kristin E. Lauter and Francisco Rodríguez-Henríquez, editors, *Progress in Cryptology — LATINCRYPT 2015*, volume 9230 of *Lecture Notes in Computer Science*, pages 329–345. Springer Verlag, 2015.

Armando Faz-Hernández, Julio López, and Ricardo Dahab.

High-performance implementation of elliptic curve cryptography using vector instructions.

ACM Transactions on Mathematical Software, 45(3):1–35, July 2019.

Armando Faz-Hernández, Julio C. López-Hernández, Eduardo Ochoa-Jiménez, and Francisco Rodríguez-Henríquez.

A faster software implementation of the supersingular isogeny Diffie-Hellman key exchange protocol.

IEEE Transactions on Computers, 67(11):1622–1636, November 2018.

H. Hişil, H. Eğrice, and M. Yassi.

Fast 4 way vectorized ladder for the complete set of Montgomery curves.

Cryptology ePrint Archive 2020/388, 2020.

References III

Dusan Kostic and Shay Gueron.

Using the new VPMADD instructions for the new post quantum key encapsulation mechanism SIKE.

In *Proceedings of the 26th IEEE Symposium on Computer Arithmetic (ARITH 2019)*, pages 215–218. IEEE, 2019.

Kaushik Nath and Palash Sarkar.

Efficient 4-way vectorizations of the Montgomery ladder.

Cryptology ePrint Archive, Report 2020/378, 2020.