SIDH on ARM:
Faster Modular Multiplications for Faster Post-Quantum Supersingular Isogeny Key Exchange.

Hwajeong Seo (Hansung University),
Zhe Liu (Nanjing University of Aeronautics and Astronautics),
Patrick Longa (Microsoft Research),
Zhi Hu (Central South University)

Outline

- Short Overview

- Post-quantum supersingular isogeny Diffie-Hellman (SIDH) key exchange
    - Supersingular isogeny key encapsulation (SIKE) protocol

- Our implementation
    - Optimized implementations for 32-bit ARMv7
    - Optimized implementations for 64-bit ARMv8

- Implementation results

- Conclusion

Post-Quantum Cryptography **(Isogeny)**

- **RSA and ECC:** integer factorization and ECDLP
  - Hard problems can be solved by **Shor's algorithm** in a quantum computer.

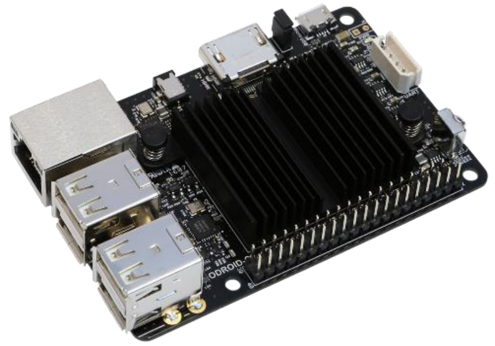
- **Quantum-Resistant Cryptography**
  - NIST launches the post-quantum cryptography standardization project.

"The goal of this process is to select a number of acceptable candidate cryptosystems for standardization."

  - Code, Lattice, Hash, Multivariate, **Isogeny**…

- **Isogeny-based cryptography:** (conjectured to be) hard for quantum computers
  - Supersingular isogeny Diffie-Hellman (SIDH) key exchange was proposed by Jao and De Feo in **2011**.
  - Among all the submitted post-quantum candidates, SIDH uses the **smallest keys**

Mobile Platform **(32-bit/64-bit ARM)**



Platform	**ARM Cortex-A15**	**ARM Cortex-A53**	**ARM Cortex-A72**
Architecture	**32-bit ARMv7**	**64-bit ARMv8**	**64-bit ARMv8**
Frequency	2.0 GHz	1.512 GHz	1.992 GHz
No. registers	15	31	31
No. registers (NEON)	16	32	32
Application	**Wearable devices**	**Smartphones**	

Previous Works

- **Hardware Implementation**
  - FPGA:
    - Koziel et al. [INDOCRYPT'16, TCAS'17]

- **Software Implementation**
  - 64-bit Intel processor:
    - Costello et al. [CRYPTO'16, EUROCRYPT'17], Faz-Hernández et al. [ToC'17], Zanon et al. [PQCrypto'18]

  - 64-bit ARM processor:
    - Jalali et al. [SAC'17] → **this work [CHES'18]**

  - 32-bit ARM processor:
    - Koziel et al. [CANS'16] → **this work [CHES'18]**

Motivation

Type	Algorithm	Advantage	Disadvantage
Code	McEliece	✓ Fast computation	✓ Long key size
Hash	XMSS, SPHINCS	✓ Security proof	✓ Long signature size
Lattice	(ring)-LWE	✓ Fast computation	✓ Difficulty of parameter selection
Multivariate	UOV, Rainbow	✓ Short signature size ✓ Fast computation	✓ Long key size
Isogeny	SIDH, SIKE	✓ Short key size	✓ Slow computation

- All PQC candidates have their own **pros** and **cons**.
- **Disadvantage of SIDH/SIKE** is slow computation.
- In this talk, we address this problem on **32-bit and 64-bit ARM processors.**

Contribution

- **Unified ARM/NEON multiplication: instruction level parallelism**
- **New Montgomery reduction: "UMAAL" + "hybrid-scanning"**
- **Efficient Implementation of SIDH:**
  - p503 (**88 msec**) / p751 (**292 msec**) on 32-bit ARMv7-A **@2.0GHz**
  - p503 (**45 msec**) on 64-bit ARMv8-A **@1.992GHz**

Outline

- Short Overview
- Post-quantum supersingular isogeny Diffie-Hellman (SIDH) key exchange
  - Supersingular isogeny key encapsulation (SIKE) protocol
- Our implementation
  - Optimized implementations for 32-bit ARMv7
  - Optimized implementations for 64-bit ARMv8
- Implementation results
- Conclusion

Post-quantum key exchange algorithm

- Supersingular Isogeny Diffie-Hellman **(SIDH)**
  - Shared key generation between two parties over an insecure communication channel.
  - SIDH works with the set of supersingular elliptic curves over $\mathbb{F}_{p^2}$ and their isogenies.

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

Supersingular Isogeny Key Encapsulation **(SIKE)**

- SIDH is not secure when keys are reused (Galbraith-Petit-Shani-Ti 2016)
- **SIKE:** (Costello–De Feo–Jao–Longa–Naehrig–Renes 2017)
  - IND-CCA secure key encapsulation based on SIDH.
- Uses a variant of Hofheinz-Hövelmanns-Kiltz (HHK) transform: **IND-CPA PKE** $\rightarrow$ **IND-CCA KEM**

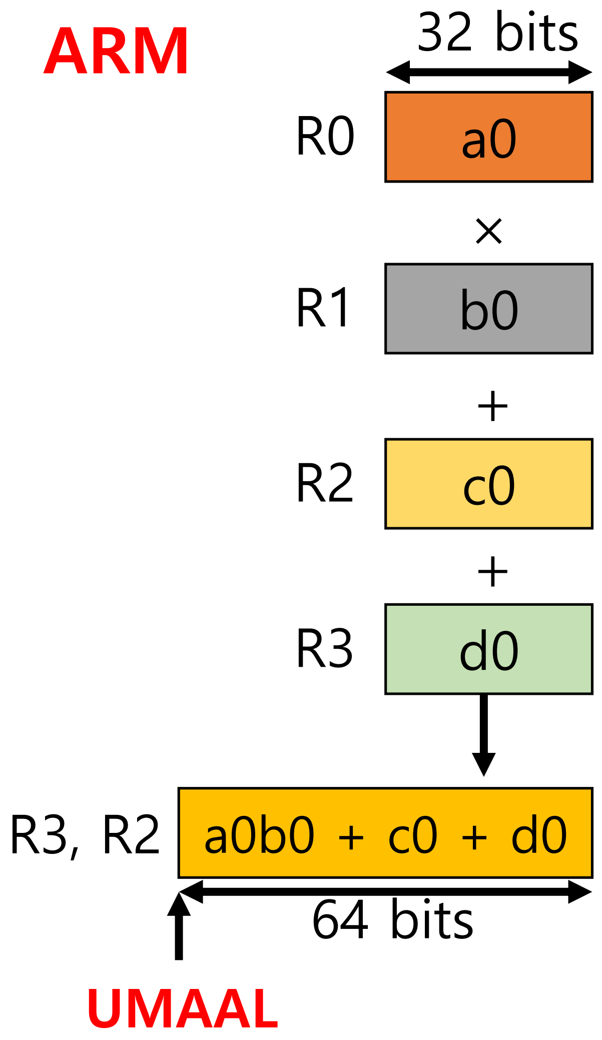
- For a starting curve $E_0/\mathbb{F}_{p^2}: y^2 = x^3 + x$, where $p = 2^{eA}3^{eB} - 1$

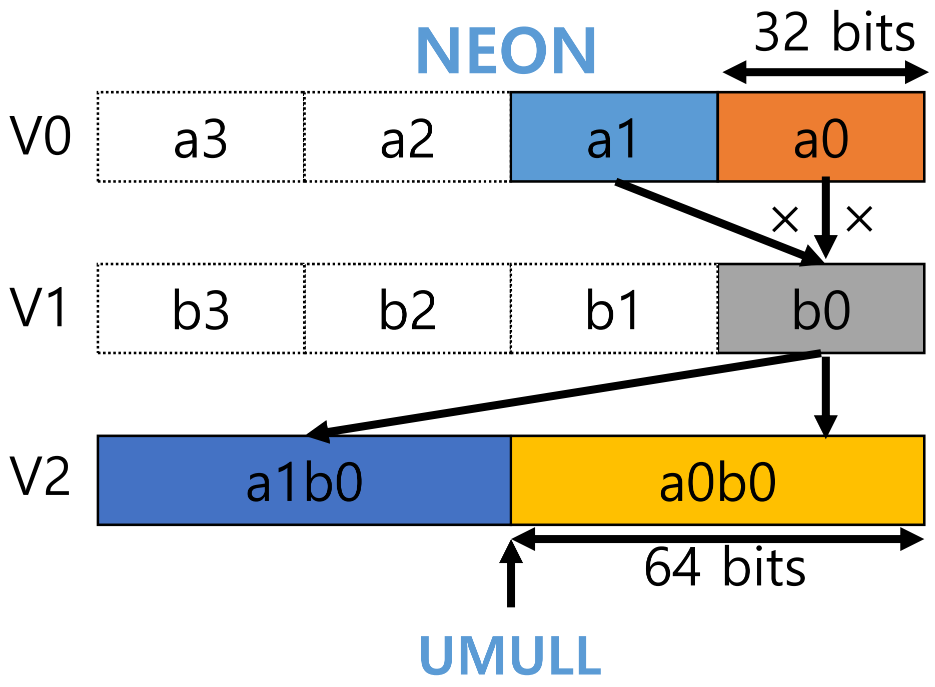
Scheme (SIKEp + $\log_2 p$)	e_A, e_B	classical sec.	quantum sec.	Security level
SIKEp503	(250,159)	126 bits	84 bits	AES-128 (NIST level 1)
SIKEp751	(372,239)	188 bits	125 bits	AES-192 (NIST level 3)
SIKEp964	(486,301)	241 bits	161 bits	AES-256 (NIST level 5)

Outline

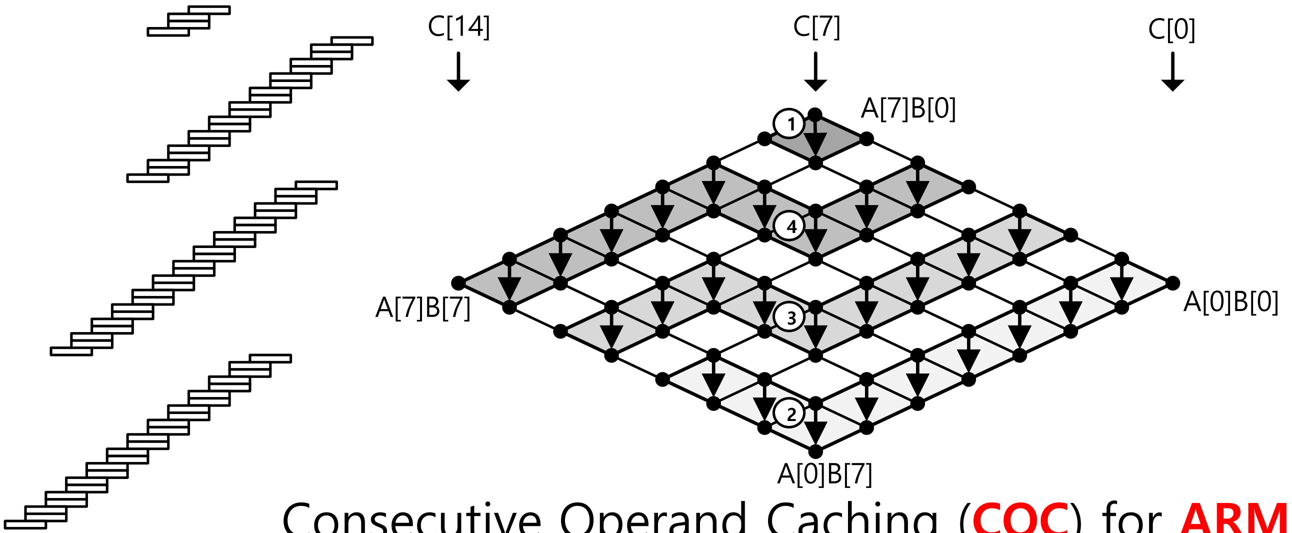
- Short Overview
- Post-quantum supersingular isogeny Diffie-Hellman (SIDH) key exchange
  - Supersingular isogeny key encapsulation (SIKE) protocol
- Our implementation
  - Optimized implementations for 32-bit ARMv7
  - Optimized implementations for 64-bit ARMv8
- Implementation results
- Conclusion

Multiplication Instruction **(32-bit ARMv7)**

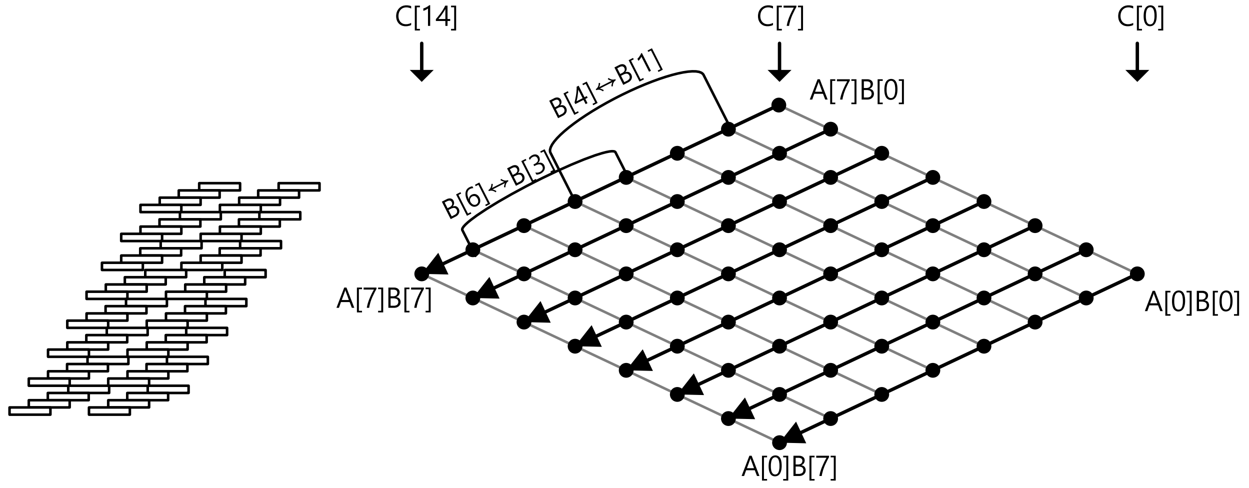




Previous Multiprecision Multiplication **(32-bit ARMv7)**



Consecutive Operand Caching (**COC**) for **ARM**



Cascade Operand Scanning (**COS**) for **NEON**

Bitlength	Method	Instruction	Timings [cc]
256-bit	**COC**	**ARM (UMAAL)**	158 BEST
	COS	NEON (UMULL)	188
512-bit	**COC**	**ARM (UMAAL)**	596 BEST
	COS	NEON (UMULL)	632

Target processor: 32-bit ARM Cortex-A15

Proposed Multiprecision Multiplication **(32-bit ARMv7)**

- **Instruction level parallelism**
  - **ARM** and **NEON** instructions are issued together

- **Karatsuba multiplication:**

m-bit multiplication $(A_H \cdot B_H \cdot 2^m + [A_H \cdot B_H + A_L \cdot B_L - |A_H - A_L| \cdot |B_H - B_L|] \cdot 2^{m/2} + A_L \cdot B_L)$

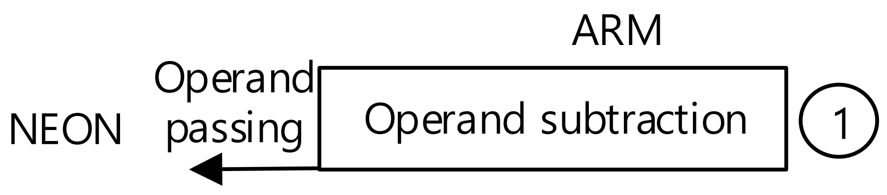
  - **Two $m/2$-bit multiplication in ARM**
  - **One $m/2$-bit multiplication in NEON**

Algorithm 1 Unified ARM/NEON multiplication

```
Input: Two $m$-bit operands $A = (A_H||A_L)$ and $B = (B_H||B_L)$
Output: $2m$-bit result $C$
1: $A_M \leftarrow |A_L - A_H|$                                   {ARM}
2: $B_M \leftarrow |B_L - B_H|$                                   {ARM}

   Interleaved section begin
3: $C_L \leftarrow A_L \cdot B_L$                                 {ARM}
4: $C_M \leftarrow A_M \cdot B_M$                                 {NEON}
5: $C_H \leftarrow A_H \cdot B_H$                                 {ARM}
   Interleaved section end

6: return $C \leftarrow C_L + (C_L + C_H - C_M) \cdot 2^{\frac{m}{2}} + C_H \cdot 2^m$   {ARM}
```

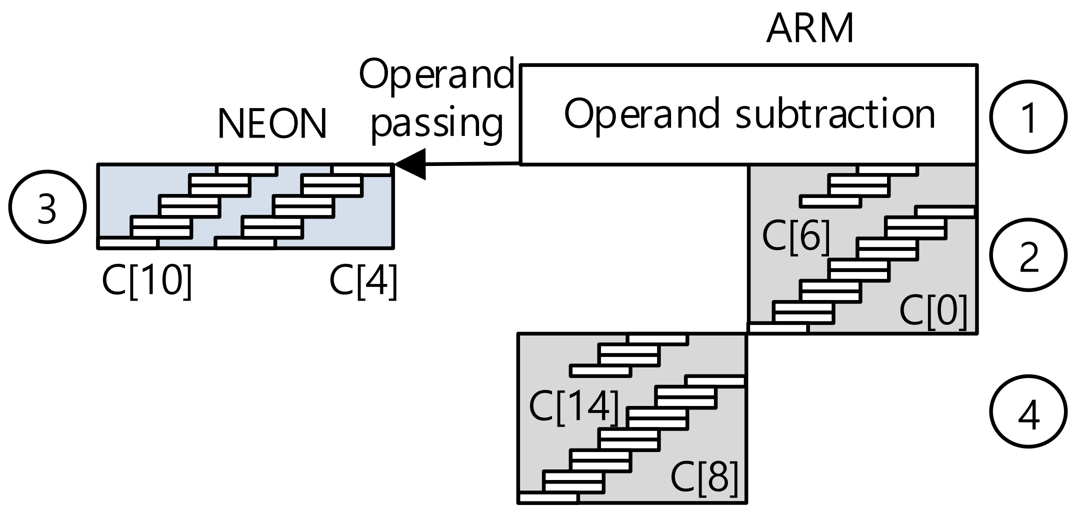


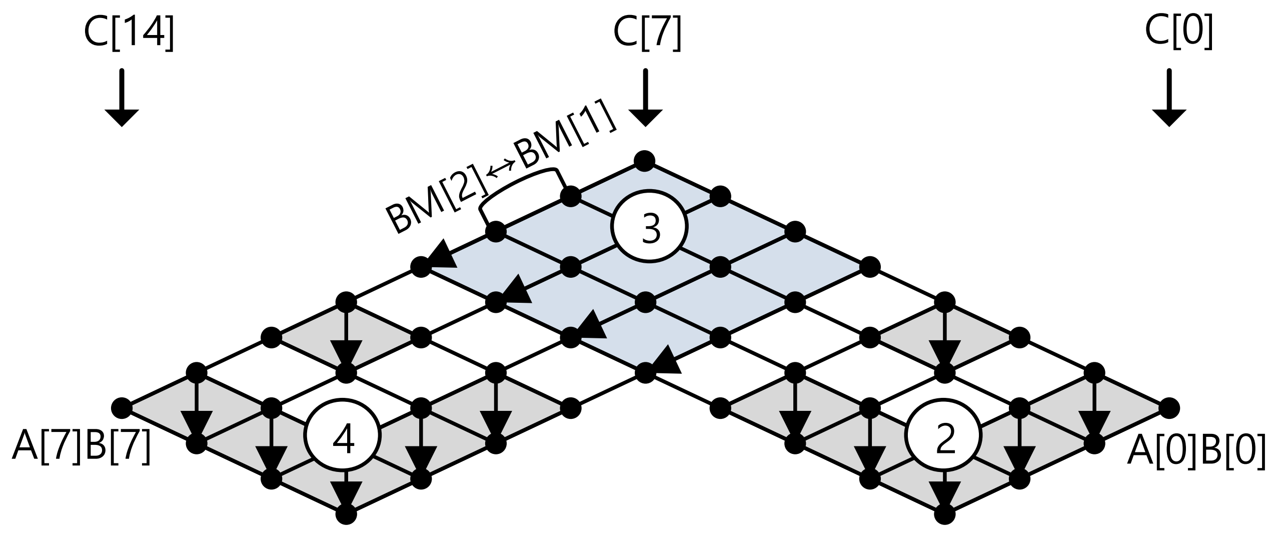
Algorithm 1 Unified ARM/NEON multiplication

```
Input: Two m-bit operands $A = (A_H||A_L)$ and $B = (B_H||B_L)$
Output: 2m-bit result $C$
1: $A_M \leftarrow |A_L - A_H|$                                   {ARM}
2: $B_M \leftarrow |B_L - B_H|$                                   {ARM}

   Interleaved section begin
3: $C_L \leftarrow A_L \cdot B_L$                                 {ARM}
4: $C_M \leftarrow A_M \cdot B_M$                                 {NEON}
5: $C_H \leftarrow A_H \cdot B_H$                                 {ARM}
   Interleaved section end

6: return $C \leftarrow C_L + (C_L + C_H - C_M) \cdot 2^{\frac{m}{2}} + C_H \cdot 2^m$   {ARM}
```



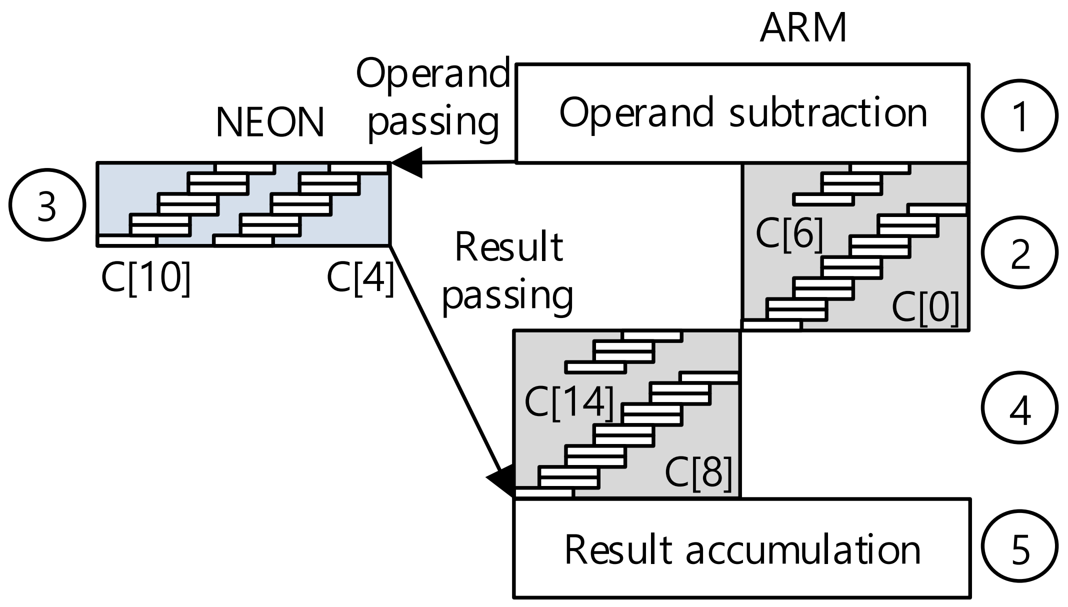


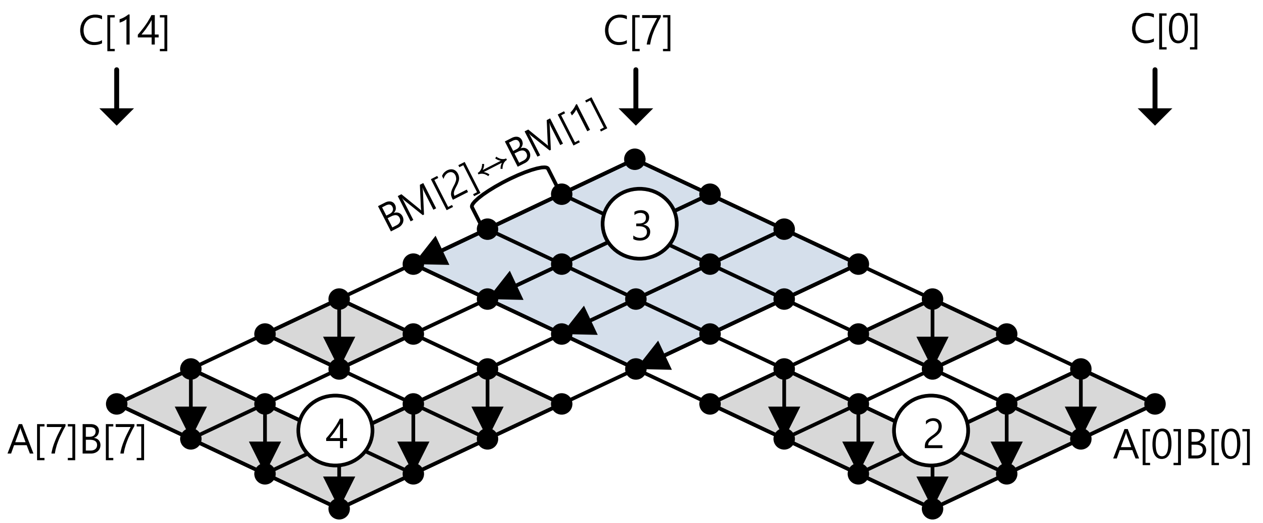
Algorithm 1 Unified ARM/NEON multiplication

```
Input: Two m-bit operands $A = (A_H||A_L)$ and $B = (B_H||B_L)$
Output: 2m-bit result $C$
1: $A_M \leftarrow |A_L - A_H|$                                   {ARM}
2: $B_M \leftarrow |B_L - B_H|$                                   {ARM}

   Interleaved section begin
3: $C_L \leftarrow A_L \cdot B_L$                                 {ARM}
4: $C_M \leftarrow A_M \cdot B_M$                                 {NEON}
5: $C_H \leftarrow A_H \cdot B_H$                                 {ARM}
   Interleaved section end

6: return $C \leftarrow C_L + (C_L + C_H - C_M) \cdot 2^{\frac{m}{2}} + C_H \cdot 2^m$   {ARM}
```





Proposed Multiprecision Multiplication **(32-bit ARMv7)**

Bitlength	Method	Instruction	Timings [cc]
512-bit	COC	ARM	596
	GMP-6.1.2	ARM	1,138
	COS	NEON	632
	This work	**ARM/NEON**	**470**
768-bit	GMP-6.1.2	ARM	2,408
	This work	**ARM/NEON**	**912**

1.26x

2.64x

Target processor: 32-bit ARM Cortex-A15

Proposed Modular Reduction (32-bit ARMv7)

- *m*-bit modular reduction using **Montgomery reduction**
  - **Two $m/2$-bit multiplication in ARM**
  - **Two $m/2$-bit multiplication in NEON**

Algorithm 4 Unified ARM/NEON Montgomery reduction

Input: An odd m-bit modulus $M = (M_H \| M_L)$, the Montgomery radix $R = 2^s$, where $s = wn$ with $w = 32$ and $n = \lceil m/w \rceil$, an operand $T \in [0, M^2 - 1]$, and pre-computed constant $M' = -M^{-1} \bmod R$

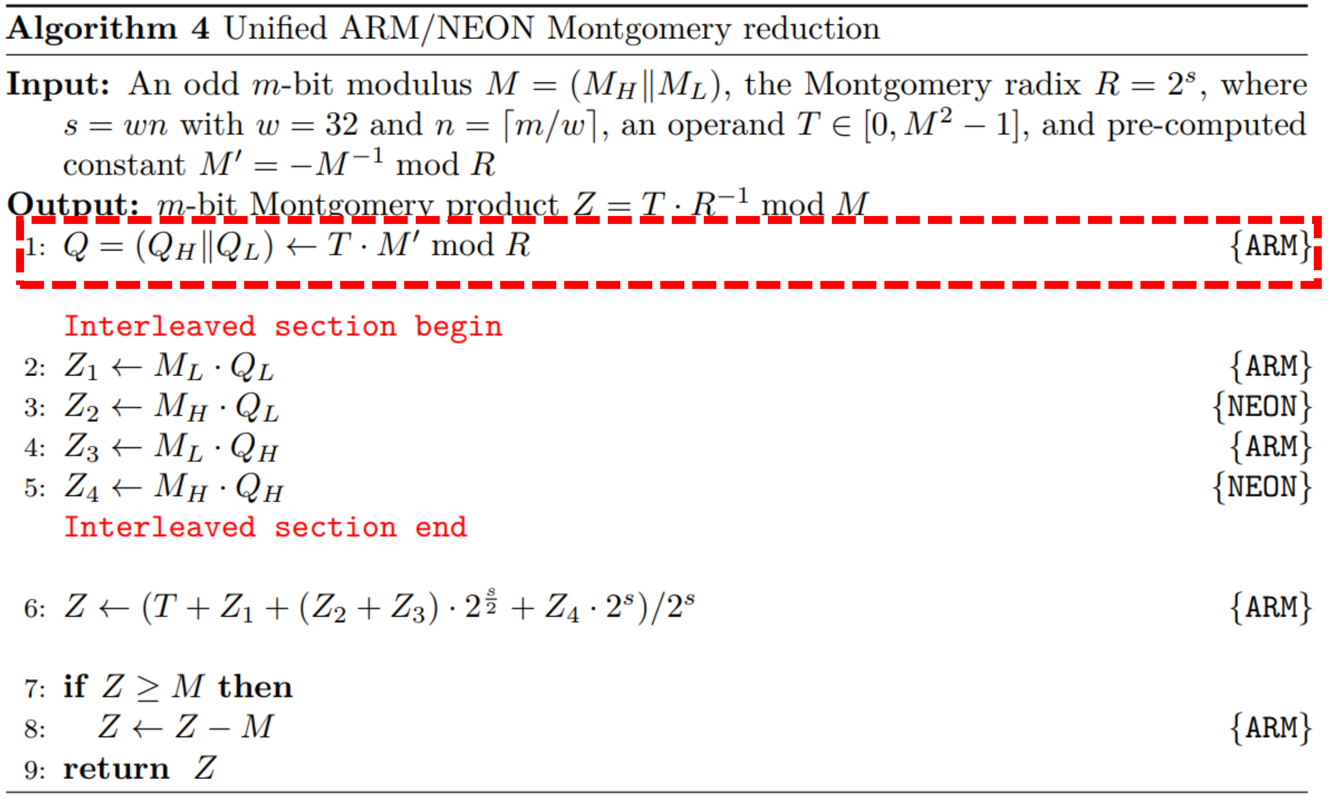
Output: m-bit Montgomery product $Z = T \cdot R^{-1} \bmod M$

```
1: $Q = (Q_H \| Q_L) \leftarrow T \cdot M' \bmod R$                {ARM}

   Interleaved section begin
2: $Z_1 \leftarrow M_L \cdot Q_L$                                    {ARM}
3: $Z_2 \leftarrow M_H \cdot Q_L$                                    {NEON}
4: $Z_3 \leftarrow M_L \cdot Q_H$                                    {ARM}
5: $Z_4 \leftarrow M_H \cdot Q_H$                                    {NEON}
   Interleaved section end

6: $Z \leftarrow (T + Z_1 + (Z_2 + Z_3) \cdot 2^{\frac{s}{2}} + Z_4 \cdot 2^s)/2^s$   {ARM}

7: if $Z \geq M$ then
8:     $Z \leftarrow Z - M$                                          {ARM}
9: return $Z$
```



Algorithm 4 Unified ARM/NEON Montgomery reduction

```
Input: An odd $m$-bit modulus $M = (M_H \| M_L)$, the Montgomery radix $R = 2^s$, where
    $s = wn$ with $w = 32$ and $n = \lceil m/w \rceil$, an operand $T \in [0, M^2 - 1]$, and pre-computed
    constant $M' = -M^{-1} \bmod R$
Output: $m$-bit Montgomery product $Z = T \cdot R^{-1} \bmod M$
1: $Q = (Q_H \| Q_L) \leftarrow T \cdot M' \bmod R$                                    {ARM}

   Interleaved section begin
2: $Z_1 \leftarrow M_L \cdot Q_L$                                                      {ARM}
3: $Z_2 \leftarrow M_H \cdot Q_L$                                                      {NEON}
4: $Z_3 \leftarrow M_L \cdot Q_H$                                                      {ARM}
5: $Z_4 \leftarrow M_H \cdot Q_H$                                                      {NEON}
   Interleaved section end

6: $Z \leftarrow (T + Z_1 + (Z_2 + Z_3) \cdot 2^{\frac{s}{2}} + Z_4 \cdot 2^s)/2^s$    {ARM}

7: if $Z \geq M$ then
8:     $Z \leftarrow Z - M$                                                            {ARM}
9: return $Z$
```

$$\begin{tikzcd}
\text{NEON} & \text{ARM} \arrow[l, "\text{Operand passing}"']
\end{tikzcd}$$

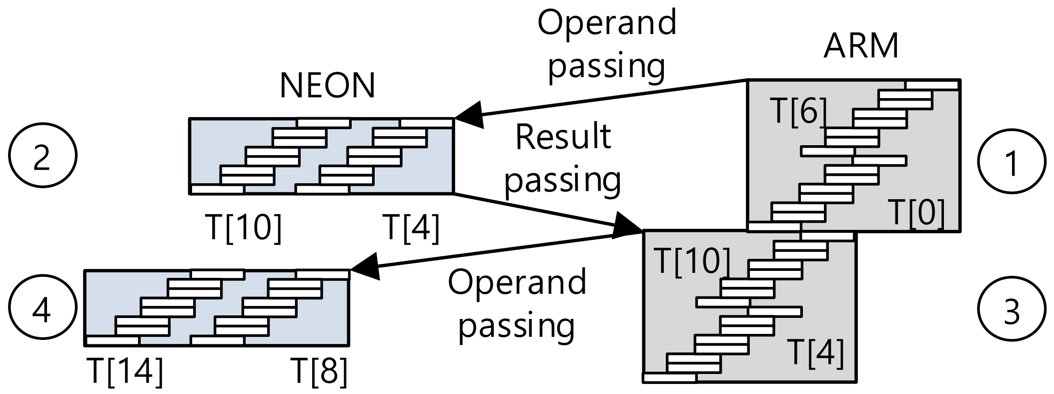
Algorithm 4 Unified ARM/NEON Montgomery reduction

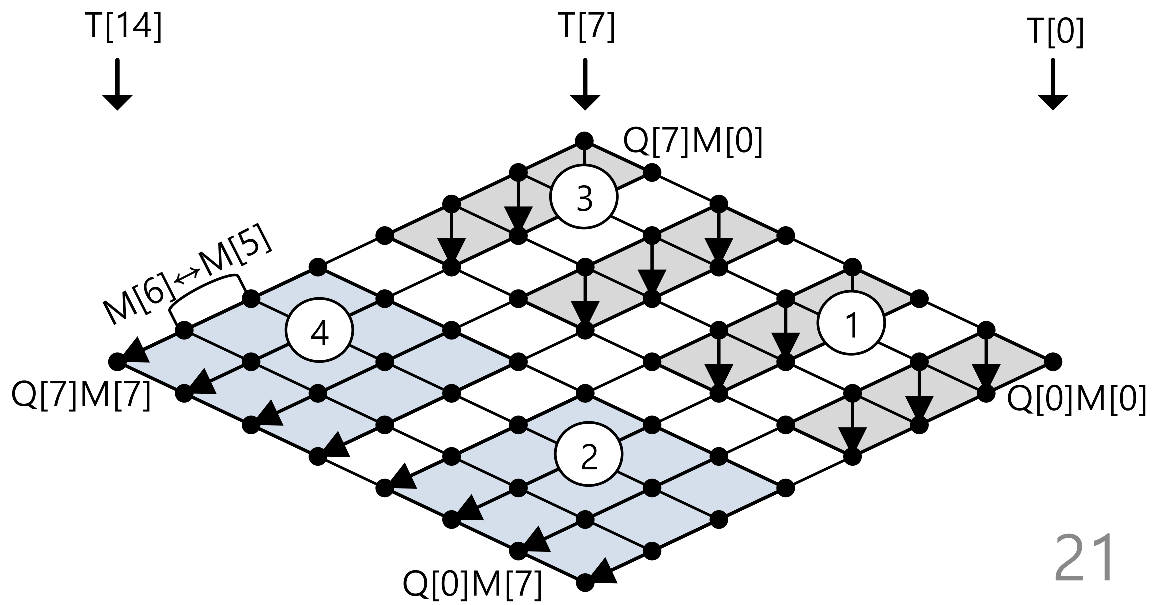
```
Input: An odd $m$-bit modulus $M = (M_H \| M_L)$, the Montgomery radix $R = 2^s$, where
    $s = wn$ with $w = 32$ and $n = \lceil m/w \rceil$, an operand $T \in [0, M^2 - 1]$, and pre-computed
    constant $M' = -M^{-1} \bmod R$
Output: $m$-bit Montgomery product $Z = T \cdot R^{-1} \bmod M$
1: $Q = (Q_H \| Q_L) \leftarrow T \cdot M' \bmod R$                                {ARM}

   Interleaved section begin
2: $Z_1 \leftarrow M_L \cdot Q_L$                                                   {ARM}
3: $Z_2 \leftarrow M_H \cdot Q_L$                                                   {NEON}
4: $Z_3 \leftarrow M_L \cdot Q_H$                                                   {ARM}
5: $Z_4 \leftarrow M_H \cdot Q_H$                                                   {NEON}
   Interleaved section end

6: $Z \leftarrow (T + Z_1 + (Z_2 + Z_3) \cdot 2^{\frac{s}{2}} + Z_4 \cdot 2^s)/2^s$        {ARM}

7: if $Z \geq M$ then
8:     $Z \leftarrow Z - M$                                                         {ARM}
9: return $Z$
```





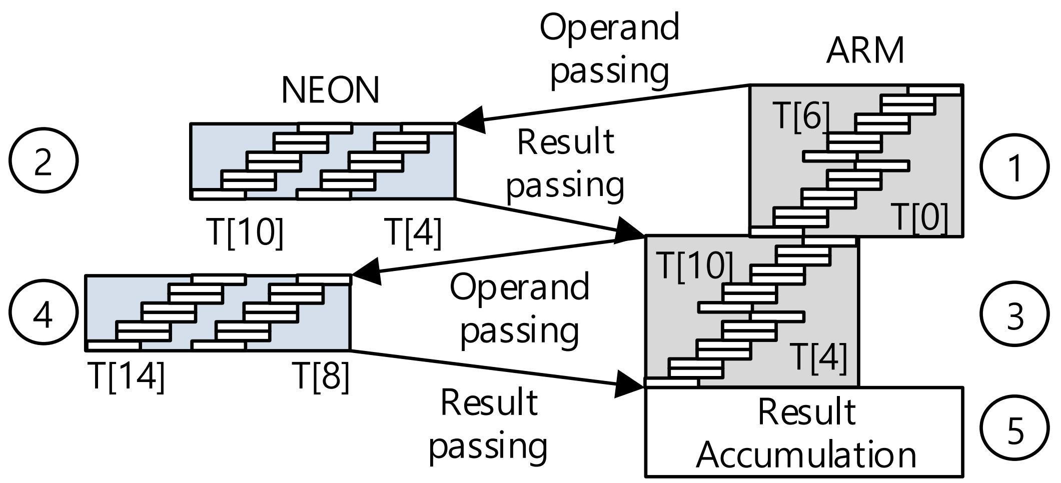
Algorithm 4 Unified ARM/NEON Montgomery reduction

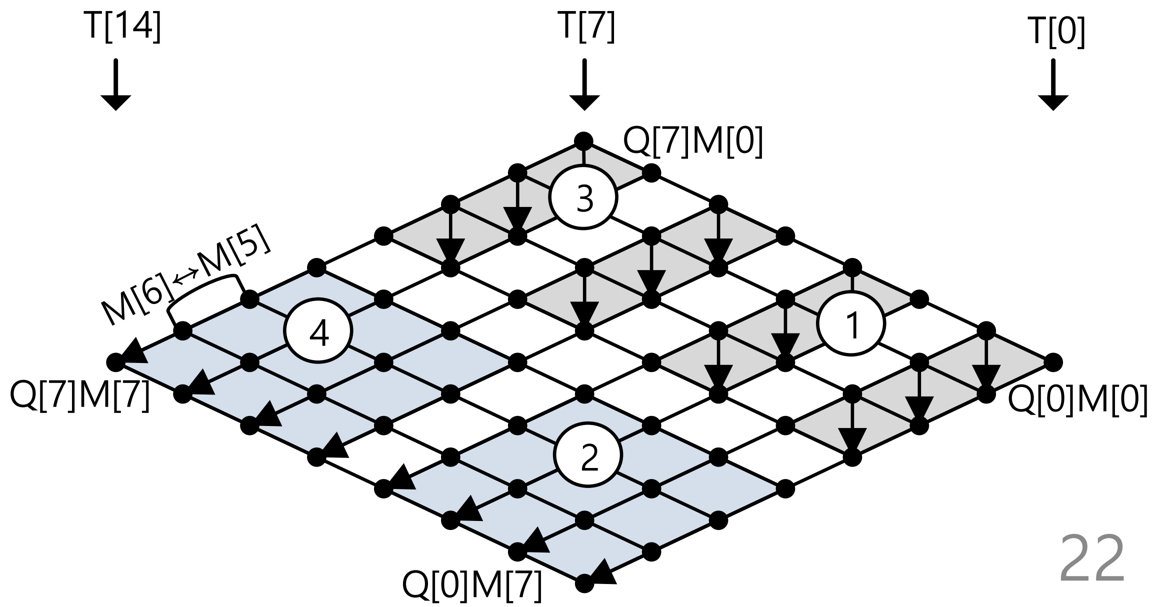
```
Input: An odd $m$-bit modulus $M = (M_H \| M_L)$, the Montgomery radix $R = 2^s$, where
    $s = wn$ with $w = 32$ and $n = \lceil m/w \rceil$, an operand $T \in [0, M^2 - 1]$, and pre-computed
    constant $M' = -M^{-1} \bmod R$
Output: $m$-bit Montgomery product $Z = T \cdot R^{-1} \bmod M$
1: $Q = (Q_H \| Q_L) \leftarrow T \cdot M' \bmod R$                          {ARM}

   Interleaved section begin
2: $Z_1 \leftarrow M_L \cdot Q_L$                                             {ARM}
3: $Z_2 \leftarrow M_H \cdot Q_L$                                             {NEON}
4: $Z_3 \leftarrow M_L \cdot Q_H$                                             {ARM}
5: $Z_4 \leftarrow M_H \cdot Q_H$                                             {NEON}
   Interleaved section end

6: $Z \leftarrow (T + Z_1 + (Z_2 + Z_3) \cdot 2^{\frac{s}{2}} + Z_4 \cdot 2^s)/2^s$   {ARM}

7: if $Z \geq M$ then
8:     $Z \leftarrow Z - M$                                                   {ARM}
9: return $Z$
```





Modular Reduction for SIDH

- **Efficient Montgomery reduction: Montgomery-friendly modulus**
  - The lower word of the modulus is $\mathbf{2^w - 1}$
    → Montgomery constant is equal to 1.

  - Multiplications with an **all-ones** word
    ($T \times 0xFFFFFFFF$ → $T \times 2^{32} - T$): **shifts and subtractions**
    - (e.g., $p503 = 2^{250}3^{159} - 1$)

0x4066F541811E1E6045C6BDDA77A4D01B9BF6C87B7E7DAF13085BDA2211E7A0AB**FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF**
(in hexadecimal)

  - A modulus ***M+1*** turns the lower part of the modulus into **all-zero** words
    - (e.g., $p503 + 1 = 2^{250}3^{159}$)

0x4066F541811E1E6045C6BDDA77A4D01B9BF6C87B7E7DAF13085BDA2211E7A0AC**00000000000000000000000000000000000000000000000000000000000000**
(in hexadecimal)

Proposed Modular Reduction for SIDH **(32-bit ARMv7)**

- *m*-bit modular reduction using Montgomery reduction
    - **One $m/2$-bit multiplication in ARM**
    - **One $m/2$-bit multiplication in NEON**

Algorithm 5 Unified ARM/NEON Montgomery reduction for SIDH-friendly primes

```
Input: $\tilde{M} = M + 1 = (\tilde{M}_H \| \tilde{M}_L)$ for an odd $m$-bit modulus $M$, the Montgomery radix
    $R = 2^s$, where $s = wn$ with $w = 32$ and $n = \lceil m/w \rceil$, an operand $T \in [0, M^2 - 1]$, and
    pre-computed constant $M' = -M^{-1} \bmod R$
Output: $m$-bit Montgomery product $Z = T \cdot R^{-1} \bmod M$
1: Set $Q = (Q_H \| Q_L) \leftarrow T \cdot M' \bmod 2^s$

   Interleaved section begin
2: $T \leftarrow T + (\tilde{M}_H \cdot Q_L) \cdot 2^{\frac{s}{2}}$            {ARM}
3: $Z_4 \leftarrow \tilde{M}_H \cdot Q_H$                                        {NEON}
   Interleaved section end

4: $Z \leftarrow (T + Z_4 \cdot 2^s - Q)/2^s$                                    {ARM}
5: if $Z \geq M$ then
6:     $Z \leftarrow Z - M$                                                      {ARM}
7: return $Z$
```

Algorithm 5 Unified ARM/NEON Montgomery reduction for SIDH-friendly primes

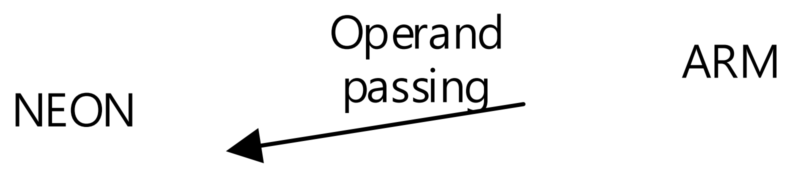
Input: $\tilde{M} = M + 1 = (\tilde{M}_H \| \tilde{M}_L)$ for an odd m-bit modulus M, the Montgomery radix $R = 2^s$, where $s = wn$ with $w = 32$ and $n = \lceil m/w \rceil$, an operand $T \in [0, M^2 - 1]$, and pre-computed constant $M' = -M^{-1} \bmod R$

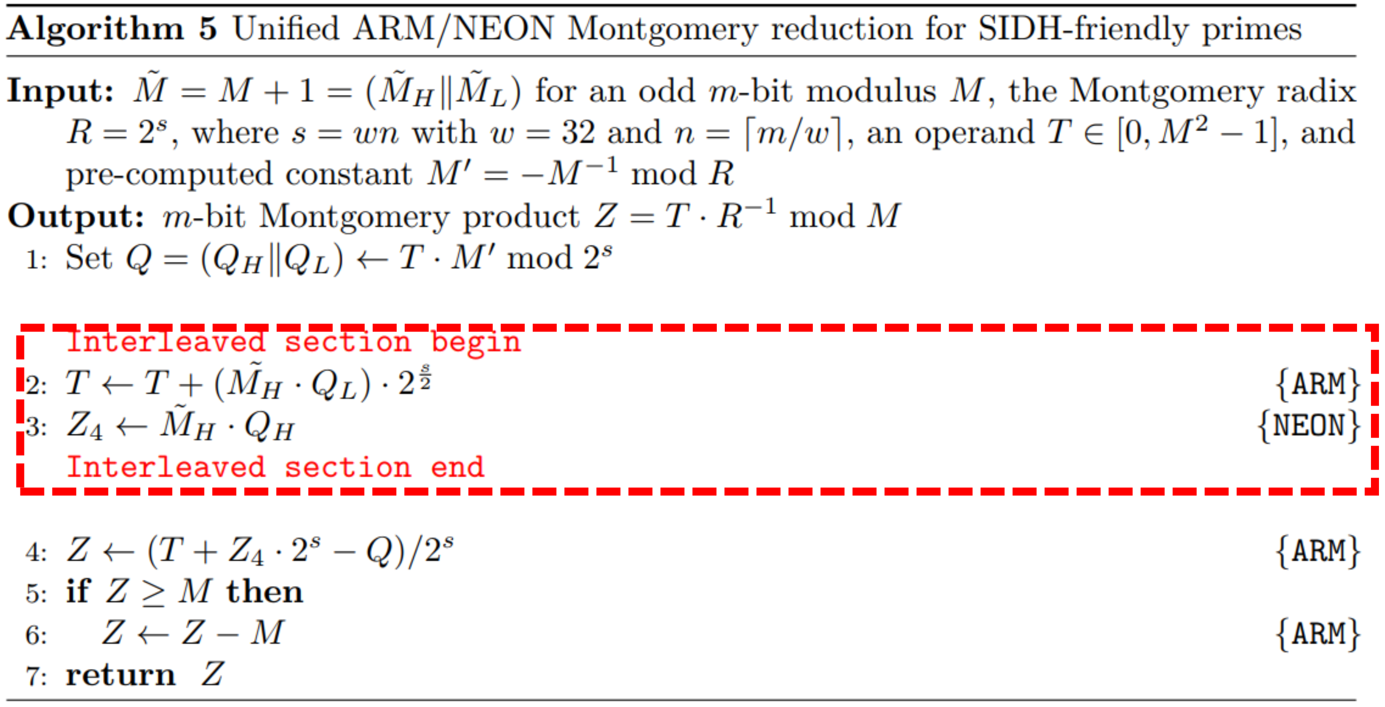
Output: m-bit Montgomery product $Z = T \cdot R^{-1} \bmod M$

```
1: Set $Q = (Q_H \| Q_L) \leftarrow T \cdot M' \bmod 2^s$

   Interleaved section begin
2: $T \leftarrow T + (\tilde{M}_H \cdot Q_L) \cdot 2^{\frac{s}{2}}$        {ARM}
3: $Z_4 \leftarrow \tilde{M}_H \cdot Q_H$                                   {NEON}
   Interleaved section end

4: $Z \leftarrow (T + Z_4 \cdot 2^s - Q)/2^s$                              {ARM}
5: if $Z \geq M$ then
6:     $Z \leftarrow Z - M$                                                {ARM}
7: return $Z$
```

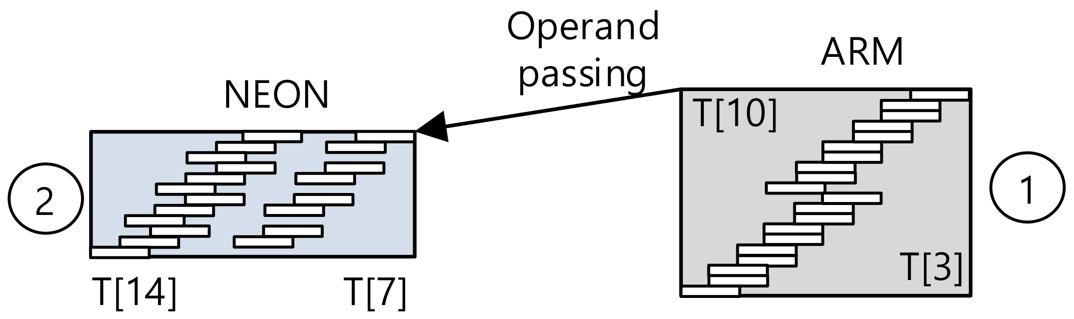


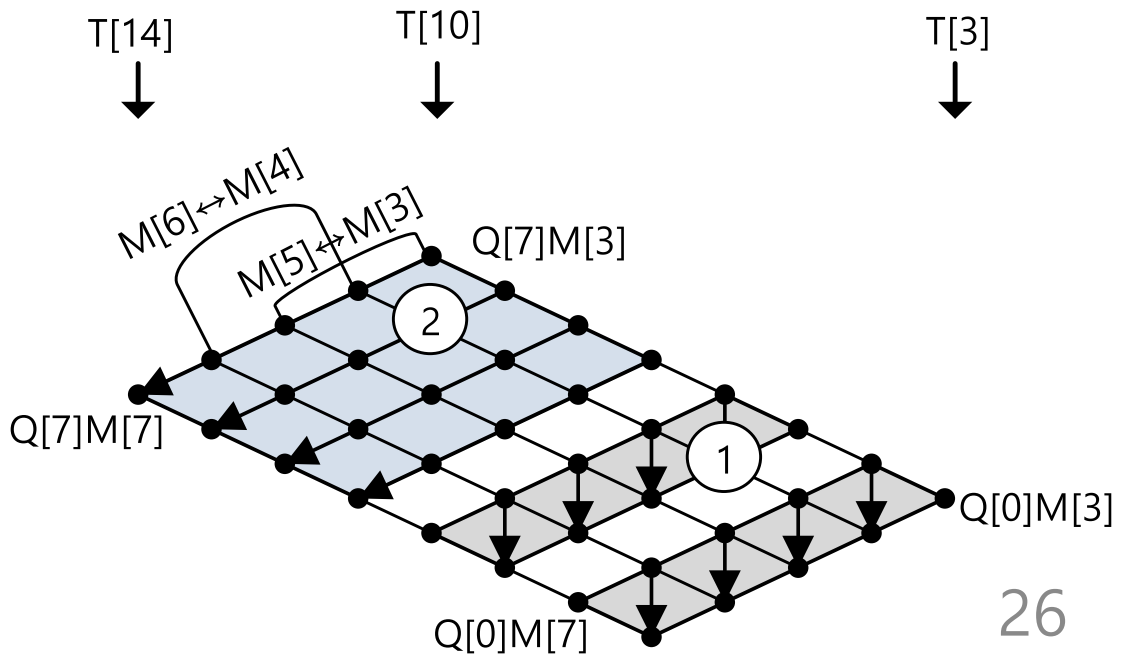


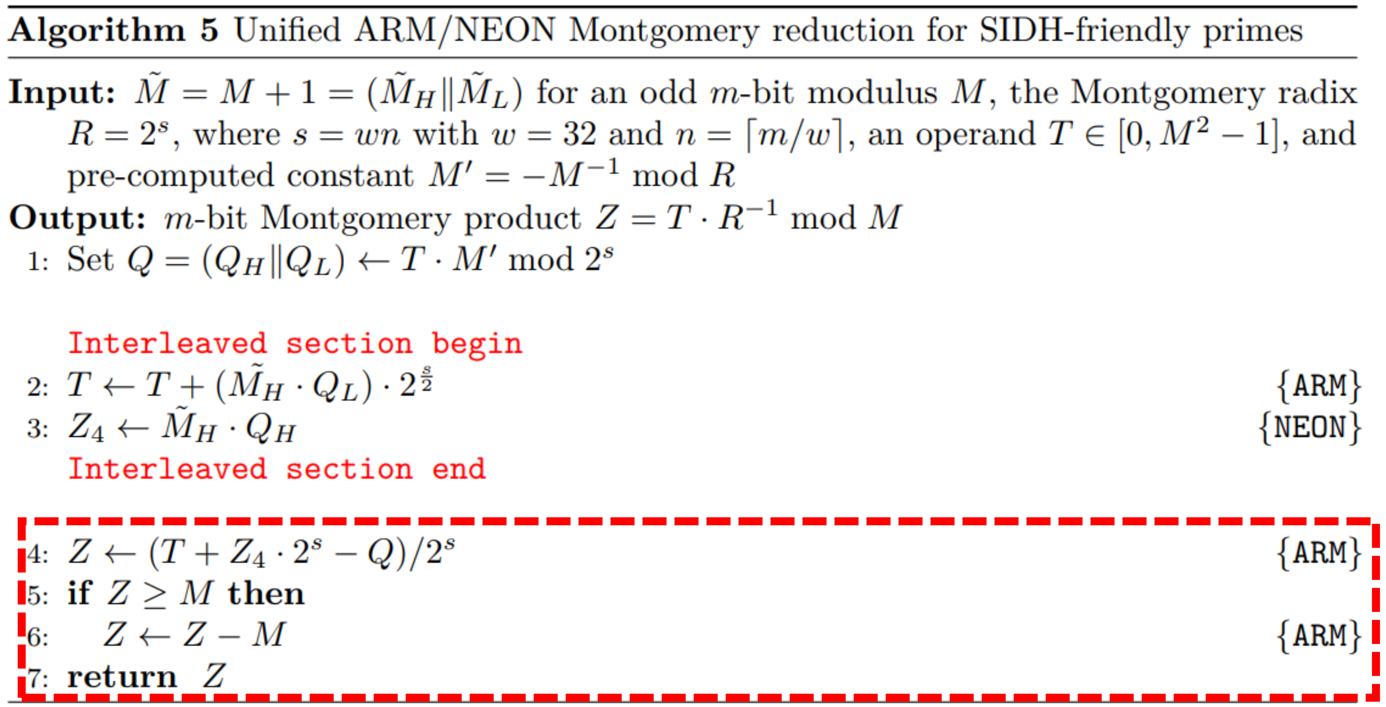
```
Algorithm 5 Unified ARM/NEON Montgomery reduction for SIDH-friendly primes
Input: $\tilde{M} = M + 1 = (\tilde{M}_H \| \tilde{M}_L)$ for an odd $m$-bit modulus $M$, the Montgomery radix
    $R = 2^s$, where $s = wn$ with $w = 32$ and $n = \lceil m/w \rceil$, an operand $T \in [0, M^2 - 1]$, and
    pre-computed constant $M' = -M^{-1} \bmod R$
Output: $m$-bit Montgomery product $Z = T \cdot R^{-1} \bmod M$
1: Set $Q = (Q_H \| Q_L) \leftarrow T \cdot M' \bmod 2^s$

    Interleaved section begin
2: $T \leftarrow T + (\tilde{M}_H \cdot Q_L) \cdot 2^{\frac{s}{2}}$                {ARM}
3: $Z_4 \leftarrow \tilde{M}_H \cdot Q_H$                                {NEON}
    Interleaved section end

4: $Z \leftarrow (T + Z_4 \cdot 2^s - Q)/2^s$                    {ARM}
5: if $Z \geq M$ then
6:     $Z \leftarrow Z - M$                                 {ARM}
7: return $Z$
```






Algorithm 5 Unified ARM/NEON Montgomery reduction for SIDH-friendly primes

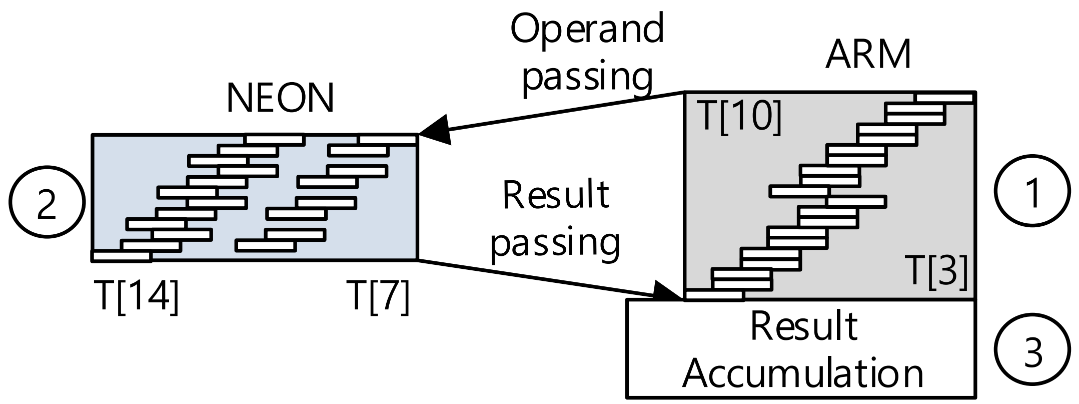
Input: $\tilde{M} = M + 1 = (\tilde{M}_H \| \tilde{M}_L)$ for an odd m-bit modulus M, the Montgomery radix $R = 2^s$, where $s = wn$ with $w = 32$ and $n = \lceil m/w \rceil$, an operand $T \in [0, M^2 - 1]$, and pre-computed constant $M' = -M^{-1} \bmod R$

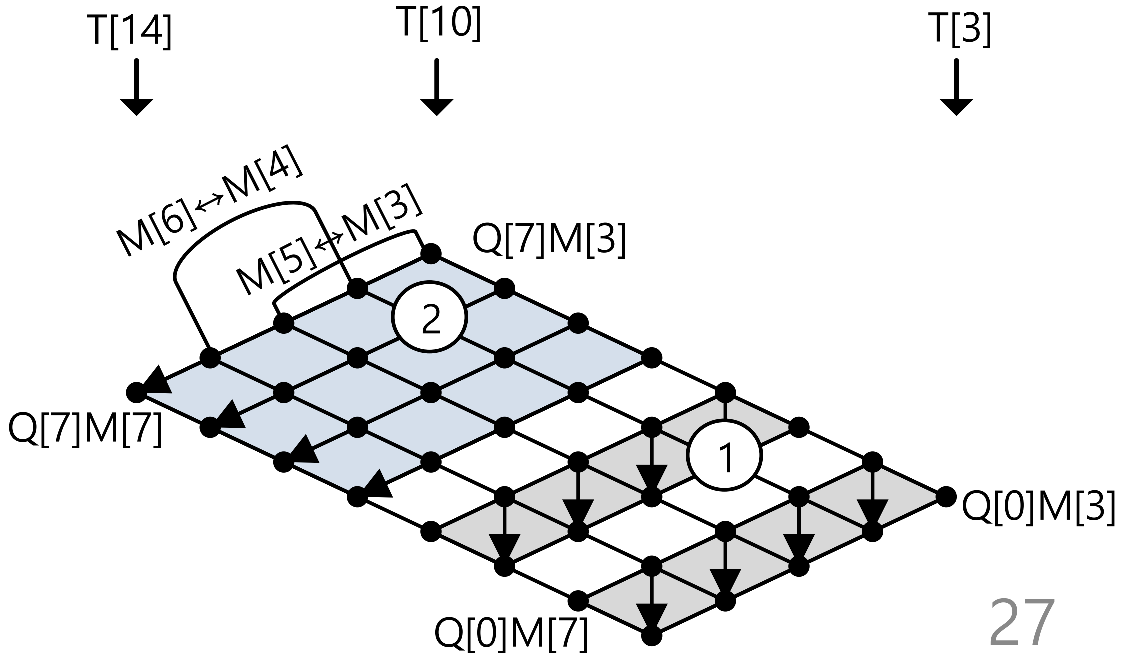
Output: m-bit Montgomery product $Z = T \cdot R^{-1} \bmod M$

```
1: Set $Q = (Q_H \| Q_L) \leftarrow T \cdot M' \bmod 2^s$

   Interleaved section begin
2: $T \leftarrow T + (\tilde{M}_H \cdot Q_L) \cdot 2^{\frac{s}{2}}$        {ARM}
3: $Z_4 \leftarrow \tilde{M}_H \cdot Q_H$        {NEON}
   Interleaved section end

4: $Z \leftarrow (T + Z_4 \cdot 2^s - Q)/2^s$        {ARM}
5: if $Z \geq M$ then
6:     $Z \leftarrow Z - M$        {ARM}
7: return $Z$
```

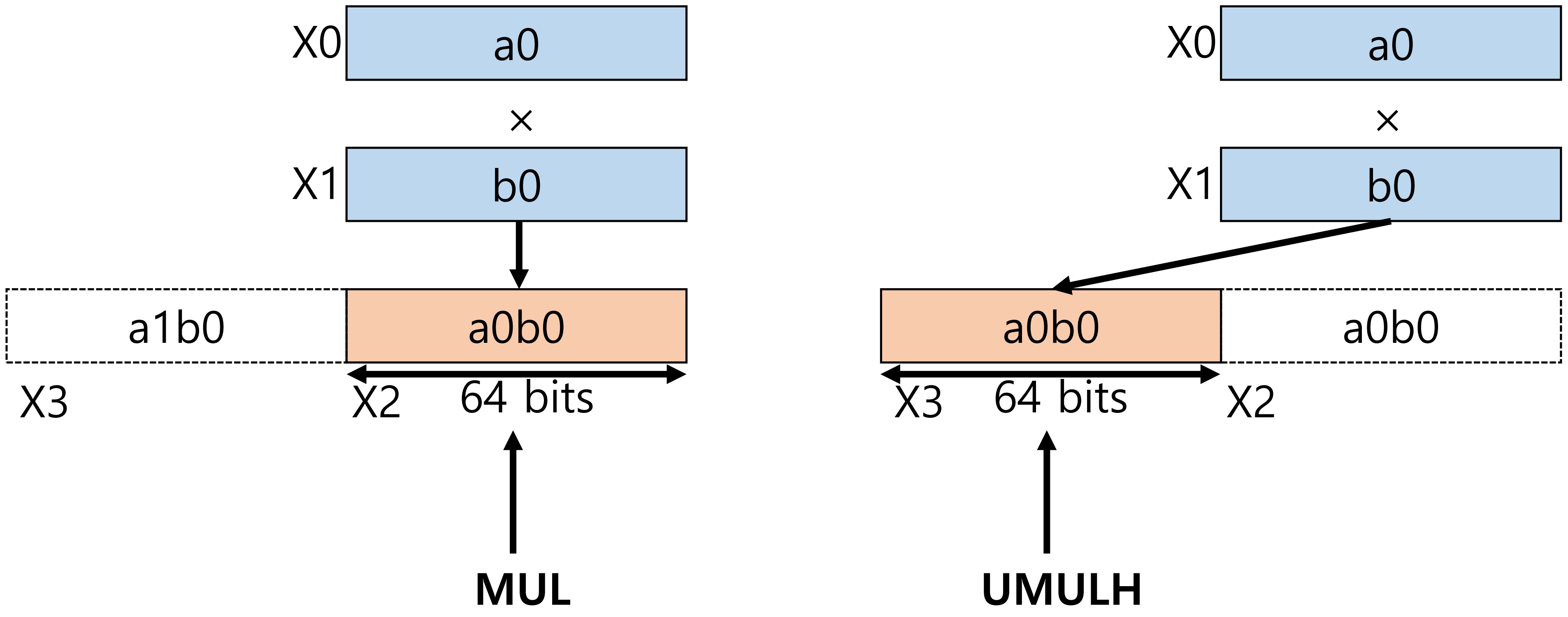




Outline

- Short Overview
- Post-quantum supersingular isogeny Diffie-Hellman (SIDH) key exchange
  - Supersingular isogeny key encapsulation (SIKE) protocol
- Our implementation
  - Optimized implementations for 32-bit ARMv7
  - Optimized implementations for 64-bit ARMv8
- Implementation results
- Conclusion

Multiplication Instruction **(64-bit ARMv8)**



Proposed Multiprecision Multiplication **(64-bit ARMv8)**

- **Instruction level parallelism** (high throughput)

Instruction	Instruction group	Latency [cc]	Throughput [cc]
ADD/ADC/SUB/SBC	ALU, basic	1	2
MUL	Multiply	3	1/3
UMULH	Multiply high	6	1/4

```
...
MUL X0, X4, X5
UMULH X1, X4, X5
ADDS X10, X10, X2
ADCS X11, X11, X3
ADC X12, XZR, XZR

MUL X2, X6, X7
UMULH X3, X6, X7
ADDS X10, X10, X0
ADCS X11, X11, X1
ADC X12, X12, XZR
...
```

- Based on these features, **the 128-bit multiplication using column-wise multiplication** is implemented at the lowest level.
  - 128-bit (column-wise multiplication) → 256-bit (1-level Karatsuba) → **512-bit (2-level Karatsuba)**

Outline

- Short Overview
- Post-quantum supersingular isogeny Diffie-Hellman (SIDH) key exchange
  - Supersingular isogeny key encapsulation (SIKE) protocol
- Our implementation
  - Optimized implementations for 32-bit ARMv7
  - Optimized implementations for 64-bit ARMv8
- Implementation results
- Conclusion

Results (32-bit ARM Cortex-A15)

SIDHp503 is about **1.7x** faster than Koziel et al.'s, and **13x** faster than (generic C) Microsoft SIDH v3.0 library

Implementation	Language	Instruction	Timings $[cc]$	Timings $[cc \times 10^6]$				
			$\mathbb{F}_p$ mul	Alice R1	Bob R1	Alice R2	Bob R2	Total
SIDHp503								
SIDH v3.0	C	Generic	8,947	597	657	487	555	2,296
Koziel et al.	ASM	NEON	1,372	83	87	66	68	302
This work	**ASM**	**ARM/NEON**	**780**	**46**	**50**	**38**	**42**	**176**

SIDHp751								
SIDH v3.0	C	Generic	36,592	2,006	2,256	1,650	1,924	7,836
Koziel et al.	C	Generic	N/A	437	474	346	375	1,632
This work	**ASM**	**ARM/NEON**	**1,502**	**150**	**170**	**120**	**144**	**584**

Results **(64-bit ARM Cortex-A53/A72)**

SIDHp503 is about **1.3x** faster than Campagna et al.'s, and **4.8x** faster than (generic C) Microsoft SIDH v3.0 library

Implementation	Language	Processor	Timings $[cc]$	Timings $[cc \times 10^6]$				
			$\mathbb{F}_p$ mul	Alice R1	Bob R1	Alice R2	Bob R2	Total
SIDHp503								
SIDH v3.0	C	Cortex-A53	4,453	167.2	136.2	184.5	155.9	643.8
Campagna et al.	ASM		1,187	44.0	35.9	48.7	41.2	169.8
This work	**ASM**		**971**	**34.5**	**28.1**	**38.3**	**32.4**	**133.3**
SIDH v3.0	C	Cortex-A72	3,942	149.1	121.5	164.3	139.4	574.3
Campagna et al.	ASM		865	28.8	23.4	31.7	26.9	110.8
This work	**ASM**		**753**	**23.4**	**19.1**	**25.9**	**21.9**	**90.3**

Results **(64-bit ARM Cortex-A53/A72)**

SIKEp503 is about **1.3x** faster than Campagna et al.'s, and **4.8x** faster than (generic C) Microsoft SIDH v3.0 library

Implementation	Language	Processor	Timings $[cc]$	Timings $[cc \times 10^6]$			
			$\mathbb{F}_p$ mul	KeyGen	Encaps	Decaps	Total
SIKEp503							
SIDH v3.0	C	Cortex-A53	4,453	184.5	303.3	323.0	626.3
Campagna et al.	ASM		1,187	48.8	80.0	85.3	165.3
This work	**ASM**		**971**	**38.4**	**62.7**	**66.9**	**129.6**
SIDH v3.0	C	Cortex-A72	3,942	164.4	270.6	287.9	558.5
Campagna et al.	ASM		865	31.8	52.2	55.6	107.8
This work	**ASM**		**753**	**25.9**	**42.5**	**45.3**	**87.8**

Outline

- Short Overview
- Post-quantum supersingular isogeny Diffie-Hellman (SIDH) key exchange
  - Supersingular isogeny key encapsulation (SIKE) protocol
- Our implementation
  - Optimized implementations for 32-bit ARMv7
  - Optimized implementations for 64-bit ARMv8
- Implementation results
- Conclusion

Conclusion

- **New implementations of modular multiplication on ARM**
  - Faster multi-precision multiplication
  - Faster Montgomery reduction
- **Record-setting SIDH and SIKE implementations**
  - On both 32-bit and 64-bit ARM
- **SIDH Library:** https://github.com/Microsoft/PQCrypto-SIDH
  - 32-bit ARMv7 code is coming soon!
  - 64-bit ARMv8 code is **already integrated**

Thank you for your attention!