SIKE Round 2 Speed Record on ARM Cortex-M4

Hwajeong Seo (Hansung University), SK
Amir Jalali (Florida Atlantic University), USA
Reza Azarderakhsh (Florida Atlantic University, PQSecure Technologies), USA

Presenter: Hyeokdong Kwon (Hansung University)

Outline

- Short Overview
- Supersingular isogeny key encapsulation (SIKE) protocol
- Our implementation
- Implementation results
- Conclusion

Quantum Threat to Information Security

Large-scale quantum computers could break some encryption schemes

Need to migrate encryption to quantum-resistant algorithms

When we should start the process?

Questions:

- When will quantum computers arrive?
- When will they be a threat?
- When do we switch?
- How do we have live and agile transition?
- What do we transition to?
- …

"In 25 years we'll be working on quantum resistant crypto standardization b ased on elliptic curves in some way!"

—Dustin Moody, NIST (NSF CCC Workshop 2019)

Post-Quantum Cryptography **(Isogeny)**

- **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**…

Quick Overview

- **[2006]:** Birth of a supersingular isogeny-based cryptosystem
  - **Charles** – **Goren** – **Lauter**
  - built hash function from supersingular isogeny graph
- **[2011]:** Supersingular isogeny key exchange
  - Jao – De Feo
- **[2017]:** Supersingular isogeny key encapsulation
  - SIKE Team

Post-Quantum Cryptography **(Isogeny)**

- Round1 (2017. 12) : 69 accepted as “complete and proper” (5 withdrew)
- Round2 (2019. 01) : 26 accepted including **SIKE**

- **SIKE Round2**
  - New parameters: SIKEp434, SIKEp610
  - Stronger security: SIKEp503 (1→2) SIKEp751 (3→5)

SIKE Team


UNIVERSITY OF
WATERLOO


UNIVERSITÉ DE
VERSAILLES
SAINT-QUENTIN-EN-YVELINES


FAU
FLORIDA ATLANTIC
UNIVERSITY


IN·DEI·NOMINE·FELICITER
Radboud
Universiteit


UNIVERSITY OF
TORONTO


Microsoft Research


amazon


LinkedIn


TEXAS
INSTRUMENTS


INFOSEC
GLOBAL


PQSecure


evolution

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.

Motivation

What NIST wants

- Performance (hardware+software) will play more of a role
  - More benchmarks
  - For hardware, NIST asks to focus on Cortex M4 (with all options) and Artix-7
    - pqc-hardware-forum
- Continued research and analysis on **ALL** of the 2nd round candidates
- See how submissions fit into applications/procotols. Any constraints?

- In this talk, we address this problem on **32-bit ARM Cortex-M4 processors.**

PQC on Low-end Microcontroller

- **STM32F4DISCOVERY**
  - ARM Cortex-M4
  - 32-bit, ARMv7E-M
  - 192 KB RAM, 168 MHz

- **Benchmark framework**
  - PQM4: github.com/mupq/pqm4

Previous Works on SIKE/SIDH

- **64-bit ARM-A processor:**
  - Jalali et al. [SAC'17] → Seo et al. [CHES'18] → Jalali et al. [TDSC'19]→ Seo et al. [WISA'19]

- **32-bit ARM-A processor:**
  - Koziel et al. [CANS'16] → Jalali et al. [SPACE'18] → Seo et al. [CHES'18]

- **32-bit ARM-M processor:**
  - **This work [CANS'19]**

Contribution

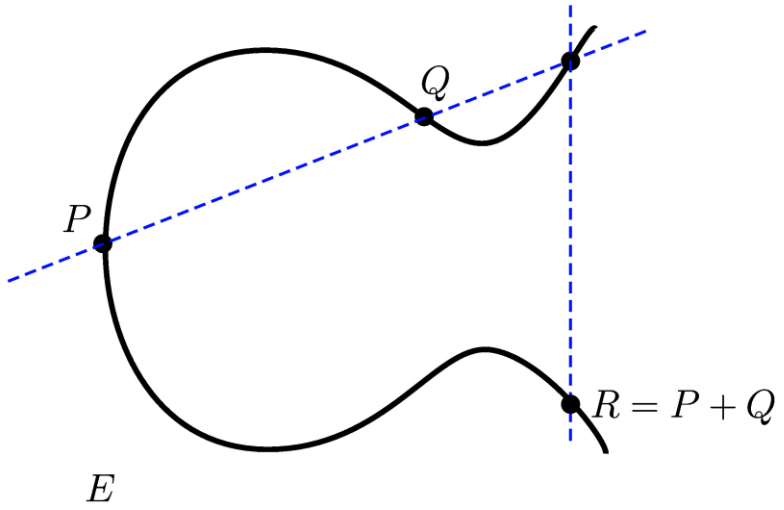
- **Optimized finite field arithmetic: super-scalar implementation**
- **Scalable implementation: well scheduled instruction**
- **Efficient Implementation of SIKE:**
  - p434 (**1.94 sec**) on 32-bit ARM Cortex-M4 **@168MHz**

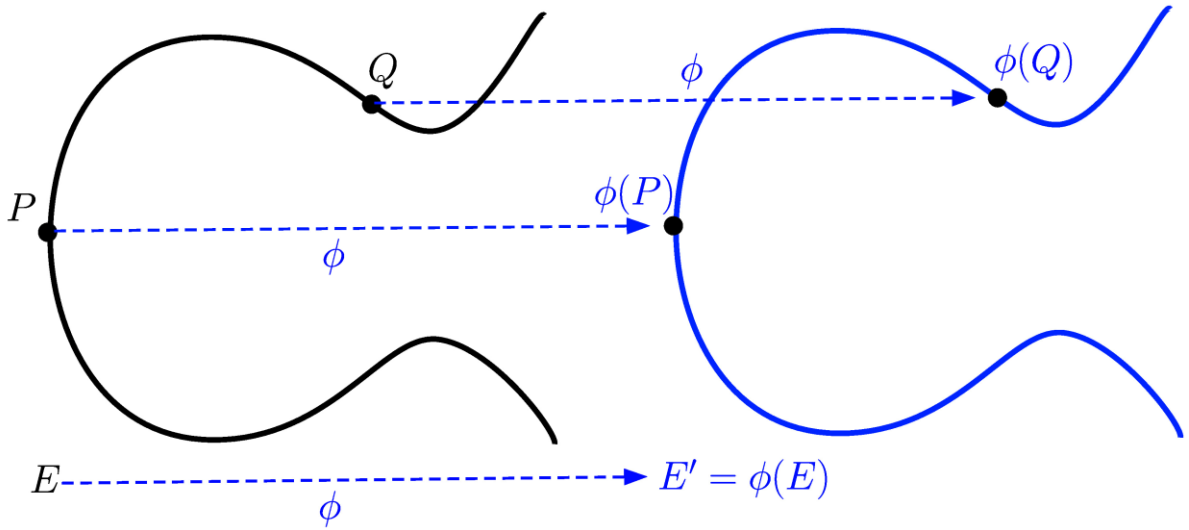
Outline

- Short Overview
- Supersingular isogeny key encapsulation (SIKE) protocol
- Our implementation
- Implementation results
- Conclusion

Isogeny-Based Cryptography

- Isogeny-based cryptography is constructed on a set of curves
  .
- Given two curve E and $E' = \phi(E)$ find ϕ?



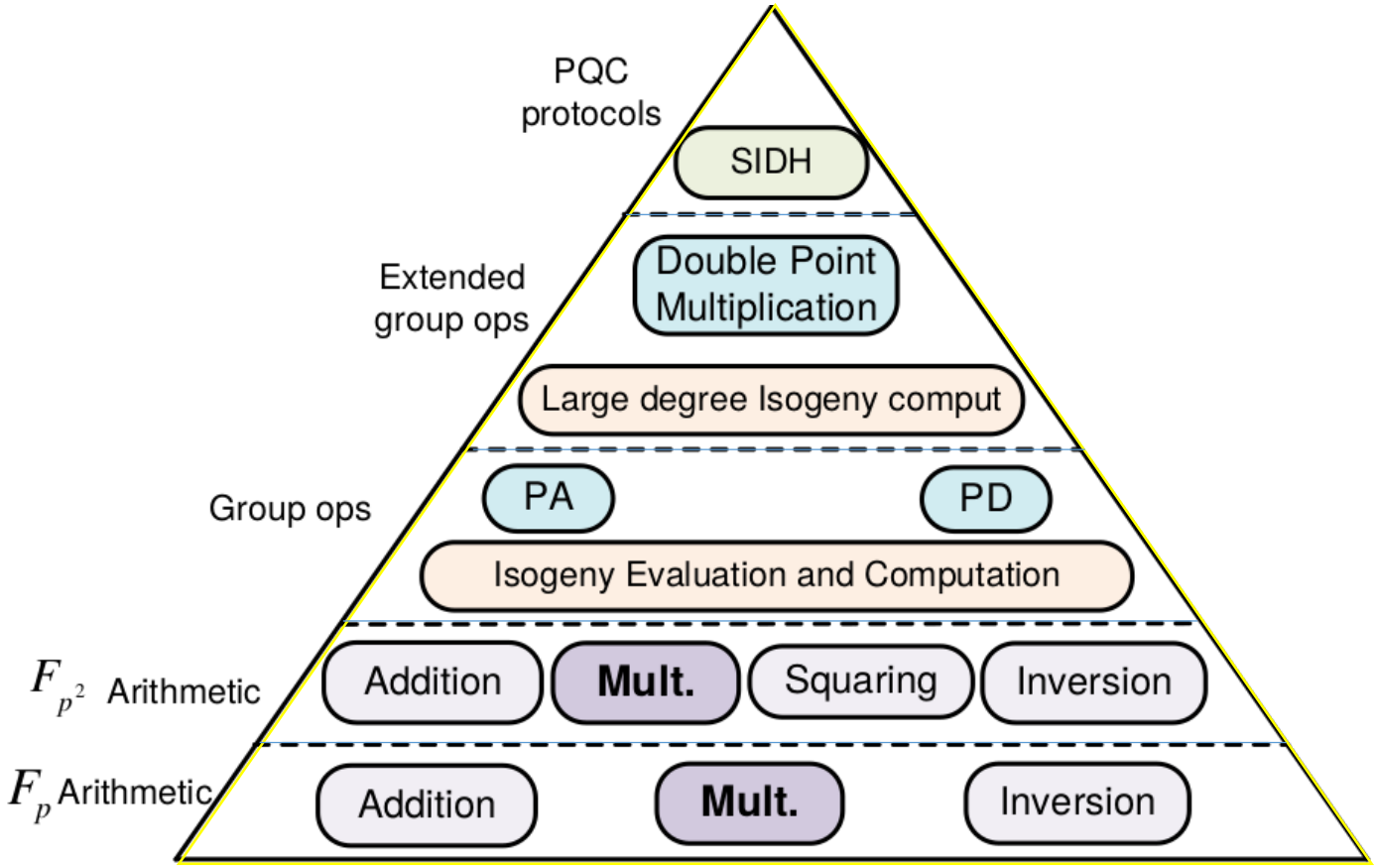


Post-quantum key exchange algorithm

- 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.

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

SIDH Computations



Supersingular Isogeny Key Encapsulation **(SIKE)**

- SIDH is not secure when keys are reused (Galbraith-Petit-Shani-Ti 2016)
- **SIKE:** IND-CCA secure key encapsulation based on SIDH.
- Uses a variant of Hofheinz-Hövelmanns-Kiltz (HHK) transform:
  IND-CPA PKE → IND-CCA KEM

- Starting curve $E_0/\mathbb{F}_{p^2}: y^2 = x^3 + x$, (Round 1) switched to $E_0/\mathbb{F}_{p^2}: y^2 = x^3 + 6x^2 + x$ (Round 2),
- where $p = 2^{eA}3^{eB} - 1$

SIKE Key sizes

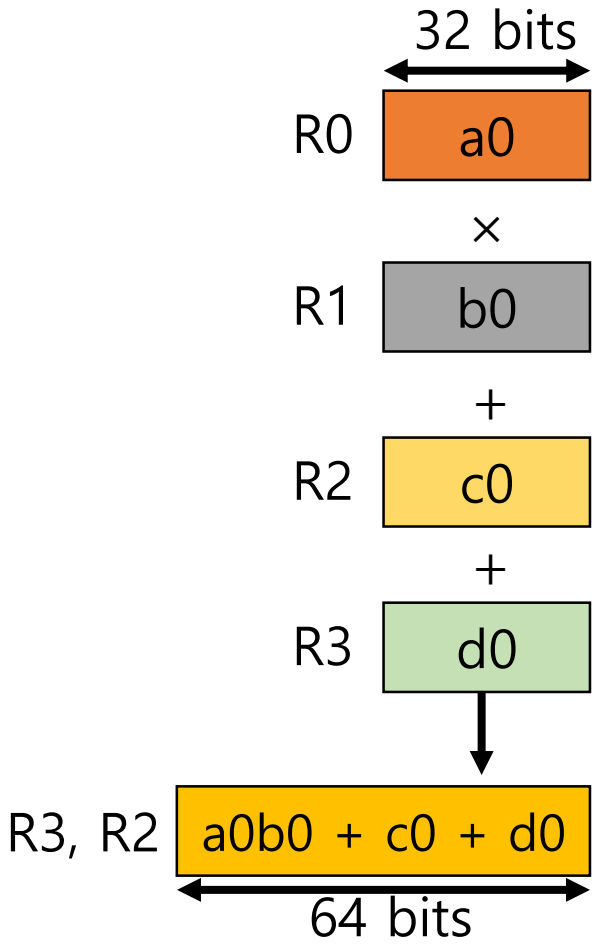
NIST Level	Prime size (bits)	Prime	Public key size (bytes)	Compressed PK size (bytes)
1	434	$2^{216}3^{137}-1$	330	196
2	503	$2^{250}3^{159}-1$	378	224
3	610	$2^{305}3^{192}-1$	462	273
5	751	$2^{372}3^{239}-1$	564	331

Outline

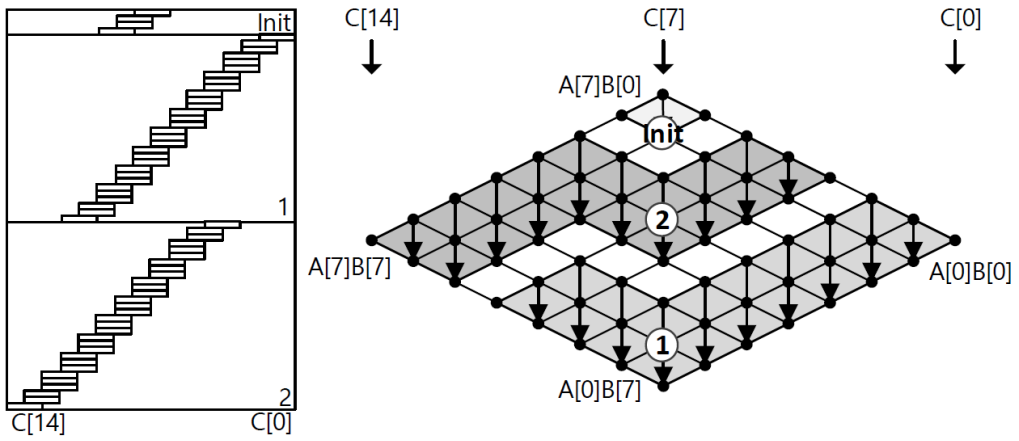
- Short Overview
- Supersingular isogeny key encapsulation (SIKE) protocol
- Our implementation
- Implementation results
- Conclusion

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

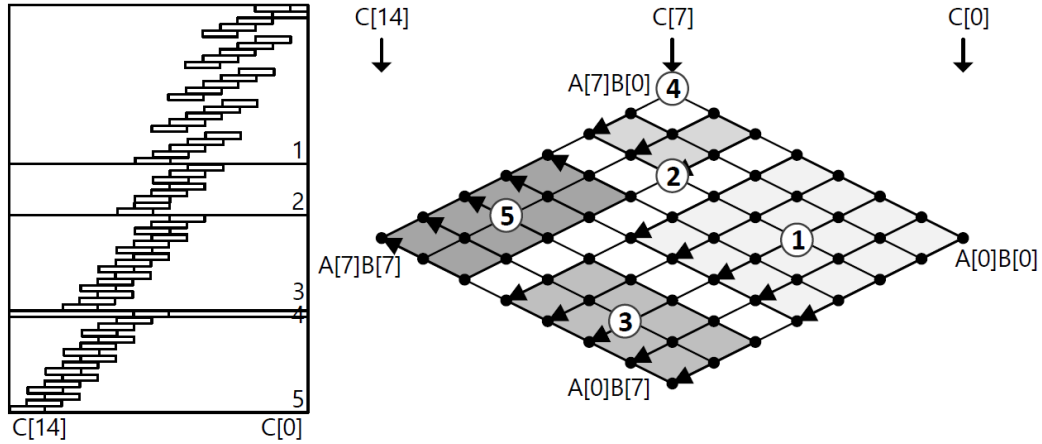
UMAAL R2, R3, R0, R1



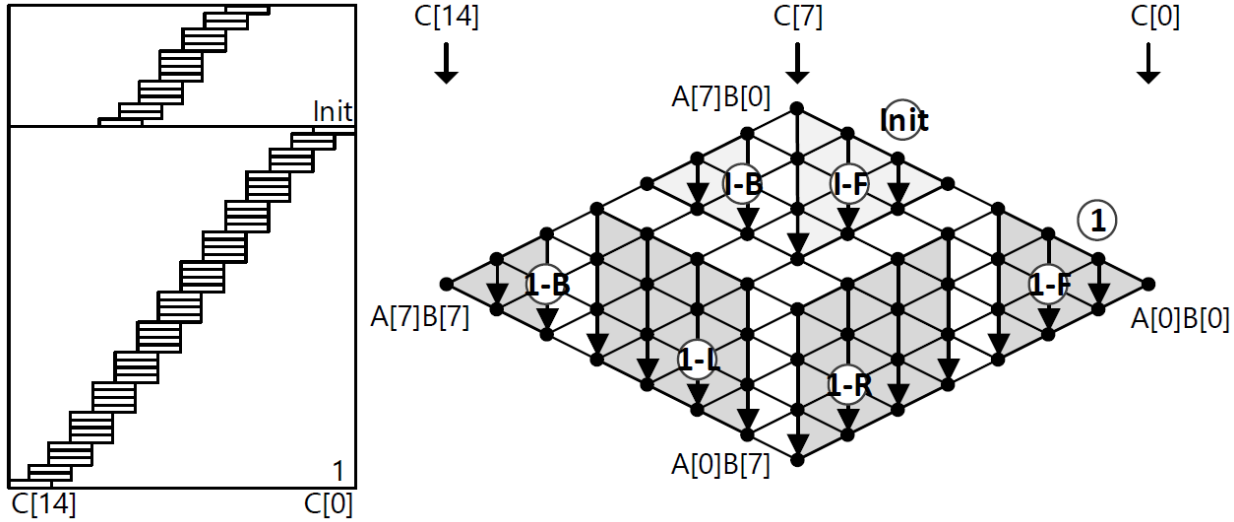
Comparison of Multiprecision Multiplication **(32-bit ARMv7-M)**



Consecutive Operand Caching (**COC**)



Operand Scanning (**OS**)



Proposed Refined Operand Caching (**ROC**)

Requirements for ROC
→ More registers

Comparison of Register Utilization

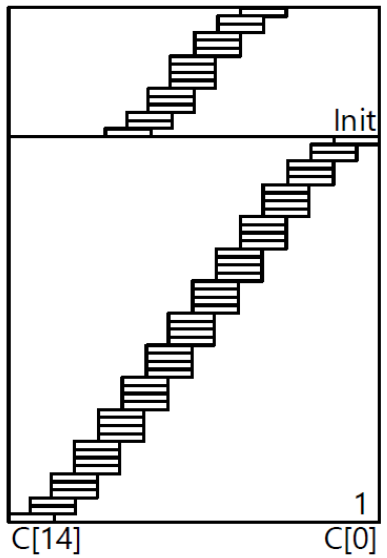
Registers	Fujii et al. [11]	Haase et al. [13]	This work
R0	Result pointer	Temporal pointer	Temporal pointer
R1	Operand A pointer	Operand A #1	Temporal register #1
R2	Operand B pointer	Operand B #1	Operand A #1
R3	Result #1	Operand B #2	Operand A #2
R4	Result #2	Operand B #3	Operand A #3
R5	Result #3	Operand B #4	Operand A #4
R6	Operand A #1	Operand B #5	Operand B #1
R7	Operand A #2	Result #1	Operand B #2
R8	Operand A #3	Result #2	Operand B #3
R9	Operand B #1	Result #3	Operand B #4
R10	Operand B #2	Result #4	Result #1
R11	Operand B #3	Result #5	Result #2
R12	Temporal register #1	Temporal register #1	Result #3
R13; SP	Stack pointer	Stack pointer	Stack pointer
R14; LR	Temporal register #2	Temporal register #2	Result #4
R15; PC	Program counter	Program counter	Program counter

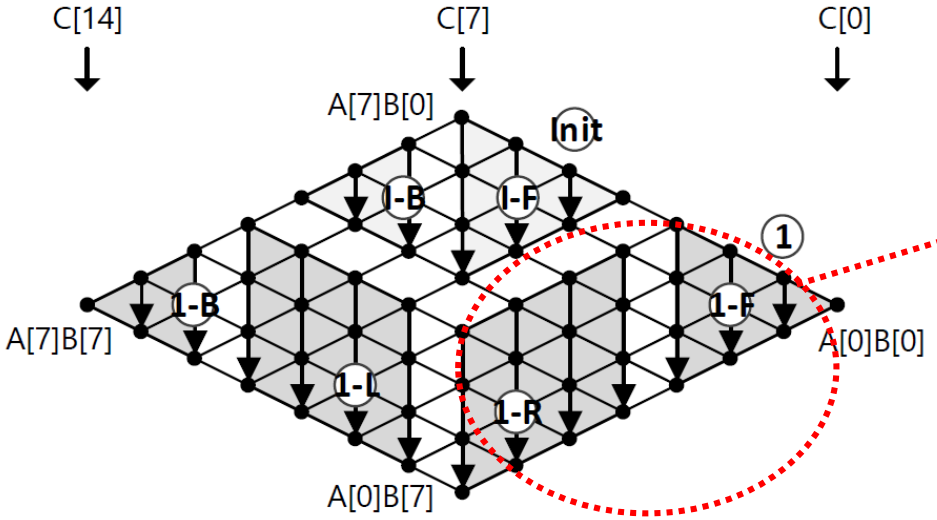
Comparison of Performance

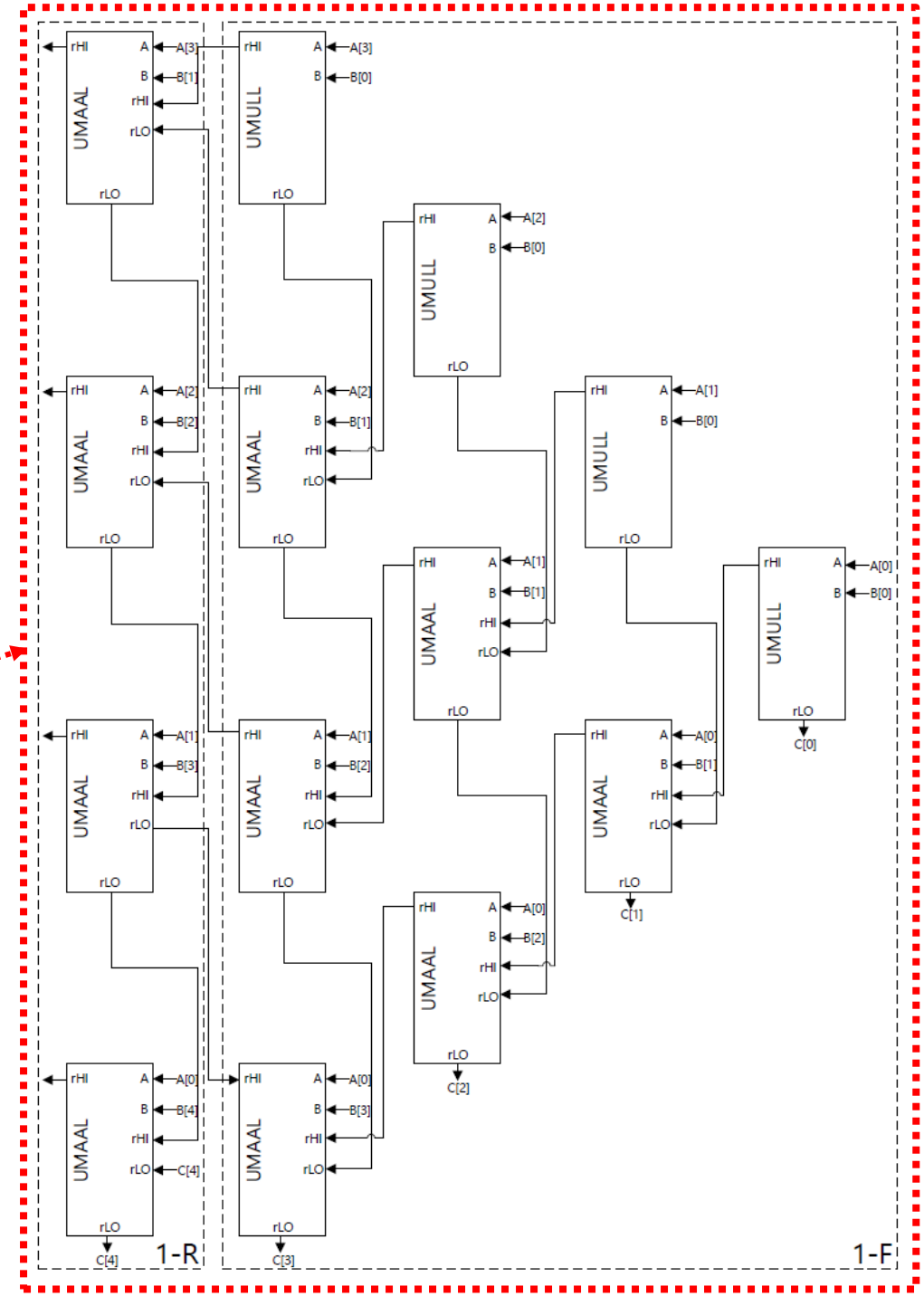
- **The number of memory access is significantly reduced**

Method	448-bit			512-bit			768-bit		
	Load	Store	Total	Load	Store	Total	Load	Store	Total
OS	406	210	616	528	272	800	1,176	600	1,776
PS	392	28	420	512	32	544	1,152	48	1,200
HS	196	28	224	256	32	288	576	48	624
OC	132	80	212	172	102	274	384	216	600
R-OC	107	63	170	140	80	220	306	168	474

Implementation in Detail







- UMULL → Initialize the result registers

Implementation in Detail

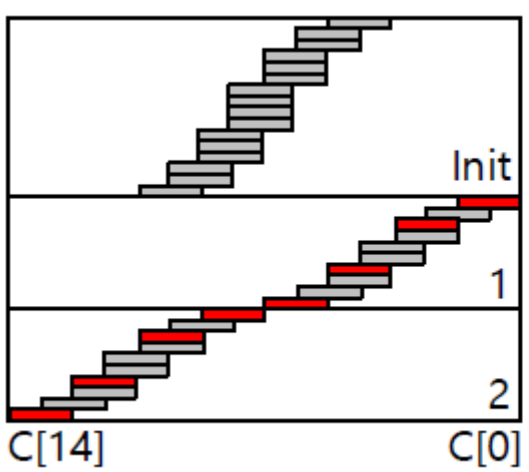
- Avoiding the pipeline stall through instruction re-ordering

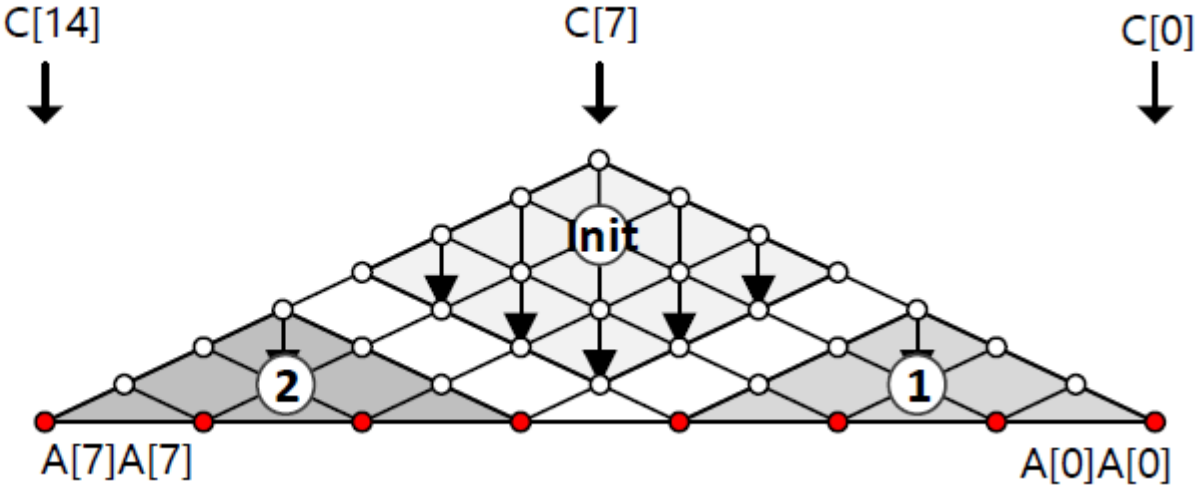
```
  ⋮
LDR   R6, [R0, #4 * 4] //Loading operand B[4] from memory
LDR   R1, [SP, #4 * 4] //Loading intermediate result C[4] from memory
UMAAL R14, R10, R5, R7 //Partial product (B[1]*A[3])
UMAAL R14, R11, R4, R8 //Partial product (B[2]*A[2])
UMAAL R14, R12, R3, R9 //Partial product (B[3]*A[1])
UMAAL R1, R14, R2, R6  //Partial product (B[4]*A[0])
  ⋮
```

Squaring in Detail

$$A[i] \times A[j] + A[j] \times A[i] = 2 \times A[i] \times A[j]$$

$$A[i] \times A[j] \rightarrow 2 \times A[i] \times A[j] \qquad 2 \times A[i] \rightarrow 2 \times A[i] \times A[j]$$





Multiplication/Squaring Result

- Multiplication

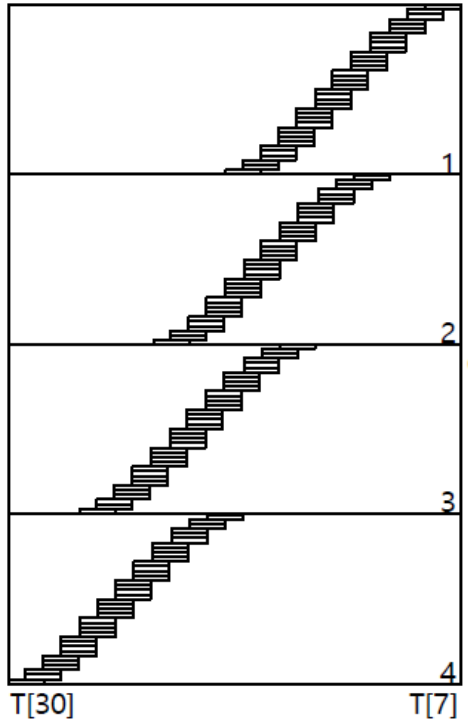
Methods	Timings [cc]	Scalability	Bit length
Fujii et al. [11]	239	✓	256
Haase et al. [13]	212	✗	256
This work	196	✓	256

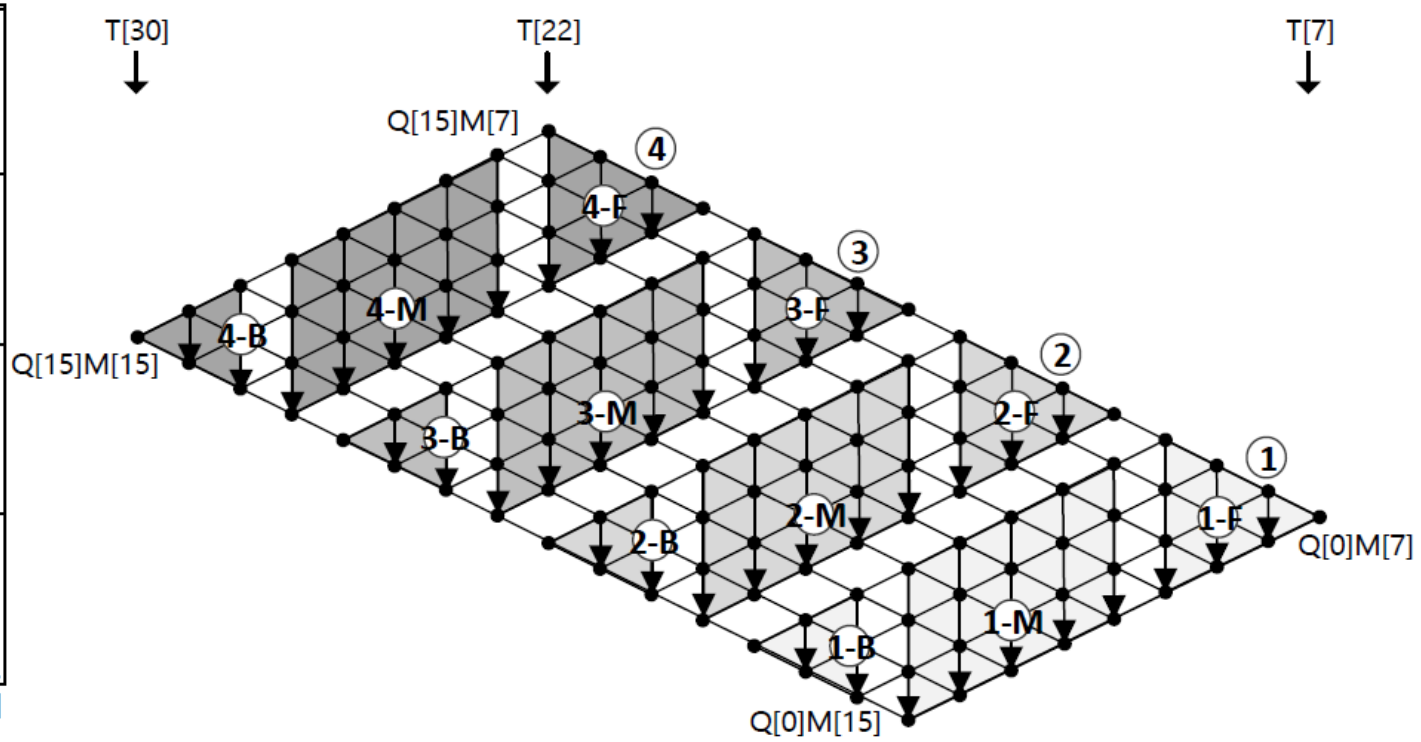
- Squaring

Methods	Timings [cc]	Scalability	Bit length
Fujii et al. [11]	218	✓	256
Haase et al. [13]	141	✗	256
This work	136	✓	255

Modular Reduction

- Montgomery Reduction: Division → Multiplication





Modular Reduction Result

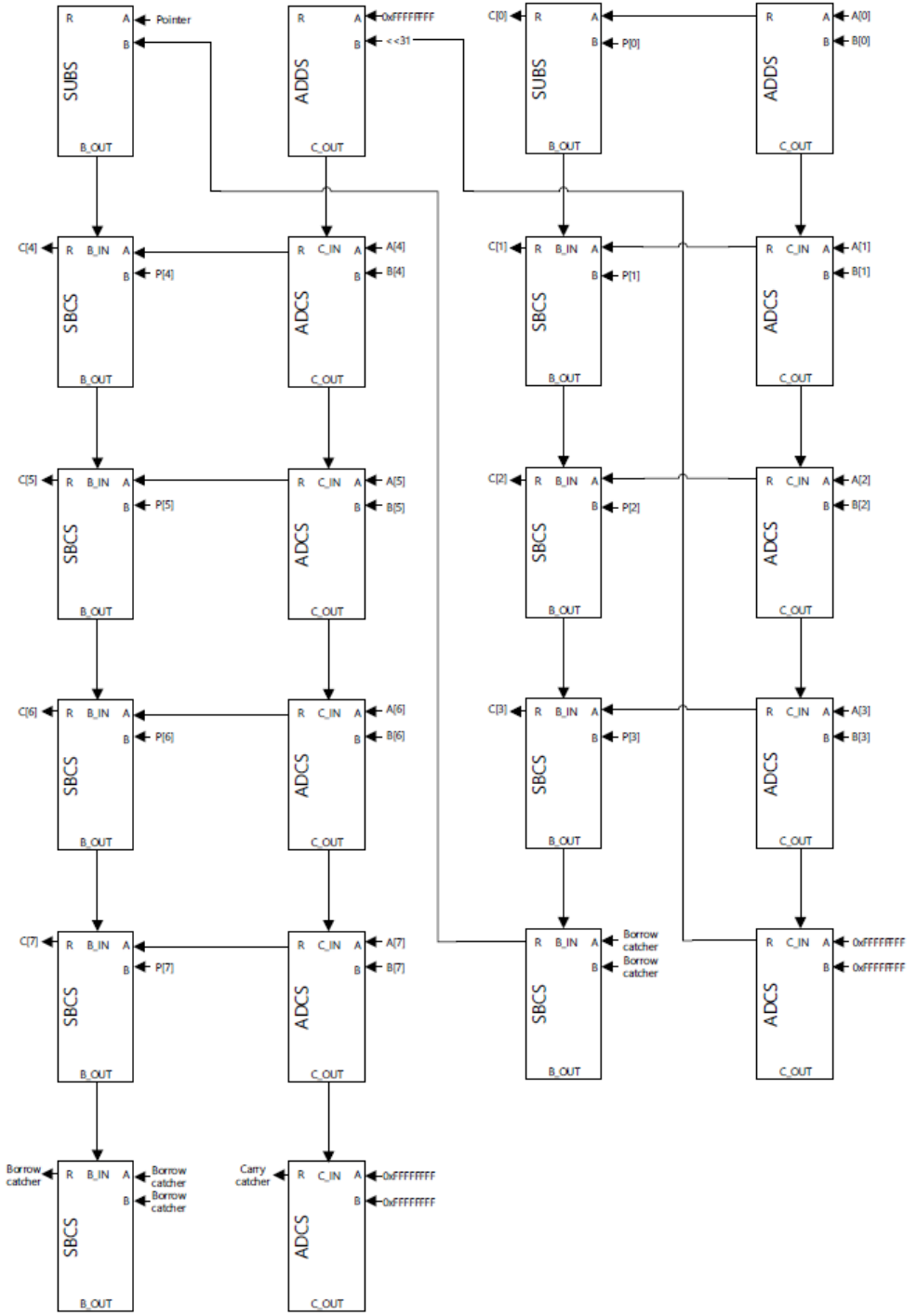
Methods	Timings [cc]			Modulus	Processor
	$\mathbb{F}_p$ mul	$\mathbb{F}_p$ sqr	reduction		
This work	1,110	981	544	$2^{216} \cdot 3^{137} - 1$	ARM Cortex-M4
SIDH v3.0 [7]	25,399	–	10,917	$2^{250} \cdot 3^{159} - 1$	ARM Cortex-M4
This work	1,333	1,139	654		
Bos et al. [4]	–	–	3,738	$2^{372} \cdot 3^{239} - 1$	ARM Cortex-A8
SIDH v3.0 [7]	55,178	–	23,484		ARM Cortex-M4
Koppermann et al. [20]	7,573	–	3,254		
This work	2,744	2,242	1,188		

Modular ADD/SUB

- Modular addition: (A+B) mod P
  ① C←A+B ② {M,C}←C-P ③ C←C+(P&M).

- Proposed modular addition: (A+B) mod P
  ① {M,C}←A+B-P ② C←C+(P&M).

Both carry and borrow catcher



Modular ADD/SUB Result

Methods	Timings [cc]		Modulus	Processor
	$\mathbb{F}_p$ add	$\mathbb{F}_p$ sub		
This work	254	208	$2^{216} \cdot 3^{137} - 1$	ARM Cortex-M4
SIDH v3.0 [7]	1,078	740	$2^{250} \cdot 3^{159} - 1$	ARM Cortex-M4
Seo et al. [25]	326	236		
This work	275	223		
SIDH v3.0 [7]	1,579	1,092	$2^{372} \cdot 3^{239} - 1$	ARM Cortex-M4
Koppermann et al. [20]	559	419		
Seo et al. [25]	466	333		
This work	388	284		

Outline

- Short Overview
- Supersingular isogeny key encapsulation (SIKE) protocol
- Our implementation
- Implementation results
- Conclusion

Results

SIDHp503 is about **19x** faster than SIDH v3.0 and SIDHp751 is about **2.7x** faster than Koppermann et al.

Implementation	Language	Timings [cc]				Timings [$cc \times 10^6$]				
		$\mathbb{F}_p$ add	$\mathbb{F}_p$ sub	$\mathbb{F}_p$ mul	$\mathbb{F}_p$ sqr	Alice R1	Bob R1	Alice R2	Bob R2	Total
SIDHp434										
This work	ASM	254	208	1,110	981	65	74	54	62	255
SIDHp503										
SIDH v3.0 [7]	C	1,078	740	25,399	–	986	1,086	812	924	3,808
This work	ASM	275	223	1,333	1,139	95	104	76	87	362
SIDHp751										
SIDH v3.0 [7]	C	1,579	1,092	55,178	–	3,246	3,651	2,669	3,112	12,678
Koppermann et al. [20]	ASM	559	419	7,573	–	1,025	1,148	967	1,112	4,252
This work	ASM	388	284	2,744	2,242	252	284	205	236	977

Comparison with NIST PQC

Implementation	Language	Timings [cc]				Timings [$cc \times 10^6$]				Memory [bytes]		
		$\mathbb{F}_p$ add	$\mathbb{F}_p$ sub	$\mathbb{F}_p$ mul	$\mathbb{F}_p$ sqr	KeyGen	Encaps	Decaps	Total	KeyGen	Encaps	Decaps
SIKEp434												
This work	ASM	254	208	1,110	981	74	122	130	326	6,580	6,916	7,260
SIKEp503												
SIDH v3.0 [7]	C	1,078	740	25,399	–	1,086	1,799	1,912	4,797	–	–	–
This work	ASM	275	223	1,333	1,139	104	172	183	459	6,204	6,588	6,974
SIKEp751												
SIDH v3.0 [7]	C	1,579	1,092	55,178	–	3,651	5,918	6,359	15,928	–	–	–
This work	ASM	388	284	2,744	2,242	282	455	491	1,228	11,116	11,260	11,852
NIST PQC Round 2 [18]												
Frodo640-AES	ASM	–	–	–	–	42	46	47	135	31,116	51,444	61,820
Frodo640-CSHAKE	ASM	–	–	–	–	81	86	87	254	26,272	41,472	51,848
Kyber512	ASM	–	–	–	–	0.7	0.9	1.0	2.6	6,456	9,120	9,928
Kyber768	ASM	–	–	–	–	1.2	1.4	1.4	4.0	10,544	13,720	14,880
Kyber1024	ASM	–	–	–	–	1.7	2.1	2.1	5.9	15,664	19,352	20,864
Newhope1024CCA	ASM	–	–	–	–	1.2	1.9	1.9	5	11,152	17,448	19,648
Saber	ASM	–	–	–	–	0.9	1.2	1.2	3.3	12.616	14,896	15,992

Results on all ARM processors

Protocol	Implementation	Platform	Freq	Latency [sec.]		Comm. [bytes]	
			[MHz]	Alice	Bob	A→B	B→A
High-end ARM Processors							
SIDHp503	[22]	32-bit ARMv7 Cortex-A8	1,000	0.216	0.229	378	378
	[22]	32-bit ARMv7 Cortex-A15	2,300	0.064	0.067	378	378
	[25]		2,000	0.042	0.046	378	378
	[2]	64-bit ARMv8 Cortex-A53	1,512	0.061	0.050	378	378
	[25]		1,512	0.050	0.041	378	378
	[2]	64-bit ARMv8 Cortex-A72	1.992	0.030	0.025	378	378
	[25]		1.992	0.025	0.021	378	378
SIDHp751	[22]	32-bit ARMv7 Cortex-A8	1,000	1.406	1.525	564	564
	[22]	32-bit ARMv7 Cortex-A15	2,300	0.340	0.368	564	564
	[25]		2,000	0.135	0.157	564	564
Low-end ARM Microcontrollers							
SIDHp434	This work	32-bit ARMv7 Cortex-M4	168	0.715	0.813	326	326
SIDHp503	This work		168	1.028	1.143	378	378
SIDHp751	[20]		120	16.590	18.833	564	564
	This work		168	2.727	3.099	564	564

Outline

- Short Overview
- Supersingular isogeny key encapsulation (SIKE) protocol
- Our implementation
- Implementation results
- Conclusion

Conclusion

- **New implementations of finite field arithemtic on ARM**
  - Faster multi-precision multiplication/squaring
  - Faster Montgomery reduction
  - Faster modular addition/subtraction

- **Record-setting SIDH and SIKE implementations**
  - On 32-bit ARM Cortex-M4

Thank you for your attention!