Fast Strategies for the Implementation of SIKE Round 3 on Cortex-M4

Mila Anastasova[1]
Reza Azarderakhsh[1,2]
Mehran Mozaffari Kermani[3]

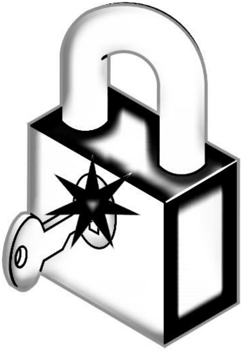
Florida Atlantic University, Boca Raton, FL.
Email addresses: (manastasova2017, razarderakhsh)@fau.edu.
PQSecure Technologies, LLC, Boca Raton, FL, USA.
University of South Florida, Tampa FL. Email address: mehran2@usf.edu.

2021/07/12

Content

Introduction

- Quantum computers threaten to break the classical cryptography schemes, such as RSA and ECC, in polynomial time. [Shor's algorithm]
- Led by the increasing capabilities of quantum computers several research teams have focused on the creation and implementation of efficient post quantum resistant schemes.



NIST Standardization Process

- The National Institute of Standards and Technology (NIST) has started a competition among the post-quantum secure algorithms.

The Supersingular Isogeny Key Encapsulation

Public Parameters: $p = \ell_A^{e_A}\ell_B^{e_B} - 1$, $E_0/\mathbb{F}_{p^2}$, $\{P_A, Q_A\}$ and $\{P_B, Q_B\}$

Bob

Key Generation

1. $sk_B \in_R \mathbb{Z}/\ell_B^{e_B}\mathbb{Z}$
2. $\phi_B : E_0 \to E_B$ with $ker(\phi_B) = \langle P_B + [sk_B]Q_B \rangle$
3. $pk_B = \{E_B, \phi_B(P_A), \phi_B(Q_A)\}$
4. $s \in_R \{0,1\}^t$

$\xrightarrow{pk_B}$

Alice

Encapsulation

1. $m \in_R \{0,1\}^t$
2. $r = H(m||pk_B) mod \ell_A^{e_A}$
3. $\phi_A : E_0 \to E_A$ with $ker(\phi_A) = \langle P_A + [r]Q_A \rangle$
4. $pk_A = \{E_A, \phi_A(P_B), \phi_A(Q_B)\}$
5. $\phi'_A : E_B \to E_{BA}$ with $ker(\phi'_A) = \langle \phi_B(P_A) + [r]\phi_B(Q_A) \rangle$
6. $c = (c_0, c_1) = (pk_A, K(j(E_{BA})) \oplus m)$
7. $ss = (J(m||c))$

$\xleftarrow{c}$

Bob

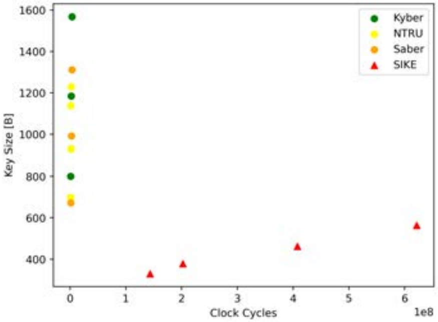
Decapsulation

1. $\phi'_B : E_A \to E_{AB}$ with $ker(\phi'_B) = \langle \phi_A(P_B) + [sk_B]\phi_A(Q_B) \rangle$
2. $m' = c_1 \oplus K(j(E_{AB}))$
3. $r' = H(m'||pk_B) mod \ell_A^{e_A}$
4. $\phi''_A : E_0 \to E_{A'}$ with $ker(\phi''_A) = \langle P_A + [r']Q_A \rangle$
5. $pk'_A = \{E_{A'}, \phi''_A(P_B), \phi''_A(Q_B)\}$
6. IF $pk'_A = c_0$
   $ss = (J(m'||c))$
   ELSE $ss = (J(s||c))$

- Based on secret isogeny maps between supersingular elliptic curves.
- Forms part of the alternate candidates in Round 3 of the NIST standardization process.
- Ensures insignificant communication latency due to smallest key sizes among the post-quantum candidates.
- Requires large execution time due to sophisticated mathematical computations.

Why SIKE?

- SIKE attracts with the smallest key sizes, which is crucial in the overall timing of the algorithm considering the communication latency.
- However, SIKE is one of the slowest post-quantum algorithms with large energy consumption.



ARM Cortex-M4

NIST recommended microcontroller for benchmarking.

STM32F407 - Discovery Board

X-NUCLEO-LPM01A

NUCLEO-F411RE

ARM Cortex-M4

NIST recommended microcontroller for benchmarking.

Features

ARMv7-M architecture

16 32-bit core registers

32 32-bit FP registers

1 CC per instruction except memory accesses

SIKE challenges

Big integer arithmetic

Repetitive memory accesses

Scheduled instructions

Optimal instruction flow sequence

Implementation strategies

Use the entire register set.

Operate on larger operand sets.

Re-organize the instruction flow for efficient design.

Modular addition

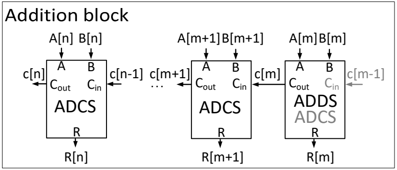
`a + b - P + (P & mask)`

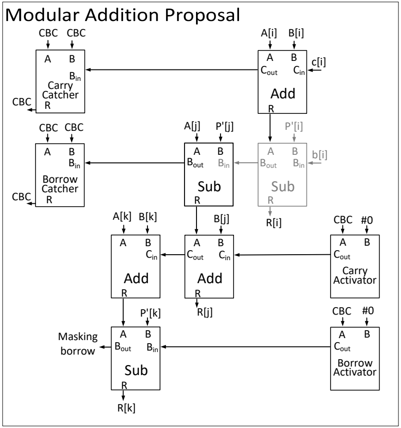
Proposed strategies :

Increase the size of the addition blocks.

Alternate the add/sub blocks.

Use `P'=P+2` to eliminate the last k additions.





Modular addition ? ? ? ?

Algorithm 1 Modular addition algorithm with 4 word operands, presenting the carry/borrow propagation when applying the add/sub block alternation technique.

```
1. (borrow0, T0) = A0 - p0
2. Result0 = T0 + B0
3. (carry0, T1) = A1 + B1
4. Result1 = T1 - p1 - borrow0
5. (borrow1, T2) = A2 - p2
6. Result2 = T2 + B2 + carry0
7. T3 = A3 + B3
8. (borrow2, Result3) = T3 - p3 - borrow1
```

`a + b - P + (P & mask)`

Proposed strategies :

New reduced instruction set for carry/borrow catcher/activator.

Use `SBC` Subtract with Carry flag for Carry/Borrow Catcher.

Use `RSBC` Reverse Subtract with Carry flag for Carry/Borrow Activator.

Modular addition

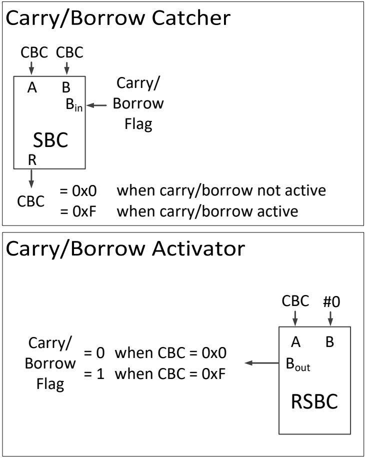
```
a + b - P + (P & mask)
```

Proposed strategies :

New reduced instruction set for carry/borrow catcher/activator.

Use `SBC` Subtract with Carry flag for Carry/Borrow Catcher.

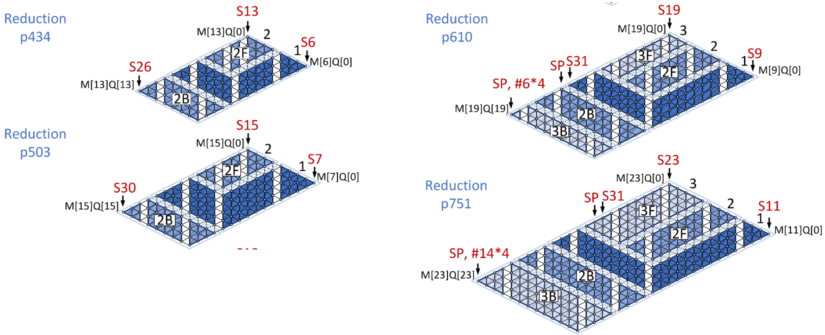
Use `RSBC` Reverse Subtract with Carry flag for Carry/Borrow Activator.



Results

Algorithm 3 Montgomery multiplication [38]

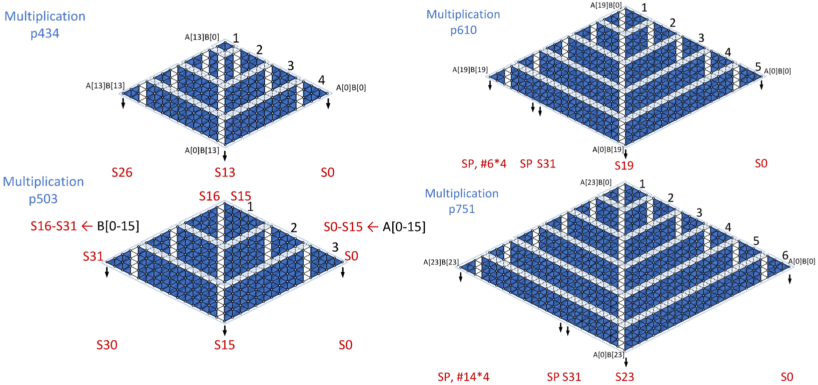
```
INPUT: M, R, M'=-M^-1 (mod R), A, B
OUTPUT: A · B · R^-1 (mod M)
1. T = A · B
2. Q = T · M' (mod R)
3. T = (T + Q · M) / R
4. IF (T > Q) RETURN T - M
5. RETURN T
```



Results Modular Addition/Subtraction

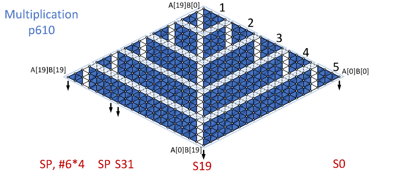
Implementation	Lang	Timing [CC] Speedup[%]							
		$\mathbb{F}_p$ add		$\mathbb{F}_p$ sub		$\mathbb{F}_p$ add		$\mathbb{F}_p$ sub	
		CC	%	CC	%	CC	%	CC	%
		SIKEp434				SIKEp503			
SIDH v3.3[1]	C	932	80.58	519	66.86	1,024	80.57	609	68.80
Seo et al.[2]	ASM	254	28.74	208	17.31	275	27.64	223	14.80
Seo et al.[3]	ASM	253	28.46	207	16.91	274	27.37	227	16.30
This work	ASM	181	-	172	-	199	-	190	-
		SIKEp610				SIKEp751			
SIDH v3.3[1]	C	1,314	81.96	836	72.73	1,570	82.36	996	74.20
Seo et al.[2]	ASM	-	-	-	-	388	28.61	284	9.51
Seo et al.[3]	ASM	331	28.40	272	16.18	387	28.42	318	19.18
This work	ASM	237	-	228	-	277	-	257	-

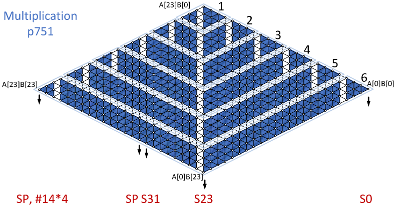
Multi-Precision Multiplication



Multi-Precision Multiplication

$$a * b = (a_{n-1}, \dots, a_0) * (b_{n-1}, \dots, b_0) = (c_{2n-1}, \dots, c_0)$$

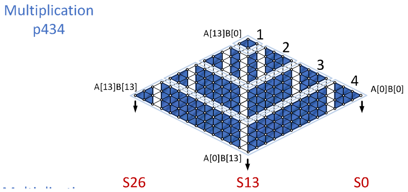




Proposed strategies :

Use the FP register set as L1 cache.

Use the 1 clock cycle instruction `VMOV` to transfer data between `R` and `S` registers to reduce the number of expensive memory accesses.



Multi-Precision Multiplication

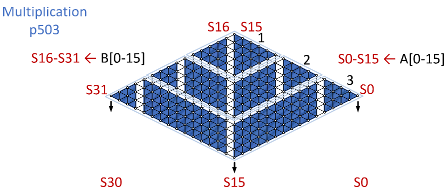
$$a * b = (a_{n-1}, \ldots, a_0) * (b_{n-1}, \ldots, b_0) = (c_{2n-1}, \ldots, c_0)$$

Proposed strategies :

Pre-load the operand values into the FP register set.

Access operand words in a single clock cycle using VMOV.

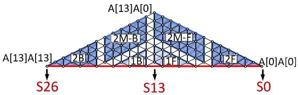
Increase the row size to 5 by re-loading one word in each iteration of the inner loop, while keeping the data access cost to 2 clock cycles per column.



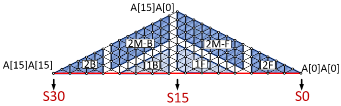
Multi-Precision Square

$$a * a = (a_{n-1}, \ldots, a_0) * (a_{n-1}, \ldots, a_0) = (c_{2n-1}, \ldots, c_0)$$

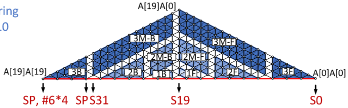
Squaring p434



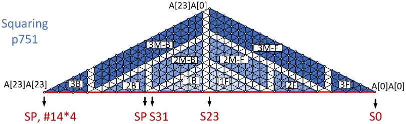
Squaring p503



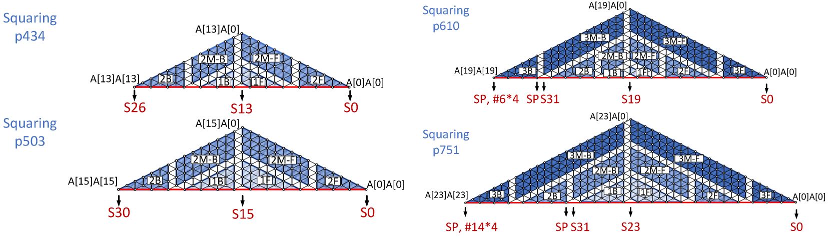
Squaring p610



Squaring p751



Multi-Precision Square



Proposed strategies :

Use the FP register set as L1 cache.

Start from the bottom center part of the rhombus to increase the length of the rows thus decrease their number.

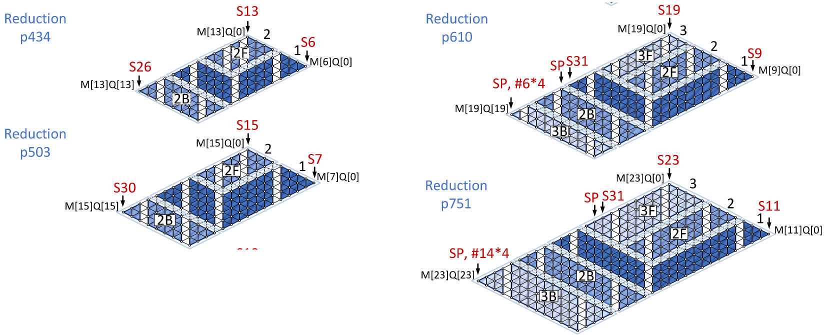
Use sub-multiplication blocks and sub-squaring blocks.

Do not pre-calculate the doubles of the values but re-compute them.

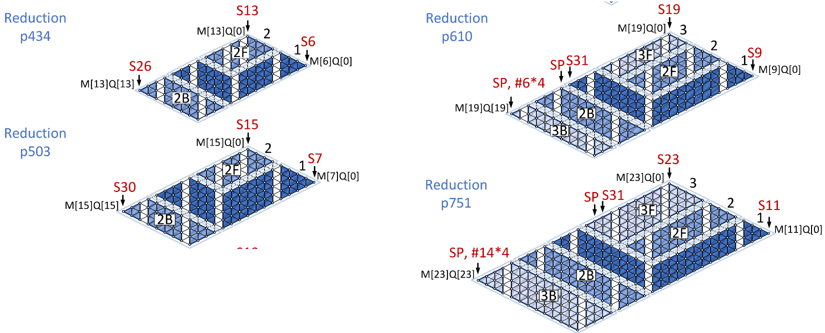
Modular Reduction

Algorithm 3 Montgomery multiplication [38]

```
INPUT: M, R, M'=-M^-1 (mod R), A, B
OUTPUT: A · B · R^-1 (mod M)
1. T = A · B
2. Q = T · M' (mod R)
3. T = (T + Q · M) / R
4. IF (T > Q) RETURN T − M
5. RETURN T
```



Modular Reduction



Proposed strategies :

Use the FP register set as L1 cache.

Change the direction of the rows in the middle of the algorithm.

Vary the size of the rows to adjust to the number of rows.

Results Modular Multiply/Square

Implementation	Timing [CC] Speedup[%]							
	$\mathbb{F}_p$ mul		$\mathbb{F}_p$ sqr		$\mathbb{F}_p$ mul		$\mathbb{F}_p$ sqr	
	CC	%	CC	%	CC	%	CC	%
	SIKEp434				SIKEp503			
SIDH v3.3[1]	17,964	95.72	17,964	96.69	23,364	95.93	23,364	96.86
Seo et al.[2]	1,110	30.72	981	39.45	1,333	28.58	1,139	35.56
Seo et al.[3]	1,011	23.94	889	33.18	1,221	22.03	1,024	28.32
This work	769	-	594	-	952	-	734	
	SIKEp610				SIKEp751			
SIDH v3.3[1]	35,047	95.70	35,047	96.66	49,722	95.77	49,722	96.90
Seo et al.[2]	-	-	-	-	2,744	23.36	2,242	31.18
Seo et al.[3]	1,869	19.42	1,535	23.71	2,577	18.39	2,066	25.31
This work	1506	-	1171	-	2103	-	1543	-

Timing/Memory Results

Implementation	Memory [B]				Timing [cc × 10^6]				Speedup
	KeyGen	**Encaps**	**Decaps**	**[%]**	**KeyGen**	**Encaps**	**Decaps**	**Total**	**[%]**
SIKEp434									
SIDH v3.3[1]	6,620	6,920	7,256	7.05	650	1,065	1,136	2,202	93.67
Seo et al.[2]	6,580	6,916	7,260	7.05	74	122	130	252	44.68
Seo et al.[3]	6,188	6,516	6,860	1.50	54	87	94	181	22.97
This work	6,092	6,420	6,756	-	41	67	72	139	-
SIKEp503									
SIDH v3.3[1]	6,244	6,620	6,996	6.28	985	1,623	1,726	3,350	94.11
Seo et al.2	6,204	6,588	6,974	6.65	104	172	183	355	44.43
Seo et al.[3]	6,700	7,084	7,468	0.16	74	121	129	250	21.10
This work	6,688	7,080	7,448	-	58	96	102	197	-
SIKEp610									
SIDH v3.3[1]	9,668	10,092	10,548	5.29	1,819	3,348	3,368	6,716	94.18
Seo et al.[3]	10,244	10,668	11,140	0.07	131	241	243	484	19.21
This work	10,244	10,668	11,124	-	106	195	196	391	-
SIKEp751									
SIDH v3.3[1]	11,156	11,300	11,884	5.88	3,296	5,347	5,742	11,089	94.48
Seo et al.[2]	11,116	11,260	11,852	6.17	282	455	491	946	35.25
Seo et al.[3]	11,852	11,996	12,564	0.29	225	365	392	757	19.08
This work	11,884	12,036	12,596	-	182	295	317	613	-

Energy Consumption Results

Implementation	Language	Speed [MHz]	Energy [mJ] KeyGen	Encaps	Decaps	Total
SIKEp434						
SIDH v3.3[1]	C	96	485.00	798.32	850.72	1,649.04
Seo et al.[2]	ASM		37.26	61.60	65.54	127.14
This work			32.96	54.16	57.85	112.01
SIKEp503						
SIDH v3.3[1]	C	96	724.96	1,198.00	1,273.00	2,471.00
Seo et al.[2]	ASM		53.03	87.89	93.55	181.44
This work			48.93	81.09	86.18	167.27
SIKEp610						
SIDH v3.3[1]	C	96	1,358.00	2,516.00	2,528.00	5,044.00
Seo et al.[2]	ASM		97.36	180.50	181.30	361.80
This work			92.03	170.13	171.15	341.28
SIKEp751						
SIDH v3.3[1]	C	96	2,435.00	3,992.00	4,273.00	8,265.00
Seo et al.[2]	ASM		172.07	280.53	301.58	582.11
This work			162.08	263.91	283.60	547.51

NIST Round 3 Comparison Results

SL	Implementation	Timing [cc $\times 10^6$] KeyGen	Encaps	Decaps	Timing [s] Total	Memory [B] KeyGen	Encaps	Decaps	Data [B] pk+ct
	Finalists								
I	mceliece348864[1]	1589.60	0.48	2.29	0.12	1,412	1,412	18,492	261,248
I	Kyber512	0.46	0.57	0.53	0.05	2,396	2,484	2,500	1,568
I	ntruhps2048509	79.66	0.56	0.54	0.05	21,392	14,068	14,800	1,398
I	lightsaber	0.36	0.49	0.46	0.04	5,332	5,292	5,308	1,408
III	Kyber768	0.76	0.92	0.86	0.07	3,276	2,9684	2,988	2,272
III	ntruhps2048677	143.73	0.82	0.82	0.07	28,504	9,036	19,728	1,862
III	saber	0.66	0.84	0.79	0.07	6,364	6,316	6,332	2,080
III	ntruhrss701	153.10	0.38	0.87	0.05	27,560	7,400	20,552	2,276
IV	Kyber1024	1.22	1.41	1.33	0.11	3,788	3,476	3,508	3,136
IV	ntruhps4096821	208.84	1.03	1.03	0.09	34,504	10,924	23,952	2,460
IV	firesaber	1.01	1.22	1.17	0.10	7,388	7,340	7,356	2,784
	Alternate								
I	BIKE L1	25.06	3.40	54.79	2.42	44,108	32,156	91,400	3,113
I	FrodoKEM640aes	48.35	47.13	46.59	3.91	31,992	62,488	83,104	19,336
I	FrodoKEM640shake	79.33	79.70	79.15	6.62	26,600	51,976	72,592	19,336
I	**SIKEp434**	**41.28**	**67.40**	**72.02**	**5.81**	**6,108**	**6,468**	**6,748**	**676**
II	**SIKEp503**	**58.12**	**95.53**	**101.73**	**8.22**	**7,360**	**7,736**	**8,112**	**780**
III	ntrulpr761	0.74	1.29	1.39	0.11	13,168	20,000	24,032	2,206
III	sntrup761	10.83	0.70	0.57	0.05	61,508	13,320	16,952	2,197
III	**SIKEp610**	**106.07**	**194.90**	**196.12**	**16.29**	**10,490**	**10,908**	**11,372**	**948**
IV	**SIKEp751**	**182.28**	**295.36**	**317.22**	**25.52**	**12,180**	**12,324**	**12,876**	**1,160**

Future work

- Continue the work on time and energy efficient SIKE implementation targeting Cortex-M4.
- Perform side-channel analysis of the post-quantum scheme.
- Target different low-end processors.