Compact Implementations of Public Key Cryptography

Cryptography Contest 2015

Korea Cryptography Forum

2015/10/14

Outline

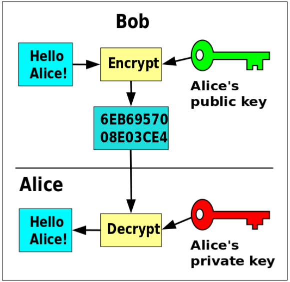
1. Short Overview
2. RSA
   - Hybrid Montgomery Reduction
   - Karatsuba Montgomery Multiplication for NEON
3. ECC
   - High Speed Microsoft ECC Curve for AVR
   - Faster ECC for P521 on NEON
   - Faster ECC for K571 on NEON
4. Ring-LWE
   - Tiny Montgomery for Ring-LWE
   - Parallel Implementation of NTT
5. Conclusion

Outline

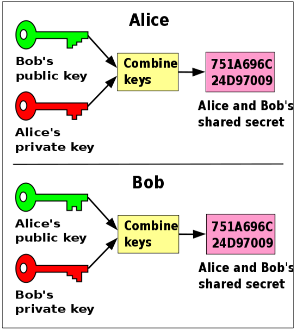
1. Short Overview
2. RSA
   - Hybrid Montgomery Reduction
   - Karatsuba Montgomery Multiplication for NEON
3. ECC
   - High Speed Microsoft ECC Curve for AVR
   - Faster ECC for P521 on NEON
   - Faster ECC for K571 on NEON
4. Ring-LWE
   - Tiny Montgomery for Ring-LWE
   - Parallel Implementation of NTT
5. Conclusion

Public Key Cryptography

- **PKC based applications**
  - Key Distribution, Encryption and Digital Signature
- **PKC algorithms**
  - RSA, ElGamal, ECC, NTRU and Ring-LWE



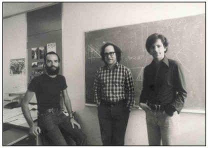
Encryption



Key exchange

Past, Present and Future of PKC

RSA
(Defacto Standard, Factoring Problem)



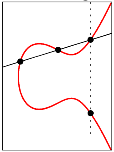
Ring-LWE
(Secure against Quantum Computer)

1977

2010

1985

ECC
(Short key, Discrete Logarithm Problem)



Internet of Things and Target Platforms

- **Internet of Things**
  - Home automation, Health care, Autonomous vehicles
  - Various platforms for things from low to high-end
- **Low-end 8-bit AVR Microcontroller**
  - Products: Arduino Uno, Yun, Due
  - Freq. 32MHz, 128KB Flash, 8KB RAM
  - Core instruction: 8-bit mul/add
- **High-end ARM-NEON SIMD Processor**
  - Products: iPhone, Galaxy, Nexus
  - Freq. 1GHz, 16GB Flash, 2GB RAM
  - Core instruction: vector-wise mul/add

Contribution & Target Journal/Conference

	Contribution	Target Journal & Conference	Collaboration
RSA	Hybrid Montgomery Reduction	ACM Transactions on Embedded Computing IEEE Transactions on Computers	Luxembourg
RSA	Karatusba Montgomery Multiplication	SECURITY AND COMMUNICATION NETWORKS	USA, Luxembourg
ECC	High Speed Microsoft ECC Curve	CHES	USA, Luxembourg
ECC	Faster ECC for P521	CHES, ICISC	USA
ECC	Faster ECC for K571	CT-RSA	Japan, Luxembourg
Ring LWE	Tiny Montgomery for Ring-LWE	IEEE Transactions on Computers	Belgium, Germany, Luxembourg
Ring LWE	Parallel Implementation of NTT	CHES, ICISC	Luxembourg

Outline

1 Short Overview

2 RSA
- Hybrid Montgomery Reduction
- Karatsuba Montgomery Multiplication for NEON

3 ECC
- High Speed Microsoft ECC Curve for AVR
- Faster ECC for P521 on NEON
- Faster ECC for K571 on NEON

4 Ring-LWE
- Tiny Montgomery for Ring-LWE
- Parallel Implementation of NTT

5 Conclusion

Outline

1. Short Overview
2. RSA
   - Hybrid Montgomery Reduction
   - Karatsuba Montgomery Multiplication for NEON
3. ECC
   - High Speed Microsoft ECC Curve for AVR
   - Faster ECC for P521 on NEON
   - Faster ECC for K571 on NEON
4. Ring-LWE
   - Tiny Montgomery for Ring-LWE
   - Parallel Implementation of NTT
5. Conclusion

Montgomery Reduction

- Montgomery reduction replaces division into multiplication
- Previous works didn't try Karatsuba for Montgomery reduction

Algorithm 1: Montgomery reduction

Require: m-bit modulus M, Montgomery radix $R = 2^m$, operand $T = A \cdot B$, constant $M' = -M^{-1} \bmod R$
Ensure: Montgomery product ($Z = \mathrm{MonRed}(T, R, M) = T \cdot R^{-1} \bmod M$)

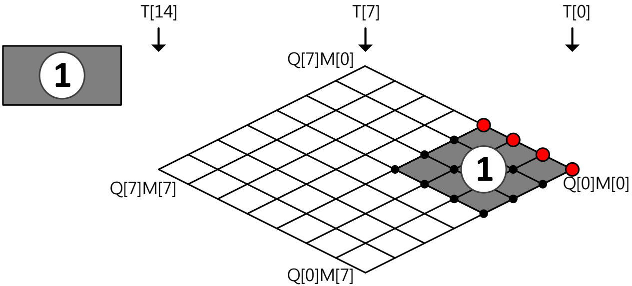
1: $Q \leftarrow T \cdot M' \bmod R$
2: $Z \leftarrow (T + Q \cdot M)/R$ {m-bit wise multiplication}
3: **if** $Z \geq M$ **then** $Z \leftarrow Z - M$ **end if**
4: **return** Z

Karatsuba Algorithm

- **Karatsuba Algorithm**
  - $\theta(n^2)$ to $\theta(n^{log_2 3})$ for n-limb
- Additive Karatsuba's method:
  - $A_H \cdot B_H \cdot 2^n + [(A_H + A_L)(B_H + B_L) - A_H \cdot B_H - A_L \cdot B_L] \cdot 2^{\frac{n}{2}} + A_L \cdot B_L$
- Subtractive Karatsuba's method:
  - $A_H \cdot B_H \cdot 2^n + [A_H \cdot B_H + A_L \cdot B_L - |A_H - A_L| \cdot |B_H - B_L|] \cdot 2^{\frac{n}{2}} + A_L \cdot B_L$
- where $A = A_H \cdot 2^{\frac{n}{2}} + A_L$ and $B = B_H \cdot 2^{\frac{n}{2}} + B_L$

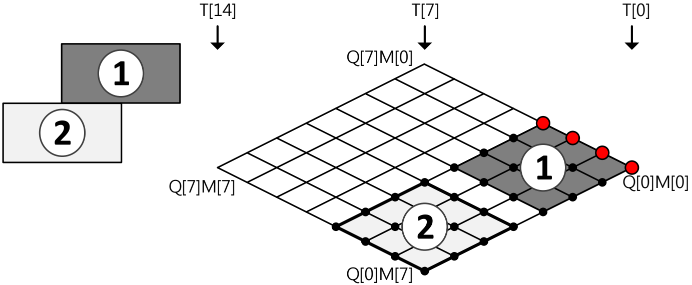
Hybrid Montgomery Reduction

- (Step 1) Ordinary Montgomery reduction



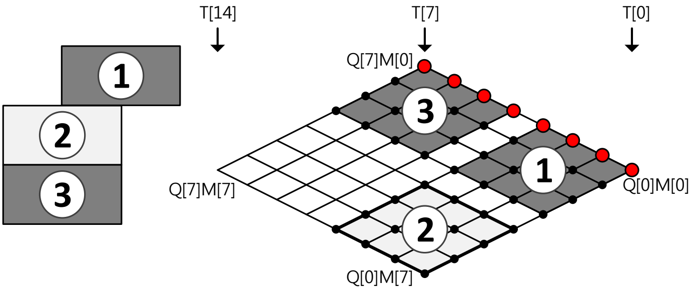
Hybrid Montgomery Reduction

- (Step 2) Ordinary multiplication → Karatsuba chance!



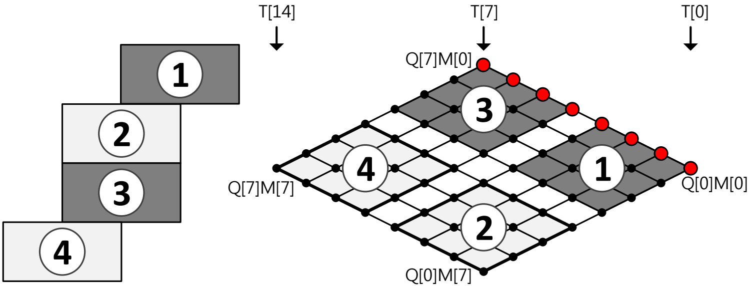
Hybrid Montgomery Reduction

- (Step 3) Ordinary Montgomery reduction



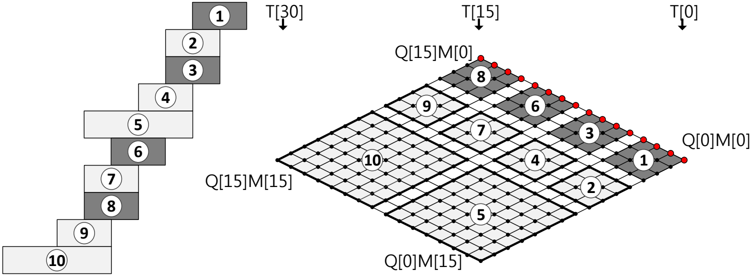
Hybrid Montgomery Reduction

- (Step 4) Ordinary multiplication $\rightarrow$ Karatsuba chance!
- Complexity reduction: $\theta(n^2 + n) \rightarrow \theta(\frac{7n^2}{8} + n)$



Hybrid Montgomery Reduction

- 2-level of Hybrid multiplication in rhombus form
- Further improvements: $\theta(n^2 + n) \rightarrow \theta(\frac{7n^2}{8} + n) \rightarrow \theta(\frac{13n^2}{16} + n)$



Results of Hybrid Montgomery Reduction

- †: Product Scanning Multiplication + Hybrid Karatsuba Reduction
- ‡: Karatsuba Multiplication + Hybrid Karatsuba Reduction

Algorithm	mul	load	store	add
Montgomery Reduction:				
OS	n^2+n	$2n^2+2n+1$	n^2+2n+1	$4n^2+2n$
PS	n^2+n	$2n^2+2n$	$2n+1$	$3n^2+6n$
HK	$\frac{7n^2}{8}+n$	$\frac{7n^2}{4}+\frac{21n}{2}+2$	$\frac{23n}{2}+4$	$\frac{21n^2}{8}+\frac{25n}{2}+4$
Montgomery Multiplication:				
FIPS	$2n^2+n$	$4n^2-n$	$2n+1$	$6n^2$
SPS	$2n^2+n$	$4n^2+2n$	$4n+1$	$6n^2+6n$
CIOS	$2n^2+n$	$4n^2+5n$	$2n^2+3n$	$8n^2+4n$
SOS	$2n^2+n$	$4n^2+3n+1$	$2n^2+3n+1$	$8n^2+2n$
CIHS	$2n^2+n$	$\frac{11n^2}{2}+\frac{7n}{2}$	$3n^2+2n$	$9n^2+5n$
FIOS	$2n^2+n$	$3n^2+4n$	n^2+n	$8n^2$
KCM	$\frac{7n^2}{4}+n$	$\frac{7n^2}{2}+11n+3$	$10n+1$	$\frac{21n^2}{4}+8n+4$
SHKM †	$\frac{15n^2}{8}+n$	$\frac{15n^2}{4}+\frac{21n}{2}+2$	$\frac{27n}{2}+4$	$\frac{45n^2}{8}+\frac{25n}{2}+4$
SHKM ‡	$\frac{13n^2}{8}+n$	$\frac{13n^2}{4}+\frac{33n}{2}+7$	$\frac{37n}{2}+3$	$\frac{39n^2}{8}+\frac{33n}{2}+6$

Results of Hybrid Montgomery Reduction

- †: Product Scanning Multiplication + Hybrid Karatsuba Reduction
- ‡: Karatsuba Multiplication + Hybrid Karatsuba Reduction
- **11%/25%** improvements for Montgomery reduction/multiplication

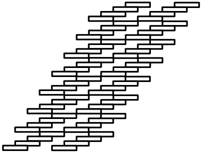
Implementation	512	1024
AVR Implementations of Montgomery Reduction:		
Product-Scanning [8]	29158	111478
Product-Scanning [9]	28265	107144
One-Level Hybrid Karatsuba	**27407**	**97153**
Two-Level Hybrid Karatsuba	**N/A**	**95055**
AVR Implementations of Montgomery Multiplication:		
Hybrid Finely Integrated Operand Scanning [8]	79760	316018
Hybrid Coarsely Integrated Hybrid Scanning [8]	74435	290549
Hybrid Separated Operand Scanning [8]	69301	268788
Hybrid Coarsely Integrated Operand Scanning [8]	65033	253787
Hybrid Separated Product Scanning [8]	57281	221044
Hybrid Finely Integrated Product Scanning [8]	56339	220596
Hybrid Separated Product Scanning [9]	54396	210139
Separated Hybrid Karatsuba †	**53616**	**197956**
Separated One-Level Hybrid Karatsuba ‡	**46146**	**160070**
Separated Two-Level Hybrid Karatsuba ‡	N/A	**157972**

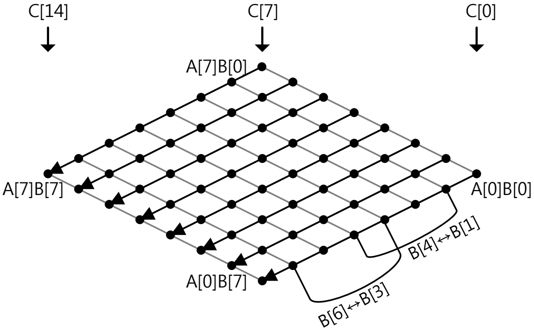
Outline

1. Short Overview
2. RSA
   - Hybrid Montgomery Reduction
   - Karatsuba Montgomery Multiplication for NEON
3. ECC
   - High Speed Microsoft ECC Curve for AVR
   - Faster ECC for P521 on NEON
   - Faster ECC for K571 on NEON
4. Ring-LWE
   - Tiny Montgomery for Ring-LWE
   - Parallel Implementation of NTT
5. Conclusion

Cascade Operand Scanning Multiplication

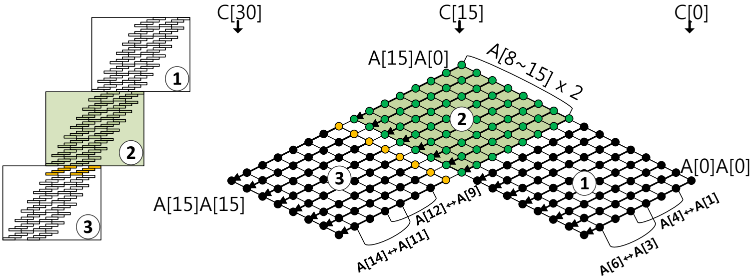
- In ICISC'14, COS multiplication for SIMD is proposed.





Double Operand Scanning Squaring

- In ePrint465, DOS squaring for SIMD is proposed.



Karatsuba Montgomery Multiplication for NEON

- **How to apply Karatsuba multiplication for NEON**
  - Tasks with high overheads (mul) $\rightarrow$ SIMD
  - Small and sequential tasks (others) $\rightarrow$ SISD

Algorithm 2: Additive Karatsuba Multiplication on SIMD

Require: An even m-bit operands $A(A_{LOW} + A_{HIGH} \cdot 2^{\frac{m}{2}})$, $B(B_{LOW} + B_{HIGH} \cdot 2^{\frac{m}{2}})$

Ensure: $2m$-bit result $C = A \cdot B$

1: $L = A_{LOW} \cdot B_{LOW}$ {SIMD}
2: $H = A_{HIGH} \cdot B_{HIGH}$ {SIMD}
3: $\{A_{CARRY}, A_{SUM}\} = A_{LOW} + A_{HIGH}$
4: $\{B_{CARRY}, B_{SUM}\} = B_{LOW} + B_{HIGH}$
5: $M = A_{SUM} \cdot B_{SUM}$ {SIMD}
6: $M = M + (AND(COM(A_{CARRY}), B_{SUM})) \cdot 2^{\frac{m}{2}}$
7: $M = M + (AND(COM(B_{CARRY}), A_{SUM})) \cdot 2^{\frac{m}{2}}$
8: $M = M + (AND(A_{CARRY}, B_{CARRY})) \cdot 2^{\frac{m}{2}}$
9: $C = L + (M - L - H) \cdot 2^{\frac{m}{2}} + H \cdot 2^m$
10: **return** C

Karatsuba Montgomery Multiplication for NEON

- **How to apply Karatsuba squaring for NEON**
  - Tasks with high overheads (sqr) $\rightarrow$ SIMD
  - Small and sequential tasks (others) $\rightarrow$ SISD

Algorithm 3: Subtractive Karatsuba Squaring on SIMD

Require: An even m-bit operand $A(A_{LOW} + A_{HIGH} \cdot 2^{\frac{m}{2}})$

Ensure: $2m$-bit result $C = A \cdot A$

1: $L = A_{LOW} \cdot A_{LOW}$ {SIMD}
2: $H = A_{HIGH} \cdot A_{HIGH}$ {SIMD}
3: $\{A_{BORROW}, A_{DIFF}\} = A_{LOW} - A_{HIGH}$
4: $A_{DIFF} = XOR(A_{BORROW}, A_{DIFF})$
5: $A_{DIFF} = A_{DIFF} + COM(A_{BORROW})$
6: $M = A_{DIFF} \cdot A_{DIFF}$ {SIMD}
7: $C = L + (L + H - M) \cdot 2^{\frac{m}{2}} + H \cdot 2^{m}$
8: **return** C

Karatsuba Montgomery Multiplication for NEON

- **18%** improvements for RSA encryption
- Our method is considered in upcoming OpenSSL

Bit	Cortex-A9					
	Our	[14]	NEON[3]	ARM[3]	GMP[6]	OpenSSL[11]
Multiplication						
1024	**3791**	4298	-	-	6256	-
2048	**13736**	17080	-	-	19618	-
Squaring						
1024	**3315**	-	-	-	4063	-
2048	**9180**	-	-	-	14399	-
Montgomery Multiplication						
1024	**8245**	8358	17464	10167	-	-
2048	**30940**	32732	63900	36746	-	-
Montgomery Squaring						
1024	**7837**	-	-	-	-	-
2048	**26860**	-	-	-	-	-
RSA encryption						
1024	**156502**	167160*	379736	245167	214064	294831
2048	**535020**	654640*	1358955	872468	791911	1029724
RSA decryption						
1024	**2965820**	-	7166897	4233862	-	4896000
2048	**20977660**	-	47205919	27547434	-	33134700

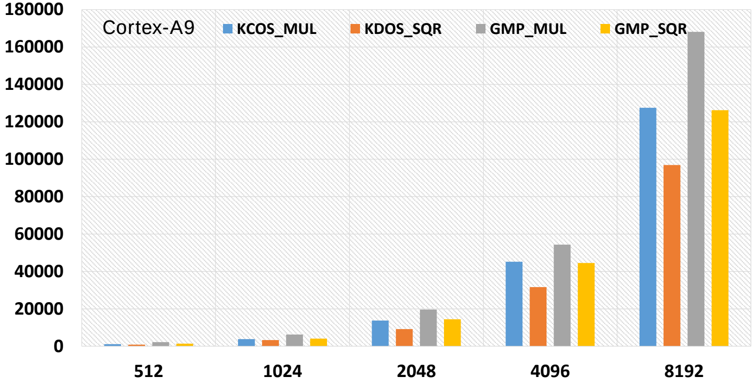
Karatsuba Montgomery Multiplication for NEON

- High reduction by **68%** and **76%** for mul and sqr

Bit	Number of Multiplication			
	[14] (mul)	KCOS (mul)	KDOS (sqr)	Karatsuba Level
512	256	256	192	-
1024	1024	768	576	1
2048	4096	2304	1728	2
4096	16384	6912	5184	3
8192	65536	20736	15552	4

Karatsuba Montgomery Multiplication for NEON

- Comparison results with **GMP**



Outline

1. Short Overview
2. RSA
   - Hybrid Montgomery Reduction
   - Karatsuba Montgomery Multiplication for NEON
3. ECC
   - High Speed Microsoft ECC Curve for AVR
   - Faster ECC for P521 on NEON
   - Faster ECC for K571 on NEON
4. Ring-LWE
   - Tiny Montgomery for Ring-LWE
   - Parallel Implementation of NTT
5. Conclusion

NIST Workshop on Elliptic Curve Cryptography Standards

- **NIST Workshop on Elliptic Curve Cryptography Standards** (June 11 - 12, 2015)
  - Security of Elliptic Curves
  - Elliptic Curve Specifications and Criteria
  - Interoperability
  - Performance
  - Intellectual Property

Outline

1 Short Overview

2 RSA
- Hybrid Montgomery Reduction
- Karatsuba Montgomery Multiplication for NEON

3 ECC
- High Speed Microsoft ECC Curve for AVR
- Faster ECC for P521 on NEON
- Faster ECC for K571 on NEON

4 Ring-LWE
- Tiny Montgomery for Ring-LWE
- Parallel Implementation of NTT

5 Conclusion

Curve Parameters for Edwards Curves

- Edwards form $x^2 + y^2 = 1 + dx^2y^2$ over $\mathbb{F}_p$
  - **NUMS256**
    - $p = 2^{256} - 189,\ d = 15531$
  - **TED379**
    - $p = 2^{379} - 19,\ d = 143305$
  - **NUMS384**
    - $p = 2^{384} - 317,\ d = 11873$

384-bit Multiplication and Squaring

Table: Execution time and code size of 384-bit mul/sqr

Integer	Combination	Time (cycles)	ROM (bytes)	Karatsuba
384-bit MUL	(1) + (3)	**11,898**	5,474	3 levels
384-bit SQR	(2) + (4)	**8,037**	3,800	3 levels
384-bit MUL	(1) + (5)	14,382	**1,704**	1 level
384-bit SQR	(6)	8,505	**832**	No

(1): Constant-time subtractive Karatsuba multiplication (in ANSI C).
(2): Constant-time subtractive Karatsuba squaring (in ANSI C).
(3): 192-bit Karatsuba multiplication [7] (in Asembly).
(4): 192-bit Karatsuba squaring [13] (in Assembly).
(5): Optimized RPS multiplication (in Assembly).
(6): Optimized RPS squaring (in Assembly).

Fast Reduction for TED379

- **Modulus conversion:** $2^{379} \equiv 19 \Longrightarrow 2^{384} \equiv 608 \bmod p_1$
- Fast reduction with **two modulus variables**

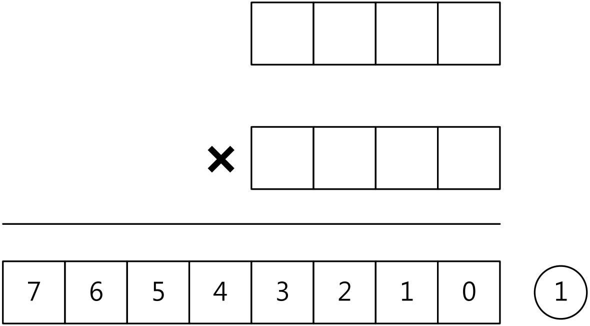
Algorithm 4: Fast (Incomplete) Reduction for p_{379}

Require: A $2n$-bit product $Z = ZH \cdot 2^{384} + ZL$, the constants $c1 = 2^5 \cdot 19$ and $c2 = 19$.
Ensure: The incomplete reduction result $R = Z \bmod p_1 \in [0, 2^{380}]$.

1: $T1[0 \sim 5] \leftarrow ZH[44 \sim 47] \times c1 + ZL[44 \sim 47]$
{The first reduction is based on the equation: $2^{384} \equiv 608 \bmod p_1$}
2: $ZL[44 \sim 47] \leftarrow ((T1[3]\&0x07) \parallel T1[0 \sim 2])$
{Get and store the bit-section, which is less than 2^{379}}
3: $T2[0 \sim 2] \leftarrow (T1[3 \sim 5] \gg 3) \times 19$
{The second reduction is based on the equation: $2^{379} \equiv 19 \bmod p_1$}
4: $R[0 \sim 47] \leftarrow (ZH[0 \sim 43] \times c1) + ZL[0 \sim 47] + T2[0 \sim 2]$
{The second reduction.}
5: **return** R

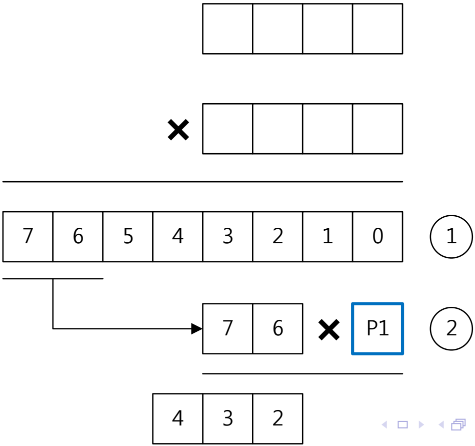
Fast Reduction for TED379

- (Step 1) multiplication



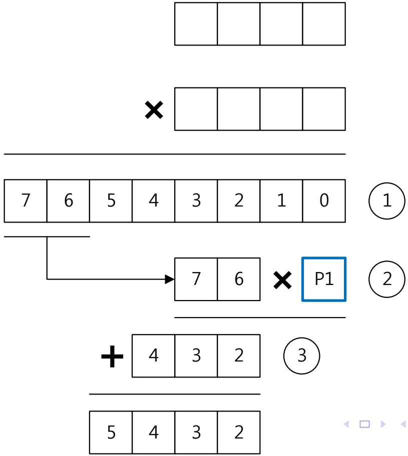
Fast Reduction for TED379

- (Step 2) reduction with P1 on higher parts



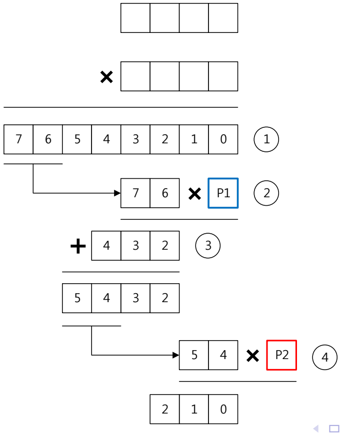
Fast Reduction for TED379

- (Step 3) updating the intermediate results



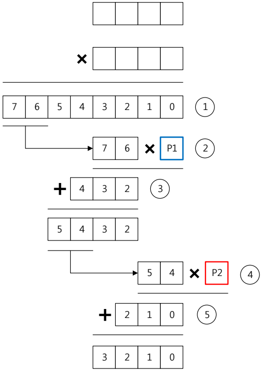
Fast Reduction for TED379

- (Step 4) reduction with P2 on lower parts



Fast Reduction for TED379

- (Step 5) updating the intermediate results



Inversion for TED379

Algorithm 5: Fermat's Little Theorem-based inversion mod p_{379}

Require: Integer A satisfying $1 \leq A \leq p-1$.
Ensure: Inverse $Z = A^{p-2} \bmod p = A^{-1} \bmod p$.

$a_2 \leftarrow a^2$	{exp: 2, cost: 1S+0M}
$a_9 \leftarrow (a_2)^{2^2} \cdot a$	{exp: 9, cost: 2S+1M}
$a_{11} \leftarrow (a_9) \cdot a_2$	{exp: 11, cost: 0S+1M}
$t_1 \leftarrow (a_{11})^2 \cdot a_9$	{exp: $2^5 - 1$, cost: 1S+1M}
$t_2 \leftarrow (t_1)^{2^5} \cdot t_1$	{exp: $2^{10} - 1$, cost: 5S+1M}
$t_3 \leftarrow (t_2)^{2^1} \cdot a$	{exp: $2^{11} - 1$, cost: 1S+1M}
$t_4 \leftarrow (t_3)^{2^{11}} \cdot t_3$	{exp: $2^{22} - 1$, cost: 11S+1M}
$t_5 \leftarrow (t_4)^{2^{22}} \cdot t_4$	{exp: $2^{44} - 1$, cost: 22S+1M}
$t_6 \leftarrow (t_5)^{2^{44}} \cdot t_5$	{exp: $2^{88} - 1$, cost: 44S+1M}
$t_7 \leftarrow (t_6)^{2^{88}} \cdot t_6$	{exp: $2^{176} - 1$, cost: 88S+1M}
$t_8 \leftarrow (t_7)^{2^{176}} \cdot t_7$	{exp: $2^{352} - 1$, cost: 176S+1M}
$t_9 \leftarrow (t_8)^{2^{22}} \cdot t_4$	{exp: $2^{374} - 1$, cost: 22S+1M}
$Z \leftarrow (t_9)^{2^5} \cdot a_{11}$	{exp: $2^{379} - 21$, cost: 5S+1M}
return Z	

Results of MS Curves for AVR

- TED379 shows the most efficient performance
- Implementation is included in Microsoft library

Table: Execution time of field arithmetic (unroll and looped fashions)

Operation	ADD	SUB	MUL	SQR	INV	CMUL
p_{256}	550	550	6,301	4,489	1,218,645	830
p_{379} (U/L)	363	686	**12,971**/15,455	**9,081**/10,496	**3,566,477**/4,132,598	1424
p_{384} (U/L)	959	959	**13,113**/15,590	**9,254**/10,663	**3,715,866**/4,287,720	1227

Table: Performance comparison of scalar multiplication

Operation	NUMS256	Ted379	NUMS384	Curve41417	Ed448
TWE.WIN ($w = 5$)	**15,807,767**	**45,350,543**	**47,096,411**†	49,919,039†	59,421,773†
MON. *	**15,125,232**	**44,245,500**	**46,482,328**†	48,727,345†	58,619,275†

†: Estimated results.
*: **Constant multiplication** $(A+2)/4$ **is used in point doubling**.

Outline

1. Short Overview
2. RSA
   - Hybrid Montgomery Reduction
   - Karatsuba Montgomery Multiplication for NEON
3. ECC
   - High Speed Microsoft ECC Curve for AVR
   - Faster ECC for P521 on NEON
   - Faster ECC for K571 on NEON
4. Ring-LWE
   - Tiny Montgomery for Ring-LWE
   - Parallel Implementation of NTT
5. Conclusion

High Security Levels of ECC

- HTTPS switches to RSA2048 (2^{112}) or 256-bit ECC (2^{128})
- **Is higher security ($2^{128} >$) useful?** **Yes**
  - **Cryptographic needs time to be reviewed for decade**
  - **Higher security level provides margin against attacks**
  - ECC protocols failed to provide 2^{128} using 256-bit curves
  - AES256 costs 40% more than AES128; ECC needs efficiency
  - **Top secret information demands high cryptography**

Curve Parameters for P521

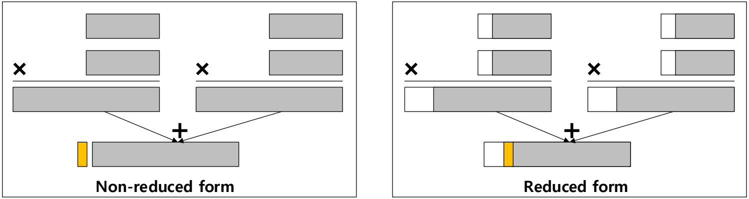
Curve $E : y^2 = x^3 + ax + b$ over $\mathbb{F}_p$ where $p = 2^{521} - 1$

$a =$ `01FF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFC`

$b =$ `0051 953EB961 8E1C9A1F 929A21A0 B68540EE A2DA725B 99B315F3 B8B48991 8EF109E1 56193951 EC7E937B 1652C0BD 3BB1BF07 3573DF88 3D2C34F1 EF451FD4 6B503F00`

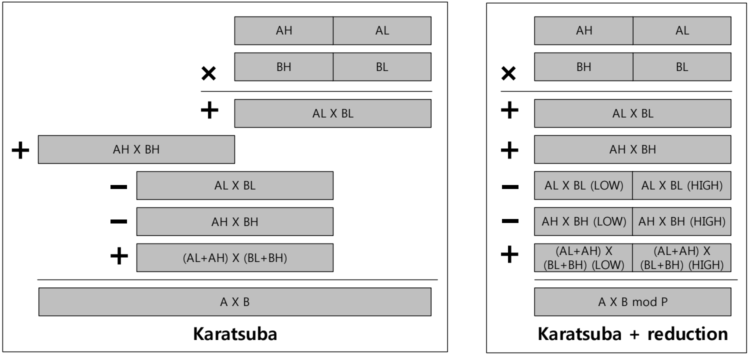
Selection of Reduced Radix

- **Modulus conversion:** $2^{521} - 1 \Longrightarrow 2^{522} - 2$
- **Reduced representation:** (522-bit / 20-limb = 26.1)
  - (27, 26, 26, 26, 26, 26, 26, 26, 26, 26) x 2



Karatsuba-based Multiplication mod p_{521}

- Combination of Karatsuba and fast reduction $(2^{522} - 2)$



Karatsuba-based Multiplication mod p_{521}

Algorithm 6: Karatsuba-based multiplication mod p_{521}

Require: Integer a, b satisfying $1 \leq a, b \leq p - 1$.
Ensure: Results $z = a \cdot b \bmod p$.
1: $a_L \leftarrow a \; mod \; 2^{261}$
2: $a_H \leftarrow a \; div \; 2^{261}$
3: $b_L \leftarrow b \; mod \; 2^{261}$
4: $b_H \leftarrow b \; div \; 2^{261}$
5: $r_L \leftarrow a_L \cdot b_L$
6: $t \leftarrow (r_L - a_H \cdot b_H \cdot 2^{261}) \; mod \; p$ {direct reduction}
7: $t_H \leftarrow t \; div \; 2^{261}$
8: $t_L \leftarrow t \; mod \; 2^{261}$
9: $t_{HL} \leftarrow t_H - t_L$
10: $a_K \leftarrow a_L + a_H$
11: $b_K \leftarrow b_L + b_H$
12: $ab_K \leftarrow (t_{HL} + t_L - t_H \cdot 2^{261} + a_K \cdot b_K \cdot 2^{261}) \; mod \; p$ {direct reduction}
13: **return** ab_K

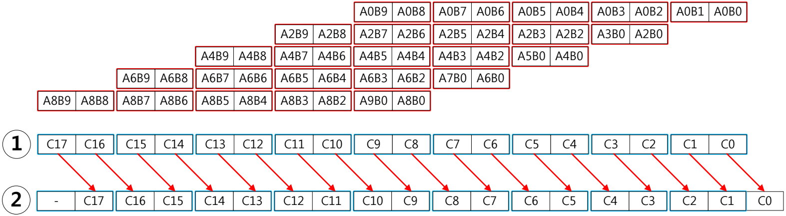
261-bit Multiplication on NEON

- (Step1): multiplication

										A0B9	A0B8	A0B7	A0B6	A0B5	A0B4	A0B3	A0B2	A0B1	A0B0
								A2B9	A2B8	A2B7	A2B6	A2B5	A2B4	A2B3	A2B2	A3B0	A2B0		
							A4B9	A4B8	A4B7	A4B6	A4B5	A4B4	A4B3	A4B2	A5B0	A4B0			
				A6B9	A6B8	A6B7	A6B6	A6B5	A6B4	A6B3	A6B2	A7B0	A6B0						
	A8B9	A8B8	A8B7	A8B6	A8B5	A8B4	A8B3	A8B2	A9B0	A8B0									
1	C17	C16	C15	C14	C13	C12	C11	C10	C9	C8	C7	C6	C5	C4	C3	C2	C1	C0

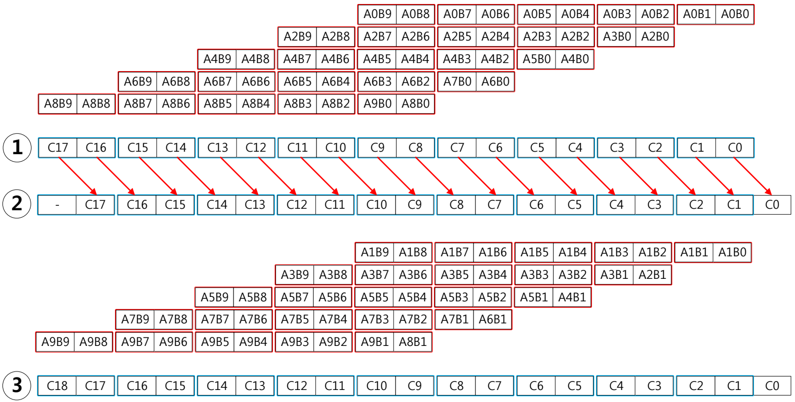
261-bit Multiplication on NEON

- (Step2): alignment of intermediate results



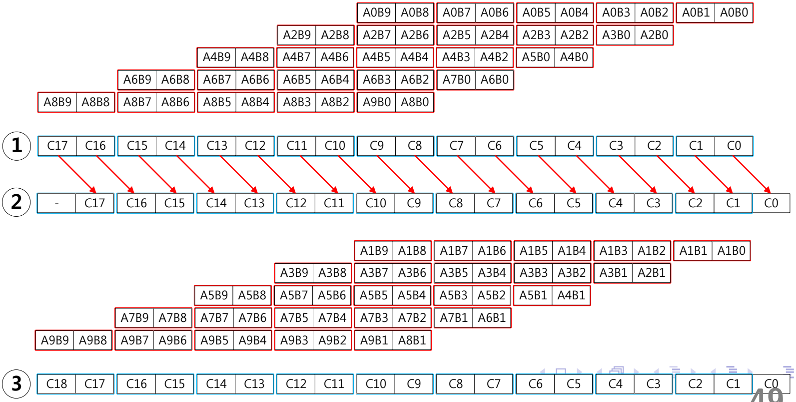
261-bit Multiplication on NEON

- (Step3): multiplication



261-bit Multiplication on NEON

- **261-bit multiplication**
  - 10-limb in 27/26 radix
  - 100 times of multiplications in 50 SIMD operations



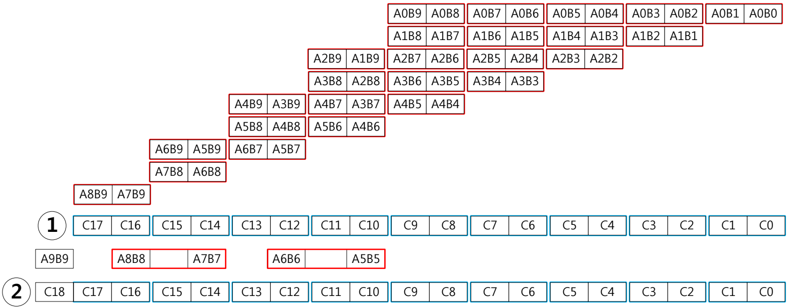
261-bit Squaring on NEON

- (Step1): multiplication

									A0B9	A0B8	A0B7	A0B6	A0B5	A0B4	A0B3	A0B2	A0B1	A0B0
									A1B8	A1B7	A1B6	A1B5	A1B4	A1B3	A1B2	A1B1		
							A2B9	A1B9	A2B7	A2B6	A2B5	A2B4	A2B3	A2B2				
							A3B8	A2B8	A3B6	A3B5	A3B4	A3B3						
					A4B9	A3B9	A4B7	A3B7	A4B5	A4B4								
					A5B8	A4B8	A5B6	A4B6										
			A6B9	A5B9	A6B7	A5B7												
			A7B8	A6B8														
	A8B9	A7B9																
1	C17	C16	C15	C14	C13	C12	C11	C10	C9	C8	C7	C6	C5	C4	C3	C2	C1	C0

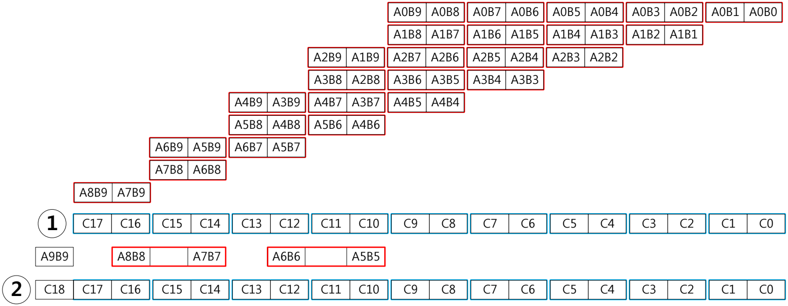
261-bit Squaring on NEON

- (Step2): remaining multiplications



261-bit Squaring on NEON

- **261-bit squaring**
  - 10-limb in 27/26 radix
  - 55 times of multiplications in 28 SIMD operations



Inversion for P521 on NEON

Algorithm 7: Fermat-based inversion mod p_{521}

Require: Integer a_1 satisfying $1 \leq a_1 \leq p-1$.
Ensure: Inverse $z = a_1^{p-2} \bmod p = a_1^{-1} \bmod p$.

1: $a_2 \leftarrow a_1^2 \cdot a_1$ {cost: 1S+1M}
2: $a_3 \leftarrow a_2^2 \cdot a_1$ {cost: 1S+1M}
3: $a_6 \leftarrow a_3^{2^3} \cdot a_3$ {cost: 3S+1M}
4: $a_7 \leftarrow a_6^2 \cdot a_1$ {cost: 1S+1M}
5: $a_8 \leftarrow a_7^2 \cdot a_1$ {cost: 1S+1M}
6: $a_{16} \leftarrow a_8^{2^8} \cdot a_8$ {cost: 8S+1M}
7: $a_{32} \leftarrow a_{16}^{2^{16}} \cdot a_{16}$ {cost: 16S+1M}
8: $a_{64} \leftarrow a_{32}^{2^{32}} \cdot a_{32}$ {cost: 32S+1M}
9: $a_{128} \leftarrow a_{64}^{2^{64}} \cdot a_{64}$ {cost: 64S+1M}
10: $a_{256} \leftarrow a_{128}^{2^{128}} \cdot a_{128}$ {cost: 128S+1M}
11: $a_{512} \leftarrow a_{256}^{2^{256}} \cdot a_{256}$ {cost: 256S+1M}
12: $a_{519} \leftarrow a_{512}^{2^7} \cdot a_7$ {cost: 7S+1M}
13: $a_1^{2^{521}-3} \leftarrow a_{519}^{2^2} \cdot a_1$ {cost: 2S+1M}
14: **return** $a_1^{2^{521}-3}$

Performance of P521 on NEON

- 2.9x/4.6x faster than OpenSSL over A9/A15

Table: Clock Cycles of OpenSSL

Curve	A9 op/s	cycles	A15 op/s	cycles
nist256	475.5	3,570,000	574.9	2,720,000
nist384	154.6	11,050,000	223.0	7,200,000
nist521	71.2	**23,800,000**	85.3	**18,720,000**

Table: Clock Cycles of Proposed Methods

Target	Unknown Point			Fixed Point			ECDH
	w=4	w=5	w=6	w=4	w=5	w=6	
Cortex-A9	6,291,936	6,098,946	6,011,768	3,056,410	2,527,714	2,147,404	**8,159,172**
Cortex-A15	3,097,904	3,003,728	2,970,976	1,503,661	1,243,027	1,056,902	**4,027,878**

Outline

1. Short Overview
2. RSA
   - Hybrid Montgomery Reduction
   - Karatsuba Montgomery Multiplication for NEON
3. ECC
   - High Speed Microsoft ECC Curve for AVR
   - Faster ECC for P521 on NEON
   - Faster ECC for K571 on NEON
4. Ring-LWE
   - Tiny Montgomery for Ring-LWE
   - Parallel Implementation of NTT
5. Conclusion

Curve Parameters for K571

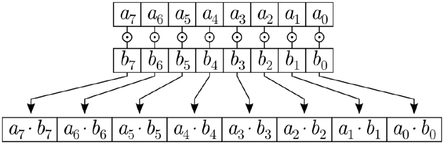
- $E : y^2 = xy = x^3 + ax^2 + b$ over $\mathbb{F}_{2^m}$ is defined by: $a = 0,\ b = 1$

$$f(x) = x^{571} + x^{10} + x^5 + x^2 + 1$$

- K-571 curve satisfies the *Frobenius map* $\tau : E(\mathbb{F}_2^m) \rightarrow E(\mathbb{F}_2^m)$

Polynomial multiplication for ARMv7

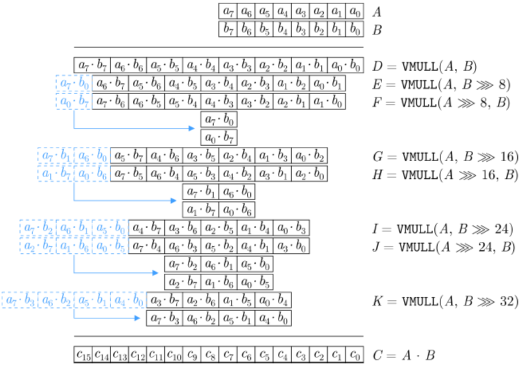
- ARMv7 provides 8-bit vectorized polynomial multiplication



$\texttt{VMULL}(A,\ B)$

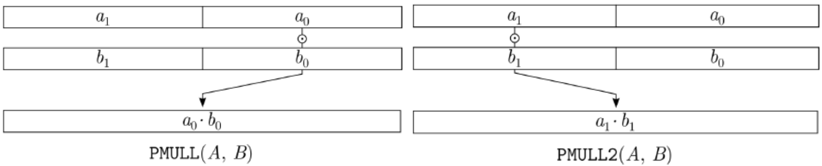
Polynomial multiplication for ARMv7

- 64-bit polynomial needs eight 8-bit vectorized multiplications
- For result alignments, shift operations are conducted



Polynomial multiplication for ARMv8

- ARMv8 provides 64-bit polynomial multiplication



Polynomial multiplication for ARMv8

- 192-bit multiplication → Ordinary method
- 576-bit multiplication → Karatsuba algorithm

Table: Comparison of 192-bit polynomial multiplication methods

Instructions	PMULL	EOR	MOVI	EXT
Ordinary	9	7	1	3
Karatsuba	6	16	1	8

Polynomial multiplication for ARMv8

- Three terms of Karatsuba multiplication for 571-bit

Algorithm 8: 571-bit Polynomial Multiplication

Require: 571-bit Operands A ($A[8 \sim 0]$) and B ($B[8 \sim 0]$).
Ensure: 1142-bit Result C ($C[17 \sim 0]$) $= A \cdot B$.

1: $A = \{A_H, A_M, A_L\} = \{(A[8], A[7], A[6]), (A[5], A[4], A[3]), (A[2], A[1], A[0])\}$
2: $B = \{B_H, B_M, B_L\} = \{(B[8], B[7], B[6]), (B[5], B[4], B[3]), (B[2], B[1], B[0])\}$
3: $C_H = (A_H \times_{192} B_H) \ll 384$ {192-bit mul}
4: $C_M = (A_M \times_{192} B_M) \ll 192$ {192-bit mul}
5: $C_L = A_L \times_{192} B_L$ {192-bit mul}
6: $T = C_H \oplus C_M \oplus C_L$
7: $C = T \oplus (T \ll 192) \oplus (T \ll 384)$
8: $C_H = ((A_H \oplus A_M) \times_{192} (B_H \oplus B_M)) \ll 576$ {192-bit mul}
9: $C_M = ((A_H \oplus A_L) \times_{192} (B_H \oplus B_L)) \ll 384$ {192-bit mul}
10: $C_L = ((A_M \oplus A_L) \times_{192} (B_M \oplus B_L)) \ll 192$ {192-bit mul}
11: $C = C_H \oplus C_M \oplus C_L$

Polynomial squaring and reduction for K571

Algorithm 9: 571-bit Polynomial Squaring

Require: 571-bit Operand A ($A[8 \sim 0]$).
Ensure: 1142-bit Result C ($C[17 \sim 0]$) $= A \times_{571} A$.

1: **for** i = 0 to 8 by 1 **do**
2: $\{C[2 \times i + 1] || C[2 \times i]\} = A[i] \times_{64} A[i]$ {series of 64-bit multiplications}
3: **end for**

Algorithm 10: Binary Field Reduction over $\mathbb{F}_{2^{571}}$

Require: 1142-bit Operands A ($A_H || A_L$).

Ensure: 571-bit Result C.

1: $r =$0x425 $\{x^{10} + x^5 + x^2 + 1\}$
2: $A_L = A \bmod 2^{571}$
3: $A_H = A \text{ div } 2^{571}$
4: $T = A_L \oplus (A_H \cdot r)$ {multiplication based reduction}
5: $T_L = T \bmod 2^{571}$
6: $T_H = T \text{ div } 2^{571}$
7: $C = T_L \oplus (T_H \cdot r)$ {multiplication based reduction}

Inversion for K571

Algorithm 11: Fermat-based inversion mod $\mathbb{F}_{2^{571}}$

Require: Integer a_1 satisfying $1 \leq a_1 \leq 2^m$.
Ensure: Inverse $z = a_1^{2^m-2} = a_1^{-1}$.

1: $a_2 \leftarrow (a_1)^{2^1} \cdot a_1$ {cost: 1S+1M}
2: $a_4 \leftarrow (a_2)^{2^2} \cdot a_2$ {cost: 2S+1M}
3: $a_8 \leftarrow (a_4)^{2^4} \cdot a_4$ {cost: 4S+1M}
4: $a_{16} \leftarrow (a_8)^{2^8} \cdot a_8$ {cost: 8S+1M}
5: $a_{17} \leftarrow (a_{16})^{2^1} \cdot a_1$ {cost: 1S+1M}
6: $a_{34} \leftarrow (a_{17})^{2^{17}} \cdot a_{17}$ {cost: 17S+1M}
7: $a_{35} \leftarrow (a_{34})^{2^1} \cdot a_1$ {cost: 1S+1M}
8: $a_{70} \leftarrow (a_{35})^{2^{35}} \cdot a_{35}$ {cost: 35S+1M}
9: $a_{71} \leftarrow (a_{70})^{2^1} \cdot a_1$ {cost: 1S+1M}
10: $a_{142} \leftarrow (a_{71})^{2^{71}} \cdot a_{71}$ {cost: 71S+1M}
11: $a_{284} \leftarrow (a_{142})^{2^{142}} \cdot a_{142}$ {cost: 142S+1M}
12: $a_{285} \leftarrow (a_{284})^{2^1} \cdot a_1$ {cost: 1S+1M}
13: $a_{570} \leftarrow (a_{285})^{2^{285}} \cdot a_{285}$ {cost: 285S+1M}
14: **return** $(a_{570})^{2^1}$ {cost: 1S}

Results for K571 over ARMv8

- **2.5x** faster than [4] (KNV/VMULL)

Table: Comparison results over K-571 curve

Algorithm	Processor	Architecture	Clock Cycles
Multiplication			
KNV [4]	Apple-A7	ARMv8	698
Proposed Method (KNP)	**Apple-A7**	**ARMv8**	**132**
Squaring			
VMULL [4]	Apple-A7	ARMv8	60
Proposed Method (PMULL)	**Apple-A7**	**ARMv8**	**41**
Inversion (Itoh-Tsujii)			
KNV/VMULL [4]	Apple-A7	ARMv8	44,631
Proposed Method (KNP/PMULL)	**Apple-A7**	**ARMv8**	**31,232**
Scalar multiplication			
KNV/VMULL (Unknown Point) [4]	Apple-A7	ARMv8	1,023,100
KNV/VMULL (Fixed Point) [4]	Apple-A7	ARMv8	929,500
Proposed Method (Unknown Point)	**Apple-A7**	**ARMv8**	**408,720**
Proposed Method (Fixed Point)	**Apple-A7**	**ARMv8**	**374,985**
ECDH Agreement			
KNV/VMULL [4]	Apple-A7	ARMv8	1,952,600
Proposed Method (KNP/PMULL)	**Apple-A7**	**ARMv8**	**783,705**

Outline

1. Short Overview
2. RSA
   - Hybrid Montgomery Reduction
   - Karatsuba Montgomery Multiplication for NEON
3. ECC
   - High Speed Microsoft ECC Curve for AVR
   - Faster ECC for P521 on NEON
   - Faster ECC for K571 on NEON
4. Ring-LWE
   - Tiny Montgomery for Ring-LWE
   - Parallel Implementation of NTT
5. Conclusion

Outline

1. Short Overview
2. RSA
   - Hybrid Montgomery Reduction
   - Karatsuba Montgomery Multiplication for NEON
3. ECC
   - High Speed Microsoft ECC Curve for AVR
   - Faster ECC for P521 on NEON
   - Faster ECC for K571 on NEON
4. Ring-LWE
   - Tiny Montgomery for Ring-LWE
   - Parallel Implementation of NTT
5. Conclusion

Ring-LWE Encryption Scheme

- **Number Theoretic Transform**
  - Polynomial multiplication in the n-th roots of unity

Algorithm 12: Iterative Number Theoretic Transform

Require: Polynomial $a(x)$, n-th root of unity ω
Ensure: Polynomial $a(x) = \mathrm{NTT}(a)$

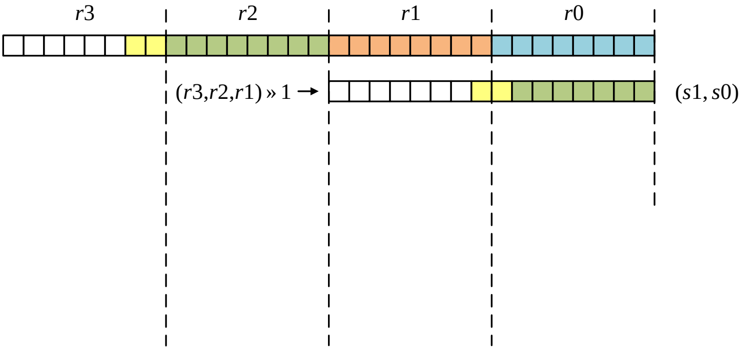
$a \leftarrow \mathrm{BitReverse}(a)$
for i from 2 by $2i$ to n **do**
  $\omega_i \leftarrow \omega_n^{n/i}$, $\omega \leftarrow 1$
  for j from 0 by 1 to $i/2 - 1$ **do**
    for k from 0 by i to $n-1$ **do**
      ① $U \leftarrow a[k+j]$, ② $V \leftarrow \omega \cdot a[k+j+i/2]$
      ③ $a[k+j] \leftarrow U + V$, ④ $a[k+j+i/2] \leftarrow U - V$
    end for
    $\omega \leftarrow \omega \cdot \omega_i$
  end for
end for
return a

Previous Optimization of NTT [CHES'15]

- **Approximation based reduction**
  - Position of 1's in $(2^w \times 1/q) \rightarrow p_1, ..., p_l$
  - $\lfloor z/q \rfloor \cong \sum_{i=1}^{l}(z \gg (w - p_i))$
  - $z \bmod q \cong z - q \times \lfloor z/q \rfloor$
  - $\lfloor z/7681 \rfloor \cong (z \gg 13) + (z \gg 17) + (z \gg 21)$

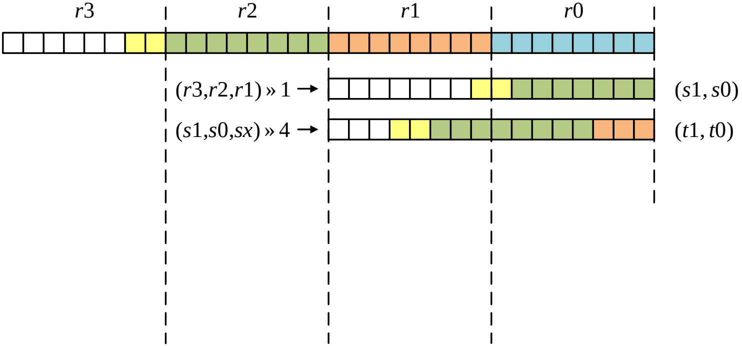
Previous Optimization of NTT [CHES'15]

- SAMS2 method, (Step1-1): shifting ($z \gg 17$)



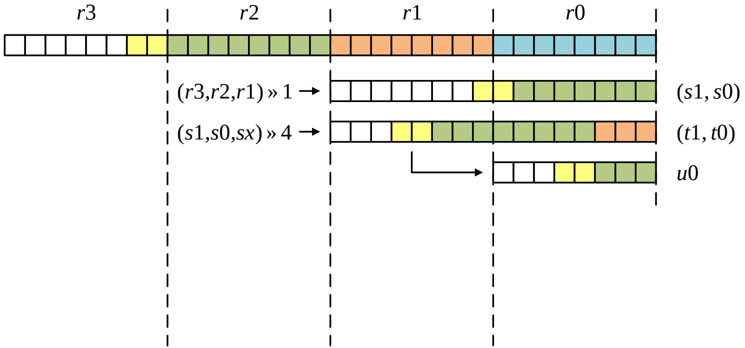
Previous Optimization of NTT [CHES'15]

- SAMS2 method, (Step1-2): shifting ($z \gg 13$)



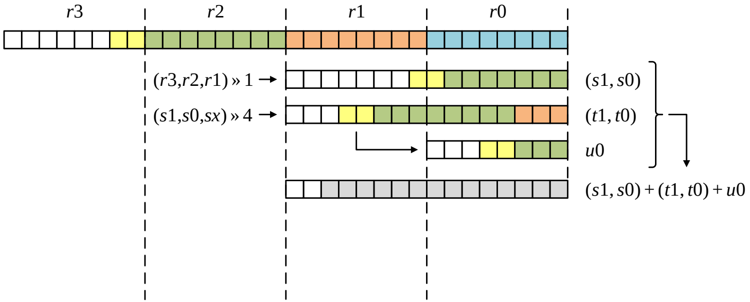
Previous Optimization of NTT [CHES'15]

- SAMS2 method, (Step1-3): shifting ($z \gg 21$)



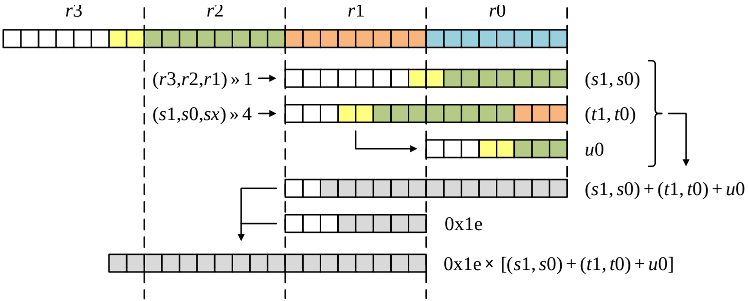
Previous Optimization of NTT [CHES'15]

- SAMS2 method, (Step2): addition $(z \gg 13) + (z \gg 17) + (z \gg 21)$



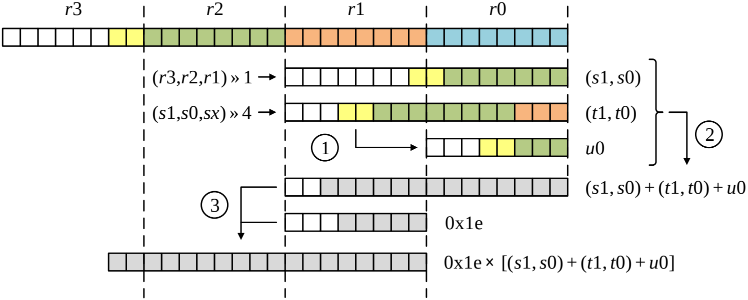
Previous Optimization of NTT [CHES'15]

- SAMS2 method, (Step3): multiplication



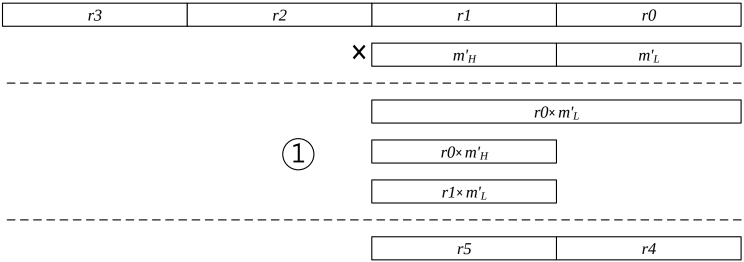
Previous Optimization of NTT [CHES'15]

- SAMS2 method, ①: shifting; ②: addition; ③: multiplication



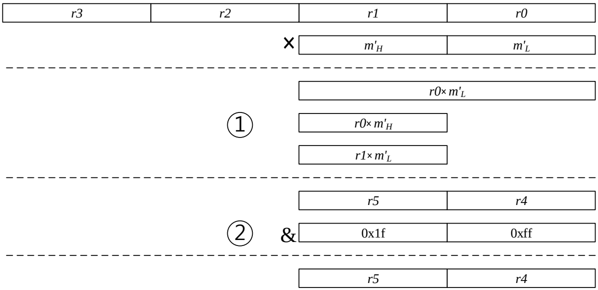
Proposed Tiny Montgomery for Ring-LWE

- Tiny Montgomery, (Step1): multiplication with inverse m'



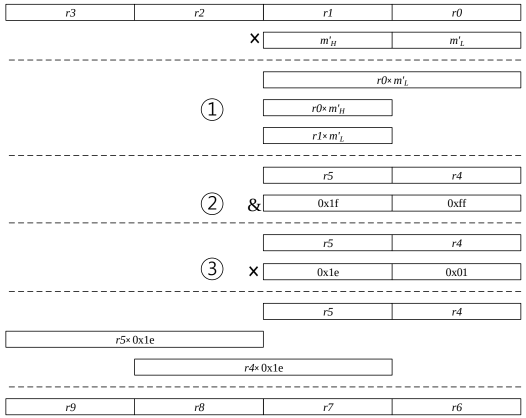
Proposed Tiny Montgomery for Ring-LWE

- Tiny Montgomery, (Step2): masking the results



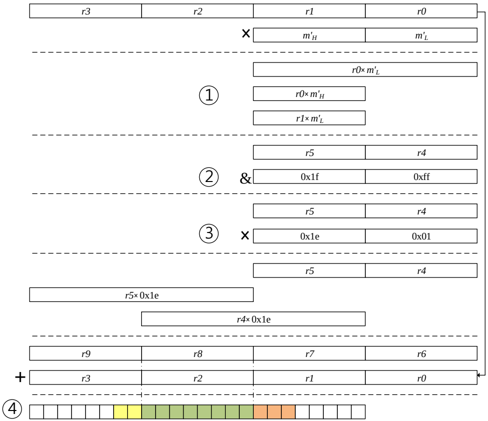
Proposed Tiny Montgomery for Ring-LWE

- Tiny Montgomery, (Step3): multiplication with modulus



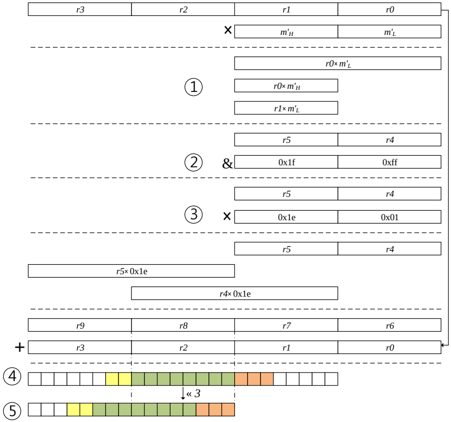
Proposed Tiny Montgomery for Ring-LWE

- Tiny Montgomery, (Step4): updating the intermeidate results



Proposed Tiny Montgomery for Ring-LWE

- Tiny Montgomery, (Step5): alignment of the results



Proposed Tiny Montgomery for Ring-LWE

- $q = 7681$
  - $a \leftarrow a - q \cdot [(a \gg 13) + (a \gg 17) + (a \gg 21)]$
  - Three times of shift operations
  - **SAMS2** method is better ☹
- $q = 12289$
  - $a \leftarrow a - q \cdot [(a \gg 14) + (a \gg 16) + (a \gg 18) + (a \gg 20) + (a \gg 22) + (a \gg 24)]$
  - Six times of shift operations
  - **Tiny Montgomery** method is better ☺
- $q = 8383489$
  - $a \leftarrow a - q \cdot [(a \gg 23) + (a \gg 34) + (a \gg 36) + (a \gg 45) + (a \gg 49)]$
  - Five times of shift operations
  - **Tiny Montgomery** method is better ☺

Proposed Tiny Montgomery for Ring-LWE

- Tiny Montgomery improves NTT case ($q = 12289$) by **4.7%**

Table: Performance comparison of software implementation

Implementations	NTT/FFT	Sampling	Gen	Enc	Dec
Implementations on 8-bit AVR processors, e.g., ATxmega64, ATxmega128:					
Boorghany et al. [1]	1,216,000	N/A	N/A	5,024,000	2,464,000
Boorghany et al. [2]	754,668	N/A	2,770,592	3,042,675	1,368,969
Pöppelmann et al. [12]	334,646	N/A	N/A	1,314,977	381,254
Zhe et al. [10] in CHES'15	193,731	26,763	589,900	671,628	275,646
This work (256)	**209,318**	26,763	637,363	725,667	315,024
Boorghany et al. [2]	2,207,787	617,600	N/A	N/A	N/A
Pöppelmann et al. [12]	855,595	N/A	N/A	3,279,142	1,019,350
Zhe et al. [10] in CHES'15	441,572	255,218	2,165,239	2,617,459	686,367
This work (512)	**420,544**	255,218	2,062,132	2,492,818	640,609

Outline

1. Short Overview
2. RSA
   - Hybrid Montgomery Reduction
   - Karatsuba Montgomery Multiplication for NEON
3. ECC
   - High Speed Microsoft ECC Curve for AVR
   - Faster ECC for P521 on NEON
   - Faster ECC for K571 on NEON
4. Ring-LWE
   - Tiny Montgomery for Ring-LWE
   - Parallel Implementation of NTT
5. Conclusion

Parallel Implementation of NTT

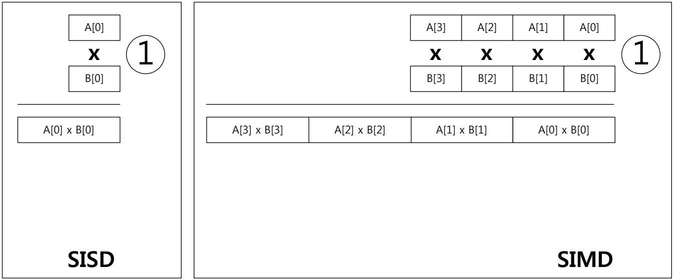
Algorithm 13: Vectorized Iterative Number Theoretic Transform

Require: A polynomial $a(x) \in \mathbb{Z}_q[x]$ of degree $n-1$ and n-th primitive $\omega \in \mathbb{Z}_q$ of unity
Ensure: Polynomial $a(x) = NTT(a) \in \mathbb{Z}_q[x]$

for i from 2 by $i = 2i$ to n **do**
  $\omega_i = \omega_n^{n/i}$, $\omega = 1$ {`LUT for twiddle factors`}
  if $i = 2$ or $i = 4$ **then**
    `sequential computation`
  else
    $\omega_{array}[0] = \omega$
    for p from 1 by 1 to $i/2 - 1$ **do**
      $\omega = \omega \cdot \omega_i$, $\omega_{array}[p] = \omega$ {`multiple computations of` ω}
    end for
    for k from 0 by i to $n-1$ **do**
      $p = 0$
      for j from 0 by 4 to $i/2 - 1$ **do**
        $U_{array} = a[k+j : k+j+3]$
        $V_{array} = \omega_{array}[p : p+3] \cdot a[k+j+i/2 : k+j+3+i/2]$
        $p = p+4$, $a[k+j : k+j+3] = U_{array} + V_{array}$
        $a[k+j+i/2 : k+j+3+i/2] = U_{array} - V_{array}$ {`parallel computation`}
      end for
    end for
  end if
end for
return a

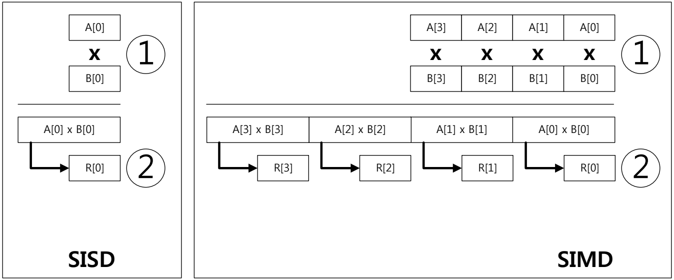
Parallel Implementation of NTT

- Parallel modular reduction, (Step1) multiplication



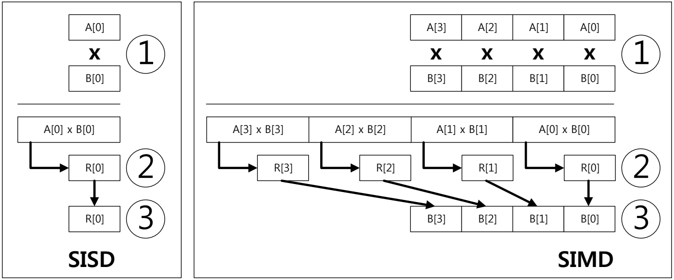
Parallel Implementation of NTT

- Parallel modular reduction, (Step2) reduction



Parallel Implementation of NTT

- Parallel modular reduction, (Step3) alignment



Parallel Implementation of NTT

- Modular operations $\rightarrow$ multiplication and subtraction (`vmls`)

Algorithm 14: Pseudo codes of vectorized NTT computation

Require: Eight 32-bit coefficients $A[0:3]$(q2), $B[0:3]$(q3), ω(q1), modulo(q0).
Ensure: Eight 32-bit results C(q5,q10).

```
vmul.i32 q3, q3, q1                  {coefficient multiplication}
vshr.u32 q4, q3, #13
vshr.u32 q5, q3, #17
vshr.u32 q6, q3, #21
vadd.i32 q4, q4, q5
vadd.i32 q4, q4, q6
vmls.i32 q3, q4, d0[0]
vshr.u32 q4, q3, #13
vmls.i32 q3, q4, d0[0]
vadd.i32 q5, q2, q3                  {coefficient addition}
vshr.u32 q4, q5, #13
vmls.i32 q5, q4, d0[0]
vshl.i32 q1, q0, #2                  {coefficient subtraction}
vadd.i32 q2, q2, q1
vsub.i32 q10, q2, q3
vshr.u32 q14, q10, #13
vmls.i32 q10, q14, d0[0]
```

Parallel Implementation of NTT

- **1.4x/1.5x** faster than previous 256/512 NTT

Table: Performance comparison of Number Theoretic Transform

Implementations	NTT
32-bit ARM-NEON processors, e.g., Cortex-A9:	
Previous work [5, 10] (256) in DATE'15 and CHES'15	39,480
This work (256)	**27,160**
Previous work [5, 10] (512) in DATE'15 and CHES'15	95,200
This work (512)	**62,160**

Outline

1. Short Overview
2. RSA
   - Hybrid Montgomery Reduction
   - Karatsuba Montgomery Multiplication for NEON
3. ECC
   - High Speed Microsoft ECC Curve for AVR
   - Faster ECC for P521 on NEON
   - Faster ECC for K571 on NEON
4. Ring-LWE
   - Tiny Montgomery for Ring-LWE
   - Parallel Implementation of NTT
5. Conclusion

Conclusion

- Contributions
  - Compact Implementations of PKC on Low/High-end Devices
- Future Works
  - Post-quantum Cryptography
  - Pairing Cryptography
  - Homomorphic Encryption
  - Light-weight Block Cipher

Thank you for your attention

A. Boorghany and R. Jalili.
Implementation and comparison of lattice-based identification protocols on smart cards and microcontrollers.
IACR Cryptology ePrint Archive, 2014:78, 2014.

A. Boorghany, S. B. Sarmadi, and R. Jalili.
On constrained implementation of lattice-based cryptographic primitives and schemes on smart cards.
ACM Transactions on Embedded Computing Systems (TECS), 14(3):42, 2015.

J. W. Bos, P. L. Montgomery, D. Shumow, and G. M. Zaverucha.
Montgomery multiplication using vector instructions.

In T. Lange, K. Lauter, and P. Lisonek, editors, *Selected Areas in Cryptography — SAC 2013*, volume 8282 of *Lecture Notes in Computer Science*, pages 471–489. Springer Verlag, 2014.

D. Câmara, C. P. Gouvêa, J. López, and R. Dahab.
Fast software polynomial multiplication on ARM processors using the NEON engine.
In *Security Engineering and Intelligence Informatics*, pages 137–154. Springer, 2013.

R. De Clercq, S. S. Roy, F. Vercauteren, and I. Verbauwhede.
Efficient software implementation of Ring-LWE encryption.
In *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition*, pages 339–344. EDA Consortium, 2015.

Free Software Foundation, Inc.

GMP: The GNU Multiple Precision Arithmetic Library.
Available for download at http://www.gmplib.org/, Feb. 2015.

M. Hutter and P. Schwabe.
Multiprecision multiplication on AVR revisited.
Journal of Cryptographic Engineering, pages 1–14, 2014.

Z. Liu and J. Großschädl.
New speed records for montgomery modular multiplication on 8-bit AVR microcontrollers.
In *Progress in Cryptology–AFRICACRYPT 2014*, pages 215–234. Springer, 2014.

Z. Liu, H. Seo, J. Groszschädl, and H. Kim.
Reverse product-scanning multiplication and squaring on 8-bit AVR processors.

In *16th International Conference on Information and Communications Security (ICICS 2014)*. Springer Verlag, 2014.

- Z. Liu, H. Seo, S. S. Roy, J. Großschädl, H. Kim, and I. Verbauwhede.
  Efficient Ring-LWE encryption on 8-bit AVR processors.

- OpenSSL.
  The open source toolkit for SSL/TLS.
  Available for download at https://www.openssl.org/, Feb. 2015.

- T. Pöppelmann, T. Oder, and T. Güneysu.
  Speed records for ideal lattice-based cryptography on AVR.

- H. Seo, Z. Liu, J. Choi, and H. Kim.
  Optimized Karatsuba squaring on 8-bit AVR processors.
  Security and Communication Networks, 2015.

H. Seo, Z. Liu, J. Großschädl, J. Choi, and H. Kim.
Montgomery modular multiplication on ARM-NEON revisited.