LSH Competition 2015: Compact Implementations of LSH

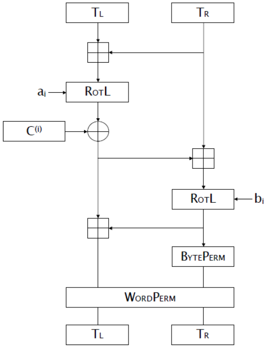
Hwajeong Seo Taehwan Park Jongseok Choi Jihyun Kim

Pusan National University

2015/08/12

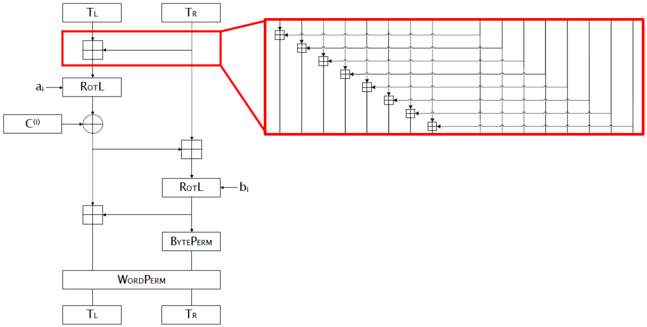
Target Hash Function (1/9)

- **LSH**
  - ARX-based hash function (Addition,Rotation,eXclusive-or)
  - 32, 64-bit wise word size
  - Hash message: 224, 256, 384, 512-bit



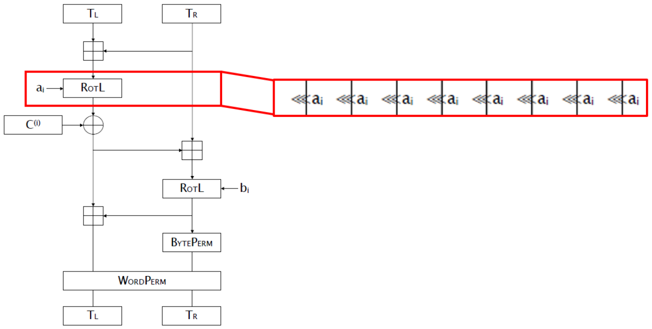
Target Hash Function (2/9)

- **LSH**
  - ARX-based hash function (Addition,Rotation,eXclusive-or)



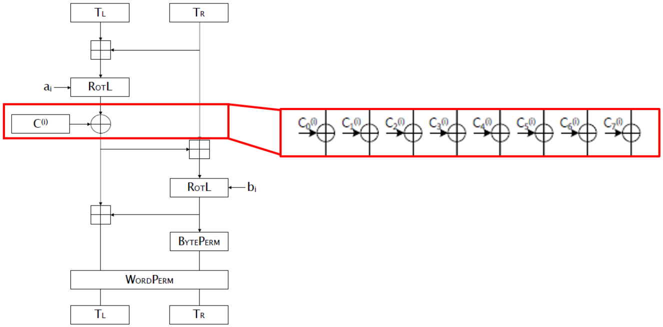
Target Hash Function (3/9)

- **LSH**
  - ARX-based hash function (Addition,Rotation,eXclusive-or)



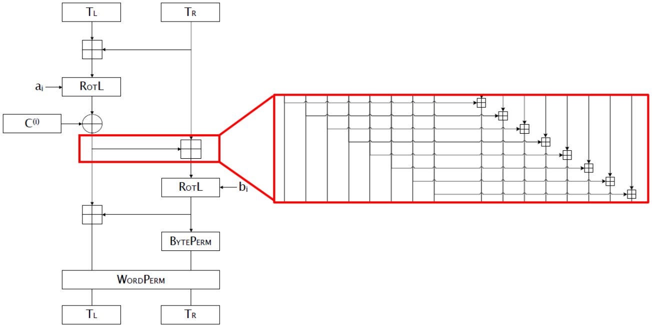
Target Hash Function (4/9)

- **LSH**
  - ARX-based hash function (Addition,Rotation,eXclusive-or)



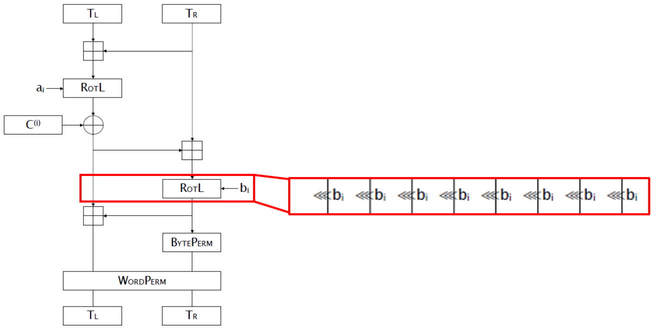
Target Hash Function (5/9)

- **LSH**
  - ARX-based hash function (Addition,Rotation,eXclusive-or)



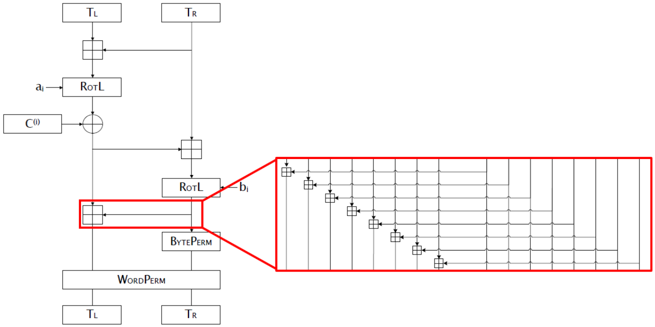
Target Hash Function (6/9)

- **LSH**
  - ARX-based hash function (Addition,Rotation,eXclusive-or)


T_L
T_R
a_i
RotL
$C^{(i)}$
RotL
b_i
$\lll b_i$ $\lll b_i$ $\lll b_i$ $\lll b_i$ $\lll b_i$ $\lll b_i$ $\lll b_i$ $\lll b_i$
BytePerm
WordPerm
T_L
T_R

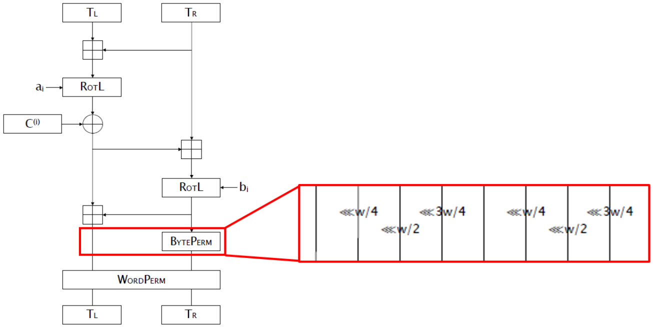
Target Hash Function (7/9)

- **LSH**
  - ARX-based hash function (Addition,Rotation,eXclusive-or)



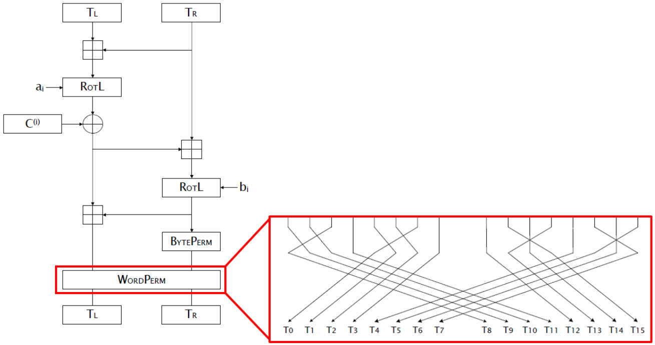
Target Hash Function (8/9)

- **LSH**
  - ARX-based hash function (Addition,Rotation,eXclusive-or)



Target Hash Function (9/9)

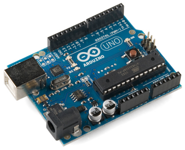
- **LSH**
  - ARX-based hash function (Addition,Rotation,eXclusive-or)



Target Devices (1/5)

- **8-bit Embedded Platform AVR**
  - 8-bit processor, 7.3728 MHz
  - 128KB EEPROM, 4KB RAM, 32 registers

Mnemonics	Operands	Description	Operation	#Clock
ADD	Rd, Rr	Add without Carry	Rd ← Rd+Rr	1
ADC	Rd, Rr	Add with Carry	Rd ← Rd+Rr+C	1
EOR	Rd, Rr	Exclusive OR	Rd ← Rd⊕Rr	1
LSL	Rd	Logical Shift Left	C|Rd ← Rd<<1	1
LSR	Rd	Logical Shift Right	Rd|C ← 1>>Rd	1
ROL	Rd	Rotate Left Through Carry	C|Rd ← Rd<<1||C	1
ROR	Rd	Rotate Right Through Carry	Rd|C ← C||1>>Rd	1



Target Devices (2/5)

- **16-bit Embedded Platform MSP**
  - 16-bit processor, 8 MHz
  - 32-48KB flash memory, 10KB RAM, 12 registers

Mnemonics	Operands	Description	Operation	#Clock
ADD	Rr, Rd	Add without Carry	Rd ← Rd+Rr	1
ADDC	Rr, Rd	Add with Carry	Rd ← Rd+Rr+C	1
AND	Rr, Rd	Logical AND	Rd ← Rd&Rr	1
XOR	Rr, Rd	Exclusive OR	Rd ← Rd⊕Rr	1
RLA	Rd	Logical Shift Left	C|Rd ← Rd<<1	1
RRA	Rd	Logical Shift Right	Rd|C ← 1>>Rd	1
RLC	Rd	Rotate Left Through Carry	C|Rd ← Rd<<1||C	1
RRC	Rd	Rotate Right Through Carry	Rd|C ← C||1>>Rd	1

Target Devices (3/5)

- **32-bit Embedded Platform ARM**
  - 32-bit processor, 1 $\sim$ 2 GHz
  - 16 registers, most instructions in a single cycle

Mnemonics	Operands	Description	Operation	#Clock
ADD	Rd, Rr	Add without Carry	Rd $\leftarrow$ Rd+Rr	1
EOR	Rd, Rr	Exclusive OR	Rd $\leftarrow$ Rd$\oplus$Rr	1
ROL	Rd	Rotate Left Through Carry	C|Rd $\leftarrow$ Rd<<1||C	1
ROR	Rd	Rotate Right Through Carry	Rd|C $\leftarrow$ C||1>>Rd	1

Target Devices (4/5)

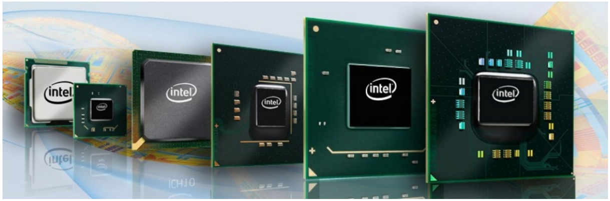
- **SIMD Embedded Platform NEON**
  - SIMD extensions on ARM
  - sixteen 128-bit registers
  - packed 8, 16, 32, 64-bit operations

Mnemonics	Operands	Description	Cycles	Source	Result	Writeback
VADD	Qd,Qn,Qm	Vector Addition	1	-,2,2	3	6
VEOR	Qd,Qn,Qm	Vector Exclusive-or	1	-,2,2	3	6
VSHL	Qd,Qm,#imm	Vector Left Shift	1	-,2,-	3	6
VSRI	Qd,Qm,#imm	Vector Right Shift with Insert	2	2,2,-	4	7

Target Devices (5/5)

- **Intel CPU Platfom**
  - 32-, 64-bit word size, high frequency with multi-core
  - Various parallel computing capabilities
    - SIMD: SSE2, SSE3, SSSE3, SSE4, AVX, AVX2, AVX512

Mnemonics	Operands	Description
_mm256_add_epi32	YMM2,YMM1,YMM0	Vector Addition
_mm256_xor_si256	YMM2,YMM1,YMM0	Vector Exclusive-or
_mm256_slli_epi32	YMM2,YMM0,#imm	Vector Left Shift
_mm256_srli_epi32	YMM2,YMM0,#imm	Vector Right Shift with Insert



Efficient Implementations of ARX operations

- 32-bit and 64-bit ARX operations over various platforms
  - Multi-precision ARX operations on 8/16/32-bit devices
  - Single word ARX operations over modern processors
  - Parallel ARX operations with SIMD architectures

8-bit AVR

32-bit Addition	32-bit Exclusive-or	32-bit Right Rotation	
ADD R12, R16	EOR R12, R16	CLR R20	ROR R16
ADC R13, R17	EOR R13, R17	LSR R19	ROR R20
ADC R14, R18	EOR R14, R18	ROR R18	EOR R19, R20
ADC R15, R19	EOR R15, R19	ROR R17	

64-bit Addition	64-bit Exclusive-or	64-bit Right Rotation	
ADD R8, R16	EOR R8, R16	CLR R24	ROR R16
ADC R9, R17	EOR R9, R17	LSR R23	ROR R20
ADC R10, R18	EOR R10, R18	ROR R22	EOR R23, R20
ADC R11, R19	EOR R11, R19	ROR R21	
ADC R12, R20	EOR R12, R20	ROR R20	
ADC R13, R21	EOR R13, R21	ROR R19	
ADC R14, R22	EOR R14, R22	ROR R18	
ADC R15, R23	EOR R15, R23	ROR R17	

Shift Offset and Direction

```
Input: direction $d$, offset $o$.
Output: direction $d$, offset $o$.
1.  $o = o \bmod 8$
2.    if o>4
3.      $o = 8 - o$
4.      $d = !d$
5.  return $d$, $o$
```

16-bit MSP

Addition	Exclusive-or	Right Rotation	
`ADD R8, R10`	`XOR R8, R10`	`CLR R12`	`RRC R12`
`ADDC R9, R11`	`XOR R9, R11`	`RRA R11`	`AND #0XEFFF, R11`
		`RRC R10`	`ADD R12, R11`

Addition	Exclusive-or	Right Rotation	
`ADD R4, R8`	`XOR R4, R8`	`CLR R12`	`RRC R8`
`ADDC R5, R9`	`XOR R5, R9`	`RRA R11`	`RRC R12`
`ADDC R6, R10`	`XOR R6, R10`	`RRC R10`	`AND #0XEFFF, R11`
`ADDC R7, R11`	`XOR R7, R11`	`RRC R9`	`ADD R12, R11`

Shift Offset and Direction

```
Input: direction $d$, offset $o$.
Output: direction $d$, offset $o$.
  1.    $o = o$ mod 16
  2.      if o>8
  3.        $o = 16 - o$
  4.        $d = !d$
  5.    return $d$, $o$
```

16-bit MSP

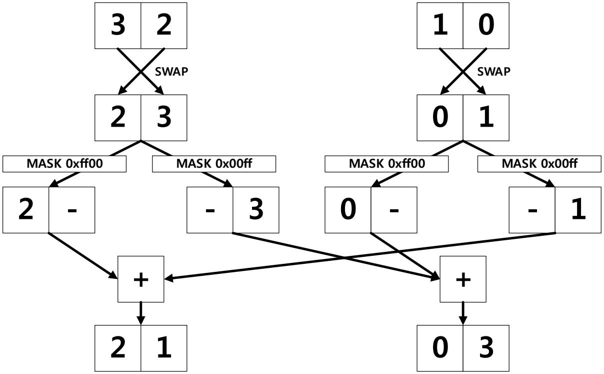


Figure: MSP: 8-bit wise rotation with swap and mask operations

32-bit ARM

Addition	Exclusive-or	Right Rotation by 31
`ADD R8, R8, R5`	`EOR R8, R8, R5`	`ROR R8, #31`

Addition	Exclusive-or	Left Rotation by 24
`ADDS R8, R8, R5`	`EOR R8, R8, R5`	`MOV R8, R5, LSL#24`
`ADCS R9, R9, R6`	`EOR R9, R9, R6`	`MOV R9, R6, LSL#24`
		`ORR R8, R8, R6, LSR#8`
		`ORR R9, R9, R5, LSR#8`

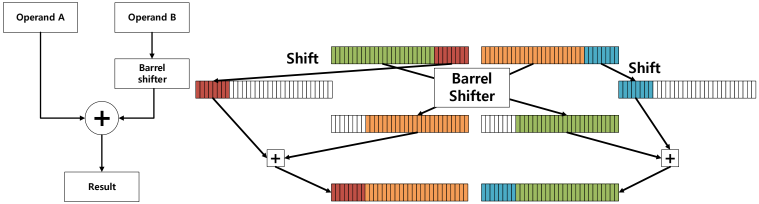
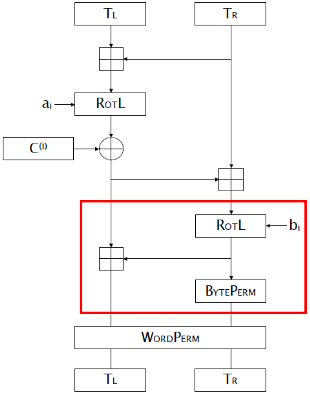


Figure: Bitwise exclusive-or with inline barrel shifter/24-bit left rotation operation with barrel shifter for 64-bit word

32-bit ARM

Sequential Way	Combined Way
`A = ROL32(A, 17)`	–
`B = B + A`	`B = B + ROL32(A, 17)//Barrel Shifter`
`A = ROL32(A,8)`	`A = ROL32(A, 25)`



Rotation

Addition

Byte Permutation (8-bit wise rotation)

SIMD Architecture: 128-bit NEON / 256-bit AVX2

4 32-bit Addition	4 32-bit Exclusive-or	4 32-bit Right Rotation by 9

`vadd.i32 q1, q1, q0`	`veor q1, q1, q0`	`vshl.i32 q1, q0, #9`
		`vsri.32 q1, q0, #23`

2 64-bit Addition	2 64-bit Exclusive-or	2 64-bit Right Rotation by 9

`vadd.i64 q1, q1, q0`	`veor q1, q1, q0`	`vshl.u64 q1, q0, #9`
		`vsri.u64 q1, q0, #23`

8 32-bit Addition	8 32-bit Exclusive-or
`C = _mm256_add_epi32(A, B)`	`C = _mm256_xor_si256(A, B)`
8 32-bit Right Rotation by 9	

`B = _mm256_slli_epi32(A, 23)`
`A = _mm256_srli_epi32(A, 9)`
	`C = _mm256_or_si256(B, A)`

4 64-bit Addition	4 64-bit Exclusive-or
`C = _mm256_add_epi64(A, B)`	`C = _mm256_xor_si256(A, B)`
4 64-bit Right Rotation by 9	

`B = _mm256_slli_epi64(A, 55)`
`A = _mm256_srli_epi64(A, 9)`
	`C = _mm256_or_si256(B, A)`

Optimizations on permutation and compression functions

- Permutation and compression functions on various platforms
  - Optimal memory access patterns on embedded processors
  - Byte and word wise permutation on various platforms

Order of compression function over embedded processor

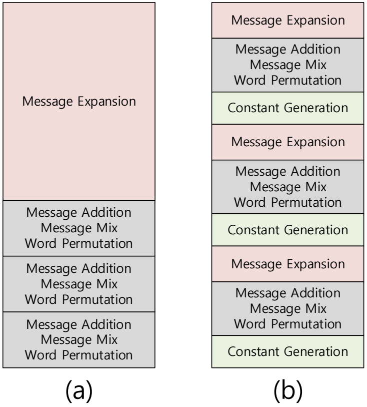


Figure: (AVR, MSP): abstraction of compression function, (a) separated, (b) on-the-fly approaches

Order of compression function over embedded processor

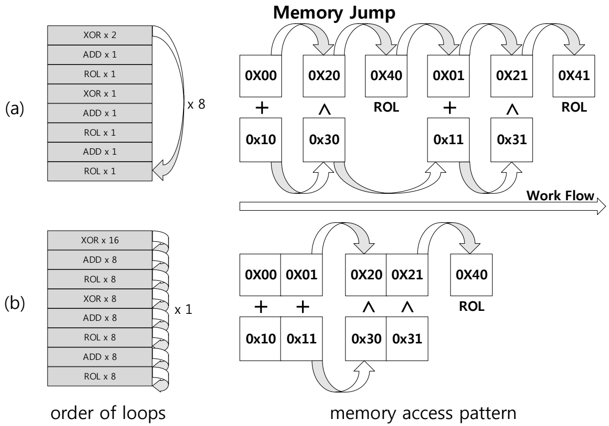


Figure: Comaprison of compression function in order of loops and memory access patterns, (a) outer-loop-first, (b) inner-loop-first

Permutation over NEON

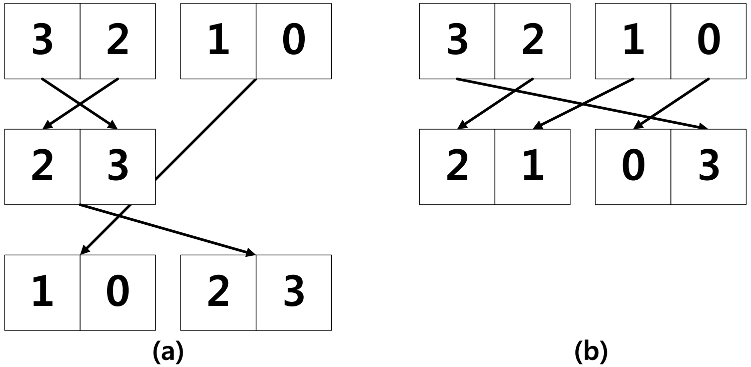


Figure: NEON: 32-bit wise word permutation, (a) $(3, 2, 1, 0) \rightarrow (2, 3, 0, 1)$, (b) (2, 3, 0, 1) $\rightarrow$ (0, 1, 2, 3)

Permutation over NEON

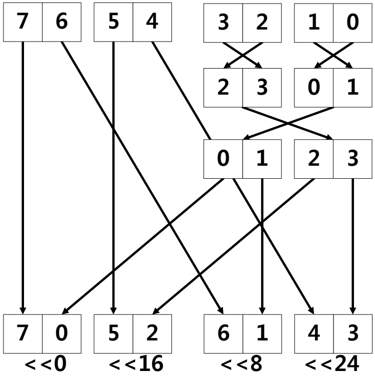
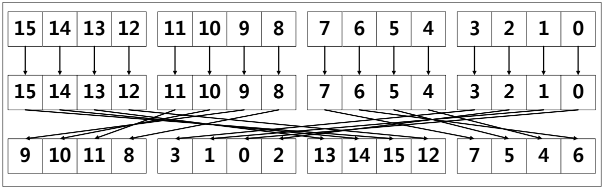
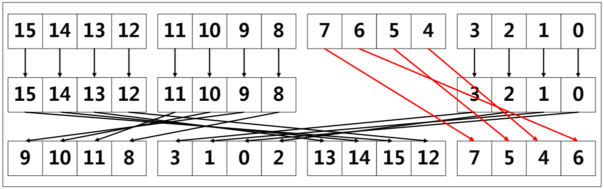


Figure: NEON: word permutation with combined shift

Permutation over Intel (SISD)



(a)



(b)

Figure: Intel: word permutation, (a) normal order, (b) direct order

Permutation over AVX2

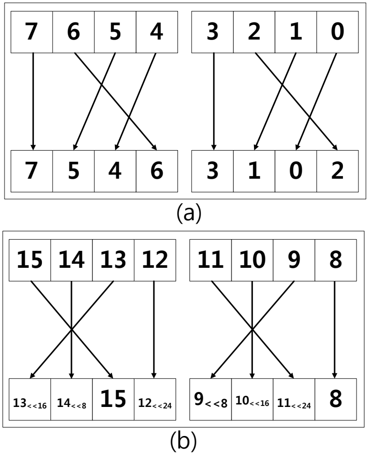


Figure: AVX2: (a) word permutation, (b) byte and word permutation

LSH with AVX512

- Analysis on Intel-AVX512
  - 8 64-bit vector wise data set
  - Vector-wise addition(_mm512_add_epi64)
  - Vector-wise exclusive-or(_mm512_xor_si512)
  - Vector-wise rotation (_mm512_rolv_epi64)
  - Vector-wise permutation (_mm512_permutexvar_ps)

8-bit AVR

- Compared with LSH-256 and LSH-512 reference results over AVR, performance are enhanced by 40 % and 64 %.

Table: Comparison results of LSH implementations on AVR

Method	500 Byte	100 Byte	50 Byte
Proposed LSH-256	**788**	**1031**	**2061**
Reference LSH-256	1319	1695	3388
SHA2-256	532	668	672
SHA3-256	1432	1794	3560
SKEIN-256	4787	7982	10647
BLAKE-256	562	708	714
GROESTL-256	685	1012	1220
JH-256	5062	7855	10490
Proposed LSH-512	**830**	**2132**	**4263**
Reference LSH-512	2351	5930	11855

16-bit MSP

- Compared with LSH-256 and LSH-512 reference results over MSP, performance are enhanced by 11 % and 33 %.

Table: Comparison results of LSH implementations on MSP

Method	500 Byte	100 Byte	50 Byte
Proposed LSH-256	**597**	**781**	**1561**
Reference LSH-256	675	878	1755
Proposed LSH-512	**652**	**1672**	**3343**
Reference LSH-512	976	2480	4960

32-bit ARM

- Compared with LSH-256 and LSH-512 reference results over ARM, performance are enhanced by 36 % and 60 %.

Table: Comparison results of LSH implementations on ARM

Method	Long Message	4096 Byte	64 Byte
Proposed LSH-256	**63.5**	**65.4**	**148.0**
Reference LSH-256	99.8	103.9	214.3
SHA2-256	22.9	23.8	80.7
SHA3-256	48.9	51.1	188.1
SKEIN-256	31.4	32.5	98.7
BLAKE-256	30.6	31.6	96.3
GROESTL-256	74.9	77.5	237.2
JH-256	165.7	168.7	363.2
Proposed LSH-512	**75.7**	**82.0**	**323.7**
Reference LSH-512	194.1	204.3	772.1
SHA2-512	47.7	49.8	133.6
SHA3-512	58.7	61.4	180.0
SKEIN-512	31.4	32.5	99.4
BLAKE-512	71.2	74.1	179.5
GROESTL-512	107.4	112.9	353.6
JH-512	165.7	168.8	362.7

128-bit NEON

- Compared with LSH-256 and LSH-512 reference results over ARM-NEON, performance are enhanced by 76 % and 88 %.

Table: Comparison results of LSH implementations on ARM-NEON

Method	Long Message	4096 Byte	64 Byte
Proposed LSH-256	**23.5**	**24.9**	**59.3**
Reference LSH-256	99.8	103.9	214.3
SHA2-256	22.9	23.7	74.3
SHA3-256	45.3	46.7	143.3
SKEIN-256	20.9	21.7	68.1
BLAKE-256	21.3	22.1	72.9
GROESTL-256	73.9	73.5	308.0
JH-256	146.0	148.7	315.6
Proposed LSH-512	**22.9**	**24.6**	**117.9**
Reference LSH-512	194.1	204.3	772.1
SHA2-512	26.6	27.9	82.2
SHA3-512	54.6	56.5	133.0
SKEIN-512	20.9	21.7	68.8
BLAKE-512	21.5	22.9	80.5
GROESTL-512	73.9	73.5	300.1
JH-512	146.2	148.9	315.4

64-bit Intel

- Compared with LSH-256 and LSH-512 reference results over Intel, performance are enhanced by 15 % and 31 %.

Table: Comparison results of LSH implementations on Intel

Method	Long Message	4096 Byte	64 Byte
Proposed LSH-256	**12.6**	**14.7**	**30.3**
Reference LSH-256	14.9	15.5	35.5
Proposed LSH-512	**5.8**	**7.4**	**31.1**
Reference LSH-512	8.5	9.1	37.5

256-bit Intel-AVX2

- Compared with LSH-256 and LSH-512 reference results over Intel-AVX2, performance are enhanced by 69 % and 65 %.

Table: Comparison results of LSH implementations on Intel-AVX2.

Method	Long Message	4096 Byte	64 Byte
Proposed LSH-256	**4.5**	**4.7**	**10.6**
Reference LSH-256	14.9	15.5	35.5
SHA2-256	12.0	12.2	63.1
SHA3-256	6.5	8.1	45.4
SKEIN-256	5.2	5.2	24.0
BLAKE-256	6.8	7.6	16.1
GROESTL-256	9.4	9.7	54.9
JH-256	14.9	14.5	64.8
Proposed LSH-512	**2.9**	**3.3**	**18.3**
Reference LSH-512	8.5	9.1	37.5
SHA2-512	8.1	8.1	16.9
SHA3-512	8.6	10.0	38.3
SKEIN-512	5.6	5.7	13.4
BLAKE-512	6.5	6.6	14.4
GROESTL-512	14.0	14.9	79.5
JH-512	14.4	14.6	43.5

Future Works

Thumb/Thumb2: Light-weight ARM implementation
ARMv8: 64-bit architecture implementation
NEON2: 32 128-bit wise register architecture
AVX512: 512-bit wise word instruction
CUDA, OpenCL: GPU based acceleration

Contributions

Novel optimization methods on various platforms
Advanced implementation results on processors

Discussion & Conclusion

Thank you for your attention.