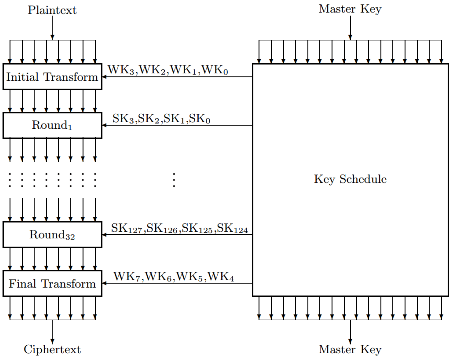
Light-weight Block Cipher: HIGHT is Back!

Cryptography Competition'16

2016/10/14

Target Lightweight Block Cipher

- **Hight block cipher**
  - Suggested by Hong et al. [CHES'06]
  - 64-bit block (8-bit word) and 128-bit key size
  - Addition, Rotation, eXclusive-or (ARX) operation



FELICS Triathlon

- **Light-weight block cipher competition: FELICS**
  - Benchmarking framework by Luxembourg University
  - Evaluation metrics: execution time, RAM, ROM
  - Two Korean block ciphers won in the competition

Table: First and second winners of FELICS (block size/key size)

Rank	First Triathlon	Second Triathlon
1	LEA (128/128)	HIGHT (64/128)
2	SPECK (64/96)	Chaskey (128/128)
3	Chaskey (128/128)	SPECK (64/128)

Target Platforms

- **Low-end 8-bit AVR processor**
  - Low frequency (8 MHz)
  - Low capacity (128KB EEPROM, 4KB RAM)
  - 8-bit wise arithmetic logic unit

Table: Instruction set summary for AVR

Mnemonics	Operands	Description	Cycles
ADD	Rd, Rr	Add without Carry	1
EOR	Rd, Rr	Exclusive OR	1
LSL	Rd	Logical Shift Left	1
ROL	Rd	Rotate Left Through Carry	1

Target Platforms

- **High-end 32-bit ARM-NEON processor**
  - 128-bit SIMD architecture for ARMv7 Cortex-A series
  - Support a length of 64-bit (`D`) and 128-bit (`Q`)
  - Vector computations in 8-, 16-, 32-, 64-bit wise

Table: Instruction set summary for NEON

Mnemonics	Operands	Description	Cycles
VADD	Qd,Qn,Qm	Vector Addition	1
VEOR	Qd,Qn,Qm	Vector Exclusive-or	1
VSHL	Qd,Qm,#imm	Vector Left Shift	1
VSRI	Qd,Qm,#imm	Vector Right Shift with Insert	2

Contributions

- **Contributions:** Light-weight HIGHT implementation on IoT devices
  - Compact rotation operations for auxiliary functions
  - Execution time, RAM and ROM optimized results
  - Vectorized LUT based for auxiliary functions

ARX Operations in Instruction Sets

Table: 8-bit instructions over 8-bit AVR, where R0 and R1 represent destination and source registers

Addition	Exclusive-or	Left rotation	
`ADD R0, R1`	`EOR R0, R1`	`LSL R0`	`ADC R0`

Table: 8-bit vectorized instructions over 32-bit ARM-NEON, where q1 and q0 represent destination and source registers

Addition	Exclusive-or	Left rotation
`vadd.i8 q1, q1, q0`	`veor q1, q1, q0`	`vshl.i8 q1, q0, #1`
		`vsri.8 q1, q0, #7`

Optimization of AVR

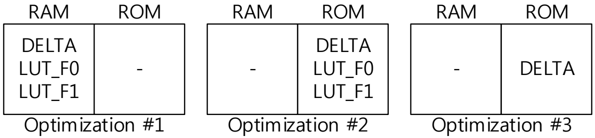


Figure: Three different storage utilizations for optimized HIGHT

- **Constant delta generation:** LFSR over $(x^7 + x^3 + 1)$
- **Two auxiliary functions:** F_0 and F_1
  - LUT based computations
  - Optimized rotation operations:
    $F_0 = x^{\lll 1} \oplus x^{\lll 2} \oplus x^{\lll 7} \rightarrow x^{\lll 1} \oplus x^{\lll 2} \oplus swap(x^{\lll 3})$;
    $F_1 = x^{\lll 3} \oplus x^{\lll 4} \oplus x^{\lll 6} \rightarrow x^{\lll 3} \oplus swap(x) \oplus swap(x^{\lll 2})$

Optimization of NEON

- **Data alignments with transpose operations**
  - Construction of byte-sliced format
  - Three times of transpose operations

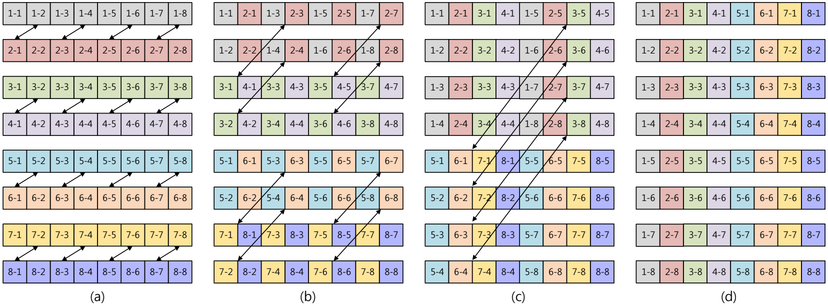


Figure: (a) 8-bit wise, (b) 16-bit wise, (c) 32-bit wise, (d) completed

Optimization of NEON

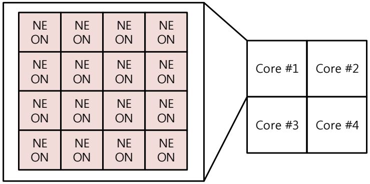
- **Vectorized LUT based auxiliary function**
  - NEON supports 4-bit wise vectorized LUT access
  - For 8-bit LUT, two 4-bit LUT accesses are performed
  - e.g. `F0(X) = F0_LOW(X mod` 2^4`)` $\oplus$ `F0_HIGH(X div` 2^4`)`

Table: Pre-computed look-up tables for auxiliary functions in 4-bit wise

Auxiliary function	Look-up table values
F0_LOW	0xA523A82E_BF39B234_91179C1A_8B0D8600
F0_HIGH	0x5A328AE2_FB932B43_1971C9A1_B8D06800
F1_LOW	0x4B13FBA3_2A729AC2_89D13961_E8B05800
F1_HIGH	0xB431BF3A_A227A92C_981D9316_8E0B8500

Thread Level Optimization

- **Parallel encryptions in multiple cores**
  - 16 encryptions are performed in each core
  - Total 64 encryptions are performed in parallel way



```
#pragma omp parallel private(id) shared(PLAINTEXT)
    #pragma omp for
        for(id=0;id<TOTAL;id++)
            Encryption(ROUNDKEY,PLAINTEXT[16×id]);
```

Performance Evaluation: Scaled Performance

- Speed: `(#add+#and+#xor+#rotation+#round_key`$\times$`2)`$\times$`#round`
- Size: `(#add+#and+#xor+#rotation+#round_key)`$\times$`2`

Table: Comparison of scaled overhead for block ciphers (64-bit block, 128-bit key), †: excluding initial and final rounds,

Features	HIGHT	SPECK	SIMON
Scaled in 8-bit add	4	4	–
Scaled in 8-bit and	–	–	4
Scaled in 8-bit xor	12	8	12
Scaled in 8-bit rotation	16	12	8
Round key in each round (byte)	4	4	4
Number of Rounds	32†	27	44
Speed (cost/penalty)	1,280/–	864/1.5	1,408/0.9
Size/round (cost/penalty)	72/–	56/1.3	56/1.3

Performance Evaluation: 8-bit AVR Processor

Table: Separated encryption results in execution time (in cycle/byte) and code size (in byte) on AVR platform; †: scaled results

Method	ENC	ENC†	ROM	ROM†	RAM
HIGHT w/ RAM LUT	216	**216**	254	**254**	640
HIGHT w/ ROM LUT	232	**232**	766	**766**	136
HIGHT w/o LUT	320	**320**	248	**248**	136
HIGHT [7]	346	**346**	286	**286**	136
SPECK [7]	125	**188**	542	**705**	100
SPECK [3]	122	**183**	628	**816**	108
SIMON [7]	224	**202**	354	**460**	176
SIMON [3]	221	**199**	436	**567**	176

Performance Evaluation: 8-bit AVR Processor

Table: On-the-fly encryption/decryption results in execution time (in cycle/byte) and code size (in byte) on AVR platform

Method	ENC	DEC	ROM	RAM
HIGHT w/ RAM LUT	325	334	672	640
HIGHT w/ ROM LUT	357	366	1,312	0
HIGHT w/o LUT	452	461	912	0
HIGHT [9]	371	371	5,672	0
HIGHT [8]	2,438	2,520	402	32

Performance Evaluation: 32-bit ARM–NEON Processor

Table: Separated encryption results in execution time (in cycle/byte) on ARM-NEON platform; †: scaled results

Method	ENC	ENC†
HIGHT w/ LUT	18.7	**18.7**
HIGHT w/o LUT	25.9	**25.9**
SPECK [15]	16.5	**24.8**
SIMON [15]	31.9	**28.7**

Table: Performance evaluation of multiple-thread encryptions

No. thread	1	2	3	4	5	6	7	8
cycle/byte	18.7	11.5	7.7	6.0	7.9	7.1	7.1	6.5

Contributions

Compact rotation operations for auxiliary functions
Execution time, RAM and ROM optimized results
Vectorized LUT based auxiliary functions

Future Works

High-speed implementations of mode of operations

Thank you for your attention