Bitsliced Implementations of the PRINCE, LED and RECTANGLE Block Ciphers on AVR 8-bit Microcontrollers

Bao Zhenzhen   Luo Peng   Lin Dongdai

State Key Laboratory of Information Security, Institute of Information Engineering, Chinese Academy of Sciences

December 8, 2015

The 17th International Conference on Information and Communications Security, ICICS 2015

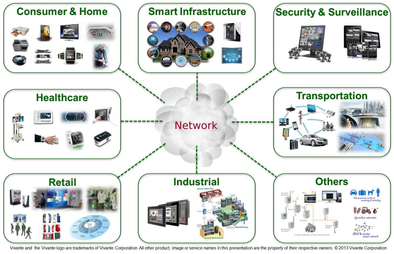
Outline

Outline

Lightweight Ciphers for the Internet of Things

Demand for low-cost cryptosystems

- Hundred billions of constrained devices are interconnected.
- Cryptographic techniques are required to provide security for devices with low-end micro controllers.
- Current standard security solution can not be implemented on all the connected objects.


Consumer & Home
Smart Infrastructure
Security & Surveillance
Healthcare
Network
Transportation
Retail
Industrial
Others
Vivante and the Vivante logo are trademarks of Vivante Corporation. All other product, image or service names in this presentation are the property of their respective owners. © 2013 Vivante Corporation

Lightweight cryptographic primitives, can relax implementation requirements, provide efficient protection, support different functionality in HW and in SW.

Lightweight Block Ciphers for the Internet of Things

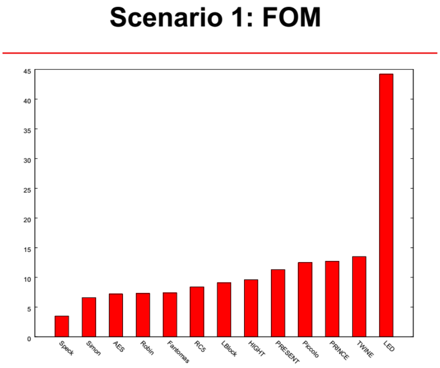
Efforts made by cryptographic community

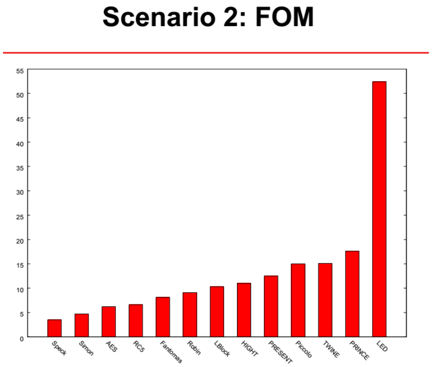
- A dozen of lightweight block ciphers spring up, excellent for some different implementation features.
- PRESENT, CLEFIA, LBlock, LED, PRINCE, SIMON and SPECK, RECTANGLE...
  `http://www.cryptolux.org/index.php/Lightweight_Block_Ciphers`.
- Consequences are the work and projects to improve and evaluated the performance of those lightweight block ciphers.

Projects Improved and Evaluated the Performance of Block Ciphers

- ECRYPT II project: Compact implementations and performance evaluation of 12 low-cost block ciphers on AVR
- BLOC project: Implementations of 21 low-cost (5 classical and 16 lightweight) block ciphers on the WSN430 sensor
- FELICS: Triathlon of Lightweight Block Ciphers for the Internet of Things, 2 usage scenarios & 3 target devices & 3 performance metrics

Triathlon of Lightweight Block Ciphers for the Internet of Things
Results





Our Motivation

- Contribute to the performance benchmarks of bad performance ciphers: PRINCE, LED.
- Contribute to the performance benchmarks of newly proposed cipher: RECTANGLE.

Usage scenarios & Target devices & Performance metrics of This Work

- Usage scenarios
  - **Scenario 1 - Encryption of 128 bytes of data using CBC mode**
  - **Scenario 2 - Encryption of 128 bits of data using CTR mode**
- Target devices
  - **8-bit AVR**
  - 16-bit MSP430
  - 32-bit ARM CortexM
- Performance metrics
  - **Execution time**
  - **RAM footprint**
  - **Code size**

Work to Improve the Performance of PRINCE

- T-table implementation: combines different operations within a round into a single table lookup operation
- Block-parallel implementation: stores two nibbles in one register and processes them in parallel wherever possible
- Nibble-slicing implementation: processes two blocks in parallel, two nibbles in the same position of two blocks are stored in one register

All of them are nibble-oriented implementations based on large LUTs

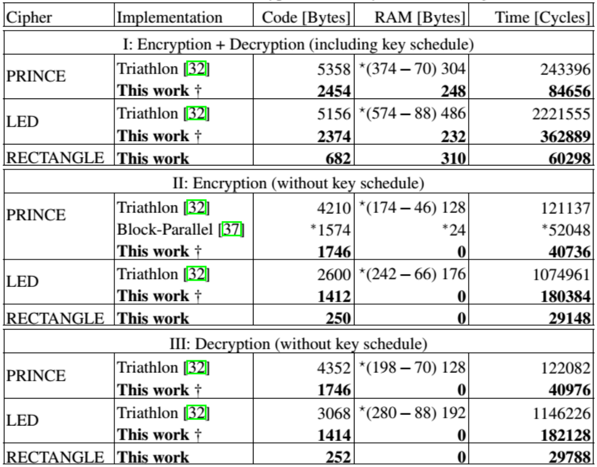
Outline

Main Techniques

- Bitslice technique instead of LUTs: minimize memory & keep high throughput
- Rearrange the state bits: each block are processed in fine granularity parallel
- Minimizing the number of instructions needed by each operation

Main Results

Table 1: Results for Scenario 1 (encryption of 128 bytes of data using CBC mode)



Cipher	Implementation	Code [Bytes]	RAM [Bytes]	Time [Cycles]
I: Encryption + Decryption (including key schedule)				
PRINCE	Triathlon [32]	5358	*(374 – 70) 304	243396
	This work †	**2454**	**248**	**84656**
LED	Triathlon [32]	5156	*(574 – 88) 486	2221555
	This work †	**2374**	**232**	**362889**
RECTANGLE	**This work**	**682**	**310**	**60298**
II: Encryption (without key schedule)				
PRINCE	Triathlon [32]	4210	*(174 – 46) 128	121137
	Block-Parallel [37]	*1574	*24	*52048
	This work †	**1746**	**0**	**40736**
LED	Triathlon [32]	2600	*(242 – 66) 176	1074961
	This work †	**1412**	**0**	**180384**
RECTANGLE	**This work**	**250**	**0**	**29148**
III: Decryption (without key schedule)				
PRINCE	Triathlon [32]	4352	*(198 – 70) 128	122082
	This work †	**1746**	**0**	**40976**
LED	Triathlon [32]	3068	*(280 – 88) 192	1146226
	This work †	**1414**	**0**	**182128**
RECTANGLE	**This work**	**252**	**0**	**29788**

Main Results

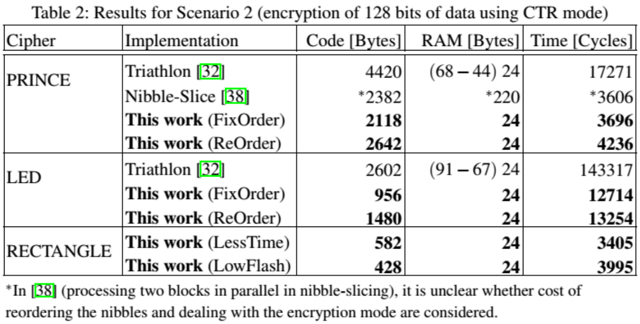


Table 2: Results for Scenario 2 (encryption of 128 bits of data using CTR mode)

Cipher	Implementation	Code [Bytes]	RAM [Bytes]	Time [Cycles]
PRINCE	Triathlon [32]	4420	(68 – 44) 24	17271
	Nibble-Slice [38]	*2382	*220	*3606
	This work (FixOrder)	**2118**	**24**	**3696**
	This work (ReOrder)	**2642**	**24**	**4236**
LED	Triathlon [32]	2602	(91 – 67) 24	143317
	This work (FixOrder)	**956**	**24**	**12714**
	This work (ReOrder)	**1480**	**24**	**13254**
RECTANGLE	**This work** (LessTime)	**582**	**24**	**3405**
	This work (LowFlash)	**428**	**24**	**3995**

*In [38] (processing two blocks in parallel in nibble-slicing), it is unclear whether cost of reordering the nibbles and dealing with the encryption mode are considered.

Outline

The PRINCE cipher

- Operates on 64-bit blocks, uses a 128-bit key $k = (k_0||k_1)$
- FX-construction & PRINCE$_{core}$
- α-reflection property: decryption can reuse the exact same procedure of encryption
- AES-like round function, operates on a 4×4 state matrix of nibbles: KeyXor and RCXor, S-box, MixColumns (involution) and ShiftRows

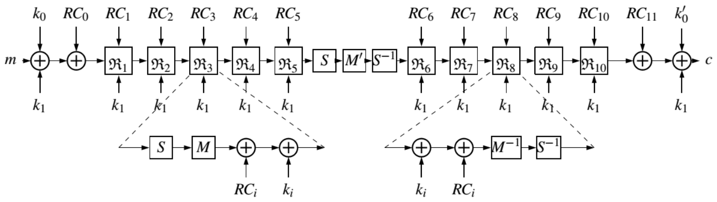
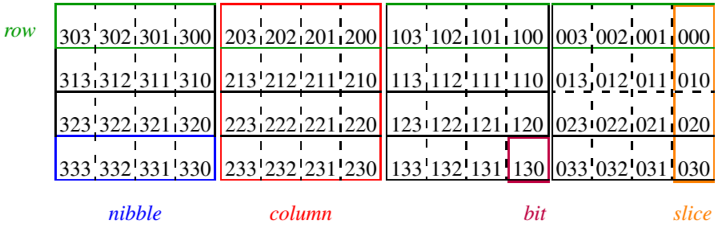


Fig. 1: The whole encryption process of PRINCE

State Bits Rearrangement

- Successive four bits from right to left is called a *nibble*
- Successive four nibbles from right to left is a *row*
- Successive four bits from top to bottom is a *slice*
- Successive four nibbles from top to bottom is a *column*



303	302	301	300	203	202	201	200	103	102	101	100	003	002	001	000
313	312	311	310	213	212	211	210	113	112	111	110	013	012	011	010
323	322	321	320	223	222	221	220	123	122	121	120	023	022	021	020
333	332	331	330	233	232	231	230	133	132	131	130	033	032	031	030

Fig. 2: The original arrangement of the state of bits for PRINCE

State Bits Rearrangement

Rearrange the state: gather the bits with index $*yz$:

- Bits with same row index and same slice index are gathered together
- Call the resulted bit set a ***lane***
- Bits in a *lane* will be settled in the same register in our implementations

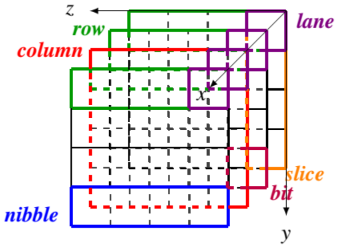


Fig. 3 (a) Rearrangement of the state of bits for PRINCE

State Bits Rearrangement

Let $\mathbf{S}$ denotes the state, then $\mathbf{S}[*, y, z]$ denotes a particular lane:

- Scenario 1: two lanes of one state $\mathbf{S}[*, y, z]$ and $\mathbf{S}[*, y+2, z]$ ($y \in \{0, 1\}$, $z \in \{0, \ldots, 3\}$) are stored in one register
- Scenario 2: two lanes of two states $\mathbf{S}[*, y, z]$ and $\mathbf{S}'[*, y, z]$ ($y \in \{0, \ldots, 3\}$, $z \in \{0, \ldots, 3\}$) are stored in one register

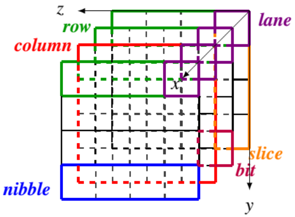


Fig. 3 (a) Rearrangement of the state of bits for PRINCE

- The rearrangement of the state takes 2 clocks per bit using ROL and ROR.
- Rearrange the key and the round constant state in the same way
- The KeyXor and RCXor operations can be merged together

State Bits Rearrangement

Let $\mathbf{S}$ denotes the state, then $\mathbf{S}[*, y, z]$ denotes a particular lane:

- Scenario 1: two lanes of one state $\mathbf{S}[*, y, z]$ and $\mathbf{S}[*, y+2, z]$ ($y \in \{0, 1\}$, $z \in \{0, \dots, 3\}$) are stored in one register
- Scenario 2: two lanes of two states $\mathbf{S}[*, y, z]$ and $\mathbf{S}'[*, y, z]$ ($y \in \{0, \dots, 3\}$, $z \in \{0, \dots, 3\}$) are stored in one register

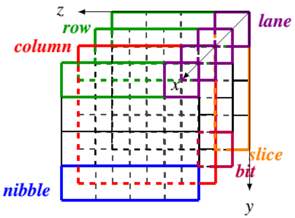


Fig. 3 (a) Rearrangement of the state of bits for PRINCE

- The rearrangement of the state takes 2 clocks per bit using ROL and ROR.
- Rearrange the key and the round constant state in the same way
- The KeyXor and RCXor operations can be merged together

Bitsliced Implementation of the S-box and the Inverse S-box

- Implement the S-box and inverse S-box using logical operations
- Compute 8 S-boxes in parallel using a logical instruction sequence

- Find the bitsliced implementation of the 4×4 S-box (resp. inverse S-box) using an automatic search tool: `http://www.gladman.me.uk/`.
- Translate the primary instruction sequences into AVR instruction sequences
- Minimize the required clock cycles and realize in place process

- Taking advantage of MOVW, and processing 16 S-boxes together, the S-layer (inverse S-layer) of PRINCE needs $17 \times 2 + 4 = 38$ (resp. $16 \times 2 + 6 = 38$) instructions.

Bitsliced Implementation of the MixColumns

MixColumns of PRINCE can be seen as being composed of three compositions:

- mirror on the rows
- addition of a parity bit
- slice-wise rotations

It can be expressed as the parallel application of 16 independent slice-wise transformations.

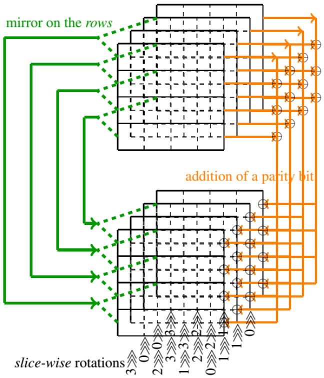


Fig. 3 (c) MixColumns of PRINCE

Bitsliced Implementation of the MixColumns

- Addition of parity bits: parity bits of 8 slices are computed in 8-way parallel
- Mirror on the rows ∘ slice-wise rotations: bits exchanging among different lanes are computed in 4-way and 8-way parallel

- Scenario 1: $28 + 34 = 62$ instructions per state
- Scenario 2: $16 + 32 = 48$ instructions per state.

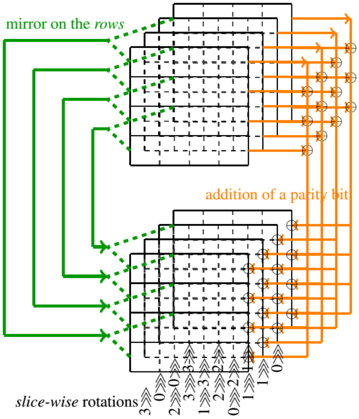


Fig. 3 (c) MixColumns of PRINCE

Bitsliced Implementation of the ShiftRows and the Inverse ShiftRows

- ShiftRows (and Inverse) correspond to *rotate bits within lanes*
- Rotate the high half and the lower half of 8-bit registers separately
- Implemented using AND, LSL, LSR, BLD, BST instructions
- Scenario 1: $4 \times 19 = 76$ instructions per state.
- Scenario 2: $4 \times 19 + 2 = 78$ instructions per 2 state.

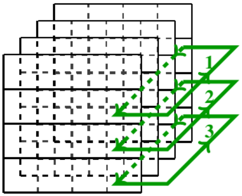


Fig. 3 (b) ShiftRows of PRINCE

Outline

The LED cipher

- 64-bit substitution-permutation network (SPN) block cipher
- uses a key size from 64 to 128 bits
- the master key k is composed of two 64-bit subparts, $k = k_1||k_2$
- the 4-round operation between two key addition is called a *step*

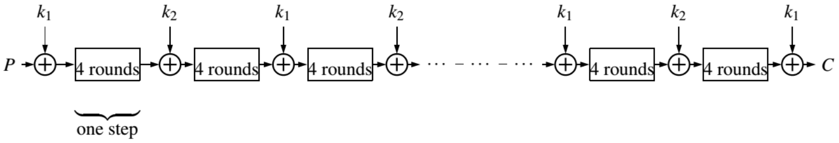


Fig. 4: The whole encryption process of LED

The LED cipher

- 64-bit substitution-permutation network (SPN) block cipher
- uses a key size from 64 to 128 bits (**LED-128**)
- the master key k is composed of two 64-bit subparts, $k = k_1||k_2$
- the 4-round operation between two key addition is called a *step*

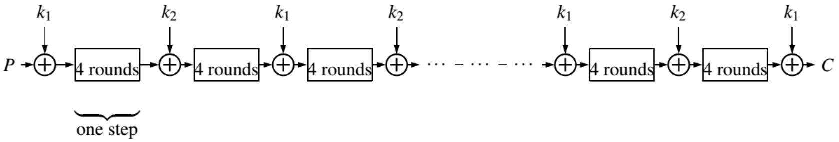


Fig. 4: The whole encryption process of LED

The LED cipher

- AES-like round function, operates on a 4×4 state matrix of nibbles: AddConstants, SubNibbles, ShiftRows and MixColumnsSerial

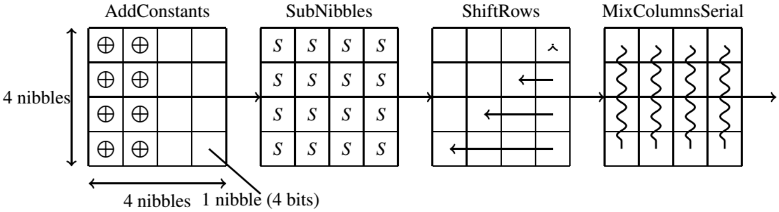


Fig. 5: An overview of a single round of LED

State Bits Rearrangement

Let **S** denotes the state, then $\mathbf{S}[*, y, z]$ denotes a particular lane:

- For Scenario 1, one lane of one state $\mathbf{S}[*, y, z]$ ($y \in \{0, 1, 2, 3\}$, $z \in \{0, 1, 2, 3\}$) is stored in one 8-bit register, *only the high half holds meaningful bits*
- For Scenario 2, two lanes of two states $\mathbf{S}[*, y, z]$ and $\mathbf{S}'[*, y, z]$ ($y \in \{0, 1, 2, 3\}$, $z \in \{0, 1, 2, 3\}$) are stored in one register

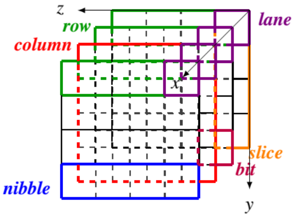
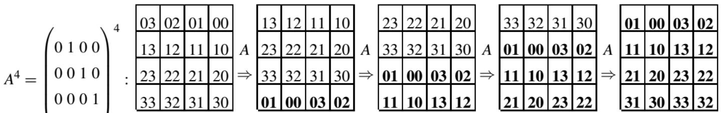


Fig. 3 (a) Rearrangement of the state of bits for PRINCE

- The rearrangement of the state takes 2 clocks per bit using ROL and ROR.
- Rearrange the subkey states in the same way

Bitsliced Implementation of the MixColumnsSerial

- Each column of the LED state is transformed by multiplying it with an MDS matrix M
- $M = A^4$: four serial applications of A, which can be implemented using XOR and bit-permutation



$$A^4 = \begin{pmatrix} 0&1&0&0 \\ 0&0&1&0 \\ 0&0&0&1 \\ 4&1&2&2 \end{pmatrix}^4 :$$

$01 = 01 \oplus 13 \oplus 22 \oplus 32$

$00 = 00 \oplus 03 \oplus 12 \oplus 21 \oplus 31$

$03 = 03 \oplus 02 \oplus 11 \oplus 20 \oplus 23 \oplus 30 \oplus 33$

$02 = 02 \oplus 10 \oplus 23 \oplus 33$

These instructions here is a sequential execution of A to update the bits to get the new nibble in the new state. Similar instructions to update the other three nibbles are omitted here.

Fig. 6: MixColumnsSerial of LED operate on one column

The four updated bits in each nibble switched from the order (3,2,1,0) to the order (1,0,3,2)

Bitsliced Implementation of the MixColumnsSerial

- Switching operation corresponds to an exchanging operation between registers
- When combined with the S-box operation ⇒ no need to exchange registers
- The four columns of the LED state go through the same MixColumnsSerial operation ⇒ MixColumnsSerial operations on columns are done in parallel

- Scenario 1: 4-way parallel implementation ⇒ 64 instructions per state
- Scenario 2: 8-way parallel implementation ⇒ 64 instructions for two states

Bitsliced Implementation of the S-box and the Inverse S-box

LED uses the PRESENT Sbox (16 Sboxes can be implemented using 32 instructions):

- Bit-permutation equivalence of LED Sbox: combine the switching operation in MixColumnsSerial with the S-box and inverse S-box operation
- Taking advantage of MOVW, and processing 16 S-boxes together, the switching SubNibbles (inverse switching SubNibbles) of LED needs 33 (resp. 34) instructions.

Bitsliced Implementation of the ShiftRows and the Inverse ShiftRows

- ShiftRows (Inverse) correspond to *rotate bits within lanes*
- rotate the high half and the lower half of 8-bit registers separately
- implemented by AND, LSL, LSR, BLD, BST instructions

- Scenario 1: $4 \times 12 = 48$ instructions per state.
- Scenario 2: $4 \times 19 = 76$ instructions per 2 state.

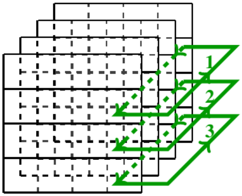


Fig. 3 (b) ShiftRows of PRINCE

Outline

The RECTANGLE cipher

- 64-bit 25-round Substitution-Permutation Network (SPN) block cipher
- Master key can be 80 or 128 bits
- Two-dimensional cipher state

$$\begin{bmatrix} a_{0,15} \cdots a_{0,2} & a_{0,1} & a_{0,0} \\ a_{1,15} \cdots a_{1,2} & a_{1,1} & a_{1,0} \\ a_{2,15} \cdots a_{2,2} & a_{2,1} & a_{2,0} \\ a_{3,15} \cdots a_{3,2} & a_{3,1} & a_{3,0} \end{bmatrix}$$

Fig. 7 (a) Arrangement of the cipher state for RECTANGLE

The RECTANGLE cipher

- 64-bit 25-round Substitution-Permutation Network (SPN) block cipher
- Master key can be 80 or 128 bits **RECTANGLE-128**
- Two-dimensional cipher state

$$\begin{bmatrix} a_{0,15} \cdots a_{0,2} & a_{0,1} & a_{0,0} \\ a_{1,15} \cdots a_{1,2} & a_{1,1} & a_{1,0} \\ a_{2,15} \cdots a_{2,2} & a_{2,1} & a_{2,0} \\ a_{3,15} \cdots a_{3,2} & a_{3,1} & a_{3,0} \end{bmatrix}$$

Fig. 7 (a) Arrangement of the cipher state for RECTANGLE

The RECTANGLE cipher

Three steps in the round transformation

- AddRoundKey: a simple XOR of the round keys
- SubColumn: the substitution layer - 16 4×4 S-boxes in parallel
- ShiftRows: the permutation layer - 3 rotations

$$\begin{tikzcd}
\begin{pmatrix} a_{0,15} \\ a_{1,15} \\ a_{2,15} \\ a_{3,15} \end{pmatrix} \arrow[d, "S"] & \cdots & \begin{pmatrix} a_{0,2} \\ a_{1,2} \\ a_{2,2} \\ a_{3,2} \end{pmatrix} \arrow[d, "S"] & \begin{pmatrix} a_{0,1} \\ a_{1,1} \\ a_{2,1} \\ a_{3,1} \end{pmatrix} \arrow[d, "S"] & \begin{pmatrix} a_{0,0} \\ a_{1,0} \\ a_{2,0} \\ a_{3,0} \end{pmatrix} \arrow[d, "S"] \\
\begin{pmatrix} b_{0,15} \\ b_{1,15} \\ b_{2,15} \\ b_{3,15} \end{pmatrix} & \cdots & \begin{pmatrix} b_{0,2} \\ b_{1,2} \\ b_{2,2} \\ b_{3,2} \end{pmatrix} & \begin{pmatrix} b_{0,1} \\ b_{1,1} \\ b_{2,1} \\ b_{3,1} \end{pmatrix} & \begin{pmatrix} b_{0,0} \\ b_{1,0} \\ b_{2,0} \\ b_{3,0} \end{pmatrix}
\end{tikzcd}$$

SubColumns of RECTANGLE

$$\begin{aligned}
(a_{0,15} \dots a_{0,2}\ a_{0,1}\ a_{0,0}) &\xrightarrow{\lll 0} (a_{0,15} \dots a_{0,2}\ a_{0,1}\ a_{0,0}) \\
(a_{1,15} \dots a_{1,2}\ a_{1,1}\ a_{1,0}) &\xrightarrow{\lll 1} (a_{1,14} \dots a_{1,1}\ a_{1,0}\ a_{1,15}) \\
(a_{2,15} \dots a_{2,2}\ a_{2,1}\ a_{2,0}) &\xrightarrow{\lll 12} (a_{2,3}\ \dots a_{2,6}\ a_{2,5}\ a_{2,4}) \\
(a_{3,15} \dots a_{3,2}\ a_{3,1}\ a_{3,0}) &\xrightarrow{\lll 13} (a_{3,2}\ \dots a_{3,5}\ a_{3,4}\ a_{3,3})
\end{aligned}$$

ShiftRows of RECTANGLE

State Bits Arrangement and Bitsliced Implementation of the S-box and the Inverse S-box

Bitsliced Implementation of the S-box and the Inverse S-box:

- Find the optimal general bitsliced implementations of the S-box and the inverse S-box (12 terms)
- Get the best translation to instruction sequences in AVR instruction set
- $26 = 12 \times 2 + 2$ (resp. $27 = 12 \times 2 + 3$) instructions

Bitsliced Implementation of the ShiftRows

- Implementing the 12-bit and 13-bit rotation of the 16-bit row with a minimized number of instructions.
  - 4-bit rotation (both left and right) of 16-bit row
  - using SWAP, MOVW, ANDI, EOR and 2 temporary registers
  - 7 instructions to perform 12-bit rotation of the 16-bit row
- ShiftRows and the inverse ShiftRows need 20 instructions per state.

Bitsliced Implementation of the Key Schedule and Adding Round Key

- There are two key bytes shared between every two successive round keys
- Use $8 + 25 \times 6$ bytes instead of 26×8 bytes to store 26 round keys
- Use 6 load instructions instead of 8 during AddRoundKey

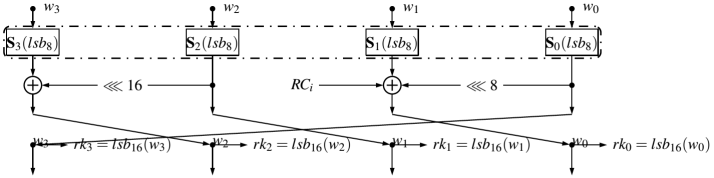


Fig. 8: One round key schedule of RECTANGLE-128, where lsb_n means taking the n least significant bits

Outline

Results Summary

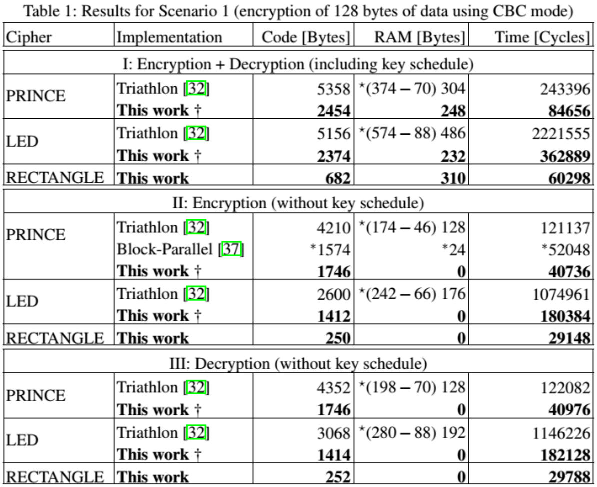


Table 1: Results for Scenario 1 (encryption of 128 bytes of data using CBC mode)

Cipher	Implementation	Code [Bytes]	RAM [Bytes]	Time [Cycles]
I: Encryption + Decryption (including key schedule)				
PRINCE	Triathlon [32]	5358	*(374 – 70) 304	243396
	This work †	**2454**	**248**	**84656**
LED	Triathlon [32]	5156	*(574 – 88) 486	2221555
	This work †	**2374**	**232**	**362889**
RECTANGLE	**This work**	**682**	**310**	**60298**
II: Encryption (without key schedule)				
PRINCE	Triathlon [32]	4210	*(174 – 46) 128	121137
	Block-Parallel [37]	*1574	*24	*52048
	This work †	**1746**	**0**	**40736**
LED	Triathlon [32]	2600	*(242 – 66) 176	1074961
	This work †	**1412**	**0**	**180384**
RECTANGLE	**This work**	**250**	**0**	**29148**
III: Decryption (without key schedule)				
PRINCE	Triathlon [32]	4352	*(198 – 70) 128	122082
	This work †	**1746**	**0**	**40976**
LED	Triathlon [32]	3068	*(280 – 88) 192	1146226
	This work †	**1414**	**0**	**182128**
RECTANGLE	**This work**	**252**	**0**	**29788**

Results Summary

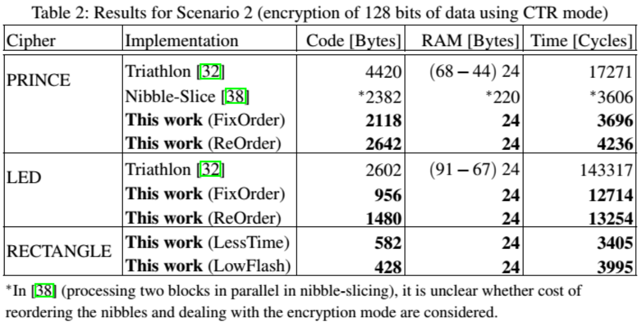


Table 2: Results for Scenario 2 (encryption of 128 bits of data using CTR mode)

Cipher	Implementation	Code [Bytes]	RAM [Bytes]	Time [Cycles]
PRINCE	Triathlon [32]	4420	(68 – 44) 24	17271
	Nibble-Slice [38]	*2382	*220	*3606
	This work (FixOrder)	**2118**	**24**	**3696**
	This work (ReOrder)	**2642**	**24**	**4236**
LED	Triathlon [32]	2602	(91 – 67) 24	143317
	This work (FixOrder)	**956**	**24**	**12714**
	This work (ReOrder)	**1480**	**24**	**13254**
RECTANGLE	**This work** (LessTime)	**582**	**24**	**3405**
	This work (LowFlash)	**428**	**24**	**3995**

*In [38] (processing two blocks in parallel in nibble-slicing), it is unclear whether cost of reordering the nibbles and dealing with the encryption mode are considered.

Results Summary

If use our results to update the results in [34], we get the following Table 3:

Table 3: Updated results for ciphers performance in Scenario 1 and Scenario 2

Scenario 1 (encryption of 128 bytes of data using CBC mode)					Scenario 2 (encryption of 128 bits of data using CTR mode)				
Cipher	Code [Bytes]	RAM [Bytes]	Time [Cycles]	p_i	Cipher	Code [Bytes]	RAM [Bytes]	Time [Cycles]	p_i
Speck†	560	280	44264	3.21	Speck†	294	24	2563	3.00
Simon†	566	320	64884	3.86	Simon†	364	24	4181	3.87
RECTANGLE†	682	310	60298	3.92	RECTANGLE†	428	24	3995	4.01
PRINCE†	2454	248	84656	7.36	AES*	1246	(81 – 49) 32	3408	6.90
AES*	3010	(408 – 70) 338	58246	8.15	LED†	956	24	12714	9.21
PRESENT*	2160	(448 – 46) 402	245232	11.13	PRINCE†	2118	24	3696	9.65
LED†	2374	232	362889	13.44	PRESENT*	1294	(56 – 32) 24	16849	11.97

$p_i = \sum_{m\in M}(w_m \times \frac{v_{i,m}}{\min_i(v_{i,m})})$, where M ={the code, the RAM, the cycles}, $w_m = 1$ [32].

*Results for assembly implementations in [34]. † Results for assembly implementations by this work.

Future Work

- Optimization work on other ciphers and on other devices.
- A more fair comparison which concern parallelization.

For Further Reading I

Bao, Z., Luo, P., Lin, D.:
Bitsliced Implementations of the PRINCE, LED and RECTANGLE Block Ciphers on AVR 8-bit Microcontrollers.
http://eprint.iacr.org/2015/1118.

Bao, Z., Zhang, W., Luo, P., Lin, D.:
Bitsliced Implementations of Block Ciphers on AVR 8-bit Microcontrollers.
http://github.com/FreeDisciplina/BlockCiphersOnAVR,
October 2015.

For Further Reading II

Dinu, D., Corre, Y. L., Khovratovich, D., Perrin, L., Großschädl, J., Biryukov, A.:
Triathlon of Lightweight Block Ciphers for the Internet of Things, http://eprint.iacr.org/2015/209.

CryptoLUX.:
FELICS Triathlon. http://www.cryptolux.org/index.php/FELICS_Triathlon, http://www.cryptolux.org/index.php/FELICS_Block_Ciphers_Detailed_Results, 1 October 2015.