All the HIGHT You Need on Cortex-M4

Hwajeong Seo[1], Zhe Liu[2]

[1]Hansung University,
[2]Nanjing University of Aeronautics and Astronautics

HSU 한성대학교 HANSUNG UNIVERSITY

CryptoCraft LAB
https://crypto.modoo.at

Background

- Data encryption is a fundamental building block for **secure communication**

- The cryptography implementation needs to satisfy **two requirements**

  - **High-speed computation** → high availability of application
  - **Secure computation** → strong security against side channel attack

Background (Side Channel Attack)

- **Side Channel Attack**:
  any attacks based on **information gained from the physical implementation** of cryptosystem

  - **Passive Attack**:
    monitoring timing information, power consumption, electromagnetic leaks or sound

  - **Active Attack**:
    fault injection on cryptosystem, manipulation of the instruction opcodes

Background (Fault Attack Model)

- **Computation Faults**: errors in the data that is processed by a program
  - **Random word**: targeting **a specific word** in a program to a random value
  - **Random byte**: targeting **a single byte** of word in a program to a random value
  - **Random bit**: targeting **a single bit** of word in a program to a random value
  - **Chosen bit pair**: targeting **a chosen bit pair of word** in a program to a random value
- **Instruction Faults**: changing the program flow
  - **Instruction skip fault model**: replacing the opcode of original instruction with a **no-operation**

Motivation

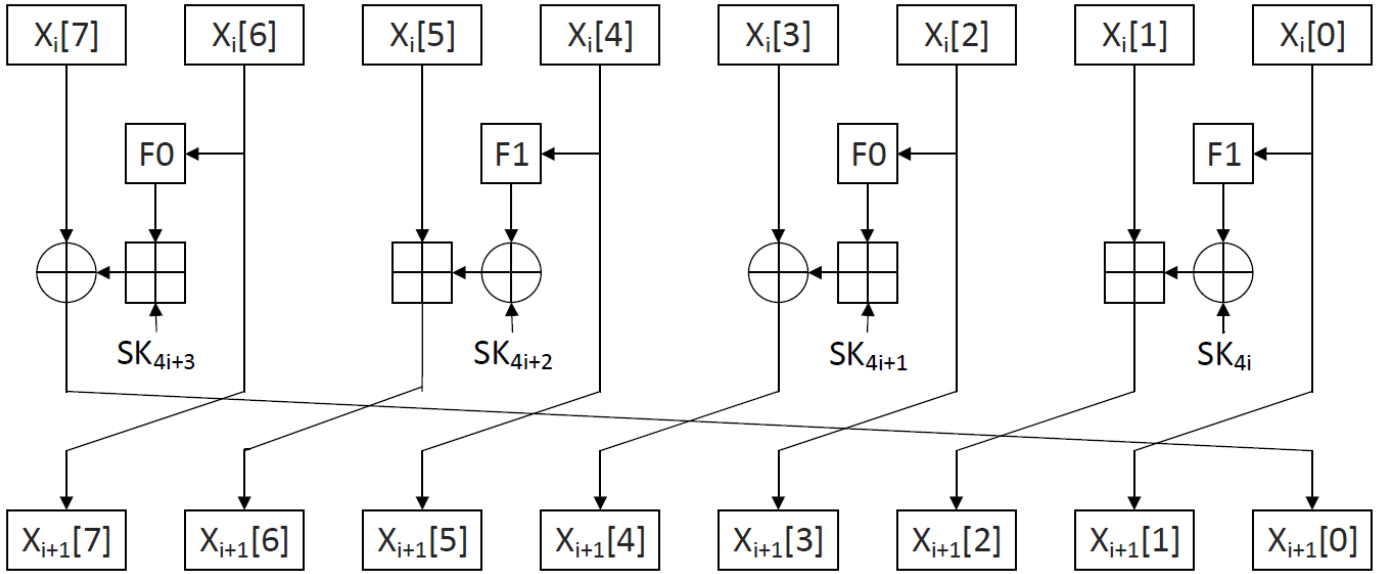
- **Lightweight fault attack resistance** is important for secure implementation
- In SAC'16, **software based secure implementation** was presented
- However, it is only practical with **bit-slicing implementation**
- In WISA'17, **high-end SIMD based implementation** was presented
- However, it is not available in **low-end devices**
- In this paper, we present **first lightweight fault attack resistance with low-end SIMD architecture**

Background (ARM Cortex-M4)

- **Target low-end architecture**
  - ARM Cortex-M4
  - 32-bit vectorized insruction
    - 4 X 8, 2 X 16 (e.g. UADD8: add 4 bytes without carry)
  - Application
    - IoT devices (Airpod, Galaxy fit)

Background (HIGHT Block Cipher)

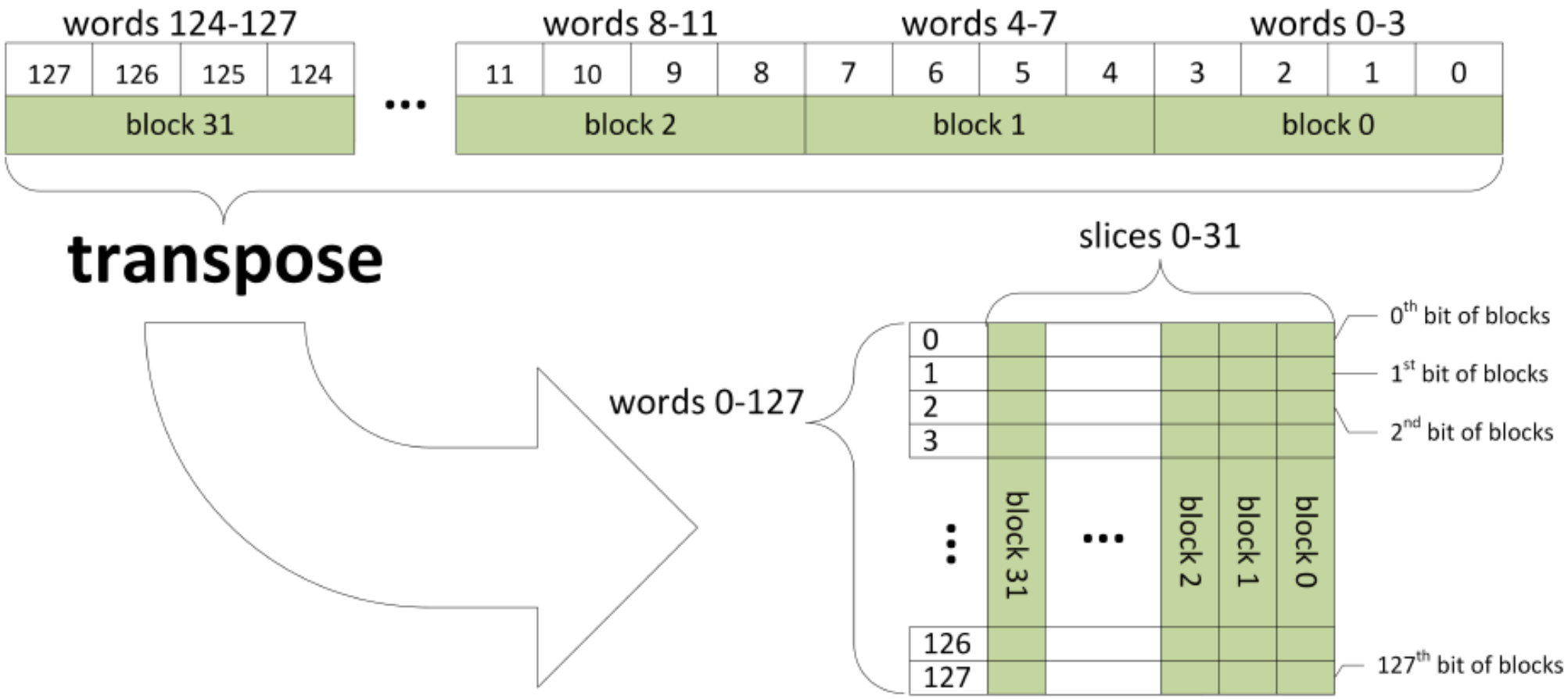
- **HIGHT (CHES'06)**
  - ISO/IEC 18033-3 international block cipher
  - 64-bit block / 8-bit word
  - 128-bit key size
  - 32 round
  - Addition/Rotation/eXclusive-or (ARX) architecture



$$F0(X) = X^{<<<1} \oplus X^{<<<2} \oplus X^{<<<7}$$

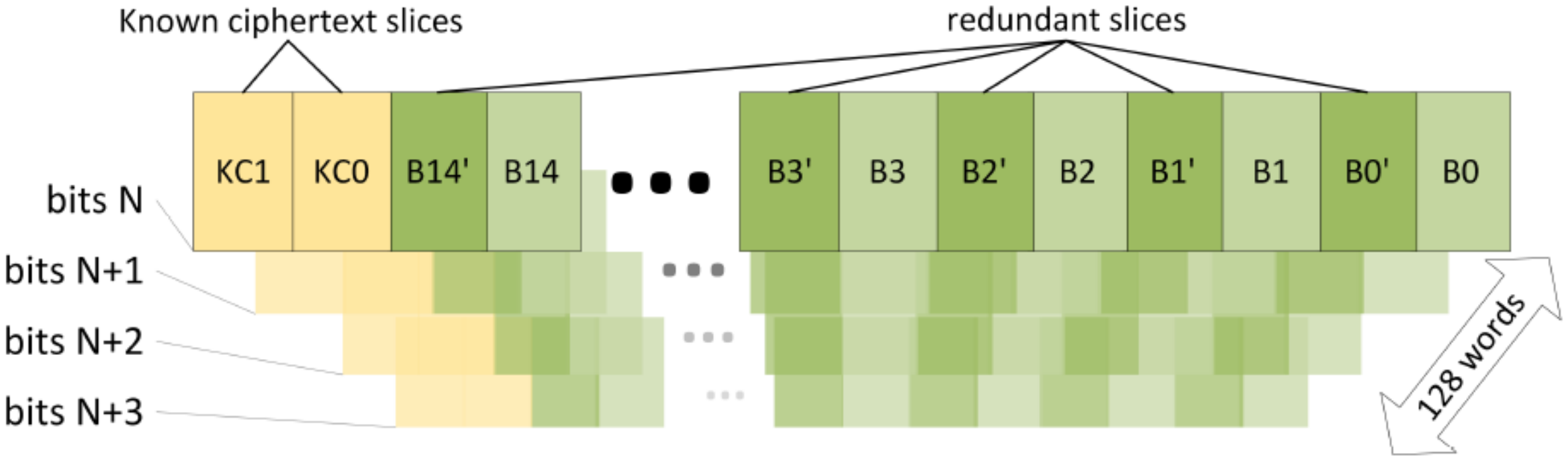
$$F1(X) = X^{<<<3} \oplus X^{<<<4} \oplus X^{<<<6}$$

Previous Works (Fault Attack Resistance in SW: SAC'16)

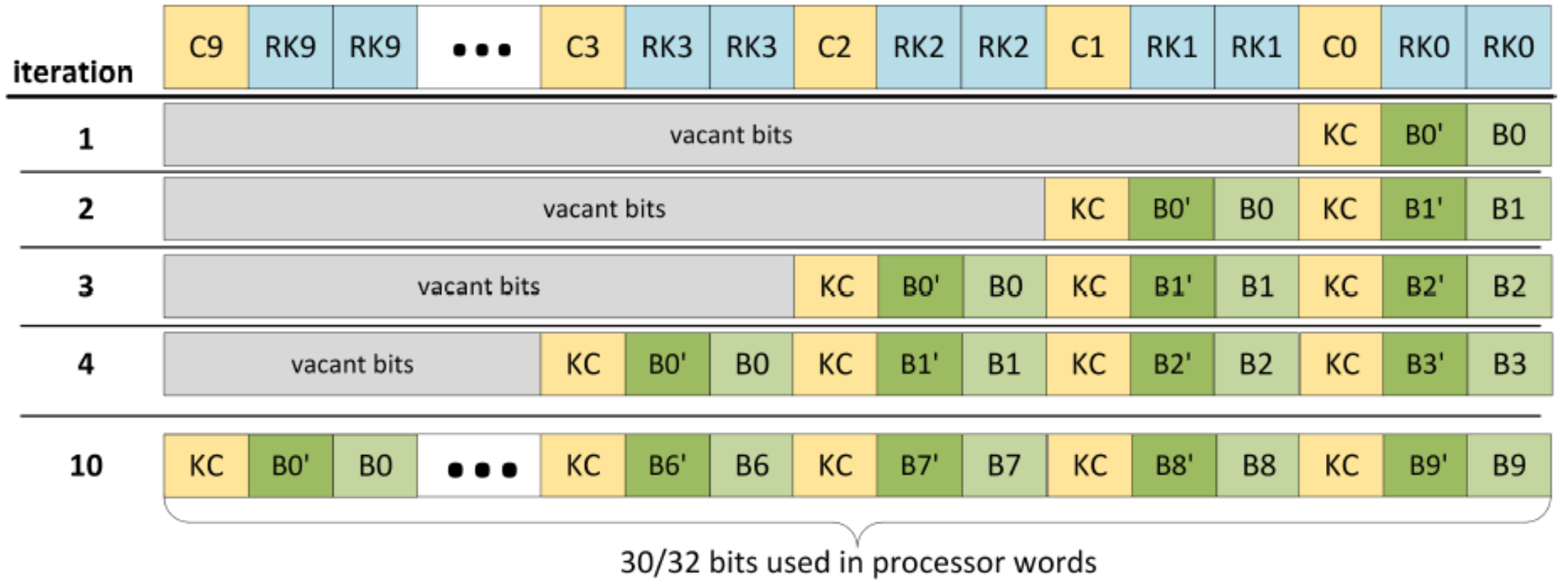


Transpose of 32 blocks (128-bit wise) to 128 32-bit words

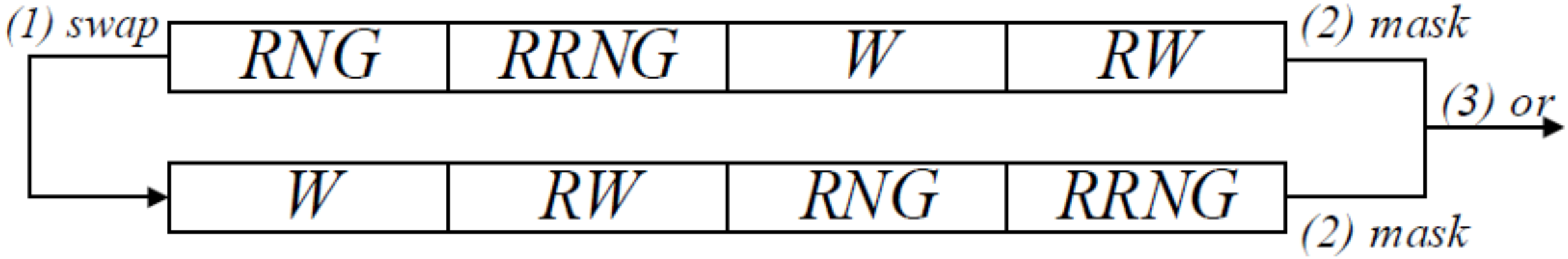
Previous Works (Fault Attack Resistance in SW: SAC'16)



Previous Works (Fault Attack Resistance in SW: SAC'16)



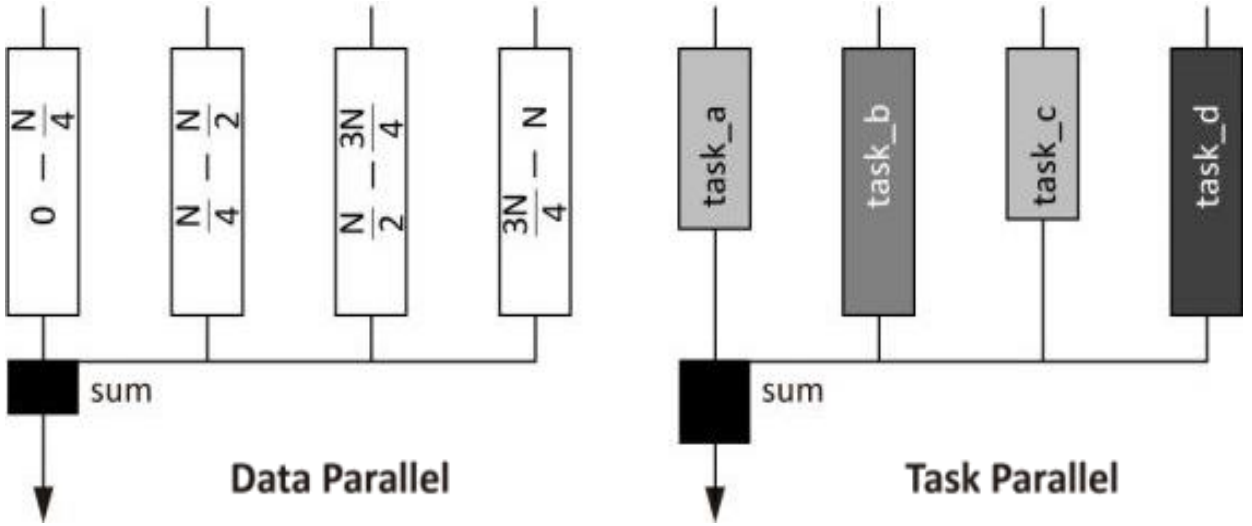
Previous Works (Random Shuffling: WISA'17)



- **Random shuffling**
  - Generating random number and shuffling the word

- **Notation**
  - **RNG** :random number generator
  - **RRNG** :redundant random number generator
  - **W** :plaintext in word
  - **RW** :redundant plaintext in word

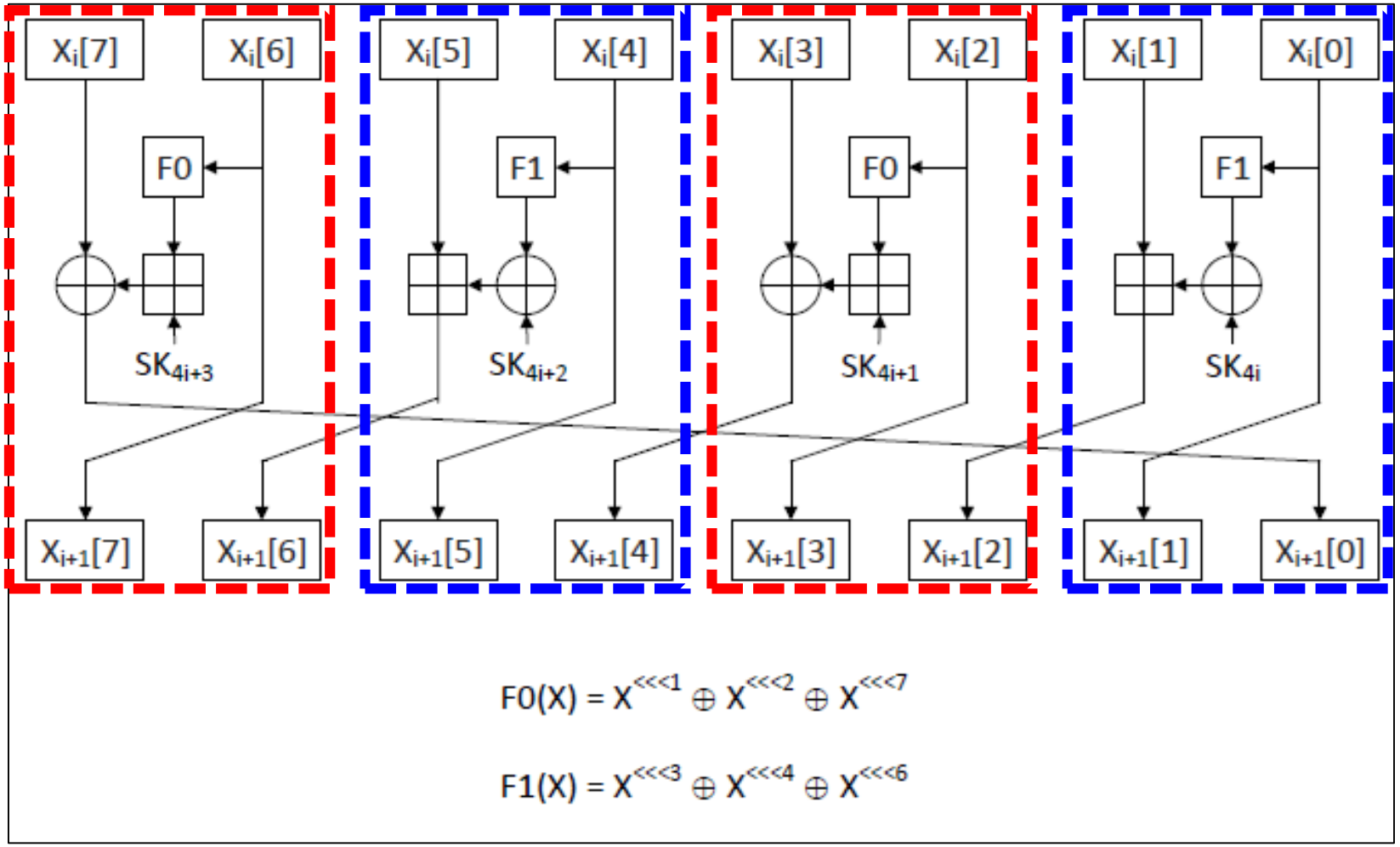
Background (Parallel Computation)



- **Proposed HIGHT implementation**
  - both data and task parallelism
  - light random shuffling

Task-parallel Implementation

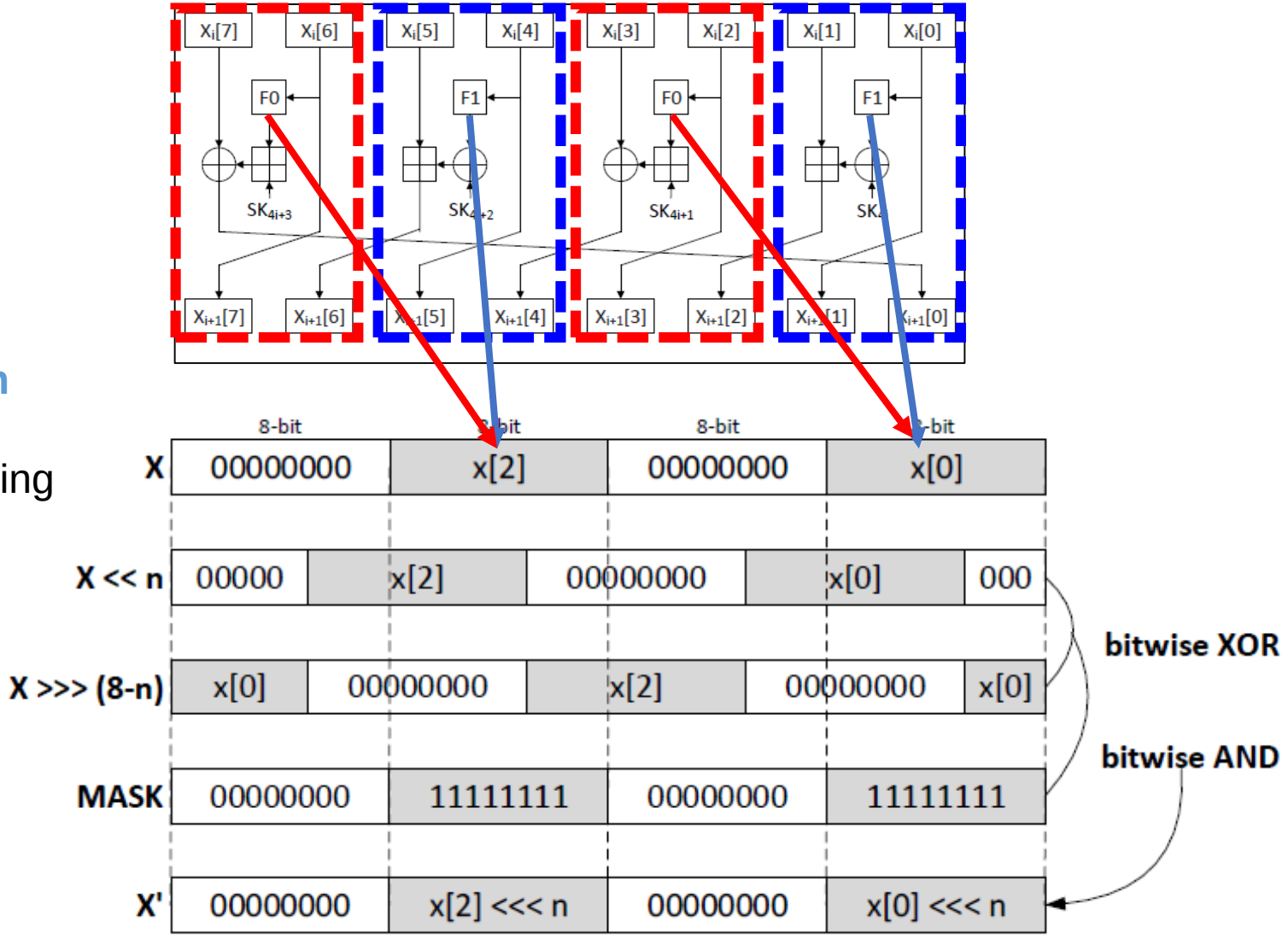
- Red and blue blocks can be computed in task parallel way



Previous works (Pseudo SIMD: TEC'18)

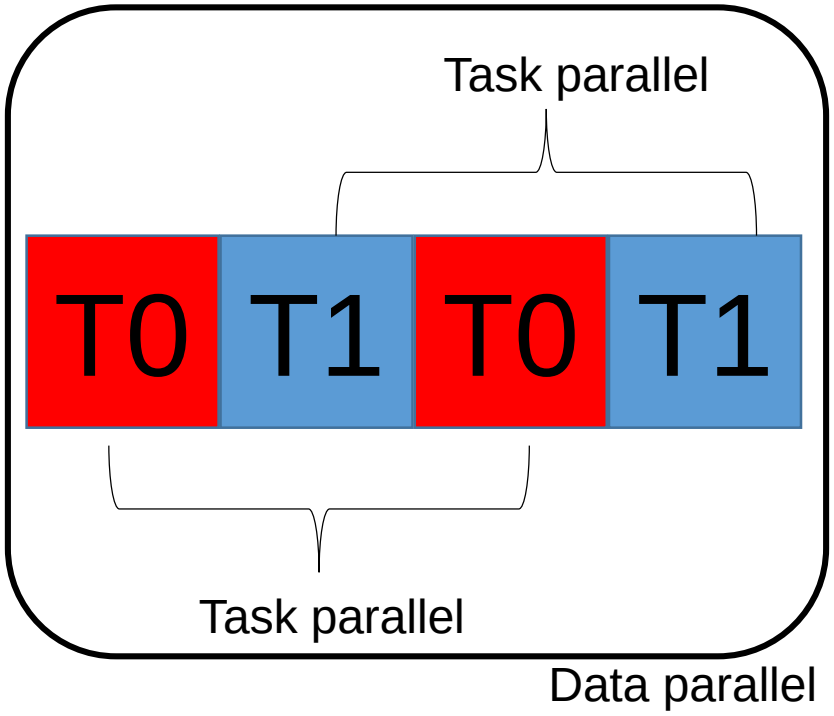
Cortex-M does not provide **SIMD-rotation**

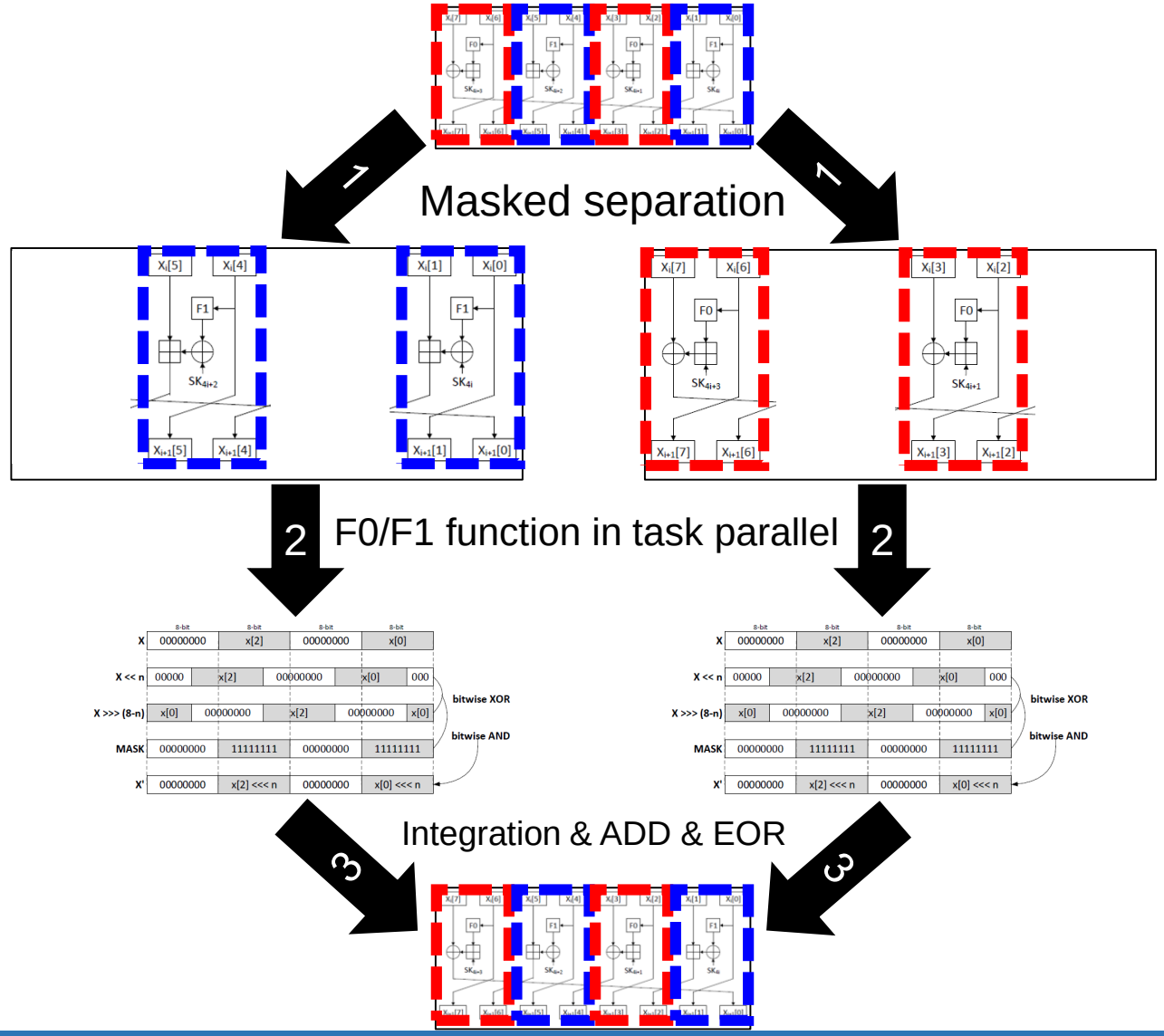
Pseudo technique is available with padding



Proposed Method: Task/Data-parallel Implementation

Operation	SIMD
Rotation	Pseudo SIMD
Addition	UADD8
eXclusive-or	EOR





Proposed Method: Task/Data-parallel Implementation (F0 function)

- **Masked separation**

Algorithm 1 F0 function for data parallel implementation.

Input: R2, temporal variables (R10 and R11)
Output: R6

```
1: AND R11,R2, #0x00FF00FF
2: LSL R10, R11,#1
3: EOR R10, R10, R11, LSR #7
4: EOR R10, R10, R11, LSL #2
5: EOR R10, R10, R11, LSR #6
6: EOR R10, R10, R11, LSL #7
7: EOR R10, R10, R11, LSR #1
8: AND R6, R10, #0x00FF00FF
9: AND R11,R2, #0xFF00FF00
10: LSL R10, R11, #1
11: EOR R10, R10, R11, LSR #7
12: EOR R10, R10, R11, LSL #2
13: EOR R10, R10, R11, LSR #6
14: EOR R10, R10, R11, LSL #7
15: EOR R10, R10, R11, LSR #1
16: AND R10, R10, #0xFF00FF00

17: ORR R6, R6, R10
```

Proposed Method: Task/Data-parallel Implementation (F0 function)

- **F0 function in task parallel**

Algorithm 1 F0 function for data parallel implementation.

Input: R2, temporal variables (R10 and R11)
Output: R6

```
 1: AND R11,R2, #0x00FF00FF
 2: LSL R10, R11,#1
 3: EOR R10, R10, R11, LSR #7
 4: EOR R10, R10, R11, LSL #2
 5: EOR R10, R10, R11, LSR #6
 6: EOR R10, R10, R11, LSL #7
 7: EOR R10, R10, R11, LSR #1
 8: AND R6, R10, #0x00FF00FF
 9: AND R11,R2, #0xFF00FF00
10: LSL R10, R11, #1
11: EOR R10, R10, R11, LSR #7
12: EOR R10, R10, R11, LSL #2
13: EOR R10, R10, R11, LSR #6
14: EOR R10, R10, R11, LSL #7
15: EOR R10, R10, R11, LSR #1
16: AND R10, R10, #0xFF00FF00
17: ORR R6, R6, R10
```

barrel shifter

barrel shifter

Proposed Method: Task/Data-parallel Implementation (F0 function)

- **Integration**

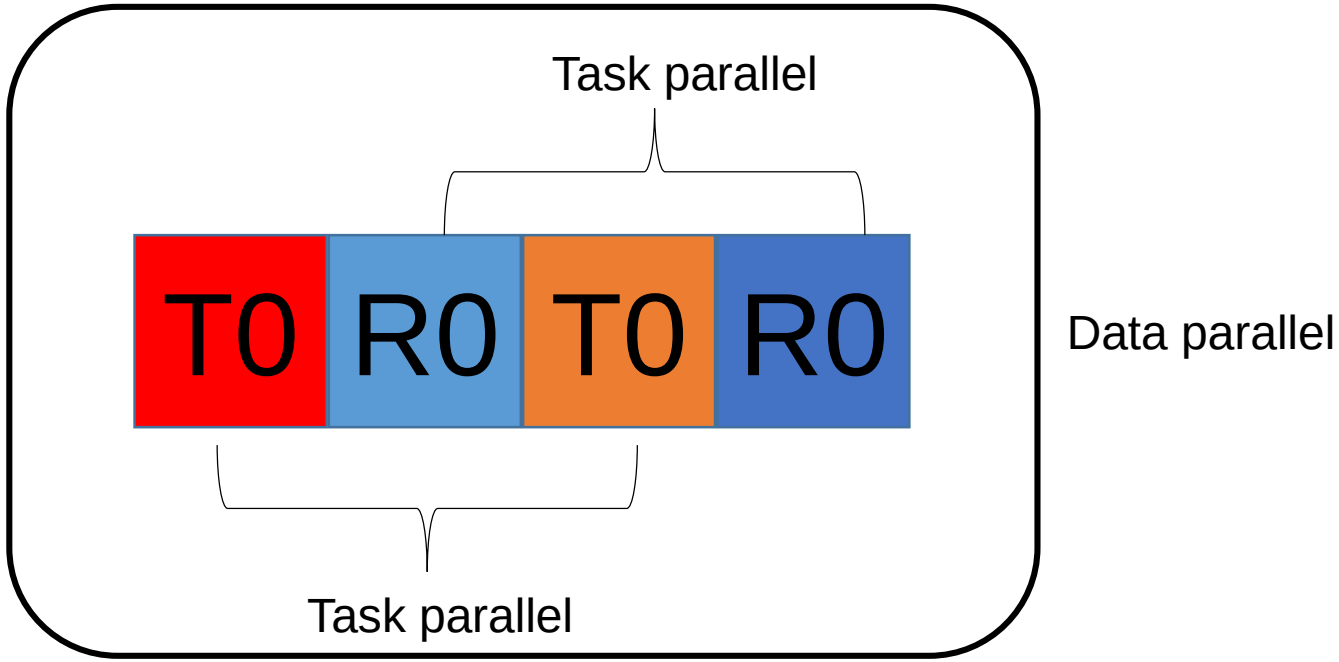
Algorithm 1 F0 function for data parallel implementation.

Input: R2, temporal variables (R10 and R11)
Output: R6

```
 1: AND R11,R2, #0x00FF00FF
 2: LSL R10, R11,#1
 3: EOR R10, R10, R11, LSR #7
 4: EOR R10, R10, R11, LSL #2
 5: EOR R10, R10, R11, LSR #6
 6: EOR R10, R10, R11, LSL #7
 7: EOR R10, R10, R11, LSR #1
 8: AND R6, R10, #0x00FF00FF
 9: AND R11,R2, #0xFF00FF00
10: LSL R10, R11, #1
11: EOR R10, R10, R11, LSR #7
12: EOR R10, R10, R11, LSL #2
13: EOR R10, R10, R11, LSR #6
14: EOR R10, R10, R11, LSL #7
15: EOR R10, R10, R11, LSR #1
16: AND R10, R10, #0xFF00FF00
17: ORR R6, R6, R10
```

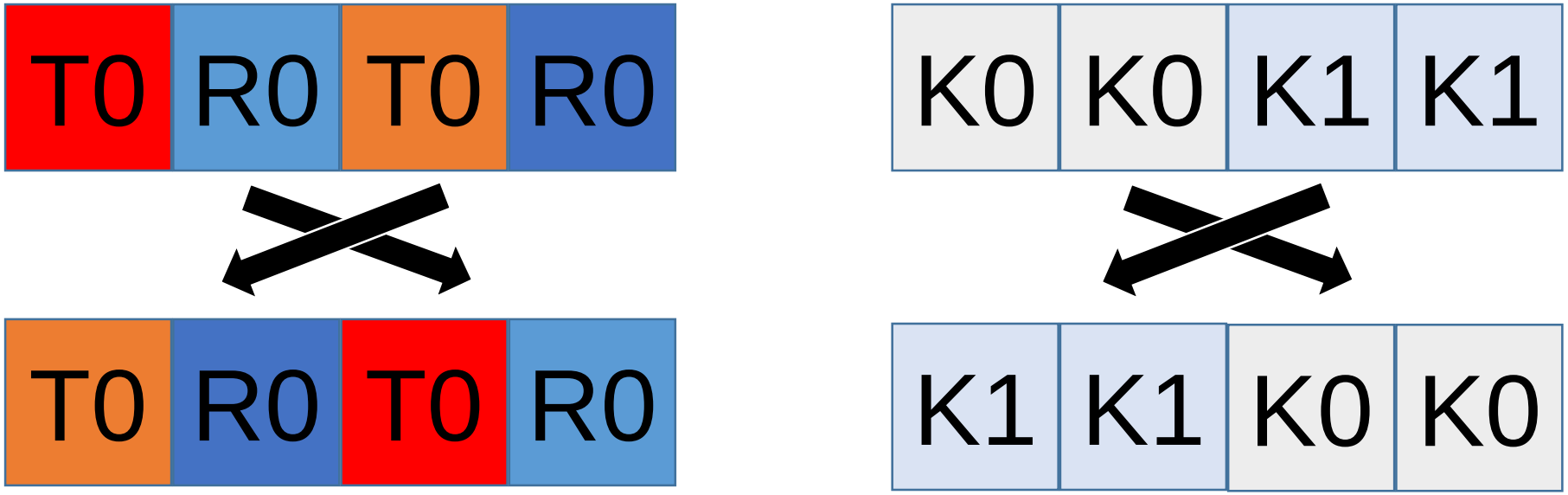
Proposed Method: Fault Attack Resistant Implementation

- **Intra-redundant implementation**
  - T0: plaintext
  - R0: copy of T0



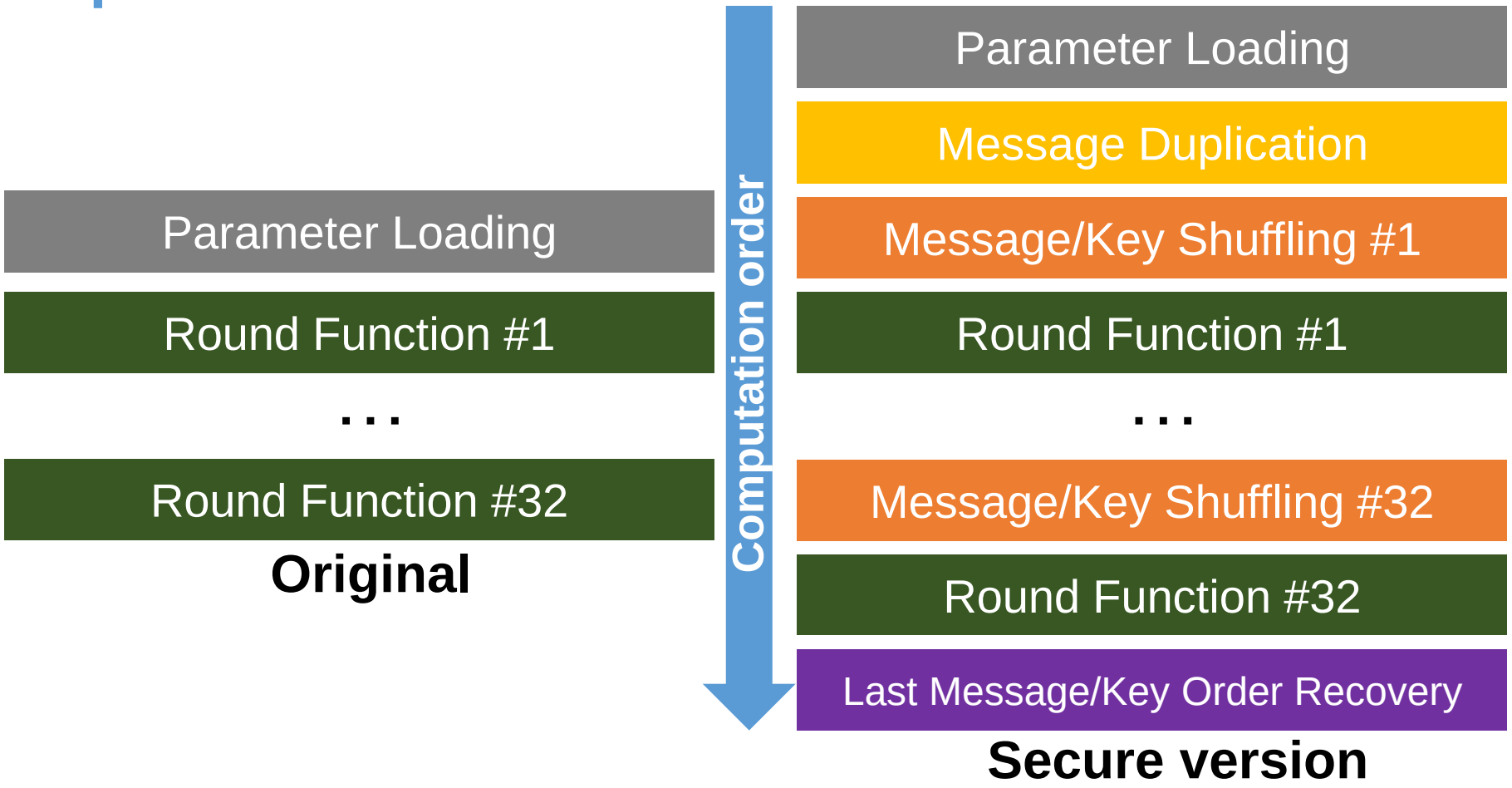
Proposed Method: Fault Attack Resistant Implementation

- **Message shuffling**
  - Plaintext and round key are **shuffed together.**
  - In last round, the order is **recovered.**



Proposed Method: Fault Attack Resistant Implementation

- **Overall process**



Evaluation (Taget Platform)

- **32-bit Cortex-M4@72MHz**
  - Teensy 3.2 development board
  - 256KB flash memory
  - 64KB RAM
  - 2KB EEPROM

Evaluation (Code Size)

- M4 provides more **optimized code size** with SIMD instruction

Method	Code size (bytes)				RAM (bytes)			Execution time (cycles per byte)		
	EKS	ENC	DEC	SUM	EKS	ENC	DEC	EKS	ENC	DEC
HIGHT 64/128										
Task (M3) [TEC'18]	154	352	352	**858**	316	180	180	33	197	234
This work										
Task (M4)	116	348	332	**796**	316	180	180	18	76	71
T/D (M4)	160	592	544	**1,296**	316	188	188	49	56	55
Fault (M4)	160	536	520	**1,216**	316	188	188	49	143	143

Evaluation (RAM)

- RAM consumption is **similar**

Method	Code size (bytes)				RAM (bytes)			Execution time (cycles per byte)		
	EKS	ENC	DEC	SUM	EKS	ENC	DEC	EKS	ENC	DEC
HIGHT 64/128										
Task (M3) [TEC'18]	154	352	352	858	316	180	**180**	33	197	234
This work										
Task (M4)	116	348	332	796	316	180	**180**	18	76	71
T/D (M4)	160	592	544	1,296	316	188	**188**	49	56	55
Fault (M4)	160	536	520	1,216	316	188	**188**	49	143	143

Evaluation (Execution Time)

- Proposed **Task/Data parallel implementation** requires **long timing for key scheduling**
- However, **encryption and decryption** show much **better performance**

Method	Code size (bytes)				RAM (bytes)			Execution time (cycles per byte)		
	EKS	ENC	DEC	SUM	EKS	ENC	DEC	EKS	ENC	DEC
HIGHT 64/128										
Task (M3) [TEC'18]	154	352	352	858	316	180	180	**33**	**197**	**234**
This work										
Task (M4)	116	348	332	796	316	180	180	**18**	**76**	**71**
T/D (M4)	160	592	544	1,296	316	188	188	**49**	**56**	**55**
Fault (M4)	160	536	520	1,216	316	188	188	**49**	**143**	**143**

Evaluation (Fault Resistant)

- Fault implementation shows **low performance than T/D implementation**.
- We provide security on **fault attack only except instruction skip**

Method	Code size (bytes)				RAM (bytes)			Execution time (cycles per byte)		
	EKS	ENC	DEC	SUM	EKS	ENC	DEC	EKS	ENC	DEC
HIGHT 64/128										
Task (M3) [TEC'18]	154	352	352	858	316	180	180	33	197	234
This work										
Task (M4)	116	348	332	796	316	180	180	18	76	71
T/D (M4)	160	592	544	1,296	316	188	188	**49**	**56**	**55**
Fault (M4)	160	536	520	1,216	316	188	188	**49**	**143**	**143**

RAND	RW	RB	Rb	CbP	IS
O	O	O	O	O	X

Conclusion

- **Achievement**
  - New intra-instruction redundancy HIGHT implementation on 32-bit ARM Cortex-M4
- **Future Works**
  - Applying proposed methods to other block cipher (CHAM, SPECK) or platforms (AVX)

Q & A