KOOKMIN UNIVERSITY SAMSUNG SAMSUNG SDS SICADA SIde Channel Analysis Design Academy

Conference on Cryptographic Hardware and Embedded Systems 2019

Atlanta, USA, August 25–28, 2019

Novel Side-Channel Attacks on Quasi-Cyclic Code-Based Cryptography

2019.08.28

Bo-Yeon Sim[1,†] , Jihoon Kwon[2], Kyu Young Choi[2], Jihoon Cho[2], Aesun Park [3,†], and Dong-Guk Han[1,3, †]

[1] Department of Mathematics, Kookmin University, Seoul, South Korea

[2] Security Research Team, Samsung SDS, Inc., Seoul, South Korea

[3] Department of Financial Information Security, Kookmin University, Seoul, South Korea

[†] SICADA(**S**ide **C**hannel **A**nalysis **D**esign **A**cademy) Laboratory

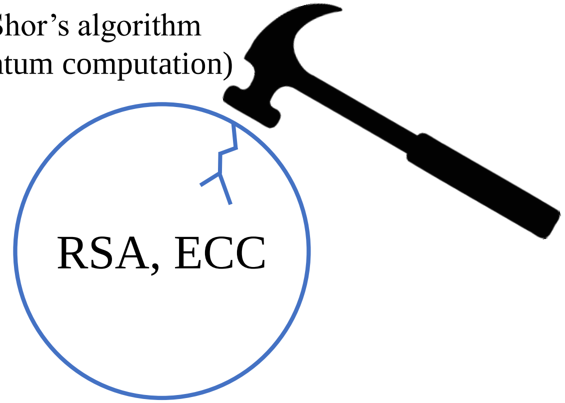
PKC (Public Key Cryptosystem)

RSA, ECC

Factoring and Discrete Logarithms

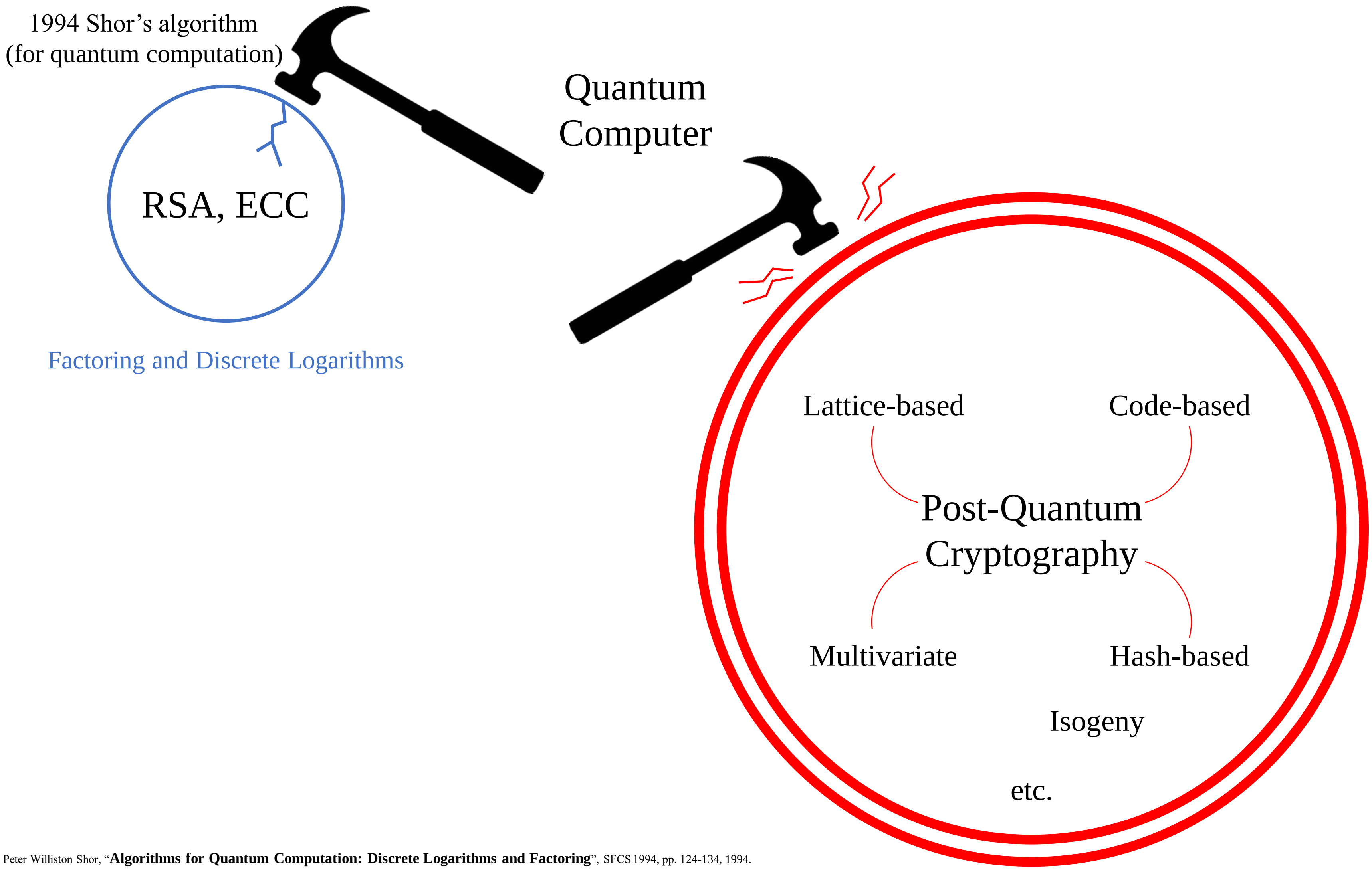
[1] Peter Williston Shor, "**Algorithms for Quantum Computation: Discrete Logarithms and Factoring**", SFCS 1994, pp. 124-134, 1994.

PKC (Public Key Cryptosystem)



[1] Peter Williston Shor, "**Algorithms for Quantum Computation: Discrete Logarithms and Factoring**", SFCS 1994, pp. 124-134, 1994.

PKC (Public Key Cryptosystem)



[1] Peter Williston Shor, "**Algorithms for Quantum Computation: Discrete Logarithms and Factoring**", SFCS 1994, pp. 124-134, 1994.

PKC (Public Key Cryptosystem)

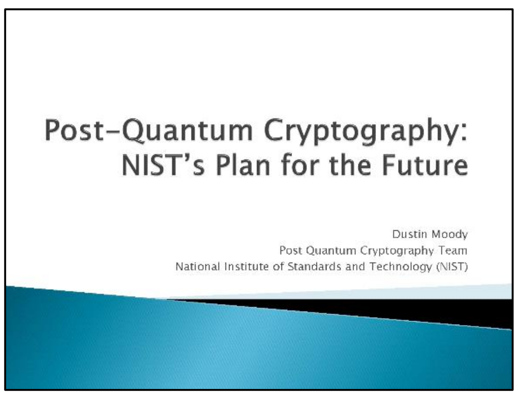
Dec 20, 2016
Formal Call for Proposals

February 24-26, 2016

April 11-13, 2018

August 22-24, 2019

PQCrypto 2016



NIST First PQC Standardization Conference

co-located with

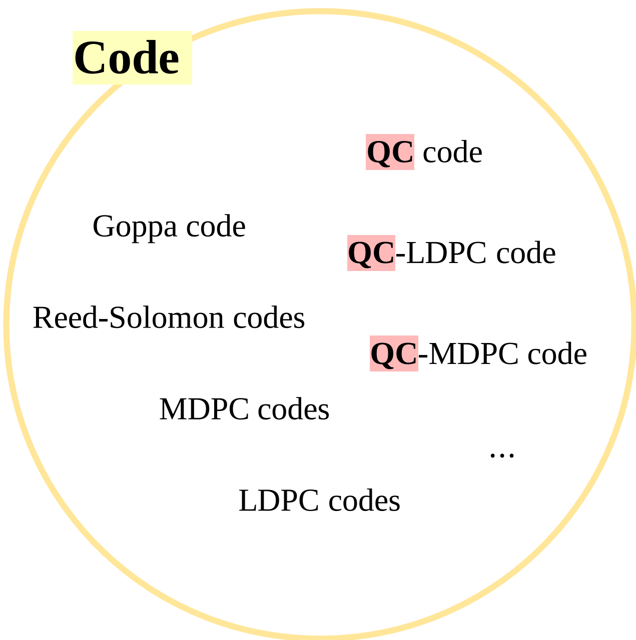
PQCrypto 2018

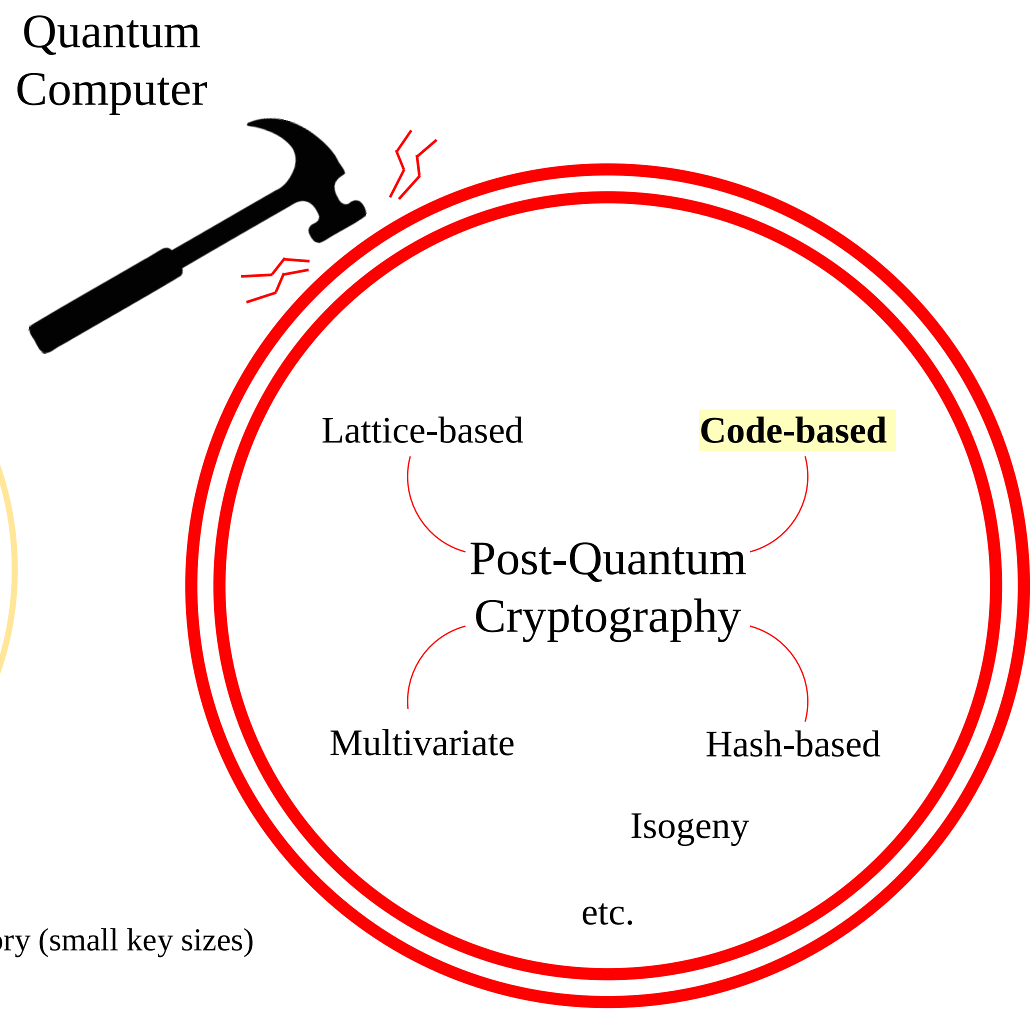
NIST Second PQC Standardization Conference

co-located with

January 30, 2019
Second Round Candidates announced
(26 algorithms)

PKC (Public Key Cryptosystem)



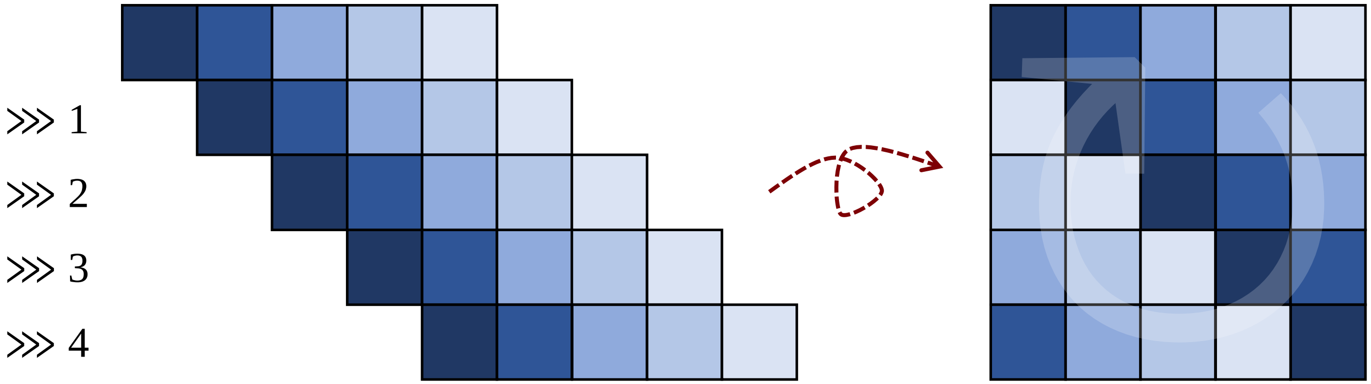


- **Quasi-Cyclic** code for saving memory (small key sizes)

QC (Quasi-Cyclic) Code

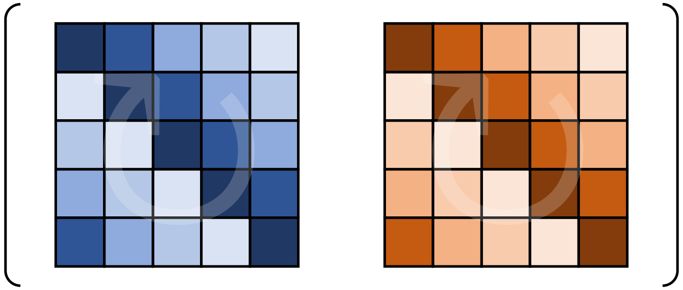
Circulant matrix

- The **top row** (or the leftmost column) of a circulant matrix is the generator of the circulant matrix



Quasi-Cyclic Matrix

$\boldsymbol{H} =$

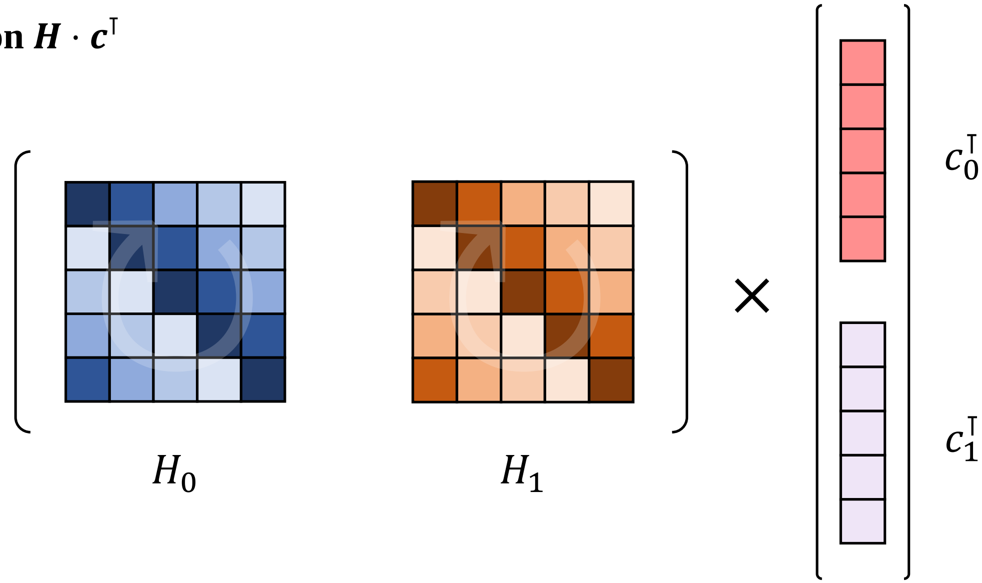


H_0 H_1

QC (Quasi-Cyclic) Code

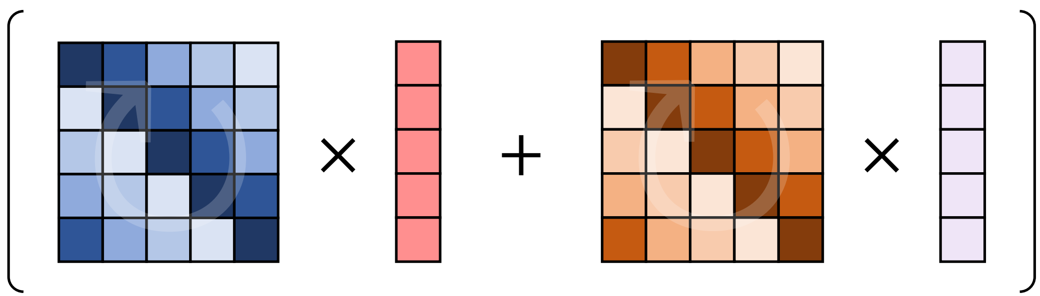
- **Syndrome computation $\boldsymbol{H} \cdot \boldsymbol{c}^\top$**

$\boldsymbol{H} \cdot \boldsymbol{c}^\top =$



H_0 H_1

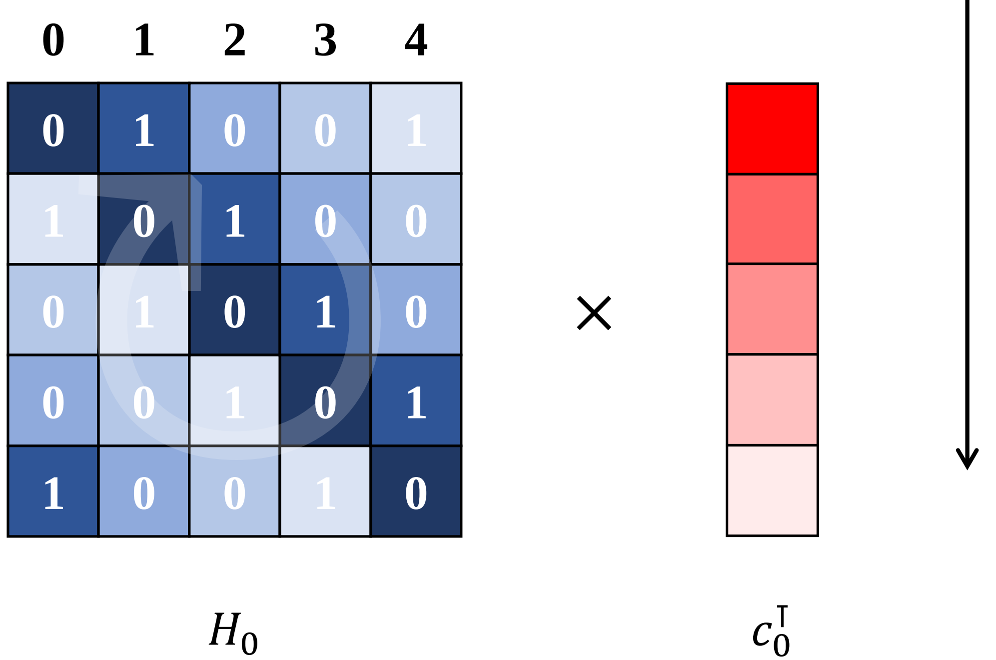
$=$



$H_0 \cdot c_0^\top$ $H_1 \cdot c_1^\top$

QC (Quasi-Cyclic) Code

- **Syndrome computation $H \cdot c^{\intercal}$**



2014 Timing Attack

(Simple Power Analysis)

Constant-Time Multiplication for QC (Quasi-Cyclic) Code

- **Syndrome computation $H \cdot c^{\intercal}$**

$$\begin{array}{ccccc} 0 & 1 & 2 & 3 & 4 \end{array}$$
$$\begin{bmatrix} 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 \end{bmatrix} \times c_0^{\intercal}$$

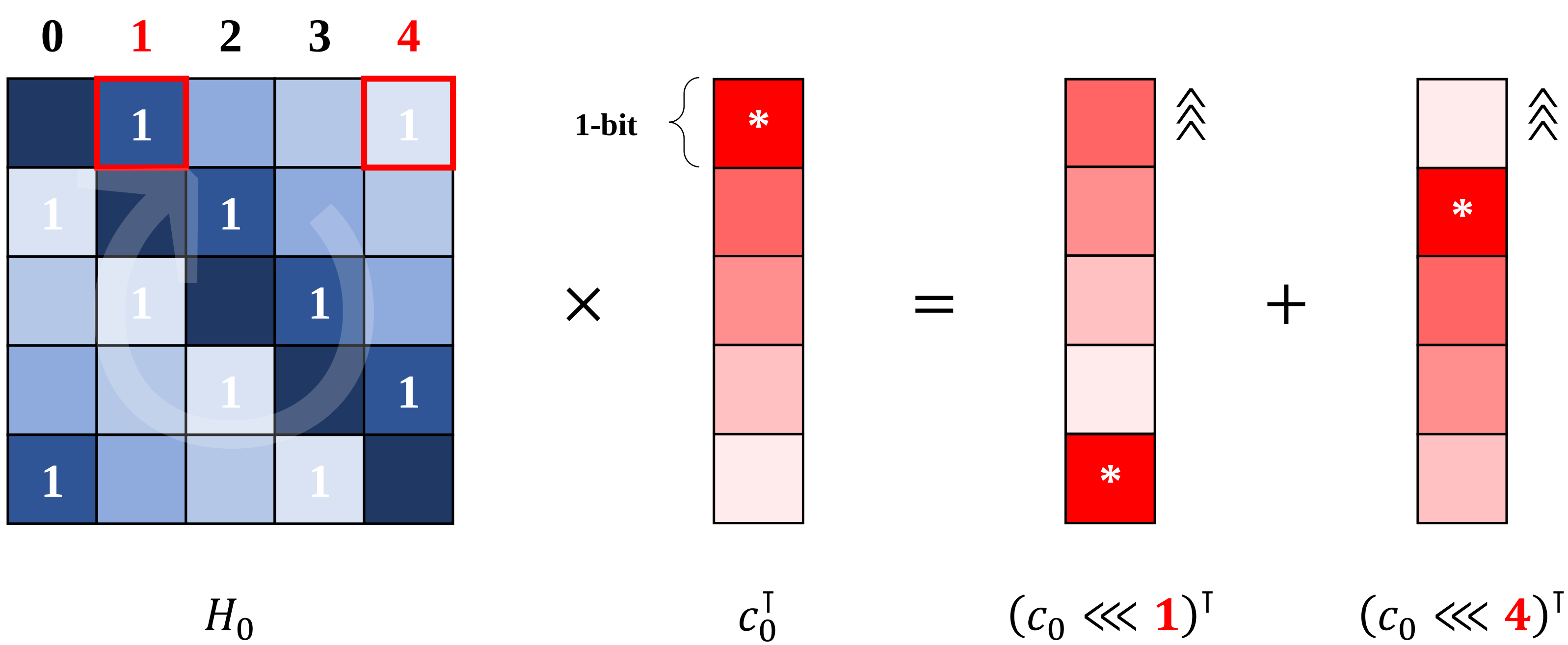
H_0 $\qquad$ $c_0^{\intercal}$

2014 Timing Attack (Simple Power Analysis)

2016 Constant-Time Implementation

▣ Constant-Time Multiplication for QC (Quasi-Cyclic) Code

❖ **Syndrome computation $H \cdot c^{\intercal}$**



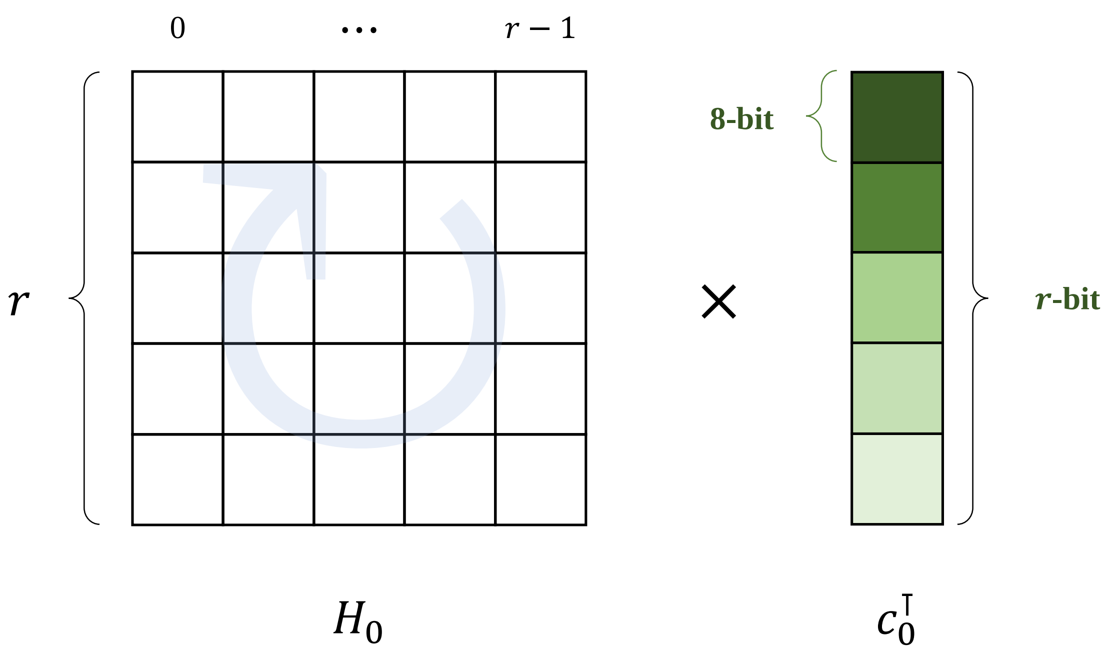
Calculated by
Constant-Time Multiplication

$* \in \{0,1\}$

■ Constant-Time Multiplication for QC (Quasi-Cyclic) Code

- **Syndrome computation $H \cdot c^\intercal$**

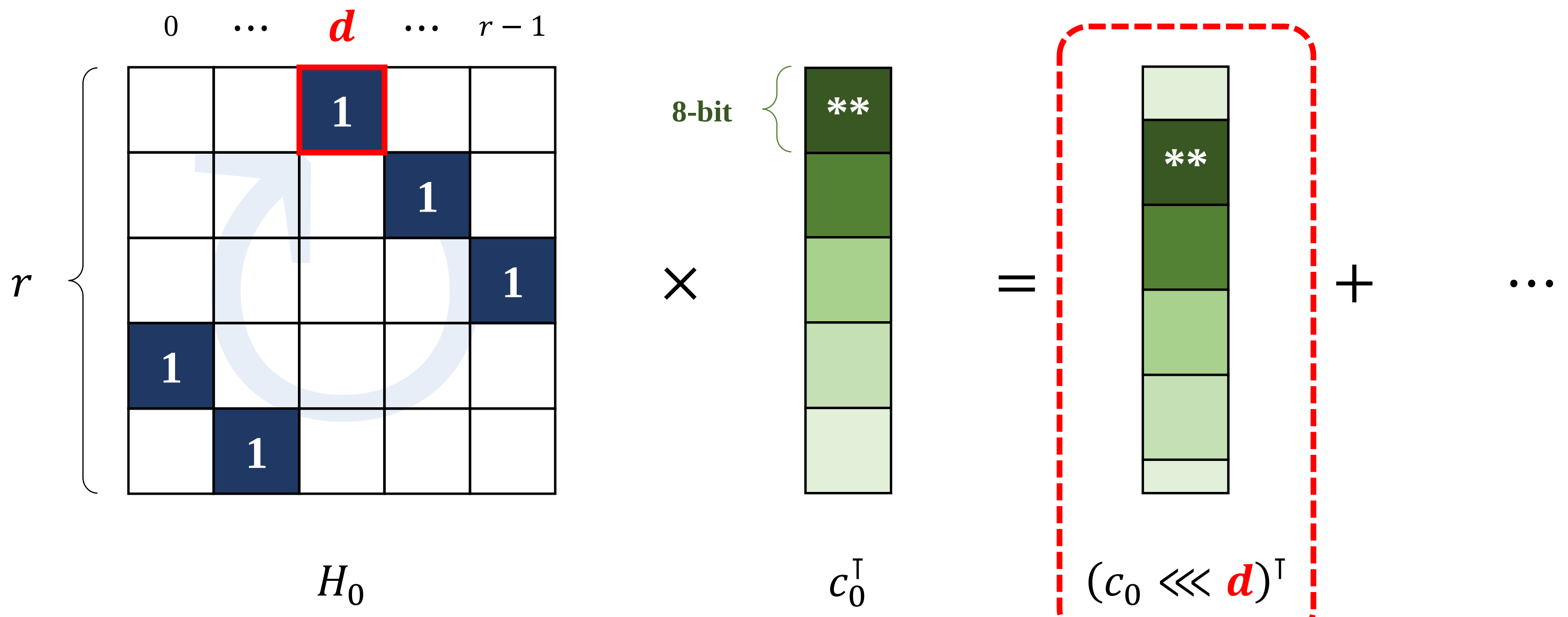
8-bit word



Constant-Time Multiplication for QC (Quasi-Cyclic) Code

- **Syndrome computation $H \cdot c^{\intercal}$**

8-bit word



$** \in \{0,1\}^8$

Constant-Time Multiplication for QC (Quasi-Cyclic) Code

- **Syndrome computation $H \cdot c^\intercal$**

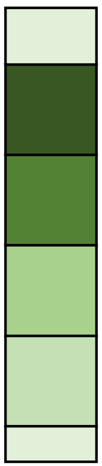
$$d = (11101010)_2$$

$\uparrow d_7$

8-bit word

2^7 = 128-bit → 16-byte

	R					unrotated
16-byte rotate <<					*R*	rotated



$$(c_0 \lll d)^\intercal$$

▣ Constant-Time Multiplication for QC (Quasi-Cyclic) Code

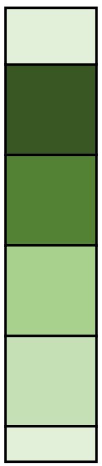
- **Syndrome computation $H \cdot c^{\intercal}$**

$$d = (11101010)_2$$

$\uparrow d_7$

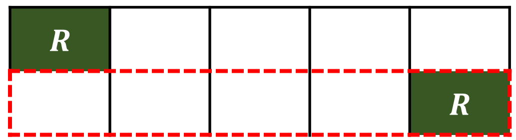
8-bit word

2^7 = 128-bit → 16-byte



$(c_0 \lll d)^{\intercal}$

16-byte rotate <<



unrotated & $0x00 \cdots 00$

rotated $d_7 = 1$ & $0xff \cdots ff$

Constant-Time Multiplication for QC (Quasi-Cyclic) Code

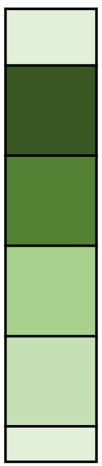
- Syndrome computation $H \cdot c^{\intercal}$

$d = (11101010)_2$

$\uparrow d_6$

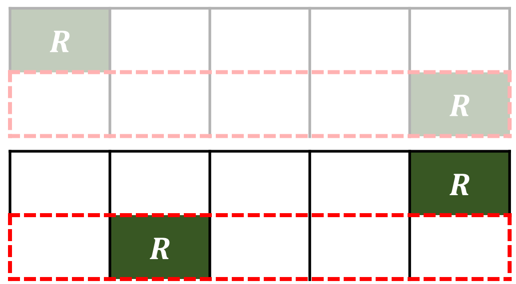
8-bit word

2^6 = 64-bit → 8-byte



$(c_0 \lll d)^{\intercal}$

16-byte rotate <<



8-byte rotate <<

unrotated

rotated $d_7 = 1$

unrotated $\& \, 0x00 \cdots 00$

rotated $d_6 = 1$ $\& \, 0xff \cdots ff$

Constant-Time Multiplication for QC (Quasi-Cyclic) Code

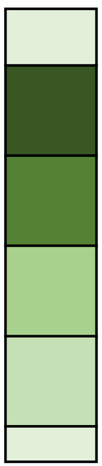
- **Syndrome computation $H \cdot c^\top$**

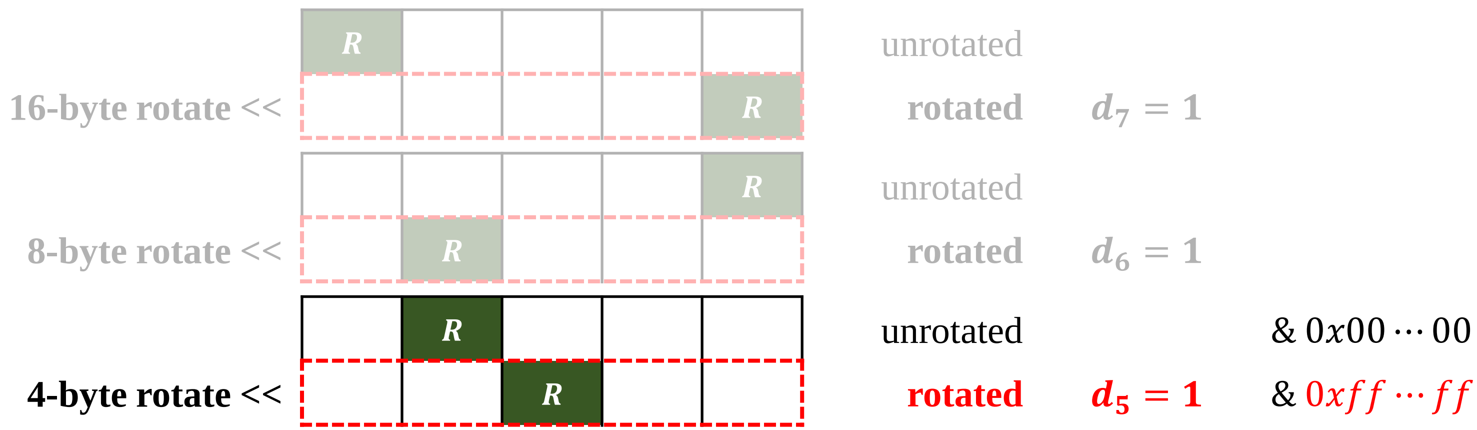
$$d = (11101010)_2$$

$\uparrow d_5$

8-bit word

2^5 = 32-bit → 4-byte





$$(c_0 \lll d)^\top$$

Constant-Time Multiplication for QC (Quasi-Cyclic) Code

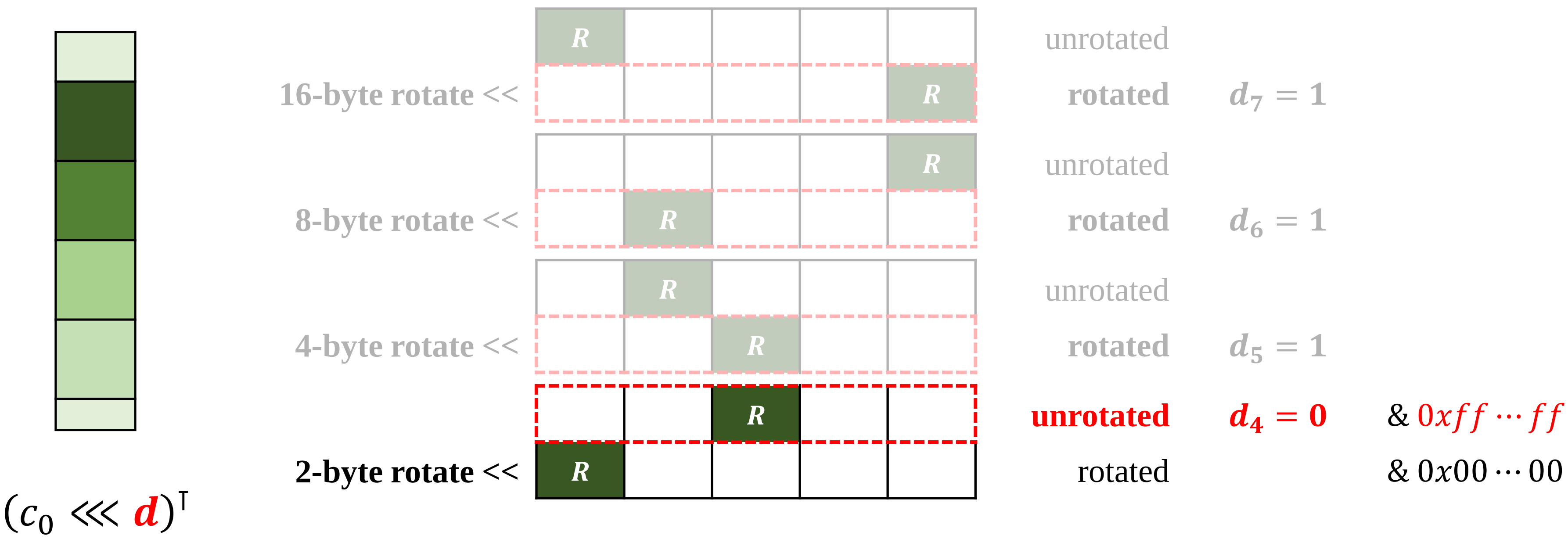
- **Syndrome computation $H \cdot c^{\intercal}$**

$$d = (11101010)_2$$

$\uparrow d_4$

8-bit word

2^4 = 16-bit → 2-byte



Constant-Time Multiplication for QC (Quasi-Cyclic) Code

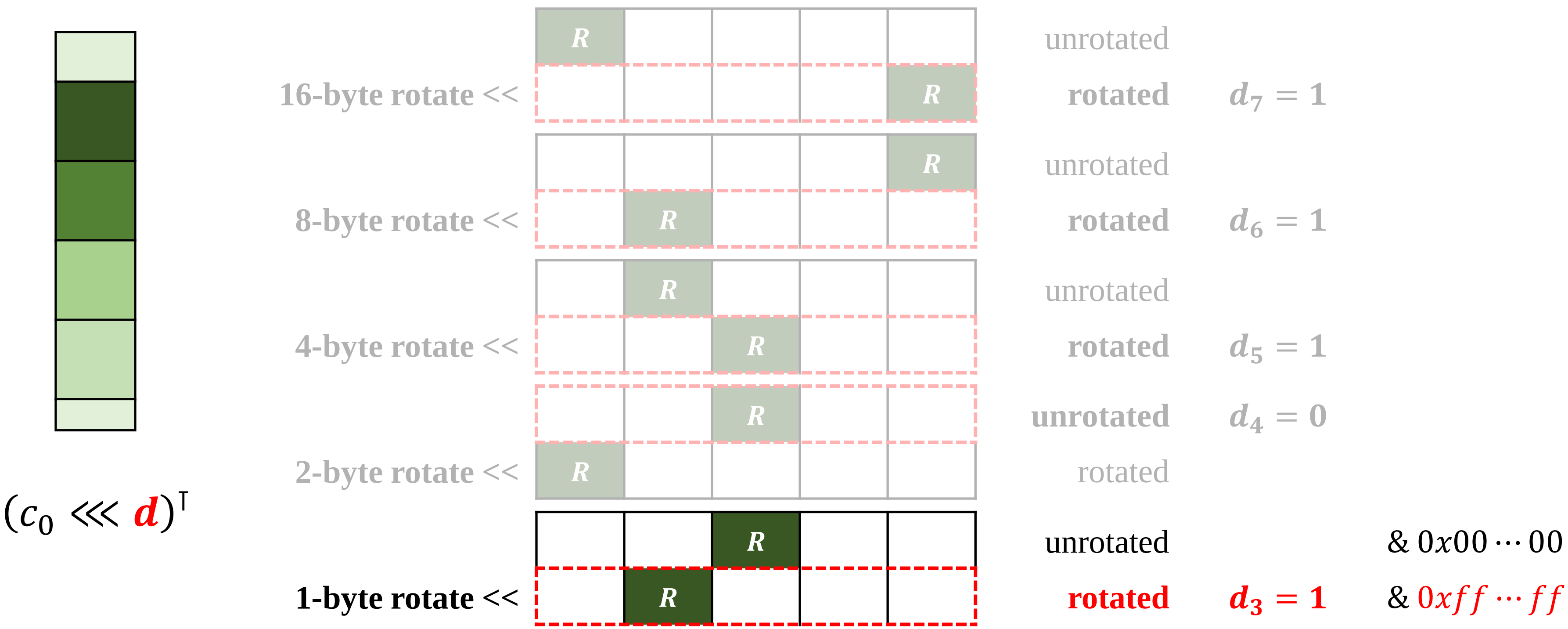
- **Syndrome computation $H \cdot c^{\intercal}$**

$$d = (11101010)_2$$

$\uparrow d_3$

8-bit word

2^3 = 8-bit → 1-byte



▣ Constant-Time Multiplication for QC (Quasi-Cyclic) Code

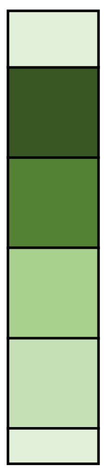
8-bit word

- **Syndrome computation $H \cdot c^\top$**

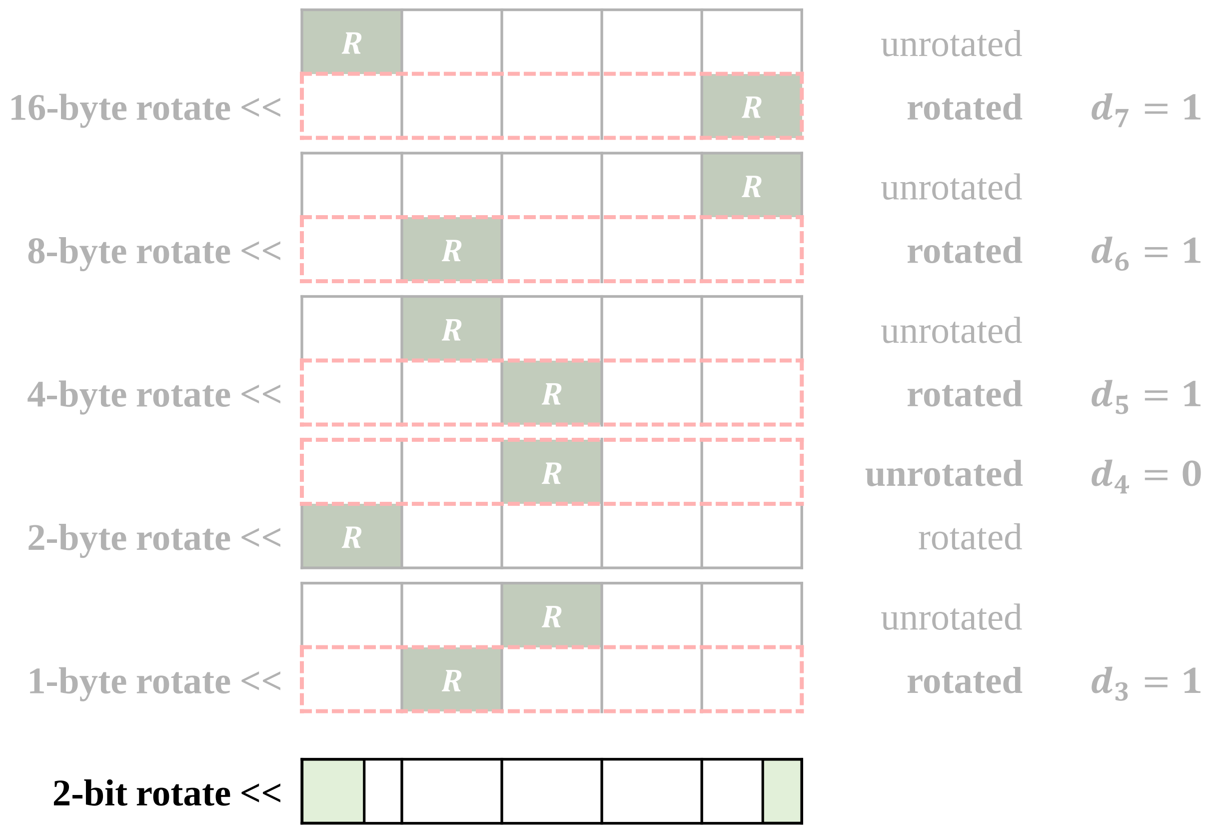
$$d = (11101010)_2$$

$$\underbrace{010}_{(d_2 d_1 d_0)_2 < \text{8-bit}}$$

$$0 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0 = \text{2-bit}$$



$(c_0 \lll d)^\top$



Constant-Time Multiplication for QC (Quasi-Cyclic) Code

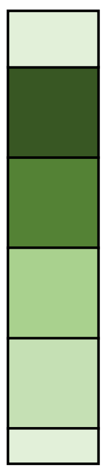
- Syndrome computation $H \cdot c^{\intercal}$

$$d = (11101010)_2$$

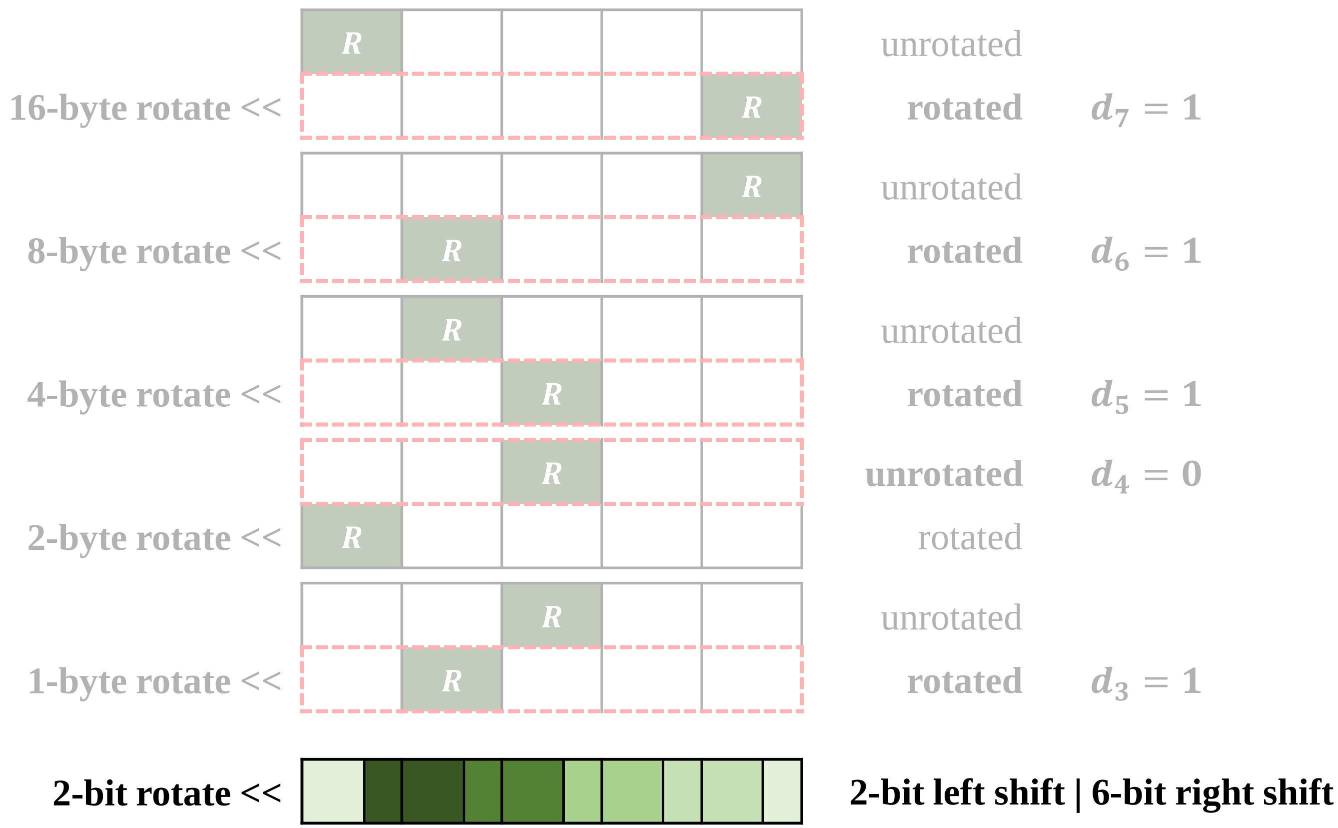
8-bit word

$(d_2 d_1 d_0)_2 < 8\text{-bit}$

$$0 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0 = 2\text{-bit}$$



$(c_0 \lll d)^{\intercal}$



Side-Channel Attacks on QC Code-Based Cryptography

Syndrome computation $\boldsymbol{H} \cdot \boldsymbol{c}^{\intercal}$

2014 Timing Attack

(Simple Power Analysis)

TA Countermeasure

2016 Constant-Time Implementation

2017 Differential Power Analysis on Constant-Time Implementation

Motivations and Contributions

Syndrome computation $\boldsymbol{H} \cdot \boldsymbol{c}^{\intercal}$

2014 Timing Attack

(Simple Power Analysis)

TA Countermeasure

2016 Constant-Time Implementation

2017 Differential Power Analysis on Constant-Time Implementation

Limitation: It could not completely recover accurate secret indices, requiring further solving linear equations to obtain entire secret information

Is there no method allows to recover accurate secret indices using only side-channel information?

Motivations and Contributions

Syndrome computation $\boldsymbol{H} \cdot \boldsymbol{c}^{\intercal}$

2014 Timing Attack

(Simple Power Analysis)

TA Countermeasure

2016 Constant-Time Implementation

2017 Differential Power Analysis on Constant-Time Implementation

Limitation: It could not completely recover accurate secret indices, requiring further solving linear equations to obtain entire secret information

Is there no method allows to recover accurate secret indices using only side-channel information?

In this paper

Enhanced Multiple-Trace Attack which can recover accurate secret indices using only side-channel information

Motivations and Contributions

Syndrome computation $\boldsymbol{H} \cdot \boldsymbol{c}^{\intercal}$

2014 Timing Attack

(Simple Power Analysis)

TA Countermeasure

2016 Constant-Time Implementation

2017 Differential Power Analysis on Constant-Time Implementation

DPA Countermeasure

2017 Codeword Randomization (Masking) or **Systems use ephemeral key pairs**

Motivations and Contributions

Syndrome computation $\boldsymbol{H} \cdot \boldsymbol{c}^{\intercal}$

2014 Timing Attack

(Simple Power Analysis)

TA Countermeasure

2016 Constant-Time Implementation

2017 Differential Power Analysis on Constant-Time Implementation

DPA Countermeasure

2017 Codeword Randomization (Masking) or **Systems use ephemeral key pairs**

Constraint : Cannot use multiple traces

Is it impossible to attack using only a single trace?

Motivations and Contributions

Syndrome computation $\boldsymbol{H} \cdot \boldsymbol{c}^{\mathsf{T}}$

2014 Timing Attack

(Simple Power Analysis)

TA Countermeasure

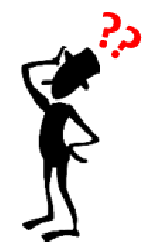
2016 Constant-Time Implementation

2017 Differential Power Analysis on Constant-Time Implementation

DPA Countermeasure

2017 Codeword Randomization (Masking) or **Systems use ephemeral key pairs**

Constraint : Cannot use multiple traces



Is it impossible to attack using only a single trace?

In this paper

Novel Single-Trace Attack on QC Code-Based Cryptography Using Masked Constant-Time Multiplication

Contributions

Enhanced Multiple-Trace Attack on QC Code-Based Cryptography Using Constant-Time Multiplication

Novel Single-Trace Attack on QC Code-Based Cryptography Using Masked Constant-Time Multiplication

BIKE
LEDAcrypt

TA + **Constant-Time Multiplication** — **It is insecure against our multiple-trace attack**

DPA + **Codeword Randomization (Masking)** — **It is insecure against our single-trace attack**

DPA + **Use the ephemeral key pairs** — **It is insecure against our single-trace attack**

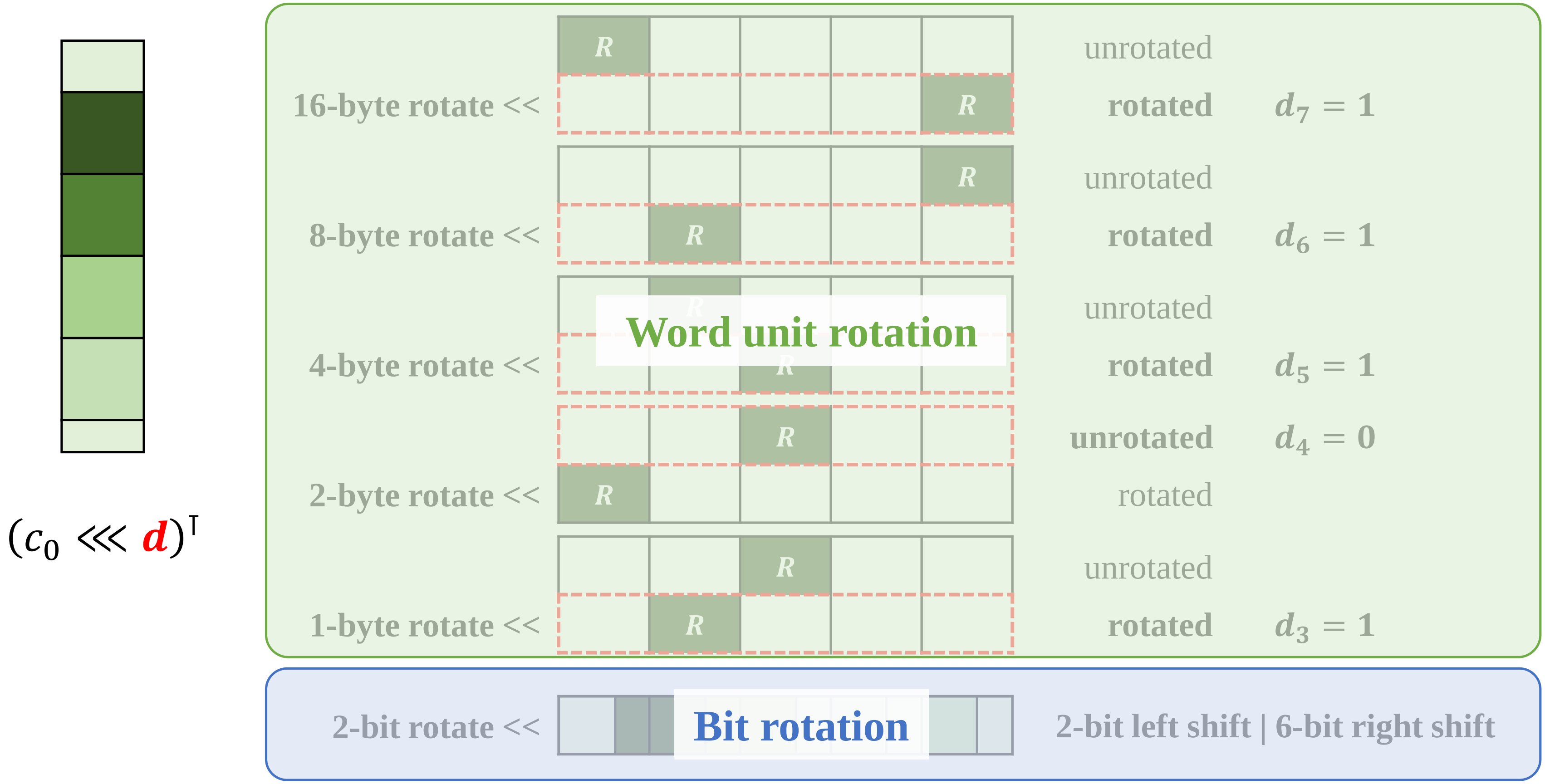
Constant-Time Multiplication for QC (Quasi-Cyclic) Code

- **Syndrome computation $H \cdot c^{\intercal}$**

$$d = (\underbrace{11101}_{\text{multiples of 8}}\underbrace{010}_{< \text{8-bit}})_2$$

8-bit word

$d = (d_7 d_6 d_5 d_4 d_3 d_2 d_1 d_0)_2, d_i \in \{0, 1\}$



▣ Multiple-Trace Attack on Constant-Time Multiplication

8-bit word

$$d = (\underbrace{d_7 d_6 d_5 d_4 d_3}_{\text{Correlation Occurring Position}} \underbrace{d_2 d_1 d_0}_{\text{Correlation Power Analysis}})_2$$

Word unit rotation

$$result = \begin{cases} unrotated & , if\ d_i = 0 \\ rotated & , if\ d_i = 1 \end{cases}$$

$$result = \begin{cases} (rotated \,\&\, 0x00) \oplus (unrotated \,\&\, 0xff) = unrotated & , if\ d_i = 0 \\ (rotated \,\&\, 0xff) \oplus (unrotated \,\&\, 0x00) = rotated & , if\ d_i = 1 \end{cases}$$

Bit rotation

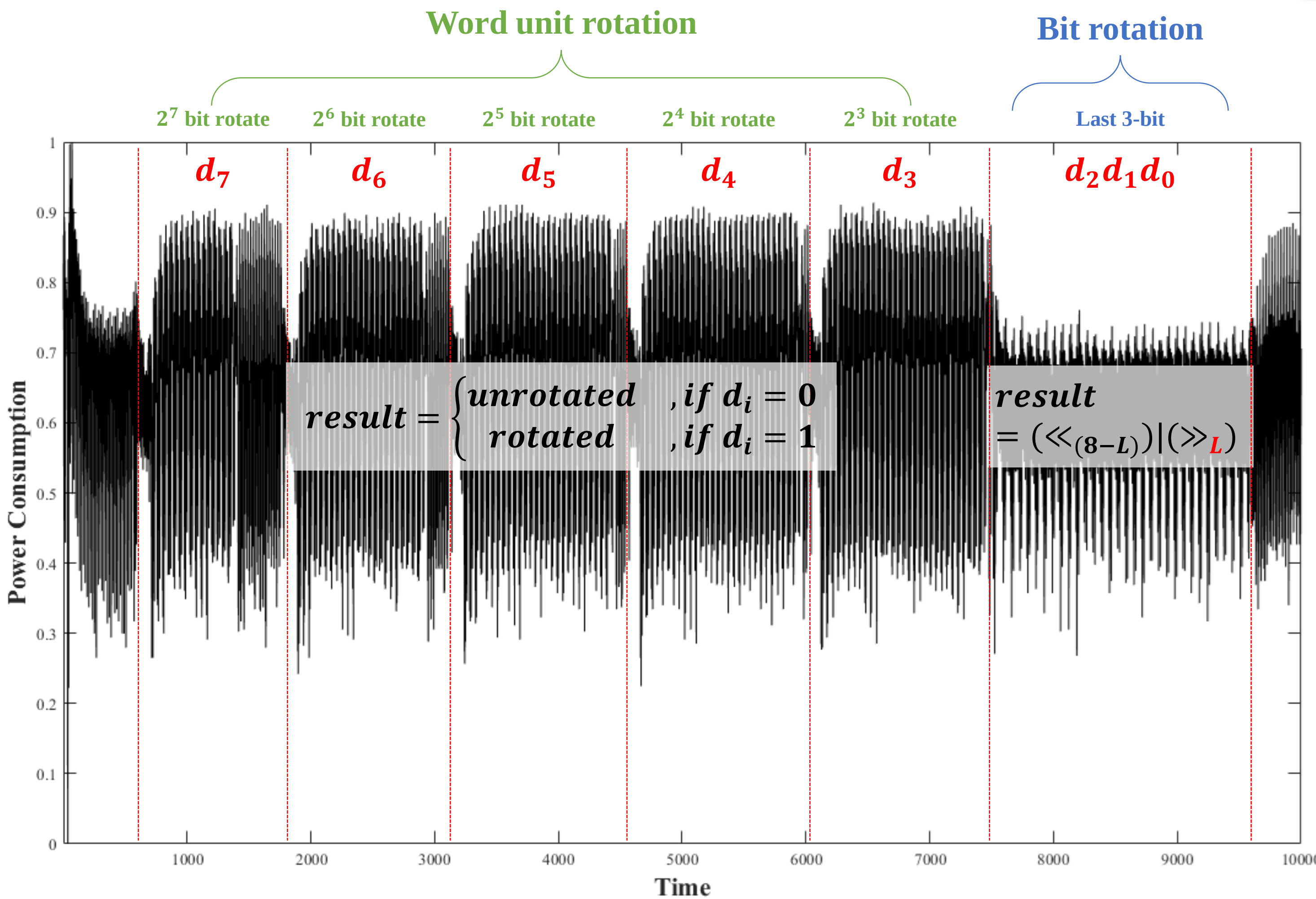
$$result = (\ll_{8-L}) | (\gg_L)$$

$$0 \le L = (d_2 d_1 d_0)_2 < 8$$

Experiment

- $d = (d_7 d_6 d_5 d_4 d_3 d_2 d_1 d_0)_2, d_i \in \{0, 1\}$

8-bit word



▣ Multiple-Trace Attack on the Word Unit Rotation

$$d = (\overbrace{d_7 d_6 d_5 d_4 d_3}^{\text{target}} d_2 d_1 d_0)_2, d_i \in \{0, 1\}$$

8-bit word

$$result = \begin{cases} unrotated & , if\ d_i = 0 \\ rotated & , if\ d_i = 1 \end{cases}$$

■ Multiple-Trace Attack on the Word Unit Rotation

$$d = (\underbrace{d_7 d_6 d_5 d_4 d_3}_{\text{target}} d_2 d_1 d_0)_2, d_i \in \{0, 1\}$$

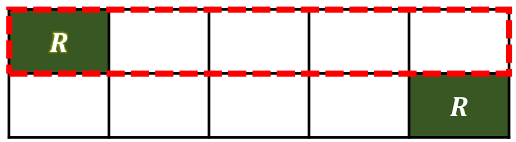
$$R \in_{Random} \{0, 1\}^8$$

8-bit word

Property 1.

$$result = \begin{cases} unrotated & , if\ d_i = 0 \\ rotated & , if\ d_i = 1 \end{cases}$$

16-byte rotate <<



Unrotated value is chosen

$$d = (\color{red}{0}\color{black}{1101010})_2$$

Multiple-Trace Attack on the Word Unit Rotation

$$d = (\overbrace{d_7 d_6 d_5 d_4 d_3 d_2 d_1 d_0}^{\text{target}})_2, d_i \in \{0,1\}$$

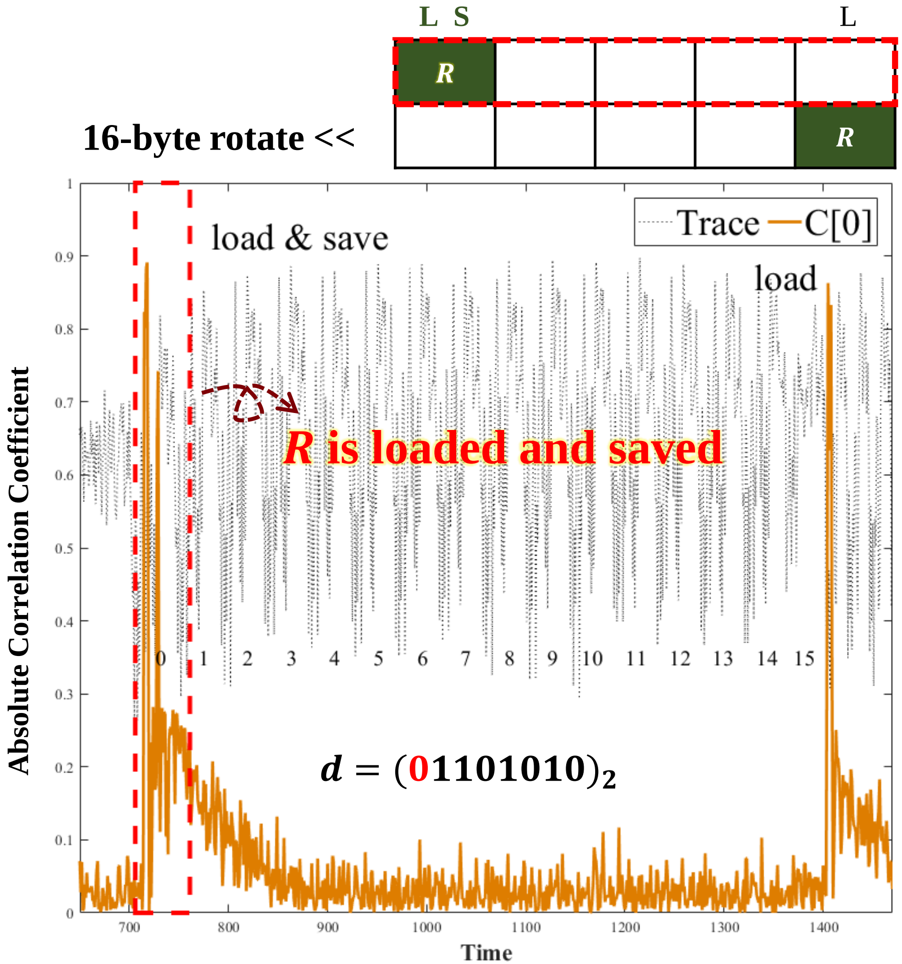
$$R \in_{Random} \{0,1\}^8$$

8-bit word

Property 1.

$$result = \begin{cases} unrotated & , if\ d_i = 0 \\ rotated & , if\ d_i = 1 \end{cases}$$

16-byte rotate <<



▣ Multiple-Trace Attack on the Word Unit Rotation

$$d = (\overbrace{d_7 d_6 d_5 d_4 d_3}^{\text{target}} d_2 d_1 d_0)_2, d_i \in \{0,1\}$$

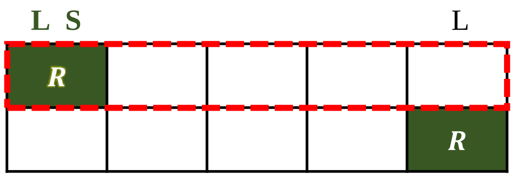
$$R \in_{Random} \{0,1\}^8$$

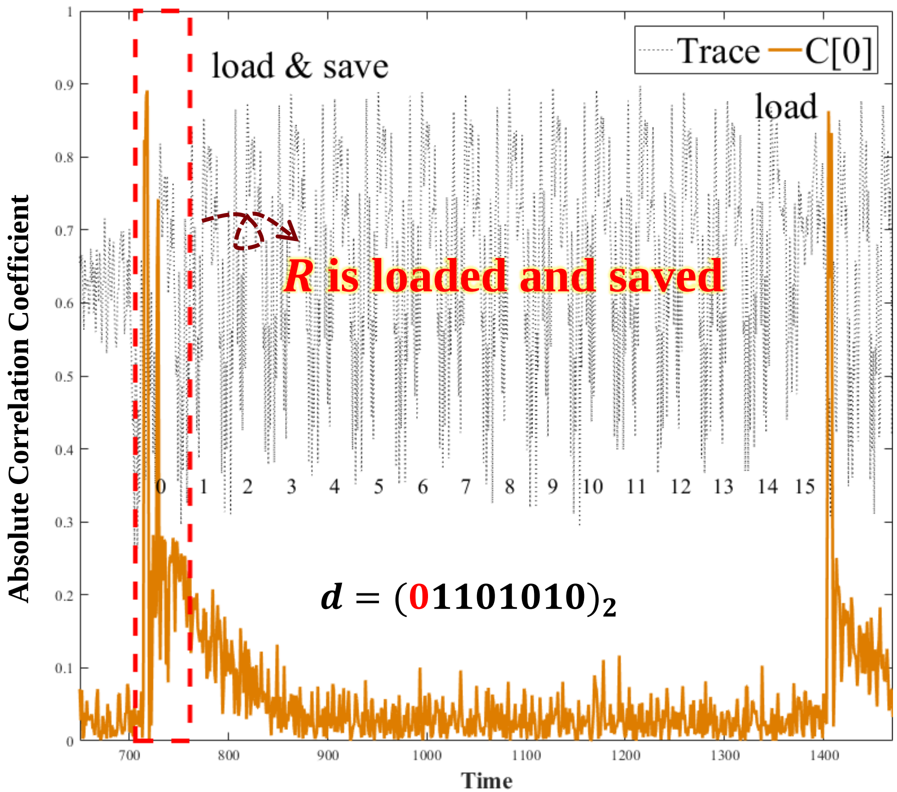
8-bit word

Property 1.

$$result = \begin{cases} unrotated & , if\ d_i = 0 \\ rotated & , if\ d_i = 1 \end{cases}$$

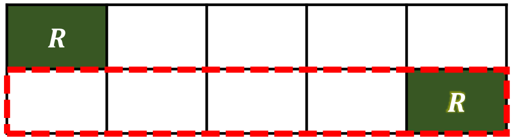
16-byte rotate <<





$$d = (01101010)_2$$

16-byte rotate <<



Rotated value is chosen

$$d = (11101010)_2$$

▣ Multiple-Trace Attack on the Word Unit Rotation

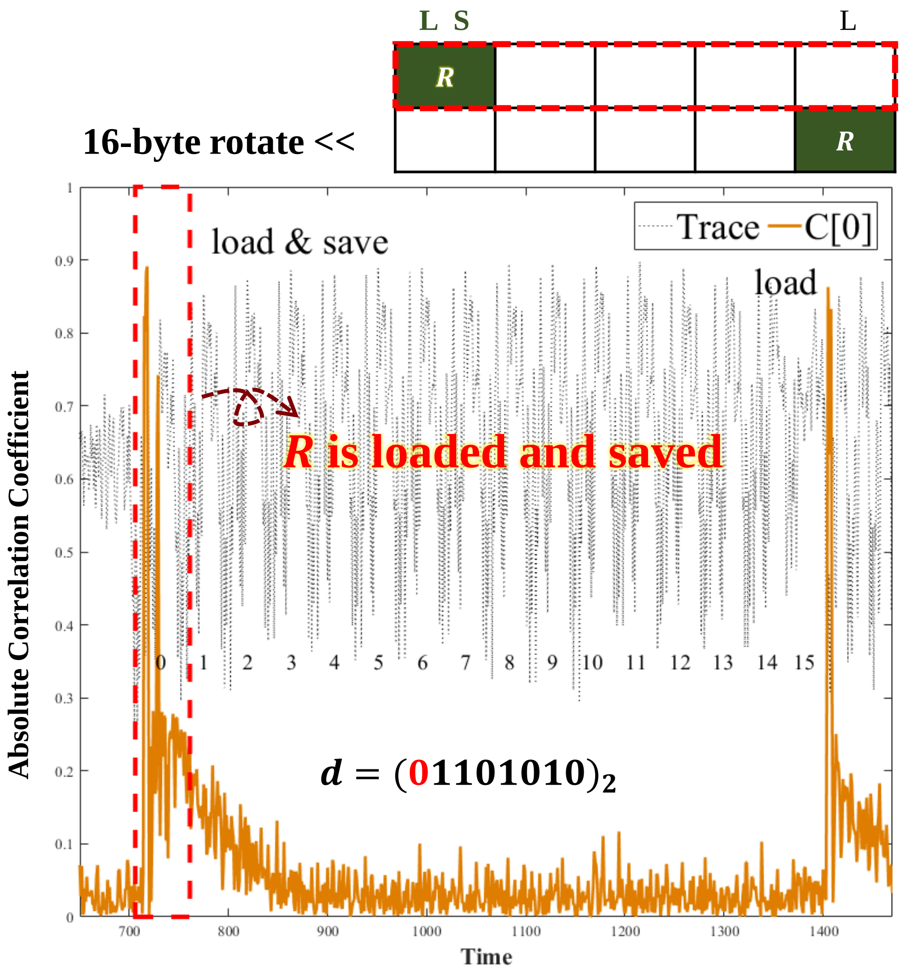
$$d = (\underbrace{d_7 d_6 d_5 d_4 d_3 d_2 d_1 d_0}_{\text{target}})_2, d_i \in \{0,1\}$$

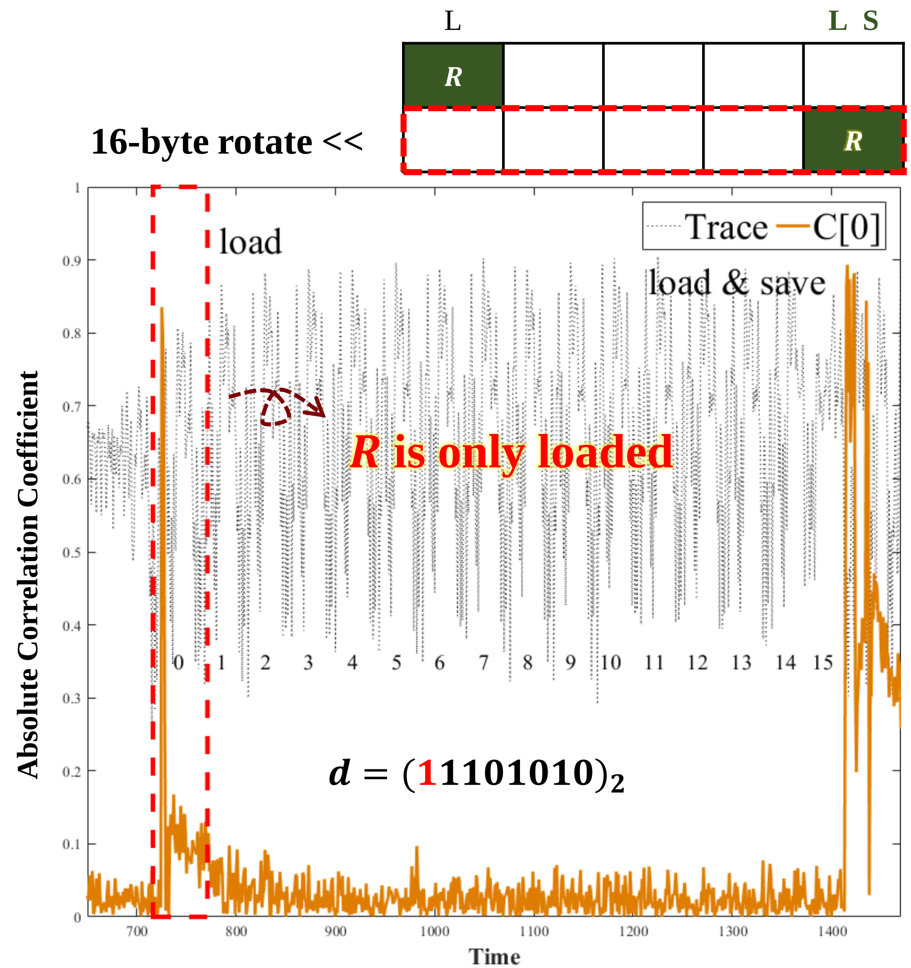
$$R \in_{Random} \{0,1\}^8$$

8-bit word

Property 1.

$$result = \begin{cases} unrotated & , if\ d_i = 0 \\ rotated & , if\ d_i = 1 \end{cases}$$





Multiple-Trace Attack on the Word Unit Rotation

$$d = (\underbrace{d_7 d_6 d_5 d_4 d_3}_{\text{target}} d_2 d_1 d_0)_2, d_i \in \{0,1\}$$

$R \in_{Random} \{0,1\}^8$

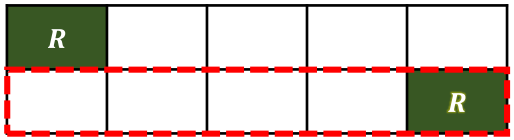
8-bit word

Property 2.

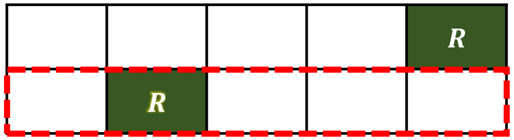
$$result = \begin{cases} unrotated & , if\ d_i = 0 \\ rotated & , if\ d_i = 1 \end{cases}$$

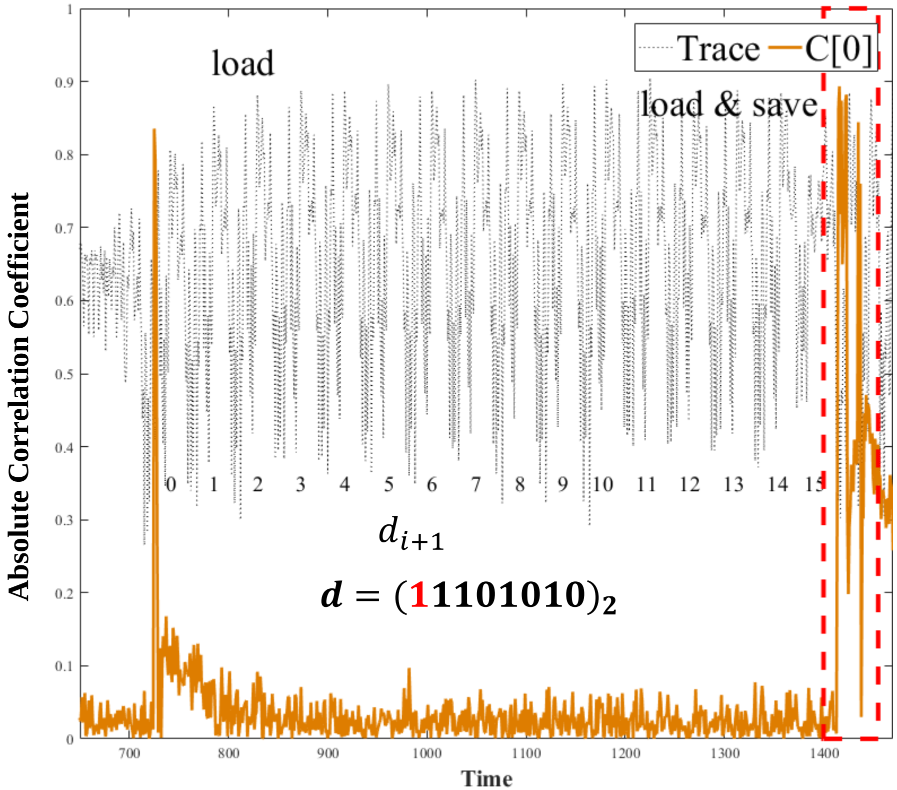
L L S

16-byte rotate <<



8-byte rotate <<





d_{i+1}

$d = (11101010)_2$

Rotated value is chosen

d_i

$d = (11101010)_2$

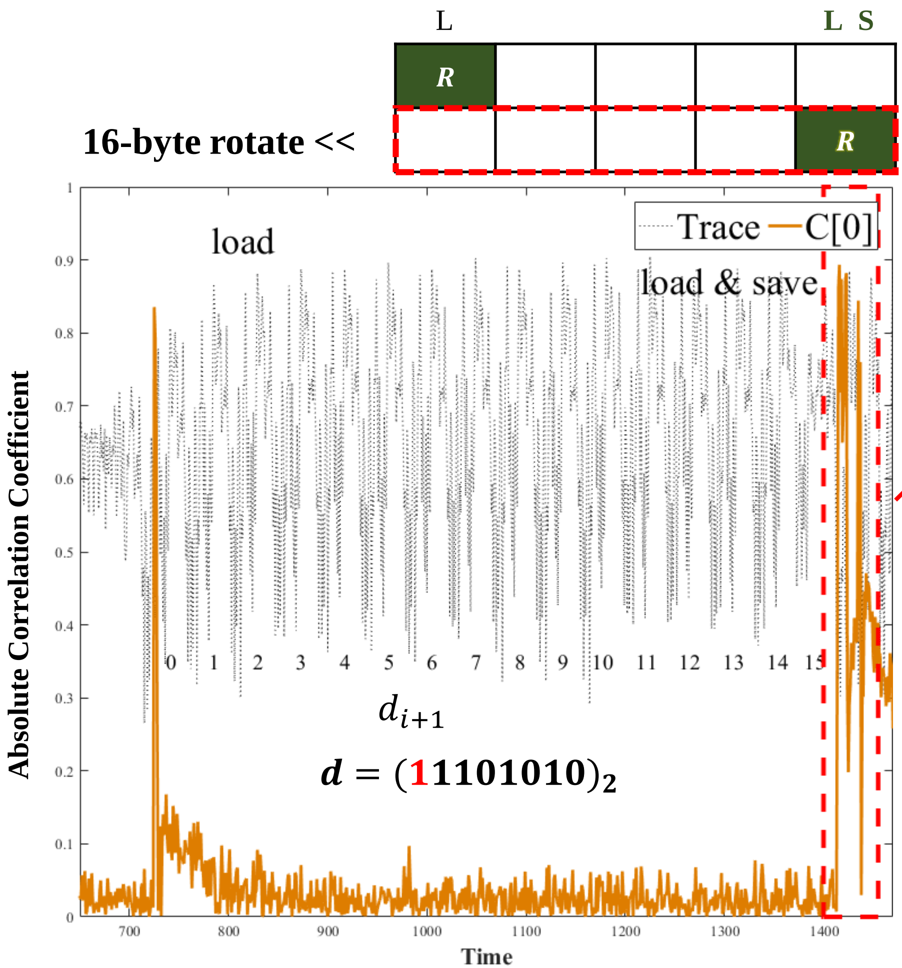
▣ Multiple-Trace Attack on the Word Unit Rotation

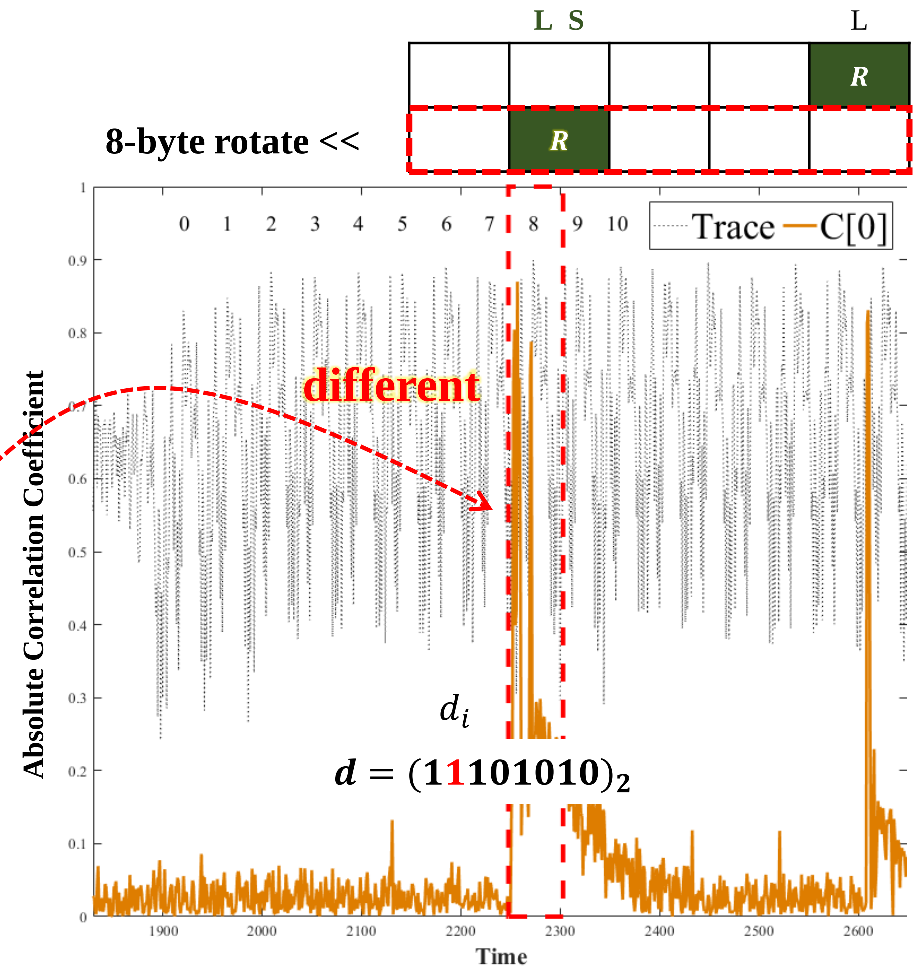
$d = (\underbrace{d_7 d_6 d_5 d_4}_{\text{target}} d_3 d_2 d_1 d_0)_2, d_i \in \{0,1\}$

$R \in_{Random} \{0,1\}^8$

8-bit word

Property 2. $result = \begin{cases} unrotated & ,if\ d_i = 0 \\ rotated & ,if\ d_i = 1 \end{cases}$





▣ Multiple-Trace Attack on the Word Unit Rotation

$d = (\underbrace{d_7 d_6 d_5 d_4 d_3 d_2 d_1 d_0}_{\text{target}})_2, d_i \in \{0,1\}$

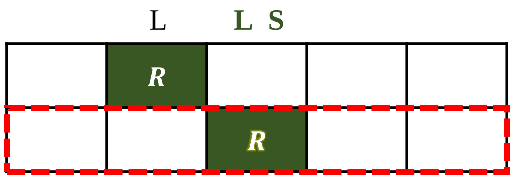
$R \in_{Random} \{0,1\}^8$

8-bit word

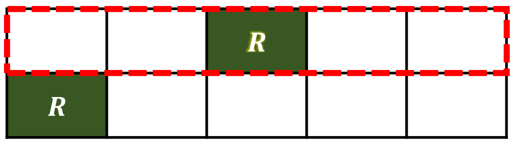
Property 2.

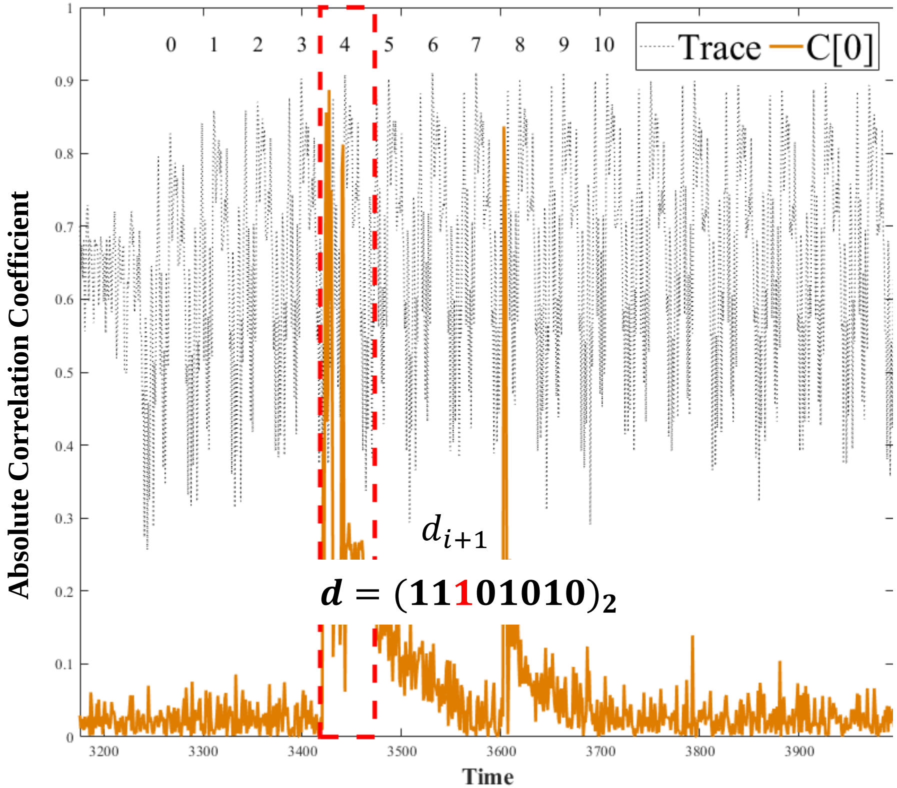
$$result = \begin{cases} unrotated & , if\ d_i = 0 \\ rotated & , if\ d_i = 1 \end{cases}$$

4-byte rotate <<



2-byte rotate <<





d_{i+1}

$d = (11\textcolor{red}{1}01010)_2$

Unrotated value is chosen

d_i

$d = (111\textcolor{red}{0}1010)_2$

SICADA Side Channel Analysis Design Academy
SAMSUNG SAMSUNG SDS

▣ Multiple-Trace Attack on the Word Unit Rotation

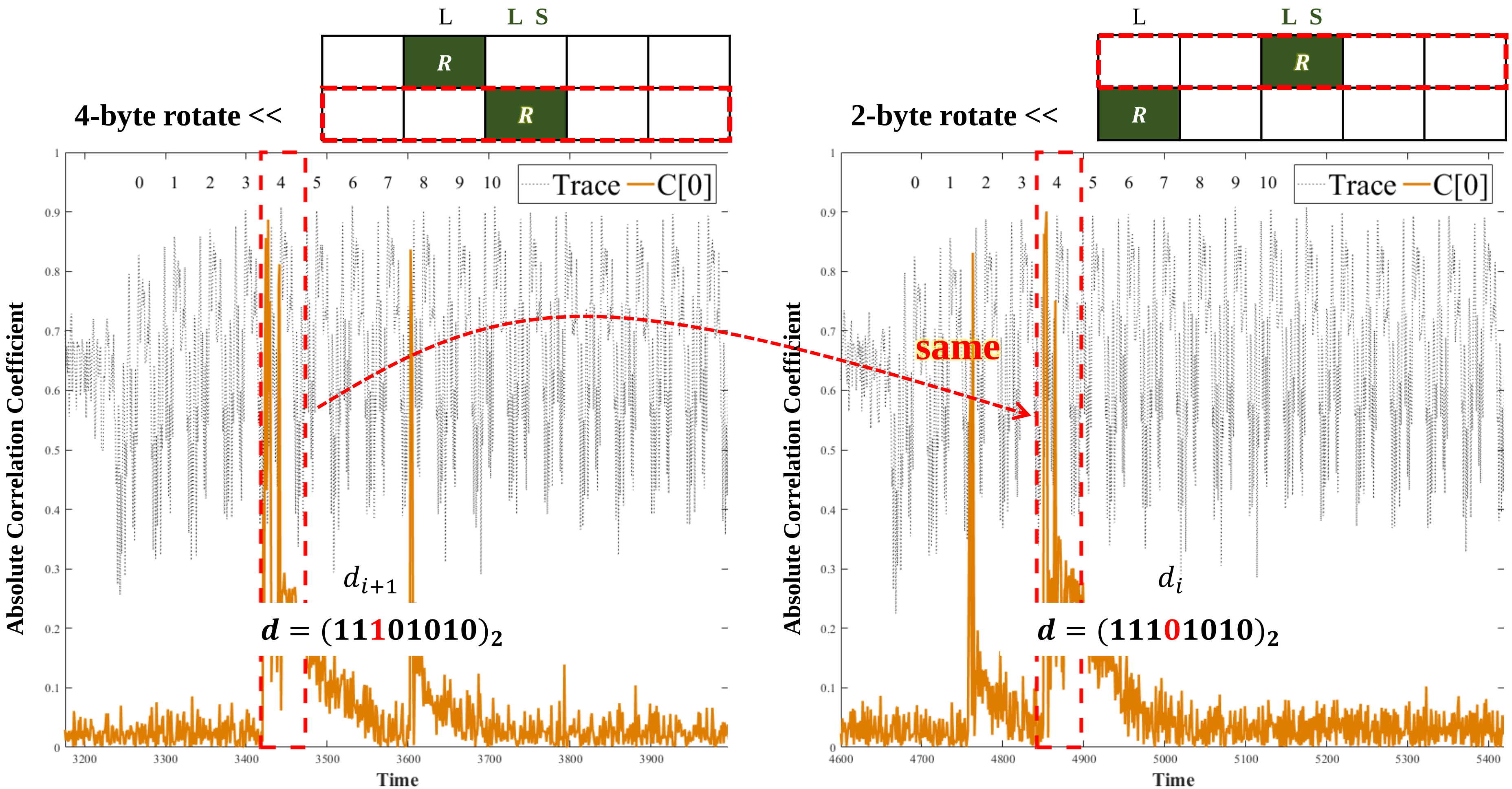
$$d = (\overbrace{d_7d_6d_5d_4d_3}^{\text{target}}d_2d_1d_0)_2, d_i \in \{0,1\}$$

$$R \in_{Random} \{0,1\}^8$$

8-bit word

Property 2.

$$result = \begin{cases} unrotated & , if\ d_i = 0 \\ rotated & , if\ d_i = 1 \end{cases}$$



SICADA Side Channel Analysis Design Academy SAMSUNG SAMSUNG SDS

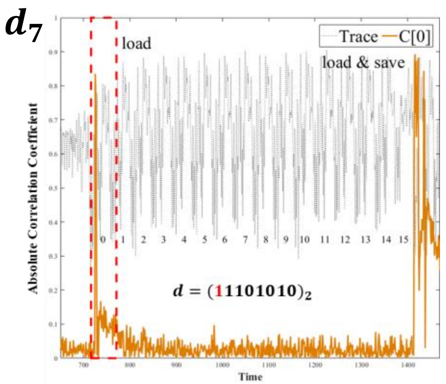
$$d = (\underbrace{d_7 d_6 d_5 d_4 d_3}_{\text{target}} d_2 d_1 d_0)_2,\ d_i \in \{0,1\}$$

$$R \in_{Random} \{0,1\}^8$$

8-bit word

▣ Multiple-Trace Attack on the Word Unit Rotation

- Step 1. Find the most significant bit d_7 based on Property 1



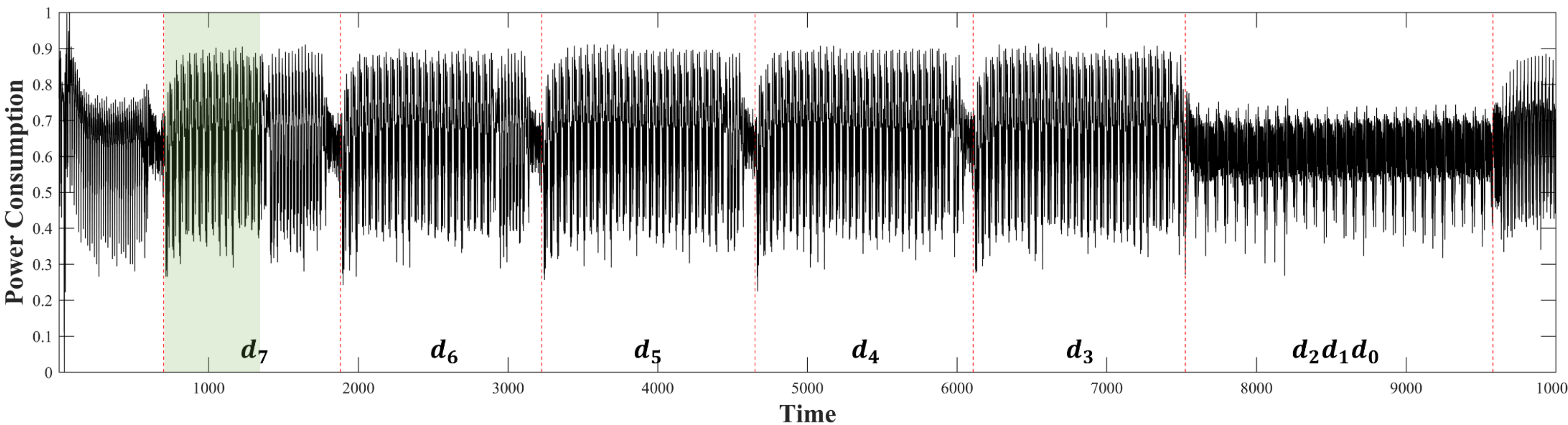
***R* is only loaded** in the first operation

Power consumption related to *R*

does not occurs sequentially twice

in the first operation part

$d_7 = 1$



Multiple-Trace Attack on the Word Unit Rotation

$d = (\underbrace{d_7 d_6 d_5 d_4 d_3}_{\text{target}} d_2 d_1 d_0)_2, d_i \in \{0, 1\}$

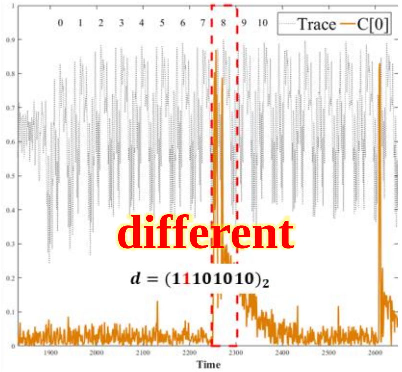
$R \in_{Random} \{0, 1\}^8$

8-bit word

- Step 2. Find from d_6 to d_3 based on Property 2

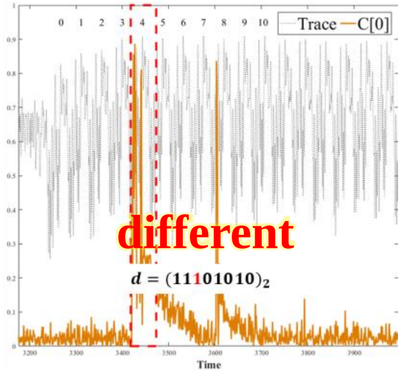
power consumption related to ***R* occurs sequentially twice in the ___ iteration**

d_6



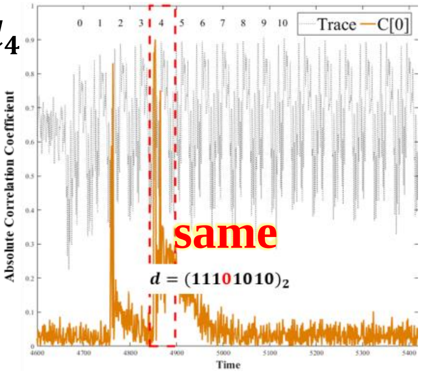
$d_6 = 1$

d_5



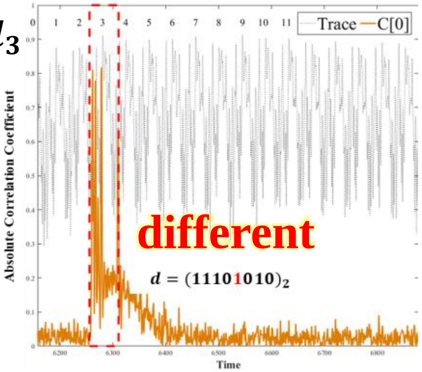
$d_5 = 1$

d_4

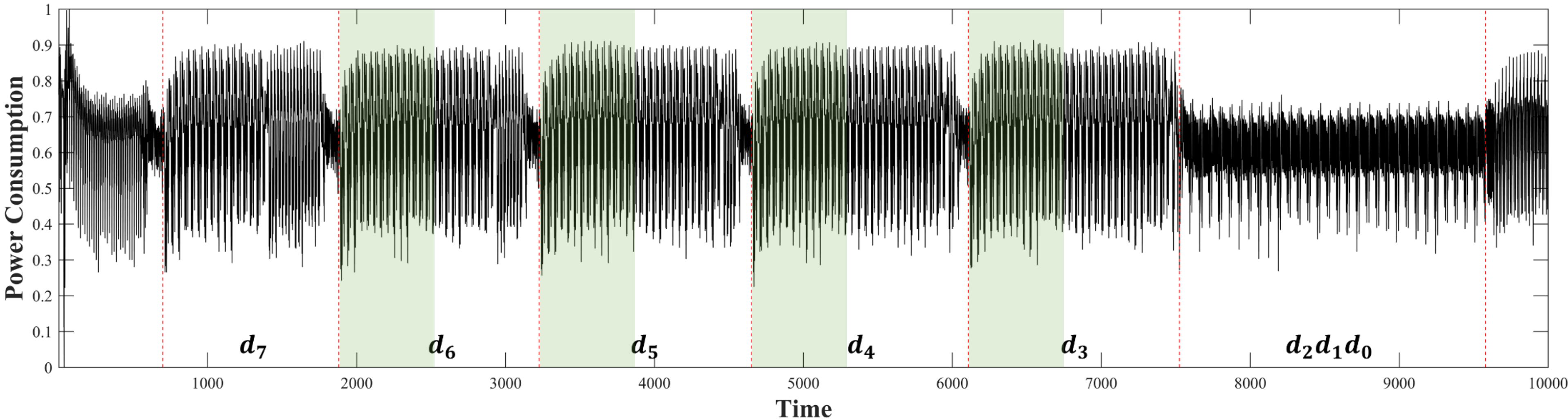


$d_4 = 0$

d_3



$d_3 = 1$



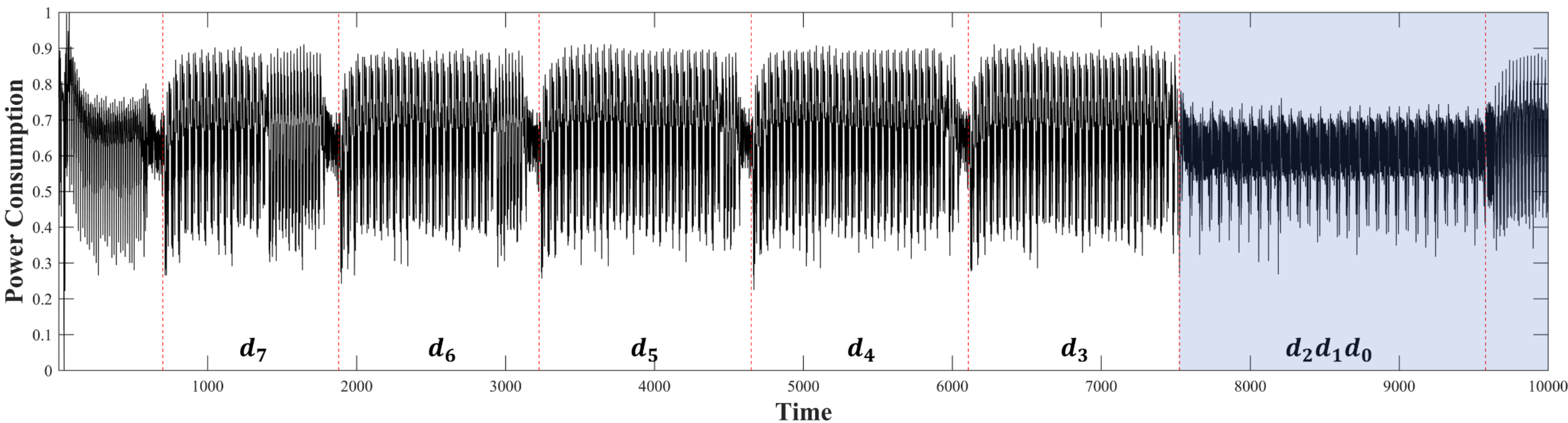
Multiple-Trace Attack on the Bit Rotation

$d = (d_7d_6d_5d_4d_3\underbrace{d_2d_1d_0}_{\text{target}})_2, d_i \in \{0,1\}$

8-bit word

$$result = (\ll_{(8-L)}) | (\gg_L)$$

$$0 \le L = (d_2d_1d_0)_2 < 8$$



Multiple-Trace Attack on the Bit Rotation

$$d = (\underbrace{d_7 d_6 d_5 d_4 d_3}_{}\overbrace{d_2 d_1 d_0}^{\text{target}})_2, d_i \in \{0, 1\}$$

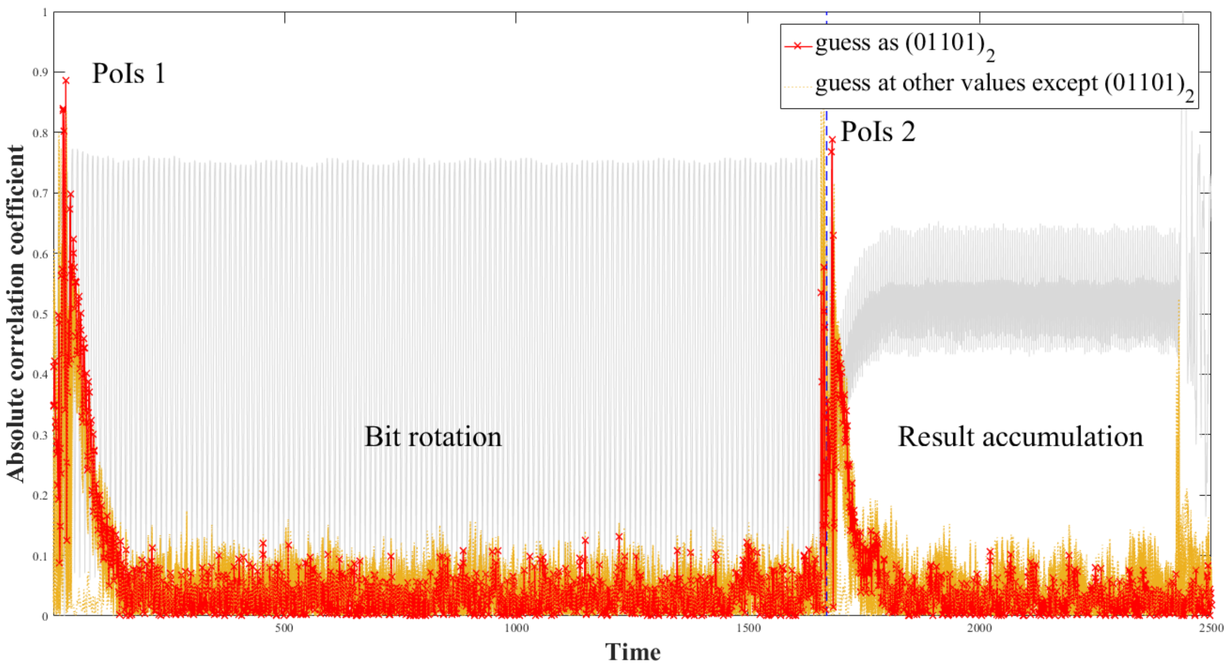
8-bit word

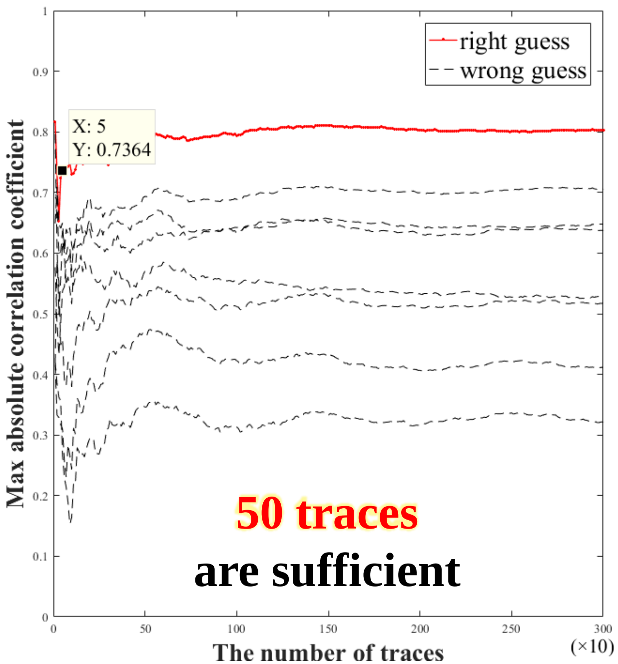
$$\boldsymbol{result} = (\ll_{(8-L)})|(\gg_L)$$

$$0 \le L = (d_2 d_1 d_0)_2 < 8$$

- Guess the L value from 0 to 7

  and calculate Pearson's correlation coefficient between traces and $\boldsymbol{result}$ values





50 traces are sufficient

▣ Multiple-Trace Attack on Constant-Time Multiplication

$$d = (\underbrace{d_7 d_6 d_5 d_4 d_3}_{\text{Correlation Occurring Position}} \underbrace{d_2 d_1 d_0}_{\text{Correlation Power Analysis}})_2$$

We can accurately recover all secret indices regardless of word size and security level

(We described the experiment results on a 32-bit processor in Appendix B)

Multiple-Trace Attack on Constant-Time Multiplication

Syndrome computation $\boldsymbol{H} \cdot \boldsymbol{c}^{\top}$

2014 Timing Attack

(Simple Power Analysis)

TA Countermeasure

2016 Constant-Time Implementation

2017 Differential Power Analysis on Constant-Time Implementation

Limitation: It could not completely recover accurate secret indices, requiring further solving linear equations to obtain entire secret information

	8-bit	16-bit	32-bit	64-bit
80-bit security	0.4 seconds	15 seconds	16 hours	**≈ 530 years**
128-bit security	2 seconds	4 minutes	≈ 7 days	**≈ 790,000 years**

It is not feasible on 64-bit processor

In this paper

Enhanced Multiple-Trace Attack which can accurately recover secret indices regardless of word size and security level

Single-Trace Attack on Constant-Time Multiplication

$$mask = \begin{cases} 0x00 & ,if\ d_i = 0 \\ 0xff & ,if\ d_i = 1 \end{cases}$$

8-bit word

$$d = (\underbrace{d_7 d_6 d_5 d_4 d_3}_{\text{Key Bit-dependent Attack}} \underbrace{d_2 d_1 d_0}_{\text{Simple Power Analysis}})_2$$

Word unit rotation

$$result = \begin{cases} unrotated & ,if\ d_i = 0 \\ rotated & ,if\ d_i = 1 \end{cases}$$

$$result = \begin{cases} (rotated\ \&\ 0x00) \oplus (unrotated\ \&\ 0xff) = unrotated & ,if\ d_i = 0 \\ (rotated\ \&\ 0xff) \oplus (unrotated\ \&\ 0x00) = rotated & ,if\ d_i = 1 \end{cases}$$

Bit rotation

$$result = (\ll_{8-L}) | (\gg_{L})$$

$$0 \le L = (d_2 d_1 d_0)_2 < 8$$

Single-Trace Attack on the Word Unit Rotation

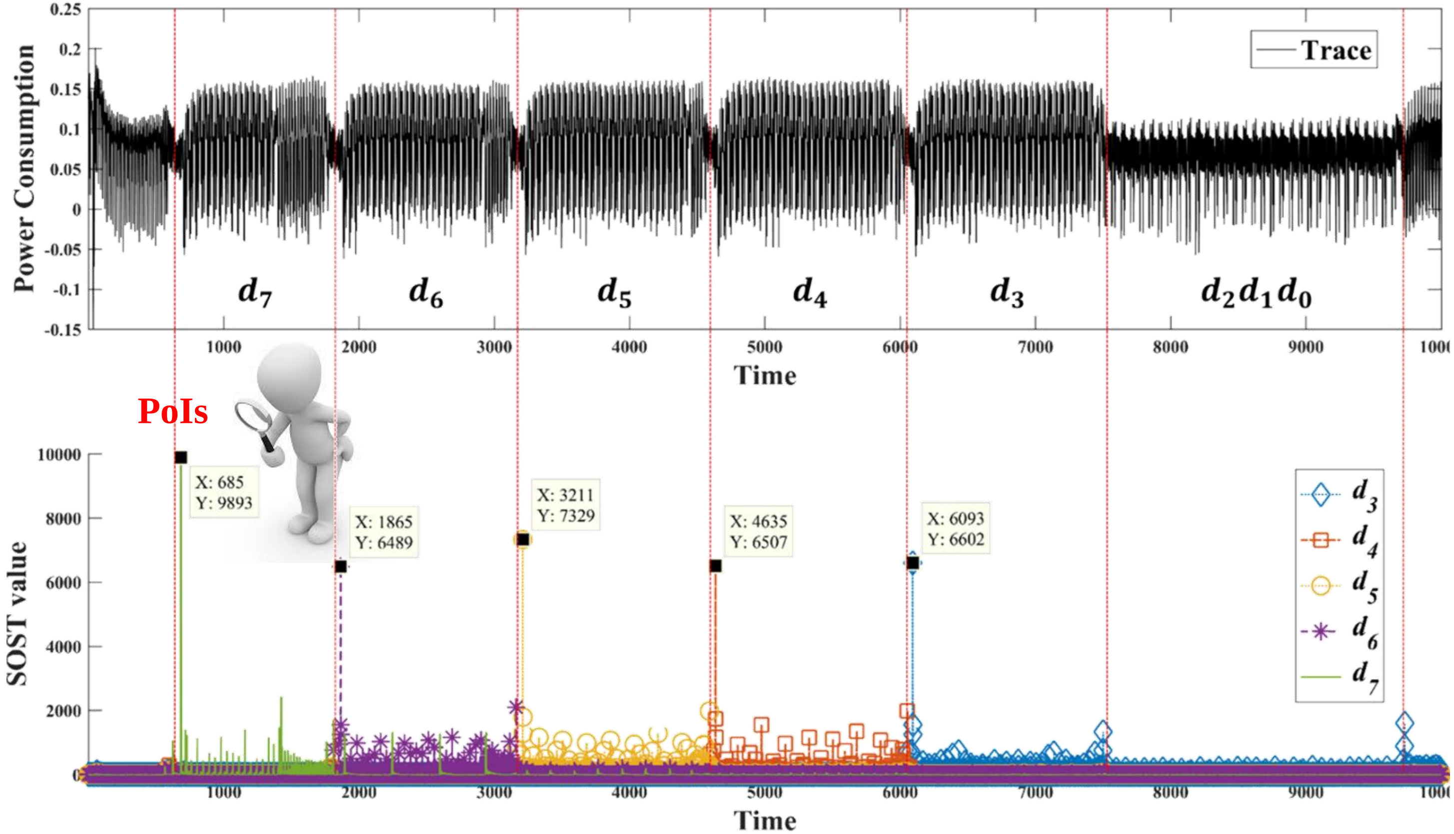
$$d = (\underbrace{d_7 d_6 d_5 d_4 d_3}_{\text{target}} d_2 d_1 d_0)_2, d_i \in \{0,1\}$$

Property 3.

$$result = \begin{cases} unrotated & , if\ d_i = 0 \\ rotated & , if\ d_i = 1 \end{cases}$$

8-bit word

$$result = \begin{cases} (rotated \,\&\, 0x00) \oplus (unrotated \,\&\, 0xff) = unrotated & , if\ d_i = 0 \\ (rotated \,\&\, 0xff) \oplus (unrotated \,\&\, 0x00) = rotated & , if\ d_i = 1 \end{cases}$$



Single-Trace Attack on the Word Unit Rotation

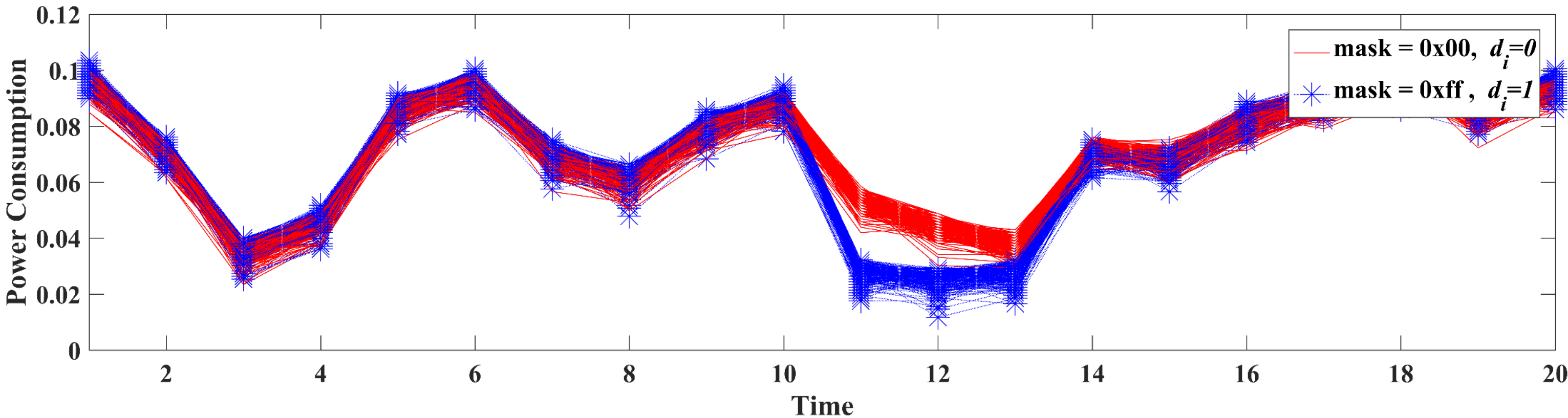
$d = (\underbrace{d_7 d_6 d_5 d_4 d_3 d_2 d_1 d_0}_{\text{target}})_2, d_i \in \{0,1\}$

- $d = (d_7 d_6 d_5 d_4 d_3 d_2 d_1 d_0)_2, d_i \in \{0,1\}$: 675 ~ 695 points

8-bit word

$$result = \begin{cases} (rotated \,\&\, 0x00) \oplus (unrotated \,\&\, 0xff) = unrotated & , if\ d_i = 0 \\ (rotated \,\&\, 0xff) \oplus (unrotated \,\&\, 0x00) = rotated & , if\ d_i = 1 \end{cases}$$

$mask$ $\neg mask$



Key Bit-dependent Property

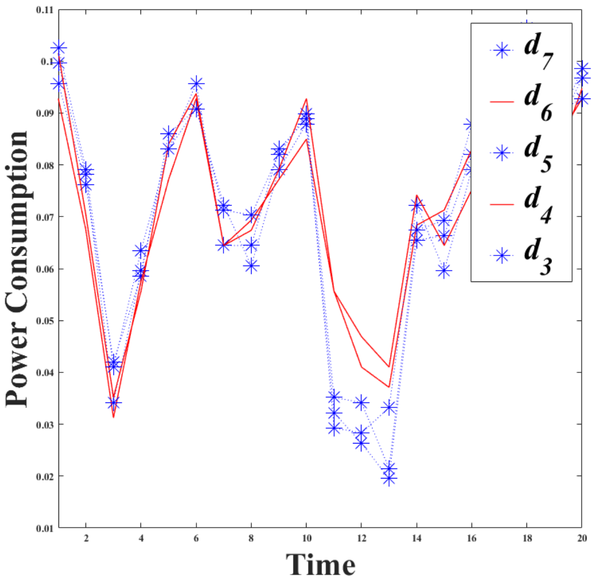
Single-Trace Attack on the Word Unit Rotation

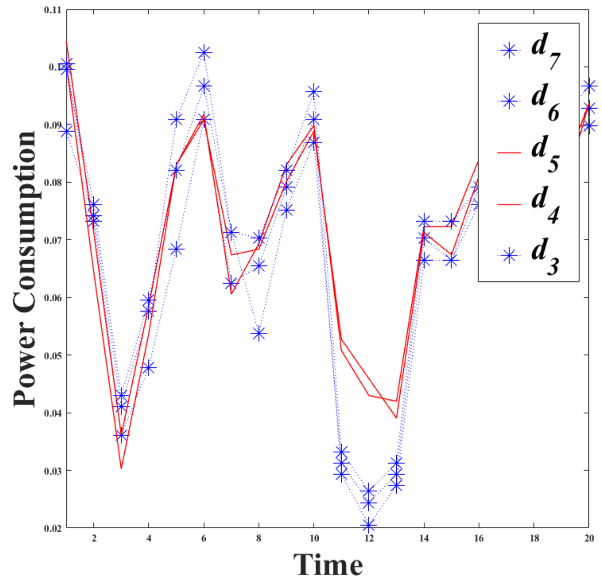
$d = (\underbrace{d_7 d_6 d_5 d_4 d_3 d_2 d_1 d_0}_{\text{target}})_2, d_i \in \{0,1\}$

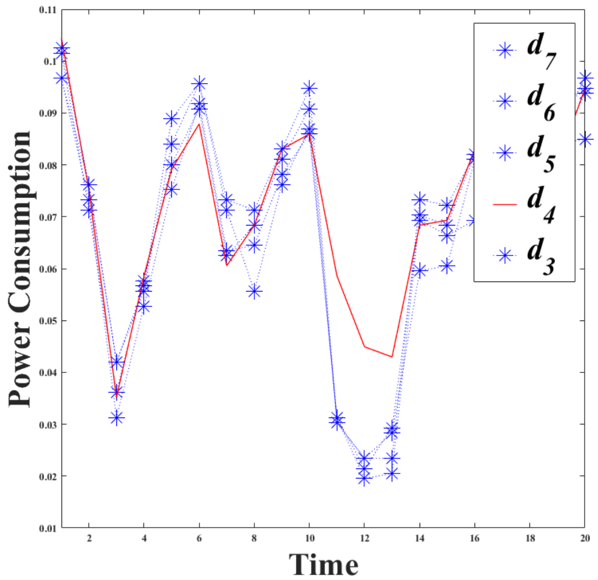
- $d = (d_7 d_6 d_5 d_4 d_3 d_2 d_1 d_0)_2, d_i \in \{0,1\}$

8-bit word

$169 = (10101001)_2$ $201 = (11001001)_2$ $233 = (11101001)_2$







✓ $W = 8$

$$mask = \begin{cases} 0x00 & , if\ d_i = 0 \\ 0xff & , if\ d_i = 1 \end{cases}$$

- K-means clustering
- Fuzzy k-means clustering
- EM (Expectation-maximization)

Single-Trace Attack on the Bit Rotation

$$\boldsymbol{d} = (\underbrace{d_7 d_6 d_5 d_4 d_3}\underbrace{d_2 d_1 d_0}_{\text{target}})_2, d_i \in \{0, 1\}$$

8-bit word

$$\boldsymbol{result} = (\ll_{8-L})|(\gg_{L})$$

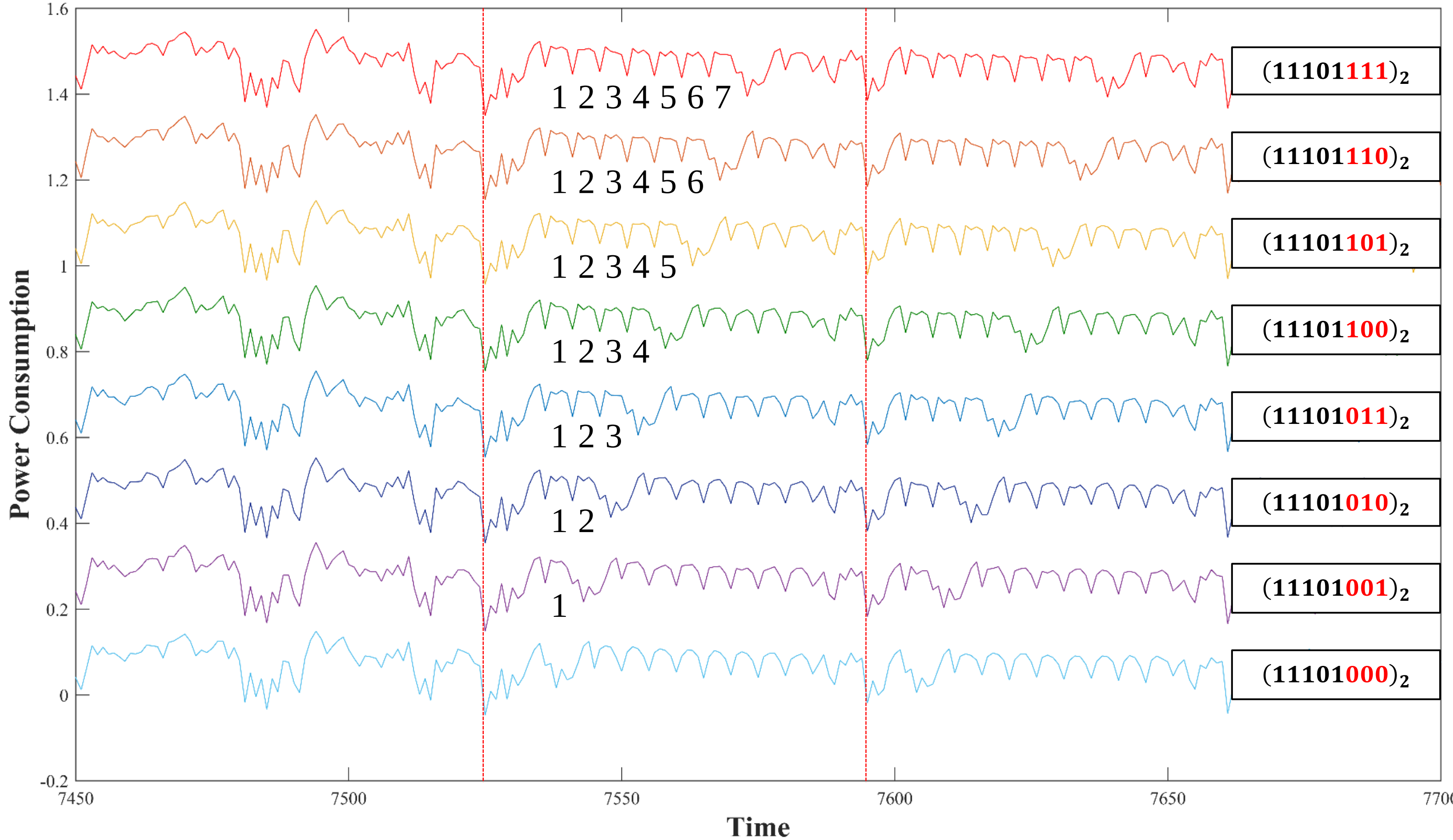
$$0 \le \boldsymbol{L = (d_2 d_1 d_0)_2} < 8$$

	Bit rotate	Left shift	Right shift	SPA
AVR 8-bit word	Single bit shift instructions	$(8 - L)$ times ($(8 - L)$ clock cycles)	L times (L clock cycles)	O
MSP 16-bit word	Single bit shift instructions	$(8 - L)$ times ($(8 - L)$ clock cycles)	L times (L clock cycles)	O

Single-Trace Attack on the Bit Rotation

$$\boldsymbol{d} = (\overbrace{d_7 d_6 d_5 d_4 d_3}\ \overbrace{d_2 d_1 d_0}^{\text{target}})_2, d_i \in \{0,1\}$$

8-bit word



Single-Trace Attack on the Bit Rotation

target

$d = (\underbrace{d_7 d_6 d_5 d_4 d_3}\underbrace{d_2 d_1 d_0}_{\text{target}})_2, d_i \in \{0,1\}$

8-bit word

$$result = (\ll_{8-L})|(\gg_L)$$

$$0 \leq L = (d_2 d_1 d_0)_2 < 8$$

	Bit rotate	Left shift	Right shift	SPA
AVR 8-bit word	Single bit shift instructions	$(8 - L)$ times ($(8 - L)$ clock cycles)	L times (L clock cycles)	O
MSP 16-bit word	Single bit shift instructions	$(8 - L)$ times ($(8 - L)$ clock cycles)	L times (L clock cycles)	O
ARM 32-bit word	Multiple bit shift instructions (ex. barrel shifter)	One clock	One clock	X
64-bit word	Multiple bit shift instructions (ex. barrel shifter)	One clock	One clock	X

✓ In the cases of 32-bit and 64-bit, we need to solve linear equations to find accurate indices

▣ Single-Trace Attack on Constant-Time Multiplication

$$\boldsymbol{d} = (\underbrace{d_7 d_6 d_5 d_4 d_3}_{\text{Key Bit-dependent Attack}} \underbrace{d_2 d_1 d_0}_{\text{Simple Power Analysis}})_2$$

We can accurately recover all secret indices

if processor provides single bit shift instructions

Even if processor does not provide single bit shift instructions,

we can extract substantial parts of secret indices

(We described the experiment results on a 32-bit processor in Section 5 and Appendix B)

Case Study: NIST Round 2 Code-Based Cryptography

Code

BIKE

Classic McElice

RQC

HQC

LEDAcrypt

ROLLO

- **BIKE**
  - **QC-MDPC**
- **LEDAcrypt**
  - **QC-LDPC**

BIKE
LEDAcrypt

TA + **Constant-Time Multiplication**

DPA + **Codeword Randomization (Masking)**

DPA + **Use the ephemeral key pairs**

Case Study: NIST Round 2 Code-Based Cryptography

- **BIKE**
  - **QC-MDPC**

Table : Keys and syndromes of BIKE

	Public key		Private key	Syndrome
BIKE-1	$F = [F_0 \mid F_1]$	$F_0 = G \cdot H_0$ $F_1 = G \cdot H_1$		$H c^\intercal$
BIKE-2	$F = [F_0 \mid F_1]$	$F_0 = I_r$ $F_1 = H_1 \cdot H_0^{-1}$	H	$H_0 c^\intercal$
BIKE-3	$F = [F_0 \mid F_1]$	$F_0 = G \cdot H_0 + H_1$ $F_1 = G$		$c_0^\intercal + H_0 c_1^\intercal$

$*$ I_r is an $r \times r$ identity matrix
$*$ G is an $r \times r$ dense circulant matrix
$*$ H_i is an $r \times r$ sparse circulant matrix, $H = [H_0 \mid H_1]$
$*$ c is a received row vector, $c = [c_0 \mid c_1]$

- **LEDAcrypt**
  - **QC-LDPC**

Table : Keys and syndromes of LEDAcrypt

	Public key	Private key	Syndrome
LEDAcrypt KEM	$P = [M \mid I_r] = L_{n_0-1}^{-1} L$	H, Q	$L_{n_0-1} c^\intercal$
LEDAcrypt PKC	$P = [Z \mid [M_0 \mid \cdots \mid M_{n_0-2}]^\intercal]$		$(HQ) c^\intercal$

$*$ I_r is an $r \times r$ identity matrix
$*$ Z is a diagonal block matrix with $n_0 - 1$ replicas of the block I_r
$*$ M_i is an $r \times r$ dense circulant matrix, $0 \leq i < n_0 - 1$, $M = [M_0 \mid \cdots \mid M_{n_0-2}]$
$*$ Q is an $n \times n$ sparse circulant matrix composed of $n_0 \times n_0$ sparse circulant blocks
$*$ H_i is an $r \times r$ sparse circulant matrix, $0 \leq i \leq n_0 - 1$, $H = [H_0 \mid \cdots \mid H_{n_0-1}]$
$*$ L_i is an $r \times r$ sparse circulant matrix, $0 \leq i \leq n_0 - 1$, $L = HQ$
$*$ c is a received row vector, $c = [c_0 \mid \cdots \mid c_{n-1}]$

Conclusion

Enhanced Multiple-Trace Attack on QC Code-Based Cryptography
Using Constant-Time Multiplication

Novel Single-Trace Attack on QC Code-Based Cryptography
Using Masked Constant-Time Multiplication

BIKE
LEDAcrypt

TA + **Constant-Time Multiplication** — It is insecure against our **multiple-trace attack**

DPA + **Codeword Randomization (Masking)** — It is insecure against our **single-trace attack**

DPA + **Use the ephemeral key pairs** — It is insecure against our **single-trace attack**