Cryptography and Embedded System Security CRAESS_I

Xiaolu Hou

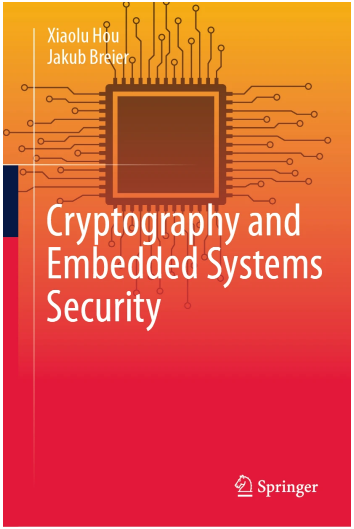
FIIT, STU
xiaolu.hou @ stuba.sk

Course Outline

- Abstract algebra and number theory
- Introduction to cryptography
- Symmetric block ciphers and their implementations
- RSA, RSA signatures, and their implementations
- Probability theory and introduction to SCA
- SPA and non-profiled DPA
- Profiled DPA
- SCA countermeasures
- FA on RSA and countermeasures
- FA on symmetric block ciphers
- FA countermeasures for symmetric block cipher
- Practical aspects of physical attacks
  - Invited speaker: Dr. Jakub Breier, Senior security manager, TTControl GmbH

Recommended reading

- Textbook
    - Sections 5.2



Lecture Outline

- Encoding-based Countermeasure for PRESENT

- Infective Countermeasure

Countermeasures

- Protocol level
  - Design the usage of cryptographic primitives in a way that certain fault attacks are not possible anymore
  - Re-keying[1]
- Cryptographic primitive level
  - Proposal of new cipher design[2]
- Implementation level
  - Infective countermeasure
  - Redundancy
    - Repeat the computation, e.g. deploy the circuit more than once
    - Using error detecting/correcting codes

[1] Medwed, M., Standaert, F. X., Großschädl, J., & Regazzoni, F. (2010, May). Fresh re-keying: Security against side-channel and fault attacks for low-cost devices. In International Conference on Cryptology in Africa (pp. 279-296). Springer, Berlin, Heidelberg.

[2] Baksi, A., Bhasin, S., Breier, J., Khairallah, M., Peyrin, T., Sarkar, S., & Sim, S. M. (2021, December). DEFAULT: Cipher Level Resistance Against Differential Fault Attack. In International Conference on the Theory and Application of Cryptology and Information Security (pp. 124-156). Springer, Cham.

Countermeasures

- Hardware level
  - Light sensors to detect the opening of chip
  - Voltage/temperature sensors to detect fault injections by voltage glitches or temperature variations[1]
  - On the other hand, there are new ways to induce faults proposed all the time. The main focus in academics is more on countermeasures that aim at managing the effect of fault induction.

[1] Hutter, M., & Schmidt, J. M. (2013, November). The temperature side channel and heating fault attacks. In International Conference on Smart Card Research and Advanced Applications (pp. 219-235). Springer, Cham.

FA countermeasures for symmetric block cipher

- Encoding-based Countermeasure for PRESENT
- Infective Countermeasure

Error-detecting code as countermeasure

- A binary code with minimum distance $\mathrm{dis}(C)$ can detect $\mathrm{dis}(C) - 1$ bit flips
- A natural choice for fault countermeasure is to consider encoding the intermediate values during the computation.
- Which code to choose and how to implement it?
- We will discuss one proposal of using anticode for the countermeasure against bit flips and instruction skips[1].
- See the original paper for
  - Formalization of encoding-based countermeasure for symmetric block ciphers
  - Calculattion of the probability for detecting any $m-$bit flip and for instruction skips

[1] Breier, J., Hou, X., & Liu, Y. (2019). On evaluating fault resilient encoding schemes in software. IEEE Transactions on Dependable and Secure Computing, 18(3), 1065-1079.

Rational for using anticode

Definition (Kerckhoffs' principle)

The security of a cryptosystem should depend only on the secrecy of the key.

- We assume the code used for the countermeasure is public – the attacker has the knowledge of all the codewords and how the information is encoded.
- Intuitively if the minimum distance of the code is too small, we know that the code cannot detect a large number of bit flips.
- On the other hand, let us consider a code of length n and size M that contains at least two codewords, say $\boldsymbol{c}_1, \boldsymbol{c}_2$, with distance n.
- If a $n-$bit flip is injected when $\boldsymbol{c}_1$ or $\boldsymbol{c}_2$ is used for the computation, then the resulting faulty value is still a codeword and cannot be detected.
- Since there are in total M codewords, the possibility for the fault to go undetected is $2/M$.
- Thus, a very big maximum distance is also not desirable.

Encoding countermeasure for PRESENT

- Each operation is implemented as a lookup table from memory.
- Before the table lookup, the destination register of an operation is precharged to $\mathbf{0}$.
- When any of the inputs is $\mathbf{0}$, the output is $\mathbf{0}$.
- When an error is detected, the output is $\mathbf{0}$ (error message).
- We assume the registers are precharged to $\mathbf{0}$ before the program starts and this process cannot be faulted.

Such a design can protect the implementation from single instruction skips – e.g. A single instruction skip of any instruction of the following algorithm will either make no change to the output or result in outputting $\mathbf{0}$ (error message).

```
LDI r0 a// load input a
LDI r1 b// load input b
EOR r2 r2// precharge register r2 to zero
LPM r2 r0 r1// execution of an operation by table lookup
```

Error message cannot be a codeword

- Do not contain $\mathbf{0}$ as a codeword.
- Of course, the error message can be changed to a different value as long as it is not a codeword and has the same bit length as the codewords

Detected and undetected faults

- In case the fault changes some encoded intermediate value to a word that is not a codeword, the table lookup will produce $\mathbf{0}$, which indicates an error.
- In the subsequent instructions, when the input of a table is $\mathbf{0}$, the output will always be $\mathbf{0}$ since $\mathbf{0}$ is not a codeword.
- In such cases, we say that the fault is *detected*.
- Otherwise, when a successful fault injection does not result in $\mathbf{0}$ output, we say the fault is *undetected*.

Lookup table for an operation – Example

Example

- As a simple example, let us consider $\{ 01, 10 \}$, a binary $(2, 2, 2, 2)-$anticode.
- Since there are two codewords, it can be used to encode one bit of information.
- Let $0 \mapsto 01$, $1 \mapsto 10$
- The lookup table for carrying out XOR between a, b ($a, b \in \mathbb{F}_2$) is shown in the following table.
- The table outputs 00 (error message) if one input is not a codeword

	00	01	10	11
00	00	00	00	00
01	00	01	10	00
10	00	10	01	00
11	00	00	00	00

Fault in the inputs – Example

Example

	00	01	10	11
00	00	00	00	00
01	00	01	10	00
10	00	10	01	00
11	00	00	00	00

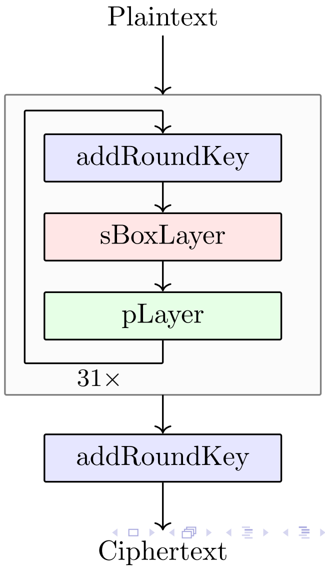
- $1-$bit flip will be detected:
  - If $1-$bit flip is injected in input 01, we get either 00 or 11, both will give output 00.
  - If $1-$bit flip is injected in input 10, we will have 00 or 11 and output will again be 00.
- A $2-$bit flip will be undetected.
  - Suppose we would like to compute $0 \oplus 0$
  - Inputs for the table lookup will be 01 and 01, the output will be 01
  - If a $2-$bit flip is injected in one of the inputs, we get 10 and 01 for table lookup and the result will be 10

PRESENT – encryption

- Block length: 64
- Number of rounds: 31
- Key length: either 80 or 128.
- Round function: addRoundKey, sBoxLayer, and pLayer.
- After 31 rounds, addRoundKey is applied again before the ciphertext output

Remark

For PRESENT specification, we consider the 0th bit of a value as the rightmost bit in its binary representation. For example, the 0th bit of $3 = 011_2$ is 1, the 1st bit is 1 and the 2nd bit is 0.



PRESENT – addRoundKey

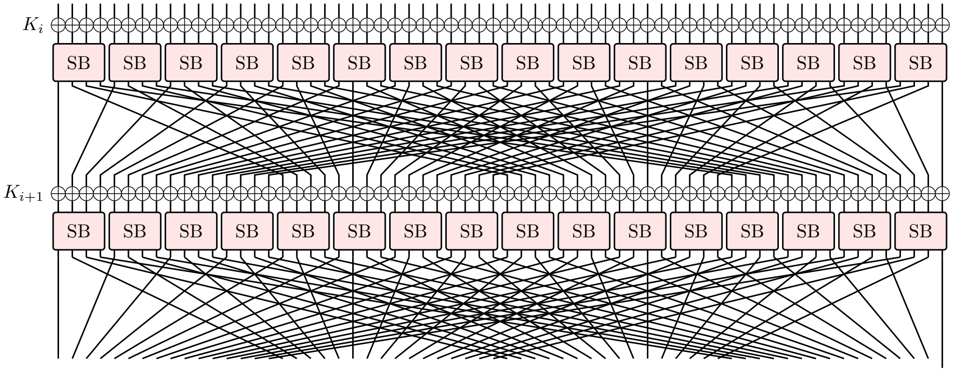
- Round key $K_i = \kappa^i_{63} \dots \kappa^i_0$, $(1 \le i \le 32)$
- Current state $b_{63} b_{62} \dots b_0$
- For $0 \le j \le 63$

$$b_j \to b_j \oplus \kappa^i_j$$

PRESENT – sBoxLayer

- sBoxLayer applies sixteen $4-$bit Sboxes to each nibble of the current cipher state.
- For example, if the input is `0`, the output is `C`.

0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
C	5	6	B	9	0	A	D	3	E	F	8	4	7	1	2



PRESENT – pLayer

pLayer permutes the 64 bits using the following formula:

$$\text{pLayer}(j) = \left\lfloor \frac{j}{4} \right\rfloor + (j \bmod 4) \times 16,$$

where j denotes the bit position.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	16	32	48	1	17	33	49	2	18	34	50	3	19	35	51
16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
4	20	36	52	5	21	37	53	6	22	38	54	7	23	39	55
32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
8	24	40	56	9	25	41	57	10	26	42	58	11	27	43	59
48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
12	28	44	60	13	29	45	61	14	30	46	62	15	31	47	63

Quotient group and Remainder group

- Before we go into details of the countermeasure implementation, we introduce the notion of Quotient group and Remainder group.
- We number the Sboxes in the ith round of PRESENT as $\mathsf{SB}_0^i, \mathsf{SB}_1^i, \ldots, \mathsf{SB}_{15}^i$, where SB_0^i is the right most Sbox
- Those Sboxes can be grouped in 2 different ways: the *Quotient group* and the *Remainder group*:

$$Qj^i := \left\{ \mathsf{SB}_{4j}^i, \mathsf{SB}_{4j+1}^i, \mathsf{SB}_{4j+2}^i, \mathsf{SB}_{4j+3}^i \right\}, \quad Rj^i := \left\{ \mathsf{SB}_{j}^i, \mathsf{SB}_{j+4}^i, \mathsf{SB}_{j+8}^i, \mathsf{SB}_{j+12}^i \right\},$$

where $j = 0, 1, 2, 3$.

- Such a grouping allows us to relate the bits for each Sbox output in round i to bits of each Sbox input in round $i + 1$ in a certain way through pLayer

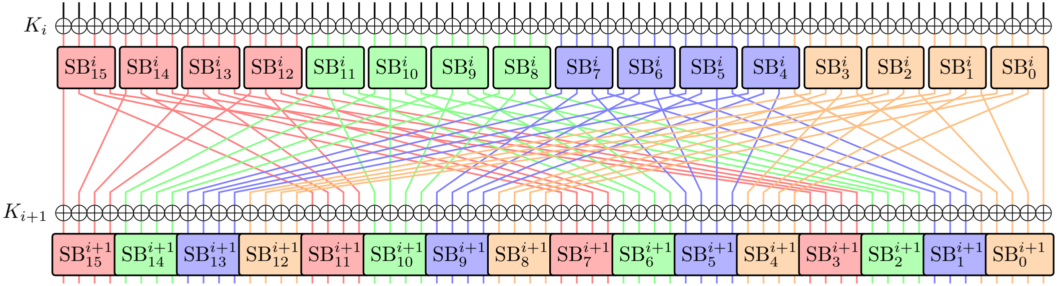
Quotient group and Remainder group

Rj^{i+1} \ Qj^i	SB$^i_{4j}$	SB$^i_{4j+1}$	SB$^i_{4j+2}$	SB$^i_{4j+3}$
SB$^{i+1}_{j}$	$(0,0)$	$(1,0)$	$(2,0)$	$(3,0)$
SB$^{i+1}_{j+4}$	$(0,1)$	$(1,1)$	$(2,1)$	$(3,1)$
SB$^{i+1}_{j+8}$	$(0,2)$	$(1,2)$	$(2,2)$	$(3,2)$
SB$^{i+1}_{j+12}$	$(0,3)$	$(1,3)$	$(2,3)$	$(3,3)$

Table: Relation between the output bits of Sboxes from the Quotient group Qj^i and the input bits of Sboxes from the corresponding Remainder group Rj^{i+1}. For example, the 0th input bit of SB$^{i+1}_{j+4}$ in Rj^{i+1} comes from the 1st output bit of SB$^i_{4j}$ in Qj^i..

- Bits of the 0th Sbox (SB$^i_{4j}$) output in Quotient group Qj^i are permuted to the 0th bits of Sbox inputs in the corresponding Remainder group Rj^{i+1};
- Bits of the 1st Sbox (SB$^i_{4j+1}$) output in Qj^i are permuted to the 1st bits of Sbox inputs in Rj^{i+1};

Quotient group and Remainder group



K_i
SB^i_{15} SB^i_{14} SB^i_{13} SB^i_{12} SB^i_{11} SB^i_{10} SB^i_{9} SB^i_{8} SB^i_{7} SB^i_{6} SB^i_{5} SB^i_{4} SB^i_{3} SB^i_{2} SB^i_{1} SB^i_{0}
K_{i+1}
SB^{i+1}_{15} SB^{i+1}_{14} SB^{i+1}_{13} SB^{i+1}_{12} SB^{i+1}_{11} SB^{i+1}_{10} SB^{i+1}_{9} SB^{i+1}_{8} SB^{i+1}_{7} SB^{i+1}_{6} SB^{i+1}_{5} SB^{i+1}_{4} SB^{i+1}_{3} SB^{i+1}_{2} SB^{i+1}_{1} SB^{i+1}_{0}

Figure: An illustration of the relation between Sbox outputs in a Quotient group to Sbox inputs in the corresponding Remainder group. Sboxes in Quotient groups $Q0^i$, $Q1^i$, $Q2^i$, $Q3^i$ and their corresponding Remainder groups $R0^{i+1}$, $R1^{i+1}$, $R2^{i+1}$, $R3^{i+1}$ are in orange, blue, green, red colors respectively.

pLayer can be considered as four identical parallel bitwise operations where each is a function: $\mathbb{F}_2^{16} \to \mathbb{F}_2^{16}$ that takes one Quotient group output and permutes it to the corresponding Remainder group input.

Choice of the anticode

- pLayer can be considered as four identical parallel bitwise operations where each is a function: $\mathbb{F}_2^{16} \to \mathbb{F}_2^{16}$
- addRoundKey is a function: $\mathbb{F}_2^{64} \to \mathbb{F}_2^{64}$.
- Present Sbox SB: $\mathbb{F}_2^{4} \to \mathbb{F}_2^{4}$.
- One convenient code choice would be those with cardinalities 16, encoding 4 bits of information.
- In particular, we are looking for a binary $(n, 16, d, \delta)-$anticode, where d is the minimum distance of the code and δ is the maximum distance of the code.

Choice of the anticode

- See the original paper[1] for an algorithm for finding anticodes that achieve a low probability of undetected faults with given length, minimum distance, and maximum distance.
- In the rest of this presentation, we will use the following binary $(8, 16, 2, 7)-$anticode as a running example

$$\{\texttt{01, 08, 02, 0B, 04, 1D, 1E, 30, 7, 65, 6A, AD, B3, CE, D9, F6}\}.$$

- 01 is the codeword for 0000, 08 in the codeword for 0001, etc.
- We write

$$\texttt{01} = \texttt{encode}(0000).$$

[1]Breier, J., Hou, X., & Liu, Y. (2019). On evaluating fault resilient encoding schemes in software. IEEE Transactions on Dependable and Secure Computing, 18(3), 1065-1079.

Encoding countermeasure - addRoundKey

- Given an anticode C, the addRoundKey operation can be implemented using an `XOR` table similar to the one we have seen before
- The table has 2^8 rows and 2^8 columns.
- Let $\widetilde{\oplus}$ denote this table lookup operation.

Example (The previous example)

- $\{\,01, 10\,\}$, a binary $(2,2,2,2)-$anticode.
- $0 \mapsto 01,\ 1 \mapsto 10$
- The lookup table for carrying out `XOR` between a, b $(a, b \in \mathbb{F}_2)$ is shown in the following table.

	00	01	10	11
00	00	00	00	00
01	00	01	10	00
10	00	10	01	00
11	00	00	00	00

Encoding countermeasure - addRoundKey

- Given an anticode C, the addRoundKey operation can be implemented using an `XOR` table similar to the one we have seen before
- The size of the table will be $2^8 \times 2^8$.
- Let $\widetilde{\oplus}$ denote this table lookup operation.

Example

$$\{\texttt{01},\ \texttt{08},\ \texttt{02},\ \texttt{0B},\ \texttt{04},\ \texttt{1D},\ \texttt{1E},\ \texttt{30},\ \texttt{07},\ \texttt{65},\ \texttt{6A},\ \texttt{AD},\ \texttt{B3},\ \texttt{CE},\ \texttt{D9},\ \texttt{F6}\}.$$

The table entry corresponding to 01 and 08 will be ?

Encoding countermeasure - addRoundKey

- Given an anticode C, the addRoundKey operation can be implemented using an `XOR` table similar to the one we have seen before
- The size of the table will be $2^8 \times 2^8$.
- Let $\widetilde{\oplus}$ denote this table lookup operation.

Example

$$\{\,\texttt{01, 08, 02, 0B, 04, 1D, 1E, 30, 07, 65, 6A, AD, B3, CE, D9, F6}\,\}.$$

The table entry corresponding to 01 and 08 will be

$$\texttt{encode}(0000 \oplus 0001) = \texttt{encode}(0001) = \texttt{08}.$$

And we write

$$\texttt{01} \widetilde{\oplus} \texttt{08} = \texttt{08}.$$

Encoding countermeasure - sBoxLayer and pLayer

- The implementation of sBoxLayer and pLayer are based on four 16×64 lookup tables, $T0, T1, T2, T3$.
- Let $\boldsymbol{x} = x_3x_2x_1x_0$ be an element in $\mathbb{F}_2^4$.
- We write

$$\mathsf{SB}(x_3x_2x_1x_0) = x_3^s x_2^s x_1^s x_0^s.$$

0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
C	5	6	B	9	0	A	D	3	E	F	8	4	7	1	2

Example

Take $\mathtt{D} = 1101$, then $x_3 = 1$, $x_2 = 1$, $x_1 = 0$, $x_0 = 1$. Since $\mathsf{SB}(\mathtt{D}) = 7 = 0111$, we have

$$x_3^s = 0,\ x_2^s = 1,\ x_1^s = 1,\ x_0^s = 1.$$

Encoding countermeasure - sBoxLayer and pLayer

$$\begin{aligned} T0 : C &\rightarrow C \times C \times C \times C \\ \mathtt{encode}(x_3x_2x_1x_0) &\mapsto \mathtt{encode}(000x_3^s), \mathtt{encode}(000x_2^s), \mathtt{encode}(000x_1^s), \mathtt{encode}(000x_0^s) \end{aligned}$$

$$\begin{aligned} T1 : C &\rightarrow C \times C \times C \times C \\ \mathtt{encode}(x_3x_2x_1x_0) &\mapsto \mathtt{encode}(00x_3^s0), \mathtt{encode}(00x_2^s0), \mathtt{encode}(00x_1^s0), \mathtt{encode}(00x_0^s0) \end{aligned}$$

$$\begin{aligned} T2 : C &\rightarrow C \times C \times C \times C \\ \mathtt{encode}(x_3x_2x_1x_0) &\mapsto \mathtt{encode}(0x_3^s00), \mathtt{encode}(0x_2^s00), \mathtt{encode}(0x_1^s00), \mathtt{encode}(0x_0^s00) \end{aligned}$$

$$\begin{aligned} T3 : C &\rightarrow C \times C \times C \times C \\ \mathtt{encode}(x_3x_2x_1x_0) &\mapsto \mathtt{encode}(x_3^s000), \mathtt{encode}(x_2^s000), \mathtt{encode}(x_1^s000), \mathtt{encode}(x_0^s000) \end{aligned}$$

Encoding countermeasure - sBoxLayer and pLayer

- Each table extracts the bits of Sbox output, permutes them and outputs the corresponding codeword.
- Each entry of the outputs of each table can be
  - $T0$: $\texttt{encode}(0000)$ or $\texttt{encode}(0001)$
  - $T1$: $\texttt{encode}(0000)$ or $\texttt{encode}(0010)$
  - $T2$: $\texttt{encode}(0000)$ or $\texttt{encode}(0100)$
  - $T3$: $\texttt{encode}(0000)$ or $\texttt{encode}(1000)$

Encoding countermeasure - sBoxLayer and pLayer

$$\begin{aligned} T0 : C &\rightarrow C \times C \times C \times C \\ \texttt{encode}(x_3x_2x_1x_0) &\mapsto \texttt{encode}(000x_3^s), \texttt{encode}(000x_2^s), \texttt{encode}(000x_1^s), \texttt{encode}(000x_0^s) \end{aligned}$$

Example

{ `01, 08, 02, 0B, 04, 1D, 1E, 30, 07, 65, 6A, AD, B3, CE, D9, F6` }.

- Suppose the input is `01` $= \texttt{encode}(0000)$.
- The corresponding Sbox output would be `C` $= 1100$, i.e. $x_3^s x_2^s x_1^s x_0^s = 1100$.
- The output of $T0$ will be ?

Encoding countermeasure - sBoxLayer and pLayer

$$\begin{aligned} T0 : C &\rightarrow C \times C \times C \times C \\ \texttt{encode}(x_3x_2x_1x_0) &\mapsto \texttt{encode}(000x_3^s), \texttt{encode}(000x_2^s), \texttt{encode}(000x_1^s), \texttt{encode}(000x_0^s) \end{aligned}$$

Example

{ `01, 08, 02, 0B, 04, 1D, 1E, 30, 07, 65, 6A, AD, B3, CE, D9, F6` }.

- Suppose the input is `01` $=$ $\texttt{encode}(0000)$.
- The corresponding Sbox output would be `C` $= 1100$, i.e. $x_3^s x_2^s x_1^s x_0^s = 1100$.
- The output of $T0$ will be

$$\begin{aligned} \texttt{encode}(0001) = \texttt{08}, \quad \texttt{encode}(0001) = \texttt{08}, \\ \texttt{encode}(0000) = \texttt{01}, \quad \texttt{encode}(0000) = \texttt{01}. \end{aligned}$$

Encoding countermeasure - sBoxLayer and pLayer

$$\begin{aligned} T1 : C &\rightarrow C \times C \times C \times C \\ \texttt{encode}(x_3x_2x_1x_0) &\mapsto \texttt{encode}(00x_3^s0), \texttt{encode}(00x_2^s0), \texttt{encode}(00x_1^s0), \texttt{encode}(00x_0^s0) \end{aligned}$$

Example

{ 01, 08, 02, 0B, 04, 1D, 1E, 30, 07, 65, 6A, AD, B3, CE, D9, F6 }.

- Suppose the input is 01 $= \texttt{encode}(0000)$.
- The corresponding Sbox output would be C $= 1100$, i.e. $x_3^sx_2^sx_1^sx_0^s = 1100$.
- The output of $T1$ will be ?

Encoding countermeasure - sBoxLayer and pLayer

$$\begin{aligned} T1 : C &\rightarrow C \times C \times C \times C \\ \texttt{encode}(x_3x_2x_1x_0) &\mapsto \texttt{encode}(00x_3^s0), \texttt{encode}(00x_2^s0), \texttt{encode}(00x_1^s0), \texttt{encode}(00x_0^s0) \end{aligned}$$

Example

{ `01`, `08`, `02`, `0B`, `04`, `1D`, `1E`, `30`, `07`, `65`, `6A`, `AD`, `B3`, `CE`, `D9`, `F6` }.

- Suppose the input is `01` $= \texttt{encode}(0000)$.
- The corresponding Sbox output would be `C` $= 1100$, i.e. $x_3^sx_2^sx_1^sx_0^s = 1100$.
- The output of $T1$ will be

$$\begin{aligned} \texttt{encode}(0010) = \texttt{02}, \quad \texttt{encode}(0010) = \texttt{02}, \\ \texttt{encode}(0000) = \texttt{01}, \quad \texttt{encode}(0000) = \texttt{01}. \end{aligned}$$

Encoding countermeasure - sBoxLayer and pLayer

$$\begin{aligned} T2 : C &\rightarrow C \times C \times C \times C \\ \texttt{encode}(x_3x_2x_1x_0) &\mapsto \texttt{encode}(0x_3^s00), \texttt{encode}(0x_2^s00), \texttt{encode}(0x_1^s00), \texttt{encode}(0x_0^s00) \end{aligned}$$

Example

{ `01, 08, 02, 0B, 04, 1D, 1E, 30, 07, 65, 6A, AD, B3, CE, D9, F6` }.

- Suppose the input is `01` $= \texttt{encode}(0000)$.
- The corresponding Sbox output would be `C` $= 1100$, i.e. $x_3^s x_2^s x_1^s x_0^s = 1100$.
- The output of $T2$ will be ?

Encoding countermeasure - sBoxLayer and pLayer

$$\begin{aligned} T2 : C &\to C \times C \times C \times C \\ \texttt{encode}(x_3x_2x_1x_0) &\mapsto \texttt{encode}(0x_3^s00), \texttt{encode}(0x_2^s00), \texttt{encode}(0x_1^s00), \texttt{encode}(0x_0^s00) \end{aligned}$$

Example

{ `01, 08, 02, 0B, 04, 1D, 1E, 30, 07, 65, 6A, AD, B3, CE, D9, F6` }.

- Suppose the input is `01` $= \texttt{encode}(0000)$.
- The corresponding Sbox output would be `C` $= 1100$, i.e. $x_3^s x_2^s x_1^s x_0^s = 1100$.
- The output of $T2$ will be

$$\begin{aligned} \texttt{encode}(0100) = \texttt{04}, \quad \texttt{encode}(0100) = \texttt{04}, \\ \texttt{encode}(0000) = \texttt{01}, \quad \texttt{encode}(0000) = \texttt{01}. \end{aligned}$$

Encoding countermeasure - sBoxLayer and pLayer

$$\begin{aligned} T3 : C &\rightarrow C \times C \times C \times C \\ \texttt{encode}(x_3x_2x_1x_0) &\mapsto \texttt{encode}(x_3^s000), \texttt{encode}(x_2^s000), \texttt{encode}(x_1^s000), \texttt{encode}(x_0^s000) \end{aligned}$$

Example

`{ 01, 08, 02, 0B, 04, 1D, 1E, 30, 07, 65, 6A, AD, B3, CE, D9, F6 }.`

- Suppose the input is `01` $= \texttt{encode}(0000)$.
- The corresponding Sbox output would be `C` $= 1100$, i.e. $x_3^s x_2^s x_1^s x_0^s = 1100$.
- The output of $T3$ will be ?

Encoding countermeasure - sBoxLayer and pLayer

$$\begin{aligned} T3 : C &\to C \times C \times C \times C \\ \texttt{encode}(x_3x_2x_1x_0) &\mapsto \texttt{encode}(x_3^s000), \texttt{encode}(x_2^s000), \texttt{encode}(x_1^s000), \texttt{encode}(x_0^s000) \end{aligned}$$

Example

{ 01, 08, 02, 0B, 04, 1D, 1E, 30, 07, 65, 6A, AD, B3, CE, D9, F6 }.

- Suppose the input is 01 $= \texttt{encode}(0000)$.
- The corresponding Sbox output would be C $= 1100$, i.e. $x_3^s x_2^s x_1^s x_0^s = 1100$.
- The output of $T3$ will be

$$\begin{aligned} \texttt{encode}(1000) &= \texttt{07}, \quad \texttt{encode}(1000) = \texttt{07}, \\ \texttt{encode}(0000) &= \texttt{01}, \quad \texttt{encode}(0000) = \texttt{01}. \end{aligned}$$

Encoding countermeasure - sBoxLayer and pLayer

- Let the original cipher state at sBoxLayer input be $b_{63}b_{62}\dots b_0$.
- For the encoding-based implementation, the corresponding cipher state will be

  $$\texttt{encode}(b_{63}b_{62}b_{61}b_{60})\texttt{encode}(b_{59}b_{58}b_{57}b_{56})\dots\texttt{encode}(b_7b_6b_5b_4)\texttt{encode}(b_3b_2b_1b_0).$$

- Each codeword in this cipher state will be passed to tables $T0, T1, T2, T3$, and the outputs will be recorded.
- Then the output of pLayer will be computed by combining those table outputs through $\widetilde{\bigoplus}$ – lookup table for XOR.

Encoding countermeasure - sBoxLayer and pLayer

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	16	32	48	1	17	33	49	2	18	34	50	3	19	35	51

Example

- The bits at positions $0, 1, 2, 3$ for the output of pLayer come from the bits at positions $0, 4, 8, 12$ of the input of pLayer.
- We first get $\texttt{encode}(000b_0^s)$ from $T0$ output, $\texttt{encode}(00b_4^s0)$ from $T1$, $\texttt{encode}(0b_8^s00)$ from $T2$, $\texttt{encode}(b_{12}^s000)$ from $T3$, then the 0th nibble of pLayer output will be ?

Encoding countermeasure - sBoxLayer and pLayer

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	16	32	48	1	17	33	49	2	18	34	50	3	19	35	51

Example

- The bits at positions $0, 1, 2, 3$ for the output of pLayer come from the bits at positions $0, 4, 8, 12$ of the input of pLayer.
- We first get $\texttt{encode}(000b_0^s)$ from $T0$ output, $\texttt{encode}(00b_4^s0)$ from $T1$, $\texttt{encode}(0b_8^s00)$ from $T2$, $\texttt{encode}(b_{12}^s000)$ from $T3$, then the 0th nibble of pLayer output will be

$$\texttt{encode}(000b_0^s)\widetilde{\oplus}\texttt{encode}(00b_4^s0)\widetilde{\oplus}\texttt{encode}(0b_8^s00)\widetilde{\oplus}\texttt{encode}(b_{12}^s000).$$

- As another example, the 3rd nibble (bits $16, 17, 18, 19$) of pLayer output is given by ?

Encoding countermeasure - sBoxLayer and pLayer

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	16	32	48	1	17	33	49	2	18	34	50	3	19	35	51

Example

- The bits at positions $0, 1, 2, 3$ for the output of pLayer come from the bits at positions $0, 4, 8, 12$ of the input of pLayer.
- We first get $\texttt{encode}(000b_0^s)$ from $T0$ output, $\texttt{encode}(00b_4^s0)$ from $T1$, $\texttt{encode}(0b_8^s00)$ from $T2$, $\texttt{encode}(b_{12}^s000)$ from $T3$, then the 0th nibble of pLayer output will be

$$\texttt{encode}(000b_0^s)\widetilde{\oplus}\texttt{encode}(00b_4^s0)\widetilde{\oplus}\texttt{encode}(0b_8^s00)\widetilde{\oplus}\texttt{encode}(b_{12}^s000).$$

- As another example, the 3rd nibble (bits $16, 17, 18, 19$) of pLayer output is given by

$$\texttt{encode}(000b_1^s)\widetilde{\oplus}\texttt{encode}(00b_5^s0)\widetilde{\oplus}\texttt{encode}(0b_9^s00)\widetilde{\oplus}\texttt{encode}(b_{13}^s000).$$

Effectiveness against fault attacks

- By the design of our implementation, when the faulty intermediate value is not a codeword, the table lookup returns $\mathbf{0}$ and the attacker will not be able to tell what the original faulty ciphertext is.
- Since both DFA and SFA require analysis of the faulty ciphertexts, they can be prevented when the fault model is bit flip and the number of bit flips is lower than the minimum distance of the binary code.

Minimum distance decoding rule

- After receiving $\boldsymbol{x}$, Bob computes

$$\boldsymbol{c}_{\boldsymbol{x}} = \arg\min_{\boldsymbol{c}} \{ \operatorname{dis}(\boldsymbol{x}, \boldsymbol{c}) \mid \boldsymbol{c} \in C \},$$

  i.e.

$$\operatorname{dis}(\boldsymbol{c}_{\boldsymbol{x}}, \boldsymbol{x}) = \min_{\boldsymbol{c}} \{ \operatorname{dis}(\boldsymbol{x}, \boldsymbol{c}) \mid \boldsymbol{c} \in C \}.$$

- If more than one codeword is identified as $\boldsymbol{c}_{\boldsymbol{x}}$, there are two options.
  - Incomplete decoding rule: Bob requests Alice for another transmission.
  - Complete decoding rule: Bob randomly selects one codeword.

Error-correcting code

Definition

A binary code C is said to be $k-$*error correcting* if minimum distance decoding is able to correct k or fewer errors. If C is $k-$error correcting but not $k+1-$error correcting, then we say that C is *exactly* k*-error correcting*.

Example

Let $C = \{ 000, 111 \}$.

- If 000 was sent and 1 bit flip occurred, the received word $\{ 001, 010, 100 \}$ will be decoded to 000.
- If 111 was sent and 1 bit flip occurred, the received word $\{ 110, 011, 101 \}$ will be decoded to 111.
- If 000 was sent and 011 was received, the decoding result will be 111.

Thus C is exactly $1-$error correcting.

Minimum distance and error-correcting

Theorem

- *A binary* $(n, M, d)-$*code is exactly* $\lfloor (d-1)/2 \rfloor-$*error correcting.*

Using error-correction code

$3-$repetition code, $0 \mapsto 000$, $1 \mapsto 111$

	000	001	010	011	100	101	110	111
000	000	000	000	111	000	111	111	111
001	000	000	000	111	000	111	111	111
010	000	000	000	111	000	111	111	111
011	111	111	111	000	111	000	000	000
100	000	000	000	111	000	111	111	111
101	111	111	111	000	111	000	000	000
110	111	111	111	000	111	000	000	000
111	111	111	111	000	111	000	000	000

- Some faults will be corrected to wrong codewords
- Better to only use error-correcting code-based countermeasure when we know at most $\lfloor (d-1)/2 \rfloor$ bits can be flipped, where d is the minimum distance of the binary code.
- In general, need to reserve one word to indicate more than one codeword is at the same smallest distance from the input word

FA countermeasures for symmetric block cipher

- Encoding-based Countermeasure for PRESENT
- Infective Countermeasure

Infective countermeasure

- The idea of infective countermeasure is to process the ciphertext in a way that the output becomes useless for an attacker when faults are injected during the computations.
- We will take the proposal from the following paper and only focus on the case for AES-128.
- Tupsamudre, H., Bisht, S., & Mukhopadhyay, D. (2014). Destroying Fault Invariant with Randomization: A Countermeasure for AES Against Differential Fault Attacks. In Cryptographic Hardware and Embedded Systems–CHES 2014: 16th International Workshop, Busan, South Korea, September 23-26, 2014. Proceedings 16 (pp. 93-111). Springer Berlin Heidelberg.

Main idea

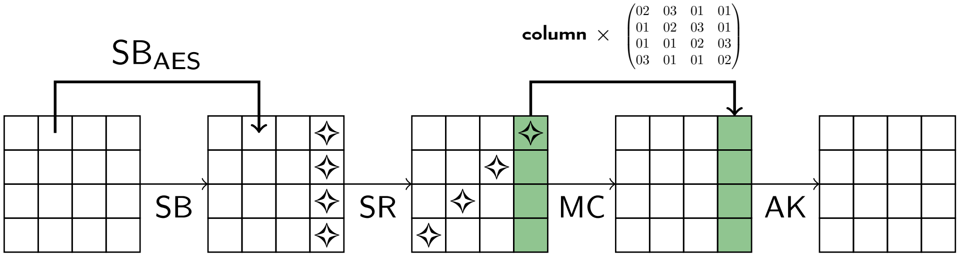
- The main methodology is to compute each round of AES encryption twice before moving to the next round.
- The results of those two rounds will be compared, if a fault is detected, the rest of the computation should produce random values.
- Computations of dummy rounds are also randomly added in between the AES rounds so that the attacker would not know where the fault was actually injected.

AES

- NIST, 1997, Call for algorithms, replacement for DES
- Advanced Encryption Standard
- October 2000, Rijndael was selected
- Invented by Belgian cryptographers Joan Daemen and Vincent Rijmen
- Optimized for software efficiency on 8 and 32 bit processors

AES encryption

- An initial AddRoundKey
- Round function for $\mathtt{Nr}-1$ rounds: SubBytes, ShiftRows, MixColumns, AddRoundKey
- Last round, round $\mathtt{Nr}$: SubBytes, ShiftRows, AddRoundKey
- AddRoundKey is bitwise `XOR` with the round key
- SubBytes is the application of $8-$bit Sboxes.
- ShiftRows permutes the bytes
- MixColumns is a function on $32-$bit values (four bytes).


SB$_{\text{AES}}$
column $\times \begin{pmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{pmatrix}$
SB
SR
MC
AK

Notations

- We define F_i $(i = 0, 1, 2, \ldots, 10)$ as follows:
  - F_0: the initial AddRoundKey operation in AES;
  - For $i = 1, 2, \ldots, 9$, F_i: the AES round function, F_i consists of the following operations: SubBytes, ShiftRows, MixColumns, and AddRoundKey;
  - F_{10}: the AES round function for the last round, consists of SubBytes, ShiftRows, AddRoundKey.
- K_i $(i = 0, 1, 2, \ldots, 10)$: the round keys for AES.
- Each F_i takes input the cipher state at end of round $i - 1$ and K_i, then outputs the cipher state at end of round i.

Notations

- We generate a random number β and the round keys for the dummy rounds, denoted κ_i $(i = 0, 1, 2, \ldots, 10)$, such that
$$F_i(\beta, \kappa_i) = \beta$$
for $i = 0, 1, 2 \ldots, 10$.
- Since F_0 is an AddRoundKey operation,
$$\kappa_0 = \mathsf{0000000000000000}.$$
- For $i = 1, 2, \ldots, 9$,
$$\kappa_i = \beta \oplus \mathsf{MixColumns}(\mathsf{ShiftRows}(\mathsf{SubBytes}(\beta))).$$
- For $i = 10$
$$\kappa_{10} = \beta \oplus \mathsf{ShiftRows}(\mathsf{SubBytes}(\beta)).$$
- We set an array of keys of size 2×11, denoted `keys` as
$$\texttt{keys}[0][i] = \kappa_i, \quad \texttt{keys}[1][i] = K_i.$$

Infective countermeasure for AES-128

- The user-specified number t determines how many dummy rounds will be added during the computation.
- The cipher state for the first computation is stored in R_0 and the cipher state in the redundant AES computation is stored in R_1 – Both are initialized to be the plaintext
- The dummy round state is stored in R_2 and initialized to be the random number β

```
Input: $p$, $\beta$, keys, $t$ // $p$ is a plaintext block; $\beta$ is a random number; keys
    contains the AES round keys $K_i$ and the dummy round keys $\kappa_i$ for
    $i = 0, 1, 2, \ldots, 10$; $t$ is a user-specified security parameter.
Output: ciphertext or infected ciphertext
1 $R_0 = p, \quad R_1 = p, \quad R_2 = \beta$
2 $\ldots$
```

Infective countermeasure for AES-128

- j counts the total number (including the redundant ones) of AES rounds computed and $i = \lfloor j/2 \rfloor$ is the actual round counter.
- The random string `rstr` contains 22 of 1s corresponding to two computations of each F_i for $i = 0, 1, \ldots, 10$ and $t - 22$ bits of 0 corresponding to dummy rounds.

```
Input: $p$, $\beta$, keys, $t$
Output: ciphertext or infected ciphertext
$R_0 = p, \quad R_1 = p, \quad R_2 = \beta$
Generate rstr$\in \mathbb{F}_2^t$
$j = 0, \quad \texttt{idx} = 1$
while $idx \leq t$ do
    $i = \lfloor j/2 \rfloor$// $i$ is the round counter
    $\lambda = \texttt{rstr}[\texttt{idx}]$// $\lambda = 0$ implies a dummy round
    $\ldots$
    $j = j + \lambda$
    $\ldots$
return $R_0$
```

Infective countermeasure for AES-128

- The random string rstr contains 22 of 1s corresponding to two computations of each F_i for $i = 0, 1, \ldots, 10$ and $t - 22$ bits of 0 corresponding to dummy rounds.
- In each loop, we go through the idx-th bit of rstr, and the value is stored in λ.
- The value of idx is increased by 1 at the end of each loop

```
Input: $p$, $\beta$, keys, $t$
Output: ciphertext or infected ciphertext
1  $R_0 = p, \quad R_1 = p, \quad R_2 = \beta$
2  Generate rstr$\in \mathbb{F}_2^t$
3  $j = 0, \quad \texttt{idx} = 1$
4  while $idx \leq t$ do
5      $i = \lfloor j/2 \rfloor$
6      $\lambda = \texttt{rstr}[\texttt{idx}]$
7      $\ldots$
8      $j = j + \lambda$
9      $\texttt{idx} = \texttt{idx} + 1$
10 return $R_0$
```

Infective countermeasure for AES-128

- If j is even (resp. odd), the least significant bit (LSB) of j is 0 (resp. 1),

$$a = ((\text{LSB of } j)\ \&\ \lambda) \oplus 2(\neg\lambda) = \begin{cases} 0 \oplus 2 = 2, & \text{if } \lambda = 0 \\ (0\ \&\ 1) \oplus 0 = 0, & \text{if } \lambda = 1 \text{ and } j \text{ is even} \\ (1\ \&\ 1) \oplus 0 = 1, & \text{if } \lambda = 1 \text{ and } j \text{ is odd} \end{cases}.$$

- $R_a = F_i(R_a, \texttt{keys}[\lambda][i])$
- When $\lambda = 0$, $a = 2$, we compute a dummy round i with

$$R_2 = F_i(R_2, \texttt{keys}[0][i]) = F_i(R_2, \kappa_i).$$

- When $\lambda = 1$ and j is even, $a = 0$, we compute AES round i with

$$R_0 = F_i(R_0, \texttt{keys}[1][i]) = F_i(R_0, K_i).$$

- When $\lambda = 1$ and j is odd, $a = 1$, we compute a redundant AES round i with

$$R_1 = F_i(R_1, \texttt{keys}[1][i]) = F_i(R_1, K_i).$$

Infective countermeasure for AES-128

When $\lambda = 0$, $a = 2$, dummy round i; When $\lambda = 1$ and j is even, $a = 0$, AES round i; When $\lambda = 1$ and j is odd, $a = 1$, redundant AES round i

---

Input: p, β, `keys`, t
Output: ciphertext or infected ciphertext

```
1  $R_0 = p, \quad R_1 = p, \quad R_2 = \beta$
2  Generate rstr$\in \mathbb{F}_2^t$
3  $j = 0, \quad \texttt{idx} = 1$
4  while $idx \leq t$ do
5      $i = \lfloor j/2 \rfloor$
6      $\lambda = \texttt{rstr}[\texttt{idx}]$
7      $a = ((\text{LSB of } j)\ \&\ \lambda) \oplus 2(\neg\lambda)$
8      $R_a = F_i(R_a, \texttt{keys}[\lambda][i])$
9      $\ldots$
10     $j = j + \lambda$
11     $\texttt{idx} = \texttt{idx} + 1$
12 return $R_0$
```

Infective countermeasure for AES-128

- Indicator function for $\mathbf{0}$ with domain $\mathbb{F}_2^{128}$

$$\begin{array}{rcl} 1_{\mathbf{0}} : \mathbb{F}_2^{128} & \to & \mathbb{F}_2 \\ \boldsymbol{x} & \mapsto & \prod_i (1 - x_i). \end{array}$$

- In other words,

$$1_{\mathbf{0}}(\boldsymbol{x}) = \begin{cases} 1 & \text{if } \boldsymbol{x} = 0 \\ 0 & \text{otherwise} \end{cases}.$$

- Then

$$\neg 1_{\mathbf{0}}(\boldsymbol{x}) = \begin{cases} 0 & \text{if } \boldsymbol{x} = 0 \\ 1 & \text{otherwise} \end{cases}.$$

Infective countermeasure for AES-128

$$\neg 1_{\mathbf{0}}(\boldsymbol{x}) = \begin{cases} 0 & \text{if } \boldsymbol{x} = 0 \\ 1 & \text{otherwise} \end{cases}.$$

We have

$$\begin{aligned} \gamma &= \lambda \,\&\, (\text{LSB of } j) \,\&\, (\neg 1_{\mathbf{0}}(R_0 \oplus R_1)) = \begin{cases} 0 & \text{if } \lambda = 0 \text{ or } j \text{ is even} \\ \neg 1_{\mathbf{0}}(R_0 \oplus R_1) & \text{otherwise} \end{cases} \\ &= \begin{cases} 0 & \text{if } \lambda = 0 \text{ or } j \text{ is even or } R_0 = R_1 \\ 1 & \lambda = 1, j \text{ is odd, and } R_0 \neq R_1 \end{cases} \end{aligned}$$

- When j is odd and $\lambda = 1$ (i.e. in the loop when the redundant AES round is computed)
  - γ indicates if the cipher state in the AES round computation, R_0, is equal to the redundant cipher state, R_1, or equivalent, whether fault happened in AES round or in the redundant round computation.
  - If there was no fault, $\gamma = 0$; otherwise, $\gamma = 1$.

Infective countermeasure for AES-128

- if j is odd and $\lambda = 1$, detect fault injection in AES
- If there was no fault, $\gamma = 0$; otherwise, $\gamma = 1$.

```
Input: $p$, $\beta$, keys, $t$
Output: ciphertext or infected ciphertext
$R_0 = p, \quad R_1 = p, \quad R_2 = \beta$
Generate rstr$\in \mathbb{F}_2^t$
$j = 0, \quad \texttt{idx} = 1$
while $idx \leq t$ do
    $i = \lfloor j/2 \rfloor$
    $\lambda = \texttt{rstr}[\texttt{idx}]$
    $a = ((\text{LSB of } j)\ \&\ \lambda) \oplus 2(\neg\lambda)$
    $R_a = F_i(R_a, \texttt{keys}[\lambda][i])$
    $\gamma = \lambda\ \&\ (\text{LSB of } j)\ \&\ (\neg 1_{\mathbf{0}}(R_0 \oplus R_1))$
    ...
    $j = j + \lambda$
    $\texttt{idx} = \texttt{idx} + 1$
return $R_0$
```

Infective countermeasure for AES-128

- Let
$$\delta = \begin{cases} 0 & \text{if } \lambda = 1 \\ \neg 1_{\mathbf{0}}(R_2 \oplus \beta) & \text{if } \lambda = 0 \end{cases} = \begin{cases} 0 & \text{if } \lambda = 1 \text{ or } R_2 = \beta \\ 1 & \text{if } \lambda = 0 \text{ and } R_2 \neq \beta \end{cases}.$$
- When $\lambda = 0$, i.e. in the loop when the dummy round is computed, δ indicates if there is a fault injected in the computation of the dummy round state R_2.
- If there was no fault, $\delta = 0$; otherwise, $\delta = 1$.

Infective countermeasure for AES-128

- When $\lambda = 0$, δ indicates if there is a fault injected in dummy round
- If there was no fault, $\delta = 0$; otherwise, $\delta = 1$.

```
Input: $p$, $\beta$, keys, $t$
Output: ciphertext or infected ciphertext
$R_0 = p, \quad R_1 = p, \quad R_2 = \beta$
Generate rstr$\in \mathbb{F}_2^t$
$j = 0, \quad \mathtt{idx} = 1$
while $idx \leq t$ do
    $i = \lfloor j/2 \rfloor$
    $\lambda = \mathtt{rstr}[\mathtt{idx}]$
    $a = ((\text{LSB of } j)\ \&\ \lambda) \oplus 2(\neg\lambda)$
    $R_a = F_i(R_a, \mathtt{keys}[\lambda][i])$
    $\gamma = \lambda\ \&\ (\text{LSB of } j)\ \&\ (\neg 1_{\mathbf{0}}(R_0 \oplus R_1))$
    $\delta = (\neg\lambda)\ \&\ (\neg 1_{\mathbf{0}}(R_2 \oplus \beta))$
    $\ldots$
    $j = j + \lambda$
    $\mathtt{idx} = \mathtt{idx} + 1$
return $R_0$
```

Infective countermeasure for AES-128

- if j is odd and $\lambda = 1$, detect fault injection in AES
- If there was no fault, $\gamma = 0$; otherwise, $\gamma = 1$.
- When $\lambda = 0$, δ indicates if there is a fault injected in dummy round
- If there was no fault, $\delta = 0$; otherwise, $\delta = 1$.
- Then

$$R_0 = (\neg(\gamma \vee \delta) \cdot R_0) \oplus ((\gamma \vee \delta) \cdot R_2) = \begin{cases} R_0 & \text{if } \gamma = 0 \text{ and } \delta = 0 \\ R_2 & \text{otherwise} \end{cases}.$$

- R_0 will be changed to a random number R_2 if a fault is detected in any of the computations.

Infective countermeasure for AES-128

- R_0 becomes a random number when fault is detected
- Consequently, the output will be a random number, or *infected ciphertext*.

```
**Input:** $p$, $\beta$, keys, $t$
**Output:** ciphertext or infected ciphertext
$R_0 = p, \quad R_1 = p, \quad R_2 = \beta$
Generate rstr$\in \mathbb{F}_2^t$
$j = 0, \quad$ idx $= 1$
**while** $idx \leq t$ **do**
    $i = \lfloor j/2 \rfloor$
    $\lambda = \texttt{rstr}[\texttt{idx}]$
    $a = ((\text{LSB of } j)\ \&\ \lambda) \oplus 2(\neg\lambda)$
    $R_a = F_i(R_a, \texttt{keys}[\lambda][i])$
    $\gamma = \lambda\ \&\ (\text{LSB of } j)\ \&\ (\neg 1_{\mathbf{0}}(R_0 \oplus R_1))$
    $\delta = (\neg\lambda)\ \&\ (\neg 1_{\mathbf{0}}(R_2 \oplus \beta))$
    $R_0 = (\neg(\gamma \vee \delta) \cdot R_0) \oplus ((\gamma \vee \delta) \cdot R_2)$
    $j = j + \lambda$
    $\texttt{idx} = \texttt{idx} + 1$
**return** $R_0$
```

Infective countermeasure for AES-128

Algorithm 1: Computation of AES round in the infective Countermeasure for AES-128 from

```
1 $j$ is even
2 $i = \lfloor j/2 \rfloor$// $i$ is the round counter
3 $\lambda = 1$
4 $a = 0$
5 $R_0 = F_i(R_0, \texttt{keys}[1][i])$// $\texttt{keys}[1][i] = K_i$ is the $i$th round key for AES
6 $\gamma = 0$
7 $\delta = 0$
8 $R_0 = R_0$
```

Infective countermeasure for AES-128

Algorithm 2: Computation of redundant AES round in the infective Countermeasure for AES-128.

```
1 $j$ is odd
2 $i = \lfloor j/2 \rfloor$ // $i$ is the round counter
3 $\lambda = 1$
4 $a = 1$
5 $R_1 = F_i(R_1, \texttt{keys}[1][i])$ // $\texttt{keys}[1][i] = K_i$ is the $i$th round key for AES
6 $\gamma = \neg 1_{\mathbf{0}}(R_0 \oplus R_1)$ // detect fault injection in AES
7 $\delta = 0$
8 $R_0 = ((\neg\gamma) \cdot R_0) \oplus (\gamma \cdot R_2)$ // if there is fault in AES computation, $R_0 = R_2$
     becomes a random number
```

Infective countermeasure for AES-128

Algorithm 3: Computation of the dummy round in the infective Countermeasure for AES-128

```
1 $\lambda = 0$
2 $a = 2$
3 $R_2 = F_i(R_2, \texttt{keys}[0][i])$// $i$ is the round counter, keys$[0][i] = \kappa_i$ is the $i$th round
    key for the dummy rounds
4 $\gamma = 0$
5 $\delta = \neg 1_{\mathbf{0}}(R_2 \oplus \beta)$// detect fault injection in dummy round
6 $R_0 = ((\neg\delta) \cdot R_0) \oplus (\delta \cdot R_2)$// if there is fault in the dummy round computation,
    $R_0 = R_2$ becomes a random number
```

Potential PhD thesis topics

- Side-channel analysis attacks and countermeasures
  - On cryptographic implementations
  - On neural networks
  - AI-assisted SCA
- Fault attacks and countermeasures
  - On cryptographic implementations
  - On neural networks
  - AI-assisted fault attacks