CRAFT: Lightweight Tweakable Block Cipher with Efficient Protection Against DFA Attacks

Christof Beierle Gregor Leander Amir Moradi
Shahram Rasoolzadeh

SnT, University of Luxembourg

Horst Görtz Institute for IT Security, Ruhr University Bochum

FSE 2019
March 25, 2019

Physical Attacks

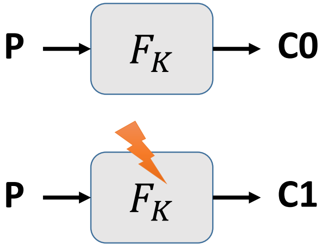
- Secrets stored in/processed by an implementation of a primitive can be recovered by **Physical Attacks**.

Physical Attacks

- Secrets stored in/processed by an implementation of a primitive can be recovered by **Physical Attacks.**
- Differential Fault Analysis (DFA) attacks are one of the most powerful class of them.

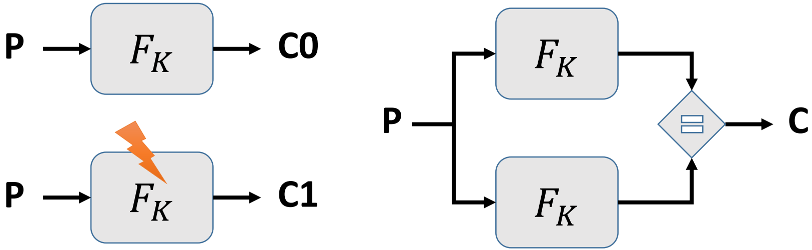
Physical Attacks

- Secrets stored in/processed by an implementation of a primitive can be recovered by **Physical Attacks.**
- Differential Fault Analysis (DFA) attacks are one of the most powerful class of them.



Physical Attacks

- Secrets stored in/processed by an implementation of a primitive can be recovered by **Physical Attacks**.
- Differential Fault Analysis (DFA) attacks are one of the most powerful class of them.



Impeccable Circuits [1]

Two general construction for Concurrent Error Detection

Adversary Model

Univariate(/Multivariate) Model, $\mathcal{M}_t$:
The adversary is able to make at most t cells of the entire circuit faulty at only one (/every) clock cycle.

[1] Aghaie et. al., *Impeccable Circuits*. IACR Cryptology ePrint Archive, 2018:203.

Impeccable Circuits [1]

Two general construction for Concurrent Error Detection

Adversary Model

Univariate(/Multivariate) Model, $\mathcal{M}_t$:
The adversary is able to make at most t cells of the entire circuit faulty at only one (/every) clock cycle.

Safe-Error and Stuck-at-0/1 models are not covered.

[1] Aghaie et. al., *Impeccable Circuits*. IACR Cryptology ePrint Archive, 2018:203.

Impeccable Circuits [1]

Two general construction for Concurrent Error Detection

Adversary Model

Univariate(/Multivariate) Model, $\mathcal{M}_t$:
The adversary is able to make at most t cells of the entire circuit faulty at only one (/every) clock cycle.

Safe-Error and Stuck-at-0/1 models are not covered.

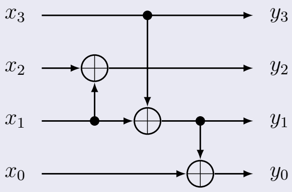
Independence Property

To prevent fault propagation, the coordinate functions of each operation have to be implemented independently.

[1] Aghaie et. al., *Impeccable Circuits*. IACR Cryptology ePrint Archive, 2018:203.

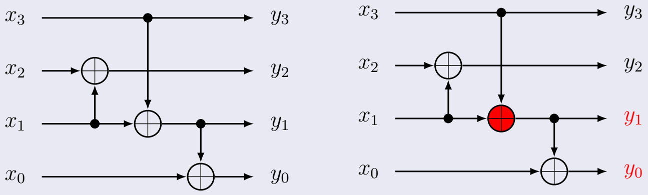
Independence Property

Example (Skinny's MixColumn:)



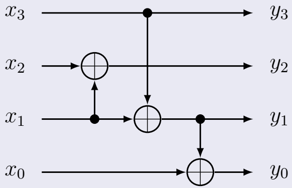
Independence Property

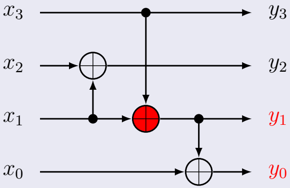
Example (Skinny's MixColumn:)

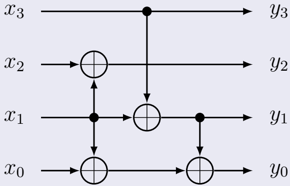


Independence Property

Example (Skinny's MixColumn:)







Motivation

Results of *Impeccable Circuits*

- Different Lightweight Block Ciphers:
  Skinny, LED, Midori, Present, Gift, Simon.

Motivation

Results of *Impeccable Circuits*

- Different Lightweight Block Ciphers:
  Skinny, LED, Midori, Present, Gift, Simon.
- There is a big gap between the implementation size of unprotected and protected circuits.

Motivation

Results of *Impeccable Circuits*

- Different Lightweight Block Ciphers: Skinny, LED, Midori, Present, Gift, Simon.
- There is a big gap between the implementation size of unprotected and protected circuits.

Goals

- Protection against DFA Attacks with efficient hardware implementation
- Tweakable and providing decryption with little implementation area overhead
- Using known design methods for easier security analysis

Motivation

Results of *Impeccable Circuits*

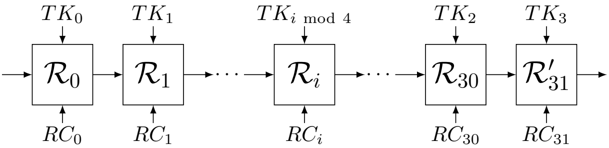
- Different Lightweight Block Ciphers:
  Skinny, LED, Midori, Present, Gift, Simon.
- There is a big gap between the implementation size of unprotected and protected circuits.

Goals

- Protection against DFA Attacks with efficient hardware implementation
- Tweakable and providing decryption with little implementation area overhead
- Using known design methods for easier security analysis

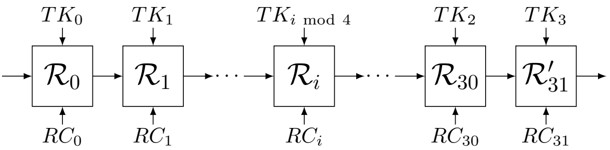
- Skinny-like structure with 128-bit key, 64-bit block & tweak

Structure

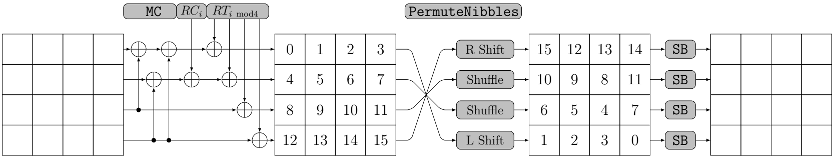


- 32 rounds: 31 identical rounds and last linear round
- Internal state: viewed as 4×4 matrix of nibbles

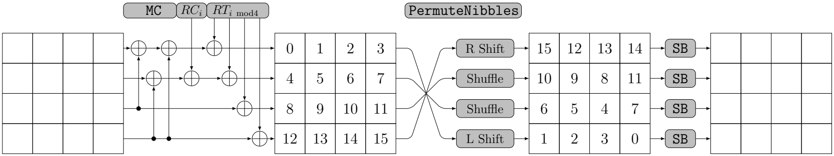
Structure



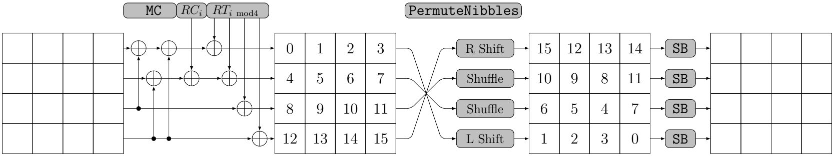
- 32 rounds: 31 identical rounds and last linear round
- Internal state: viewed as 4×4 matrix of nibbles



Round Functions

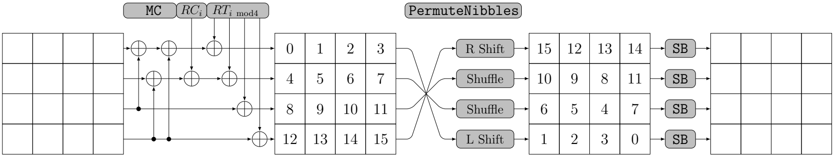


Round Functions



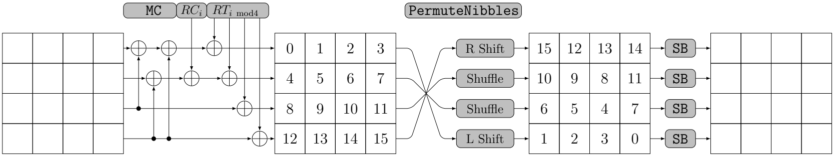
- `MixColumn(MC):`
  Involutory binary matrix *M* is multiplied to each column.

Round Functions



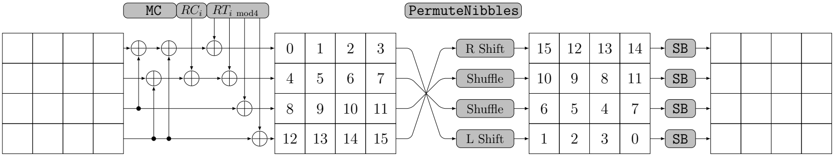
- `MixColumn(MC)`:
  Involutory binary matrix *M* is multiplied to each column.
- `AddConstants`$_i$ `(ARC`$_i$`)`:
  4-bit value a_i and 3-bit b_i are xored to the 4th & 5th nibbles.

Round Functions



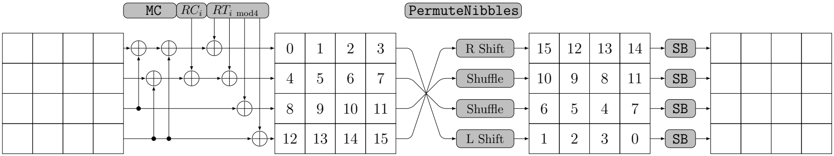
- `MixColumn(MC)`:
  Involutory binary matrix M is multiplied to each column.
- `AddConstants`$_i$ `(ARC`$_i$`)`:
  4-bit value a_i and 3-bit b_i are xored to the 4th & 5th nibbles.
- `AddTweakey`$_i$ `(ATK`$_i$`)`:
  Tweakey $TK_{i \bmod 4}$ is xored to the state.

Round Functions



- MixColumn (MC):
  Involutory binary matrix M is multiplied to each column.
- AddConstants$_i$ (ARC$_i$):
  4-bit value a_i and 3-bit b_i are xored to the 4th & 5th nibbles.
- AddTweakey$_i$ (ATK$_i$):
  Tweakey $TK_{i \bmod 4}$ is xored to the state.
- PermuteNibbles (PN):
  Involutory permutation P is applied on the nibble positions.

Round Functions



- `MixColumn(MC)`:
  Involutory binary matrix M is multiplied to each column.
- `AddConstants`$_i$`(ARC`$_i$`)`:
  4-bit value a_i and 3-bit b_i are xored to the 4th & 5th nibbles.
- `AddTweakey`$_i$`(ATK`$_i$`)`:
  Tweakey $TK_{i \bmod 4}$ is xored to the state.
- `PermuteNibbles(PN)`:
  Involutory permutation P is applied on the nibble positions.
- `SubBox(SB)`:
  4-bit involutory Sbox S is applied to each nibble.

Tweakey Schedule

If (K_0, K_1) are two 64-bit halves of the key and T is the tweak, then

$$\begin{aligned} TK_0 &= K_0 \oplus T \\ TK_1 &= K_1 \oplus T \end{aligned}$$

Tweakey Schedule

If (K_0, K_1) are two 64-bit halves of the key and T is the tweak, then

$$\begin{aligned} TK_0 &= K_0 \oplus T \\ TK_1 &= K_1 \oplus T \\ TK_2 &= K_0 \oplus Q(T) \\ TK_3 &= K_1 \oplus Q(T) \end{aligned}$$

where Q is a circular permutation on the position of tweak nibbles:

$$[12, 10, 15, 5,\ 14, 8, 9, 2,\ 11, 3, 7, 4,\ 6, 0, 1, 13]$$

One structure for both encryption & decryption

Lemma 1:

CRAFT decryption is the same as its encryption with modified tweakeys and reverse order of round constants.

One structure for both encryption & decryption

Lemma 1:

CRAFT decryption is the same as its encryption with modified tweakeys and reverse order of round constants.

$$\mathrm{SB} \circ \mathrm{PN} = \mathrm{PN} \circ \mathrm{SB}$$

$$\mathrm{MC} \circ \mathrm{ARC} \circ \mathrm{ATK} = \mathrm{ATK}' \circ \mathrm{ARC} \circ \mathrm{MC}$$

$$\mathrm{TK}' = \mathrm{MC}(\mathrm{TK})$$

One structure for both encryption & decryption

Lemma 1:

CRAFT decryption is the same as its encryption with modified tweakeys and reverse order of round constants.

$$\mathcal{DEC}\,{}^{RC_0,\cdots,RC_{31}}_{TK_0,\cdots,TK_3} =$$

$$= (\mathrm{ATK}_3 \circ \mathrm{ARC}_{31} \circ \mathrm{MC} \circ \mathrm{SB} \circ \mathrm{PN} \circ \mathrm{ATK}_2 \circ \mathrm{ARC}_{30} \circ \mathrm{MC} \circ \cdots \circ$$

$$\circ \mathrm{SB} \circ \mathrm{PN} \circ \mathrm{ATK}_0 \circ \mathrm{ARC}_0 \circ \mathrm{MC})^{-1}$$

One structure for both encryption & decryption

Lemma 1:

CRAFT decryption is the same as its encryption with modified tweakeys and reverse order of round constants.

$$\mathcal{DEC}\,{}^{RC_0,\cdots,RC_{31}}_{TK_0,\cdots,TK_3} =$$

$$= (\mathrm{ATK}_3 \circ \mathrm{ARC}_{31} \circ \mathrm{MC} \circ \mathrm{SB} \circ \mathrm{PN} \circ \mathrm{ATK}_2 \circ \mathrm{ARC}_{30} \circ \mathrm{MC} \circ \cdots \circ$$

$$\circ \mathrm{SB} \circ \mathrm{PN} \circ \mathrm{ATK}_0 \circ \mathrm{ARC}_0 \circ \mathrm{MC})^{-1}$$

$$= \mathrm{MC} \circ \mathrm{ARC}_0 \circ \mathrm{ATK}_0 \circ \mathrm{PN} \circ \mathrm{SB} \circ \cdots \circ$$

$$\circ \mathrm{MC} \circ \mathrm{ARC}_{30} \circ \mathrm{ATK}_2 \circ \mathrm{PN} \circ \mathrm{SB} \circ \mathrm{MC} \circ \mathrm{ARC}_{31} \circ \mathrm{ATK}_3$$

One structure for both encryption & decryption

Lemma 1:

CRAFT decryption is the same as its encryption with modified tweakeys and reverse order of round constants.

$$\mathcal{DEC}\,{}^{RC_0,\cdots,RC_{31}}_{TK_0,\cdots,TK_3} =$$

$$= (\mathtt{ATK}_3 \circ \mathtt{ARC}_{31} \circ \mathtt{MC} \circ \mathtt{SB} \circ \mathtt{PN} \circ \mathtt{ATK}_2 \circ \mathtt{ARC}_{30} \circ \mathtt{MC} \circ \cdots \circ$$

$$\circ \mathtt{SB} \circ \mathtt{PN} \circ \mathtt{ATK}_0 \circ \mathtt{ARC}_0 \circ \mathtt{MC})^{-1}$$

$$= \mathtt{MC} \circ \mathtt{ARC}_0 \circ \mathtt{ATK}_0 \circ \mathtt{PN} \circ \mathtt{SB} \circ \cdots \circ$$

$$\circ \mathtt{MC} \circ \mathtt{ARC}_{30} \circ \mathtt{ATK}_2 \circ \mathtt{PN} \circ \mathtt{SB} \circ \mathtt{MC} \circ \mathtt{ARC}_{31} \circ \mathtt{ATK}_3$$

One structure for both encryption & decryption

Lemma 1:

CRAFT decryption is the same as its encryption with modified tweakeys and reverse order of round constants.

$$\mathcal{DEC}^{RC_0,\cdots,RC_{31}}_{TK_0,\cdots,TK_3} =$$

$$= (\mathrm{ATK}_3 \circ \mathrm{ARC}_{31} \circ \mathrm{MC} \circ \mathrm{SB} \circ \mathrm{PN} \circ \mathrm{ATK}_2 \circ \mathrm{ARC}_{30} \circ \mathrm{MC} \circ \cdots \circ$$

$$\circ \mathrm{SB} \circ \mathrm{PN} \circ \mathrm{ATK}_0 \circ \mathrm{ARC}_0 \circ \mathrm{MC})^{-1}$$

$$= \mathrm{MC} \circ \mathrm{ARC}_0 \circ \mathrm{ATK}_0 \circ \mathrm{PN} \circ \mathrm{SB} \circ \cdots \circ$$

$$\circ \mathrm{MC} \circ \mathrm{ARC}_{30} \circ \mathrm{ATK}_2 \circ \mathrm{PN} \circ \mathrm{SB} \circ \mathrm{MC} \circ \mathrm{ARC}_{31} \circ \mathrm{ATK}_3$$

$$= \mathrm{ATK}'_0 \circ \mathrm{ARC}_0 \circ \mathrm{MC} \circ \mathrm{SB} \circ \mathrm{PN} \circ \cdots \circ$$

$$\circ \mathrm{ATK}'_2 \circ \mathrm{ARC}_{30} \circ \mathrm{MC} \circ \mathrm{SB} \circ \mathrm{PN} \circ \mathrm{ATK}'_3 \circ \mathrm{ARC}_{31} \circ \mathrm{MC}$$

One structure for both encryption & decryption

Lemma 1:

CRAFT decryption is the same as its encryption with modified tweakeys and reverse order of round constants.

$$
\begin{aligned}
&\mathcal{DEC}^{RC_0,\cdots,RC_{31}}_{TK_0,\cdots,TK_3} = \\
&= (\mathrm{ATK}_3 \circ \mathrm{ARC}_{31} \circ \mathrm{MC} \circ \mathrm{SB} \circ \mathrm{PN} \circ \mathrm{ATK}_2 \circ \mathrm{ARC}_{30} \circ \mathrm{MC} \circ \cdots \circ \\
&\circ \mathrm{SB} \circ \mathrm{PN} \circ \mathrm{ATK}_0 \circ \mathrm{ARC}_0 \circ \mathrm{MC})^{-1} \\
&= \mathrm{MC} \circ \mathrm{ARC}_0 \circ \mathrm{ATK}_0 \circ \mathrm{PN} \circ \mathrm{SB} \circ \cdots \circ \\
&\circ \mathrm{MC} \circ \mathrm{ARC}_{30} \circ \mathrm{ATK}_2 \circ \mathrm{PN} \circ \mathrm{SB} \circ \mathrm{MC} \circ \mathrm{ARC}_{31} \circ \mathrm{ATK}_3 \\
&= \mathrm{ATK}'_0 \circ \mathrm{ARC}_0 \circ \mathrm{MC} \circ \mathrm{SB} \circ \mathrm{PN} \circ \cdots \circ \\
&\circ \mathrm{ATK}'_2 \circ \mathrm{ARC}_{30} \circ \mathrm{MC} \circ \mathrm{SB} \circ \mathrm{PN} \circ \mathrm{ATK}'_3 \circ \mathrm{ARC}_{31} \circ \mathrm{MC} \\
&= \mathcal{ENC}^{RC_{31},\cdots,RC_0}_{TK'_3,\cdots,TK'_0}.
\end{aligned}
$$

Sbox

Sbox

Sbox & Redundant Sbox

For each Sbox, we need to implement a *Redundant Sbox*.

Sbox

Sbox & Redundant Sbox

For each Sbox, we need to implement a *Redundant Sbox*.
For example, in case of 4-bit redundancy:

$$S_4 = F_4 \circ S \circ F_4^{-1}$$

where F_4 is a multiplication with

$$\begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix}$$

Sbox

Sbox & Redundant Sbox

For each Sbox, we need to implement a *Redundant Sbox*.
For example, in case of 4-bit redundancy:

$$S_4 = F_4 \circ S \circ F_4^{-1}$$

where F_4 is a multiplication with

$$\begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix}$$

Problem

There are 46 206 736 involutory 4-bit Sboxes which implementing and synthesizing all of them is impossible.

Sbox

Solution

- Because of *Independence Property*, each coordinate function of Sbox must be implemented separately.

Sbox

Solution

- Because of *Independence Property*, each coordinate function of Sbox must be implemented separately.
- Implementation cost of each operation is sum of area size for its coordinate functions.

Sbox

Solution

- Because of *Independence Property*, each coordinate function of Sbox must be implemented separately.
- Implementation cost of each operation is sum of area size for its coordinate functions.
- For each Sbox, size of 13 Boolean functions are important.

Sbox

Solution

- Because of *Independence Property*, each coordinate function of Sbox must be implemented separately.
- Implementation cost of each operation is sum of area size for its coordinate functions.
- For each Sbox, size of 13 Boolean functions are important.
- There are 12 870 four-bit balanced Boolean functions.

Sbox

Solution

- Because of *Independence Property*, each coordinate function of Sbox must be implemented separately.
- Implementation cost of each operation is sum of area size for its coordinate functions.
- For each Sbox, size of 13 Boolean functions are important.
- There are 12 870 four-bit balanced Boolean functions.
- Up to bit permutation-equivalence, there are only 730.

Sbox

Solution

- Because of *Independence Property*, each coordinate function of Sbox must be implemented separately.
- Implementation cost of each operation is sum of area size for its coordinate functions.
- For each Sbox, size of 13 Boolean functions are important.
- There are 12 870 four-bit balanced Boolean functions.
- Up to bit permutation-equivalence, there are only 730.

Results for Sbox

Among all the smallest found Soxes, we use the Midori's one.

Tweakey Schedule

Key Schedule

- Round key updating method needs at least 128 registers.
- Round key alternating method needs 64 multiplexers.

Tweakey Schedule

Key Schedule

- Round key updating method needs at least 128 registers.
- Round key alternating method needs 64 multiplexers.

Tweak Schedule

- Xoring the tweak with key.

Tweakey Schedule

Key Schedule

- Round key updating method needs at least 128 registers.
- Round key alternating method needs 64 multiplexers.

Tweak Schedule

- Xoring the tweak with key.
- To prevent Time-Data-Memory Trade-off attacks, tweak cannot be always the same when round keys are equal.

Tweakey Schedule

Key Schedule

- Round key updating method needs at least 128 registers.
- Round key alternating method needs 64 multiplexers.

Tweak Schedule

- Xoring the tweak with key.
- To prevent Time-Data-Memory Trade-off attacks, tweak cannot be always the same when round keys are equal.
- Solution: using 64 multiplexers to choose T or a nibble-wise permutation of it, $Q(T)$.

Tweakey Schedule

Key Schedule

- Round key updating method needs at least 128 registers.
- Round key alternating method needs 64 multiplexers.

Tweak Schedule

- Xoring the tweak with key.
- To prevent Time-Data-Memory Trade-off attacks, tweak cannot be always the same when round keys are equal.
- Solution: using 64 multiplexers to choose T or a nibble-wise permutation of it, $Q(T)$.
- To provide maximum possible security against TDM-TO attack, Q must be circular (there are $15! \approx 2^{40}$).
- Trying 1000 of them, Q is the one with most active Sboxes in related-tweak differential attack.

Security

Security Analysis

- Time-Data-Memory Trade-off
- (Truncated / Impossible) (ST/RT) Differential
- (Linear Hulls / Zero-Correlation) Linear
- Integral
- Meet in the Middle
- (Linear Subspace/Nonlinear) Invariant Attacks

Security

Security Analysis

- Time-Data-Memory Trade-off
- (Truncated / Impossible) (ST/RT) Differential
- (Linear Hulls / Zero-Correlation) Linear
- Integral
- Meet in the Middle
- (Linear Subspace/Nonlinear) Invariant Attacks

Security Claim

- 124 bit security in the related-tweak model
- No claim in chosen-key, known-key or related-key models

Accelerated Exhaustive Search

Related-key Property:

If $\Delta = (x, x, \ldots, x)$, since $Q(\Delta) = \Delta$, both (K_0, K_1, T) and $(K_0 + \Delta, K_1 + \Delta, T + \Delta)$ cause the same tweakeys:

$$TK_i = TK'_i \quad (0 \leq i \leq 3)$$

Accelerated Exhaustive Search

Related-key Property:

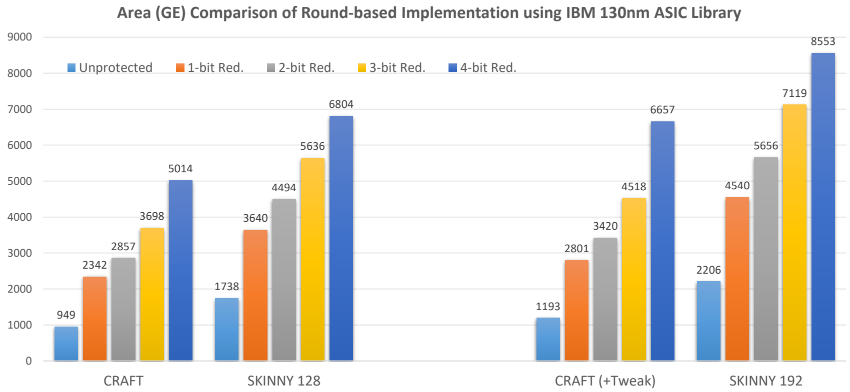
If $\Delta = (x, x, \ldots, x)$, since $Q(\Delta) = \Delta$, both (K_0, K_1, T) and $(K_0 + \Delta, K_1 + \Delta,\ T + \Delta)$ cause the same tweakeys:

$$TK_i = TK_i' \quad (0 \leq i \leq 3)$$

Attack Procedure:

- Attacker asks for encryption of the same plaintext P under 16 different tweaks of $T, T + \Delta_1, \ldots, T + \Delta_{15}$: $C_0, C_1, \ldots, C_{15}$.
- By setting one of the key nibbles to zero, for each of 2^{124} possible key candidate (K_0^*, K_1^*), he computes C^*, the encryption of P using K_0^*, K_1^* and T.
- If C^* is equal to C_x, then $(K_0^* + \Delta_x,\ K_1^* + \Delta_x)$ is a candidate for the master key.

Hardware Implementations



Summary

CRAFT:

Implementation

- A lightweight tweakable block cipher with effiCient pRotection Against DFA aTtacks
- The smallest block cipher with 128-bit key in the round-based implementation (950 GE)
- Lower area overhead to support a 64-bit tweak (245 GE)
- Lower area overhead to support decryption (140 GE)

Summary

CRAFT:

Implementation

- A lightweight tweakable block cipher with effiCient pRotection Against DFA aTtacks
- The smallest block cipher with 128-bit key in the round-based implementation (950 GE)
- Lower area overhead to support a 64-bit tweak (245 GE)
- Lower area overhead to support decryption (140 GE)

Security

Providing 124-bit security in the related-tweak model

The End

Thank you for your attention.

Looking forward for further analysis by you

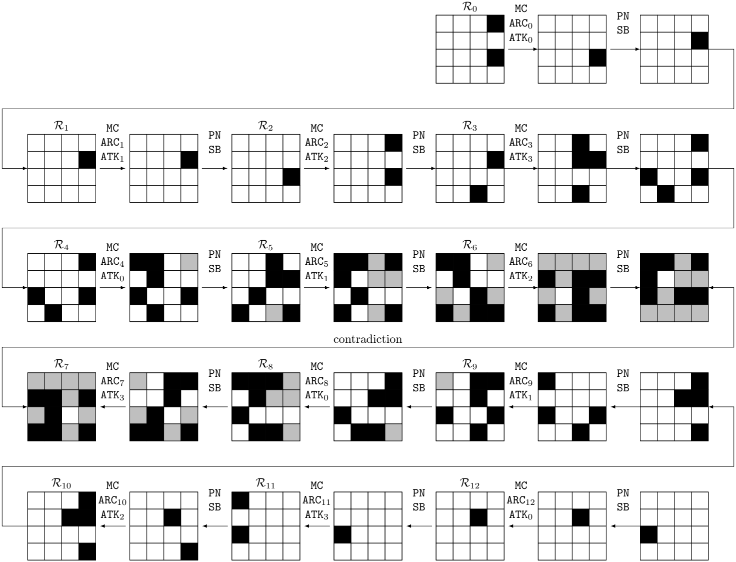
Time-Data-Memory Trade-off Attack

- Attacker fixes the tweakeys to $TK_0 = 0$, $TK_1 = X$, $TK_2 = T'$ and $TK_3 = X + T'$.
- For plaintext P and all possible X and T', he computes the ciphertext $C_{T',X}$ and saves X in the index (T', C) of table $\mathcal{T}$.

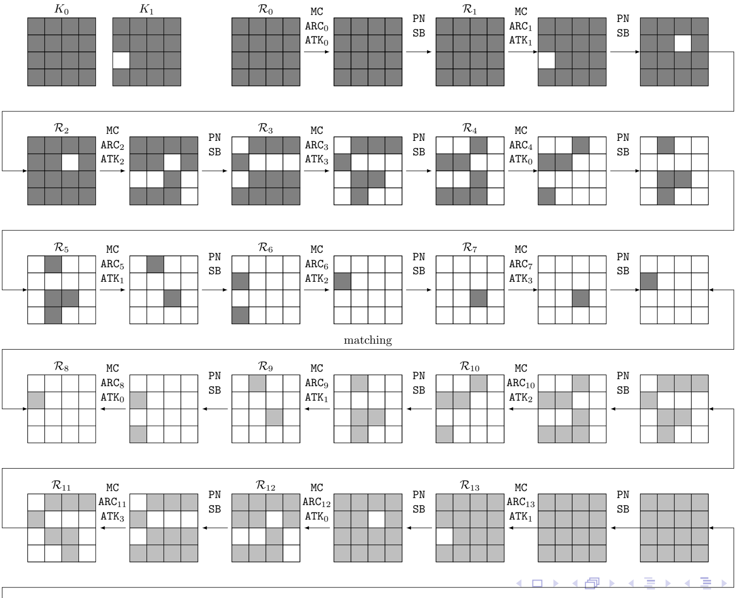
- For all possible tweaks T, attacker requests for encryption of P; C_T.
- For each of T, he gets a candidate for $K_0 + K_1$ by looking up to the index $(T + Q(T), C_T)$ of $\mathcal{T}$.

- $2^{64+\dim\{T+Q(T)\}}$ pre-computations, $2^{64+\dim\{T+Q(T)\}}$ memory, 2^{65} online computions and 2^{64} data.
- All online attack: $2^{64+\dim\{T+Q(T)\}}$ computations, 2^{64} data and memory.

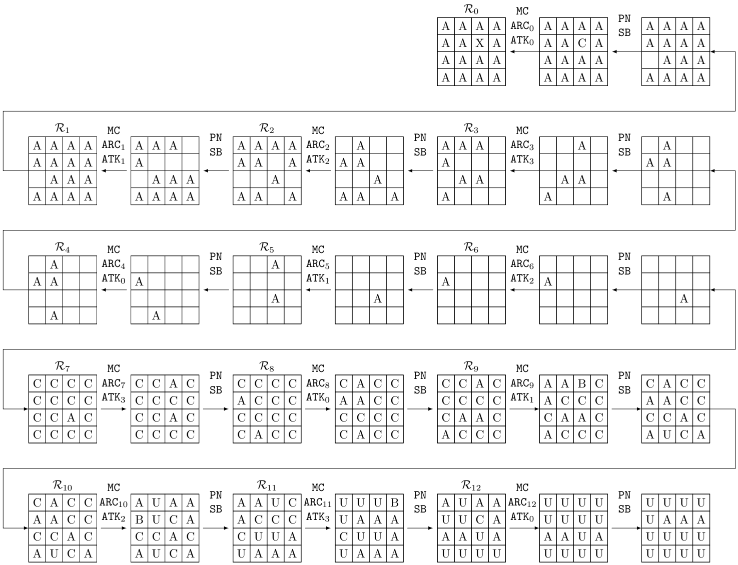
13-Round Impossible Truncated Differentials



15-Round Meet-in-the-Middle Attacks



13-Round Integral Distinguisher



Enc. & Dec. Algorithms

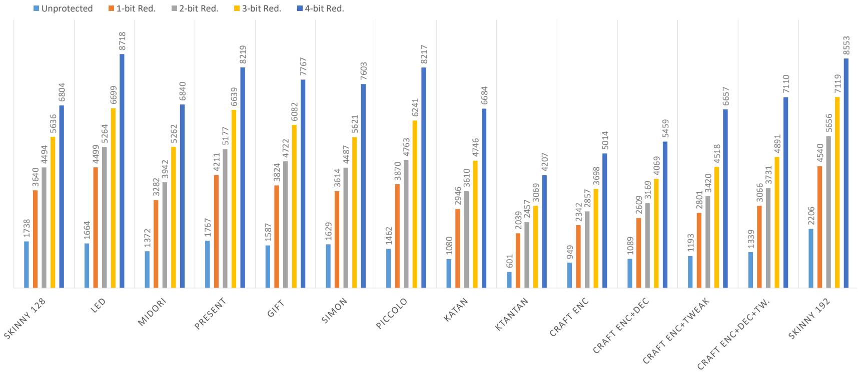
Input : X: *plaintext*
$K_0||K_1$: *cipher key*
T: *tweak*
Output : Y: *ciphertext*

$TK_0 \leftarrow K_0 \oplus T$
$TK_1 \leftarrow K_1 \oplus T$
$TK_2 \leftarrow K_0 \oplus Q(T)$
$TK_3 \leftarrow K_1 \oplus Q(T)$
$Y \leftarrow X$
for $i \leftarrow 0$ **to** 31 **do**
  $Y \leftarrow \texttt{MC}(Y)$
  $Y_{4,5} \leftarrow Y_{4,5} \oplus RC_i$
  $Y \leftarrow Y \oplus TK_{i \bmod 4}$
  if $i \neq 31$ **then**
    $Y \leftarrow \texttt{PN}(Y)$
    $Y \leftarrow \texttt{SB}(Y)$
  end
end

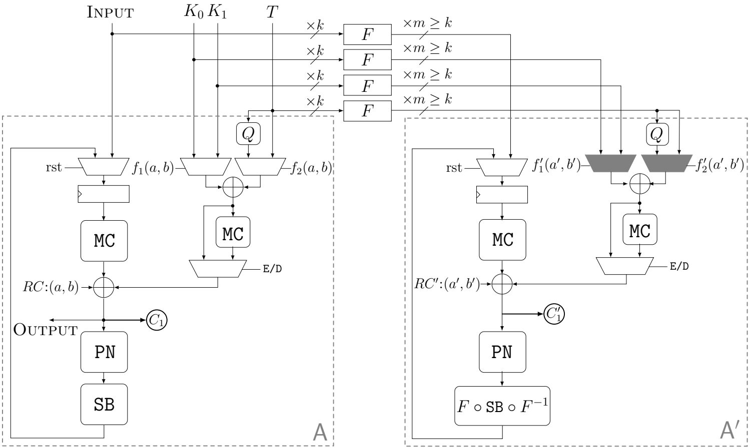
Input : X: *ciphertext*
$K_0||K_1$: *cipher key*
T: *tweak*
Output : Y: *plaintext*

$TK_0 \leftarrow \texttt{MC}(K_0 \oplus T)$
$TK_1 \leftarrow \texttt{MC}(K_1 \oplus T)$
$TK_2 \leftarrow \texttt{MC}(K_0 \oplus Q(T))$
$TK_3 \leftarrow \texttt{MC}(K_1 \oplus Q(T))$
$Y \leftarrow X$
for $i \leftarrow 31$ **to** 0 **do**
  $Y \leftarrow \texttt{MC}(Y)$
  $Y_{4,5} \leftarrow Y_{4,5} \oplus RC_i$
  $Y \leftarrow Y \oplus TK_{i \bmod 4}$
  if $i \neq 0$ **then**
    $Y \leftarrow \texttt{PN}(Y)$
    $Y \leftarrow \texttt{SB}(Y)$
  end
end

Implementation Results



Round-based Implementation with Fault Detection



Round-based Implementation with Fault Detection