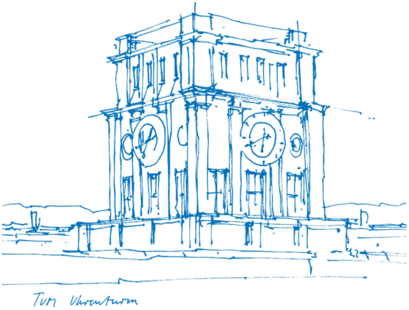
# Differential Fault Attacks on KLEIN

Michael Gruber[1], Bodo Selmke[2]

[1]Department of Electrical and Computer Engineering, Technical University of Munich
[2]Fraunhofer Institute for Applied and Integrated Security, Garching, Germany

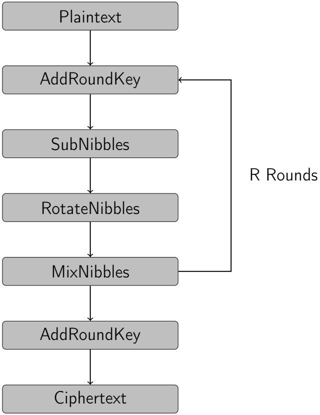
COSADE 2019



# Table of contents

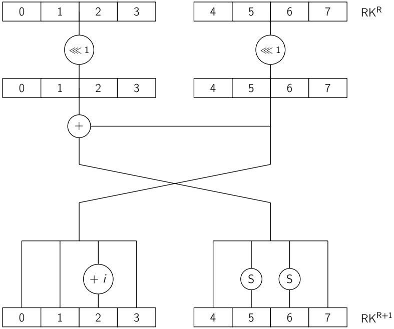
# KLEIN - Round Function

- Gong et al. 2012 [3]
- Round function similar to AES
- Substitution Permutation Network
- Key sizes of {64, 80, 96} bit
- Block size of 64 bit
- State of 4 bit nibbles
- Involutive 4 bit S-Box
- All functions are linear, except *SubNibbles*
- All rounds are equal



# KLEIN - Key Schedule

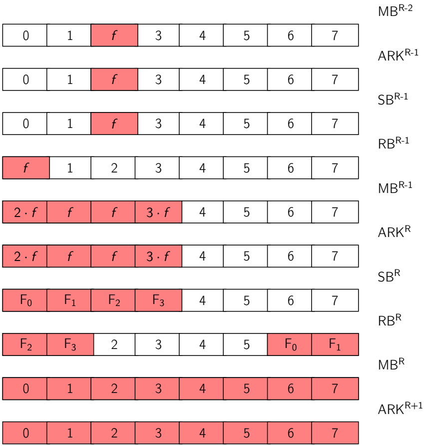
- Balanced Feistel network
- Byte oriented
- State size of {64, 80, 96} bit
- Circular left shift by 1 byte
- Addition on $GF(2)$
- Addition with round counter $i$
- Reuse S-Box (round function)



# Overview

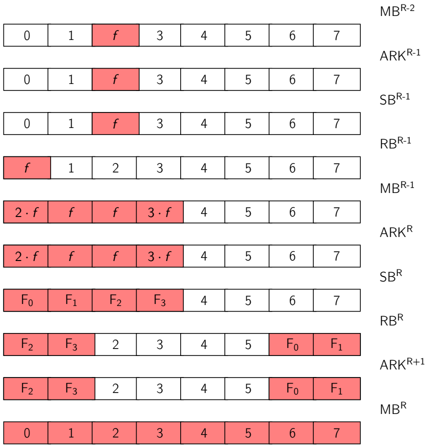
# DFA - State

- Similar [5–7] (AES)
- Applicable to all Variants
- Assume byte oriented state
- Random byte fault model
- 4 possible fault locations
- Recoverable half of $ARK^R$ depends on fault location
- *MixBytes* distributes the fault **over one half**!



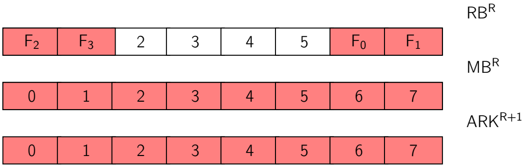
# DFA - State

- Similar [5–7] (AES)
- Applicable to all Variants
- Assume byte oriented state
- Random byte fault model
- 4 possible fault locations
- Recoverable half of $ARK^R$ depends on fault location
- *MixBytes* distributes the fault **over one half**!

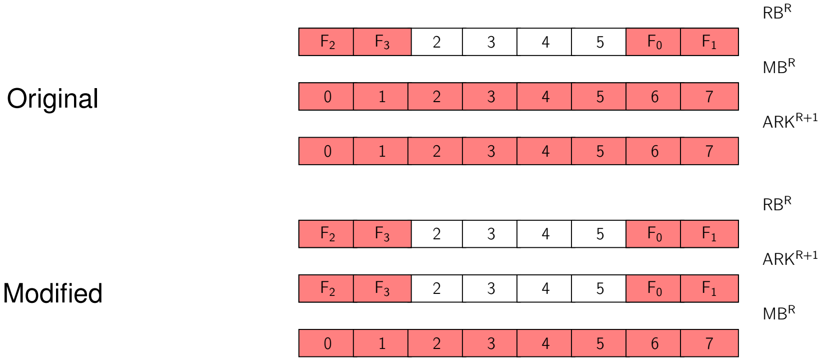


# Modified Representation of KLEIN

Original



# Modified Representation of KLEIN




Application of *invMixBytes* to the last round key required

# Overview

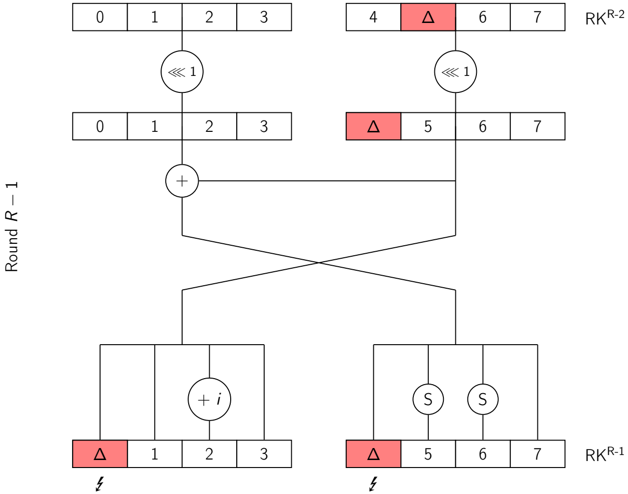
# DFA - Key Schedule

- Similar to the attacks on the AES key schedule [1, 2, 4]
- Applicable to KLEIN-64
- Random byte fault model
- Two possible positions to inject a fault
- Recovery of the whole state

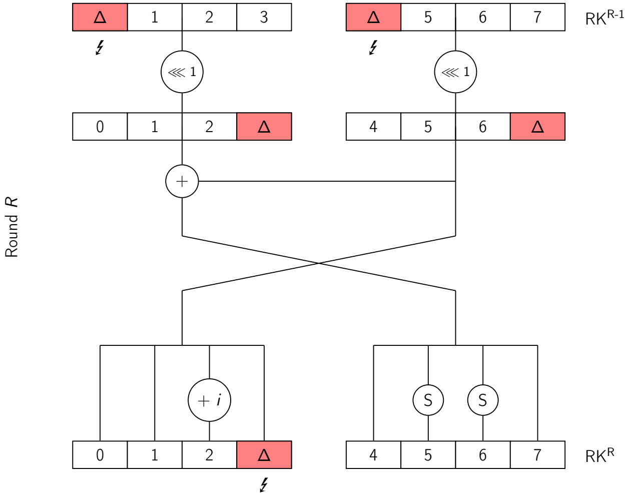
**Fault Location**

- Problem:
  - Limited fault propagation of the round function
- Idea:
  - Exploit Feistel structure of the key schedule
  - Choose fault location to be within a path of only linear functions
  - Cancel out the fault asap

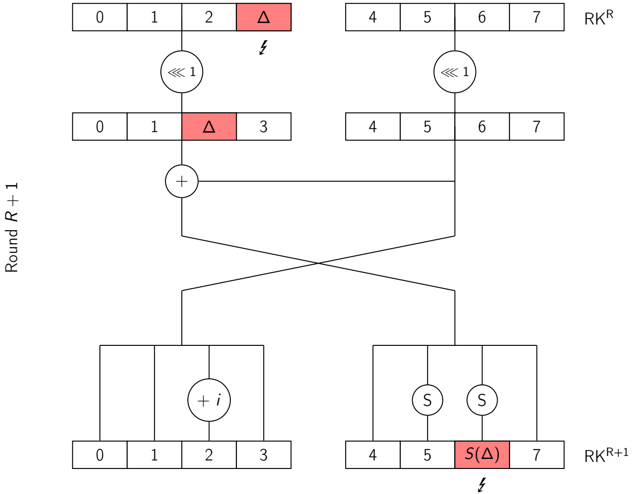
# Fault Propagation Key Schedule I



# Fault Propagation Key Schedule II

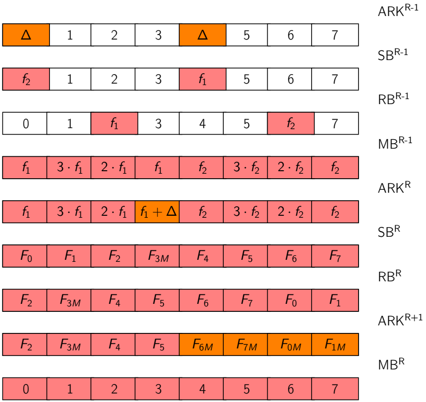


# Fault Propagation Key Schedule III



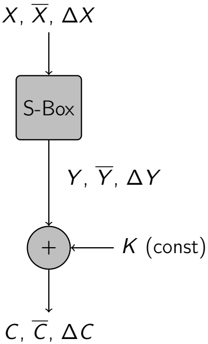
# Fault Propagation State

- Fault from the key schedule
- Two bytes are perturbed with $\Delta$
- One byte is perturbed with $f_1 + \Delta$
- Right half of $ARK^{R+1}$ is perturbed (modified rep.)
- $ARK^{R+1}$ is observable



# Attack on the Substitution Layer

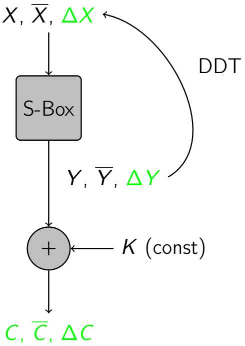
- Objective:
  - Recovery of $X$
- Conditions:
  - $\Delta C$, the S-Box is known
  - $K$ is constant
  - $\overline{X}$, $\Delta X$ is variable
- Approach:
  - Exhaustive search $X$, $\Delta X$
  - Exploit relationsship between $\Delta X$ and $\Delta Y$ (DDT)




$$\Delta C = SubByte(X) + SubByte(X + \Delta X) = filter(X, \Delta X)$$

# Attack on the Substitution Layer

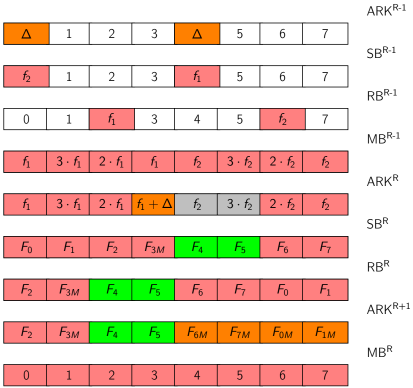
- Objective:
  - Recovery of $X$
- Conditions:
  - $\Delta C$, the S-Box is known
  - $K$ is constant
  - $\overline{X}$, $\Delta X$ is variable
- Approach:
  - Exhaustive search $X$, $\Delta X$
  - Exploit relationsship between $\Delta X$ and $\Delta Y$ (DDT)




$$\Delta C = SubByte(X) + SubByte(X + \Delta X) = filter(X, \Delta X)$$

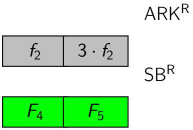
# Approach I

- *MixBytes* in the last round is invertible
- *AddRoundKey* is linear
- $ARK^{R+1}$ is observable, and so are the faults
- Faults from the key schedule
- Faulty state byte

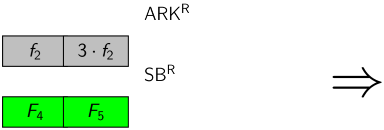



Example: Observable faults, bytes under attack

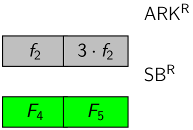
# Approach II



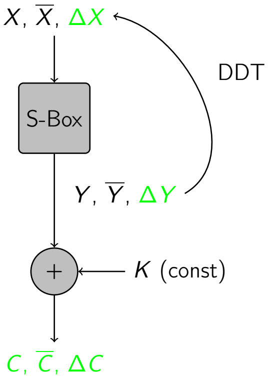
# Approach II



# Approach II




$\Rightarrow$



## Approach III

$$\begin{aligned} F_4 &= filter(ARK_4^R, f_2) \\ F_5 &= filter(ARK_5^R, 3 \cdot f_2) \end{aligned} \tag{1}$$

$$\begin{aligned} T_{poss} &= \{ARK_4^R \times ARK_5^R \times \{1, ..., 255\}\} \\ T_{valid} &= \{(x, y, f) \in T_{poss} \mid F_4 \equiv filter(x, f) \wedge F_5 \equiv filter(y, 3 \cdot f)\} \\ ARK_4^R &= \{\ x \mid (x, y, f) \in T_{valid}\ \} \\ ARK_5^R &= \{\ y \mid (x, y, f) \in T_{valid}\ \} \\ f_2 &= \{\ f \mid (x, y, f) \in T_{valid}\ \} \end{aligned} \tag{2}$$

# Overview

KLEIN

DFA - State

DFA - Key Schedule

Simulation

Practical Evaluation

Conclusion

# Simulation I

- Implemented in Python, core of the attack as C-Extension for Python
- Simulated on an desktop CPU [1]
- Amount of RAM can be neglected
- Attack requires up to 3 minutes

**Approach**

1. Perform 1 correct encryption
2. Perform 100[2] faulty encryptions
3. Store the remaining brute force complexity for the n-th faulty encryption

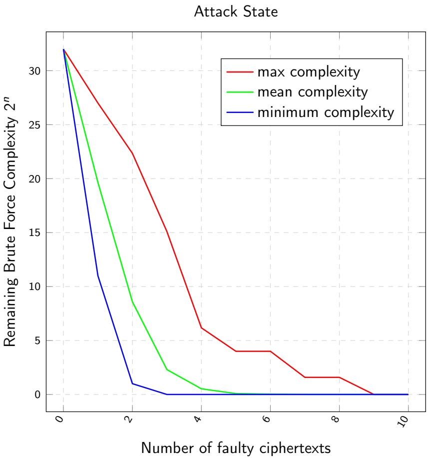
[1] `Intel(R) Core(TM) i7-6700 CPU @ 3.40GHz`

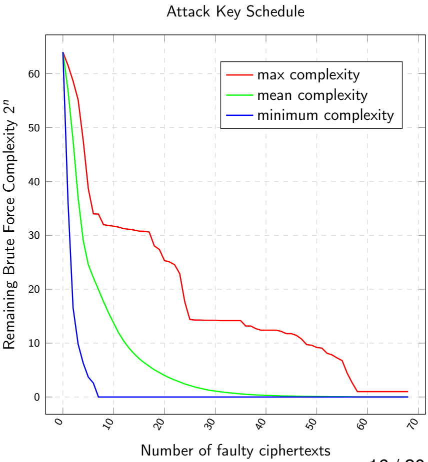
[2] 100 was found to be a reliable upper bound

# Simulation II

| | State | Key schedule |
|---|---|---|
| Injection strategy | one half | key schedule |
| Decrease in complexitiy | $2^{32}$ to $2^{0}$ (one half) | $2^{64}$ to $2^{32}$ (both halves) |
| # faulty encryptions | 5 | 4 |
| Remaining complexity | $2^{32}$ (the other half) | $2^{32}$ (both halves) |

# Simulation III

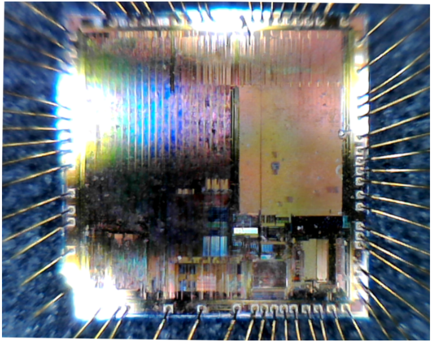


# Overview

# Evaluation - DUT[3]

- ARM Cortex-M0 STM32F051R8T6
- 64 Kbytes of Flash memory
- 8 Kbytes of SRAM
- Running at 8 MHz
- PLL disabled
- Minimize usage of peripherals
- front side decapped
- C implementation



[3] See extended version of original paper

# Evaluation - Attack - Key Schedule

**Settings**

- Temporal:
  - Vary temporal location in 50 ns steps
- Spatial:
  - Area 2.5 mm $\times$ 2.5 mm
  - 0.1 mm per step (675 locations)
  - 108 injections at each coordinate
- EMFI:
  - Discharge voltage 330 V fixed
  - Discharge duration 10 ns fixed

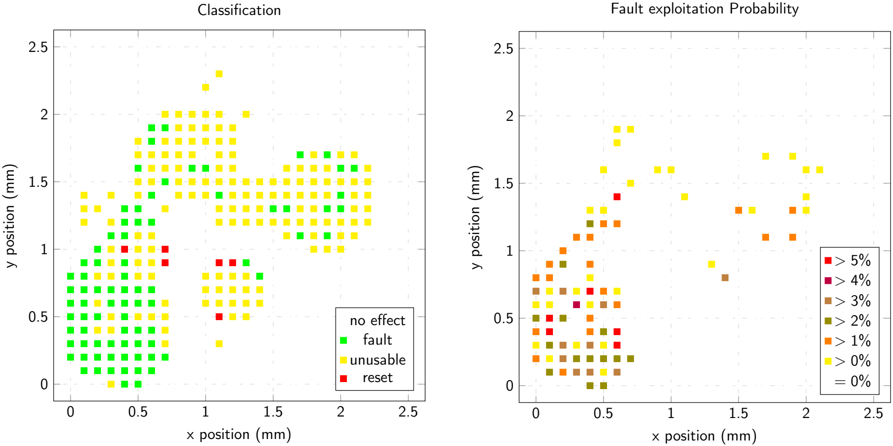
**Classification**

- no effect
- exploitable fault
- unusable
- reset

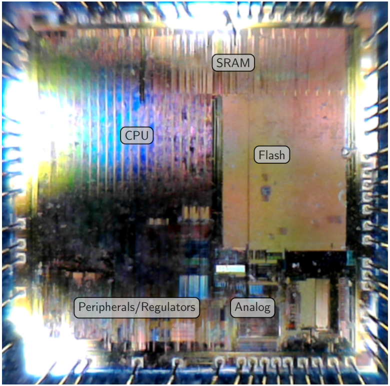
**Fault exploitation probability**

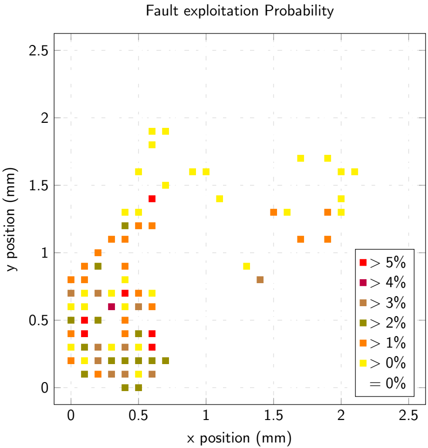
$$P_{exp} = \frac{\#\ \textit{exploitable faults}}{108}$$

# Evaluation - Attack - Key Schedule - Results



# Evaluation - Attack - Key Schedule - Results



# Overview

# Conclusion

- DFA on the **state** of **KLEIN** requires **five** faulty encryptions
- DFA on the **key schedule** of **KLEIN-64** requires **four** faulty encryptions

# Thank you for your attention!

`m.gruber@tum.de`
`https://www.sec.ei.tum.de`

# References


[1] S. S. Ali and D. Mukhopadhyay. “Differential Fault Analysis of AES-128 Key Schedule Using a Single Multi-byte Fault”. In: *Smart Card Research and Advanced Applications*. Springer Berlin Heidelberg, 2011, pp. 50–64.

[2] C.-N. Chen and S.-M. Yen. “Differential Fault Analysis on AES Key Schedule and Some Countermeasures”. In: *Information Security and Privacy*. Springer Berlin Heidelberg, 2003, pp. 118–129.

[3] Z. Gong, S. Nikova, and Y. W. Law. “KLEIN: A New Family of Lightweight Block Ciphers”. In: *RFID. Security and Privacy*. Springer Berlin Heidelberg, 2012, pp. 1–18.

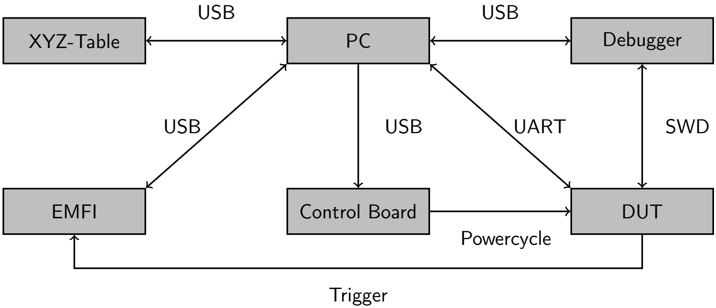
[4] C. H. Kim. “Improved Differential Fault Analysis on AES Key Schedule”. In: *IEEE Transactions on Information Forensics and Security* 7.1 (2012), pp. 41–50.

[5] D. Mukhopadhyay. “An Improved Fault Based Attack of the Advanced Encryption Standard”. In: *Progress in Cryptology – AFRICACRYPT 2009*. Springer Berlin Heidelberg, 2009, pp. 421–434.

[6] G. Piret and J.-J. Quisquater. “A Differential Fault Attack Technique against SPN Structures, with Application to the AES and Khazad”. In: *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2003, pp. 77–88.

[7] M. Tunstall, D. Mukhopadhyay, and S. Ali. “Differential Fault Analysis of the Advanced Encryption Standard Using a Single Fault”. In: *Information Security Theory and Practice. Security and Privacy of Mobile Devices in Wireless Communication*. Springer Berlin Heidelberg, 2011, pp. 224–233.

# Evaluation - EMFI - Setup



# Modified Representation of KLEIN

**Algorithm 1** KLEIN

$sk^1 \leftarrow KEY$
$STATE \leftarrow PLAINTEXT$
**for** $i = 1$ *to* $R$ **do**
    $AddRoundKey(STATE, sk^i)$
    $SubNibbles(STATE)$
    $RotateNibbles(STATE)$
    $MixNibbles(STATE)$
    $sk^{i+1} \leftarrow KeySchedule(sk^i, i)$
**end for**
$CIPHERTEXT \leftarrow AddRoundKey(STATE, sk^{R+1})$

**Algorithm 2** KLEIN modified

$sk^1 \leftarrow KEY$
$STATE \leftarrow PLAINTEXT$
**for** $i = 1$ *to* $R - 1$ **do**
    $AddRoundKey(STATE, sk^i)$
    $SubBytes(STATE)$
    $RotateBytes(STATE)$
    $MixBytes(STATE)$
    $sk^{i+1} \leftarrow KeySchedule(sk^i, i)$
**end for**
$AddRoundKey(STATE, sk^R)$
$SubBytes(STATE)$
$RotateBytes(STATE)$
$sk^{R+1} \leftarrow KeySchedule(sk^R, R)$
$AddRoundKey(STATE, invMixBytes(sk^{R+1}))$
$CIPHERTEXT \leftarrow MixBytes(STATE)$