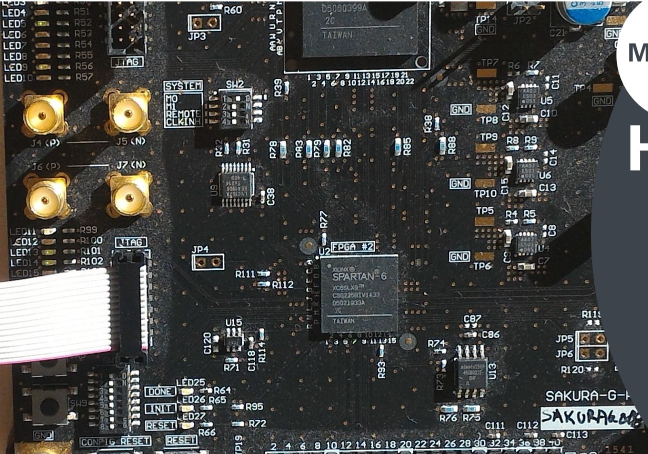
Universität Stuttgart



Maël Gay

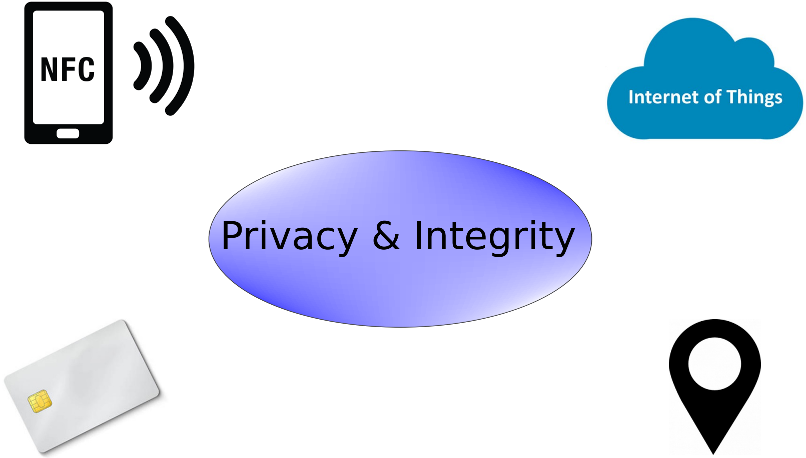
AutoFault: Hardware-Oriented AFA Framework Multiple Fault Support

Table of Contents

Introduction

1

Introduction



Cryptographic Primitives

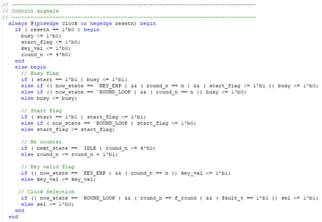
- Cryptographic primitives

Cryptographic Primitives

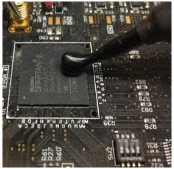

```
// -----------------------------------------------------------------------------
// Control signals
// -----------------------------------------------------------------------------
  always @(posedge clock or negedge resetn) begin
    if ( resetn == 1'b0 ) begin
      busy <= 1'b0;
      start_flag <= 1'b0;
      key_val <= 1'b0;
      round_n <= 4'h0;
    end
    else begin
      // Busy flag
      if ( start == 1'b1 ) busy <= 1'b1;
      else if (( now_state == `KEY_EXP ) && ( round_n == n ) && ( start_flag != 1'b1 )) busy <= 1'b0;
      else if (( now_state == `ROUND_LOOP ) && ( round_n == n )) busy <= 1'b0;
      else busy <= busy;

      // Start flag
      if ( start == 1'b1 ) start_flag <= 1'b1;
      else if ( now_state == `ROUND_LOOP ) start_flag <= 1'b0;
      else start_flag <= start_flag;

      // Nr counter
      if ( next_state == `IDLE ) round_n <= 4'h0;
      else round_n <= round_n + 1'b1;

      // Key valid flag
      if (( now_state == `KEY_EXP ) && ( round_n == n )) key_val <= 1'b1;
      else key_val <= key_val;

     // Clock Selection
      if (( now_state == `ROUND_LOOP ) && ( round_n == f_round ) && ( fault_t == 1'b1 )) sel <= 1'b1;
      else sel <= 1'b0;
    end
  end
```

- Hardware design -> physical restrictions

Cryptographic Primitives

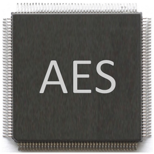


```
// ----------------------------------------------------------------------------
// Control signals
// ----------------------------------------------------------------------------
 always @(posedge clock or negedge resetn) begin
   if ( resetn == 1'b0 ) begin
     busy <= 1'b0;
     start_flag <= 1'b0;
     key_val <= 1'b0;
     round_n <= 4'h0;
   end
   else begin
     // Busy flag
     if ( start == 1'b1 ) busy <= 1'b1;
     else if (( now_state == `KEY_EXP ) && ( round_n == n ) && ( start_flag != 1'b1 )) busy <= 1'b0;
     else if (( now_state == `ROUND_LOOP ) && ( round_n == n )) busy <= 1'b0;
     else busy <= busy;

     // Start flag
     if ( start == 1'b1 ) start_flag <= 1'b1;
     else if ( now_state == `ROUND_LOOP ) start_flag <= 1'b0;
     else start_flag <= start_flag;

     // Nr counter
     if ( next_state == `IDLE ) round_n <= 4'h0;
     else round_n <= round_n + 1'b1;

     // Key valid flag
     if (( now_state == `KEY_EXP ) && ( round_n == n )) key_val <= 1'b1;
     else key_val <= key_val;

    // Clock Selection
     if (( now_state == `ROUND_LOOP ) && ( round_n == f_round ) && ( fault_t == 1'b1 )) sel <= 1'b1;
     else sel <= 1'b0;
   end
 end
```

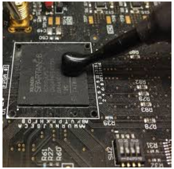


- Vulnerabilities, especially if physical access is allowed

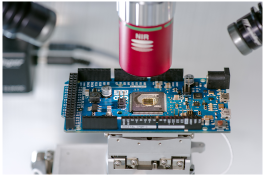
Cryptographic Primitives




```
// ---------------------------------------------------------------------------
// Control signals
// ---------------------------------------------------------------------------
  always @(posedge clock or negedge resetn) begin
    if ( resetn == 1'b0 ) begin
      busy <= 1'b0;
      start_flag <= 1'b0;
      key_val <= 1'b0;
      round_n <= 4'h0;
    end
    else begin
      // Busy flag
      if ( start == 1'b1 ) busy <= 1'b1;
      else if (( now_state == `KEY_EXP ) && ( round_n == n ) && ( start_flag != 1'b1 )) busy <= 1'b0;
      else if (( now_state == `ROUND_LOOP ) && ( round_n == n )) busy <= 1'b0;
      else busy <= busy;

      // Start flag
      if ( start == 1'b1 ) start_flag <= 1'b1;
      else if ( now_state == `ROUND_LOOP ) start_flag <= 1'b0;
      else start_flag <= start_flag;

      // Nr counter
      if ( next_state == `IDLE ) round_n <= 4'h0;
      else round_n <= round_n + 1'b1;

      // Key valid flag
      if (( now_state == `KEY_EXP ) && ( round_n == n )) key_val <= 1'b1;
      else key_val <= key_val;

     // Clock Selection
      if (( now_state == `ROUND_LOOP ) && ( round_n == f_round ) && ( fault_t == 1'b1 )) sel <= 1'b1;
      else sel <= 1'b0;
    end
  end
```

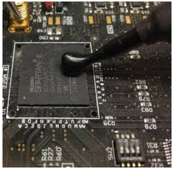


- Side-channel attacks: DPA, DFA...

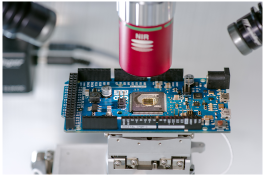


Cryptographic Primitives

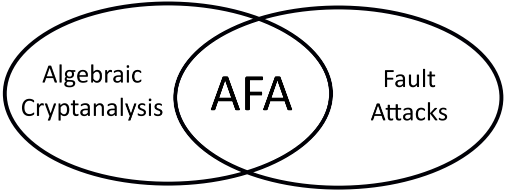

```
// ------------------------------------------------------------------------------
// Control signals
// ------------------------------------------------------------------------------
  always @(posedge clock or negedge resetn) begin
    if ( resetn == 1'b0 ) begin
      busy <= 1'b0;
      start_flag <= 1'b0;
      key_val <= 1'b0;
      round_n <= 4'h0;
    end
    else begin
      // Busy flag
      if ( start == 1'b1 ) busy <= 1'b1;
      else if (( now_state == `KEY_EXP ) && ( round_n == n ) && ( start_flag != 1'b1 )) busy <= 1'b0;
      else if (( now_state == `ROUND_LOOP ) && ( round_n == n )) busy <= 1'b0;
      else busy <= busy;

      // Start flag
      if ( start == 1'b1 ) start_flag <= 1'b1;
      else if ( now_state == `ROUND_LOOP ) start_flag <= 1'b0;
      else start_flag <= start_flag;

      // Nr counter
      if ( next_state == `IDLE ) round_n <= 4'h0;
      else round_n <= round_n + 1'b1;

      // Key valid flag
      if (( now_state == `KEY_EXP ) && ( round_n == n )) key_val <= 1'b1;
      else key_val <= key_val;

     // Clock Selection
      if (( now_state == `ROUND_LOOP ) && ( round_n == f_round ) && ( fault_t == 1'b1 )) sel <= 1'b1;
      else sel <= 1'b0;
    end
  end
```



- Focus: Algebraic Fault Attacks (AFA)



Algebraic Fault Attacks



- Input: description of the cipher, fault model & faulty values
- AFA frameworks:
  - Fault propagation and evaluation of the reduction of the key space
  - Solver that feeds functional description of the cipher and fault model to a SAT solver

AutoFault Framework Summary

- Objectives: automatic construction of fault attacks & evaluation of hardware implementations of cryptographic primitives
- Our framework focuses on:
  - Checking for vulnerabilities throughout each phase of the design
  - Evaluation of possible countermeasures
- Hardware description of the cipher as input
- Differences compared to previous frameworks:
  - Multiple fault injections
  - Different fault models
  - Support several SAT solvers
  - Speed-up of several orders of magnitude
- Easily repeatable for any changes in the hardware implementation
- May also be used to find new attacks

AutoFault Framework

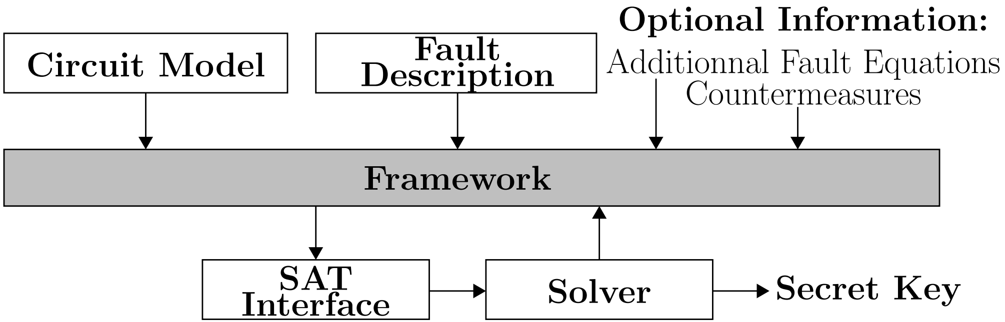
2

Framework Features

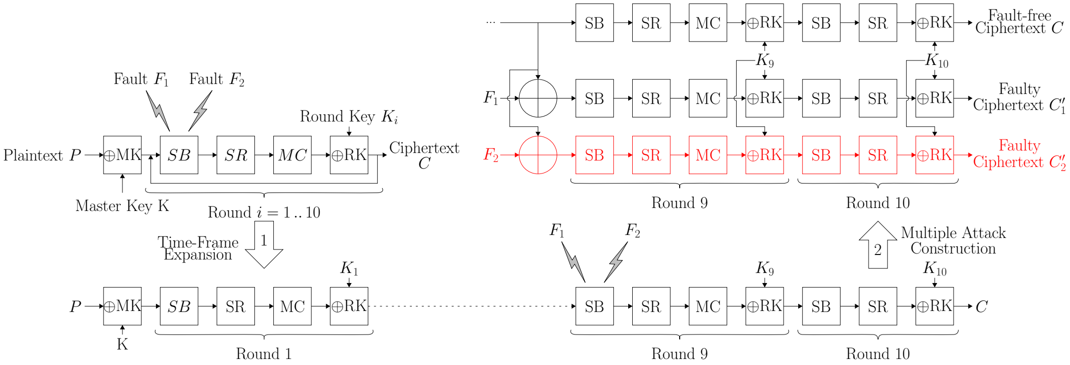
- Features:
  - Pre-silicon analysis
  - Post-silicon analysis
  - Validation of countermeasures

- Tools
  - Automated attack constructor **with multiple fault support**
  - Hardware to CNF converter
  - SAT solver interface
  - Attack simulation

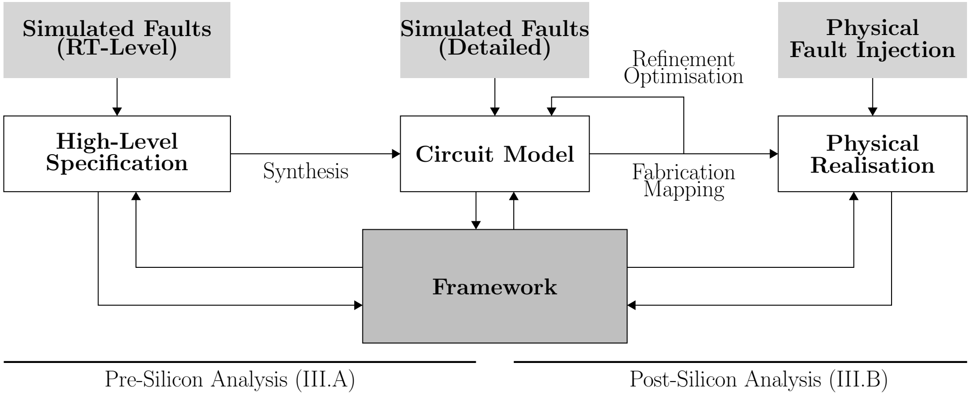
Overall Structure



Attack Construction



AutoFault during design flow

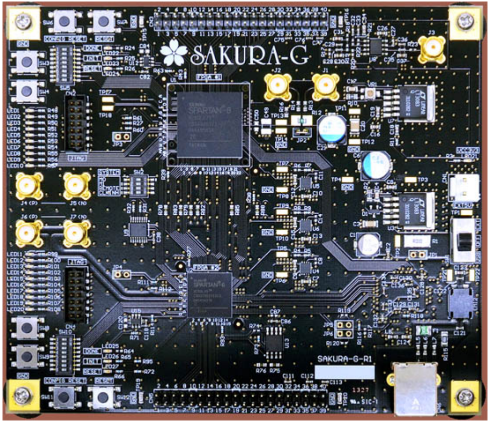


Experimental Results

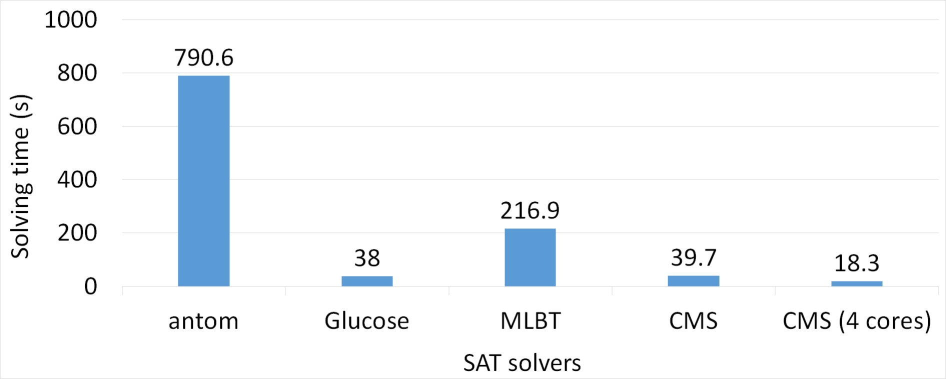
3

Experimental Setup

- Intel Xeon processor with 4 cores at 3.3GHz
- SAKURA-G FPGA board
- Focus on Substitution and Permutation Network (SPN) ciphers



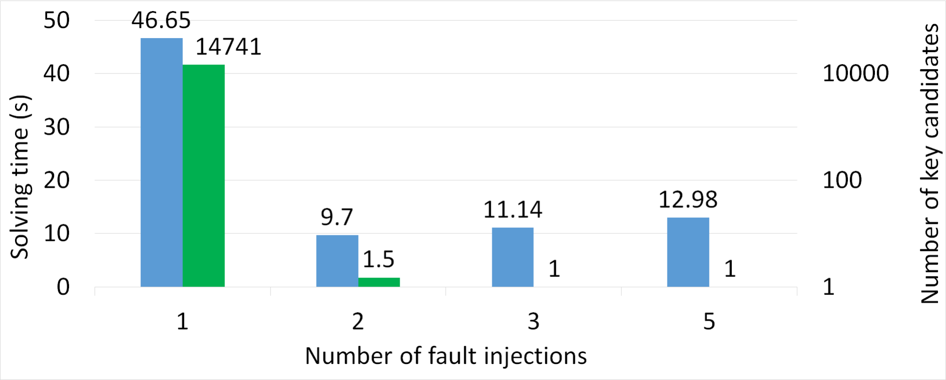
SAT Solver Comparison



Average solving times comparison between different SAT solver

- Small scale AES 444 & 2 fault injections
- MLBT: MapleLCMDistChronoBT & CMS: CryptoMiniSAT
- CMS was the best: in some instances, 2 orders of magnitude faster

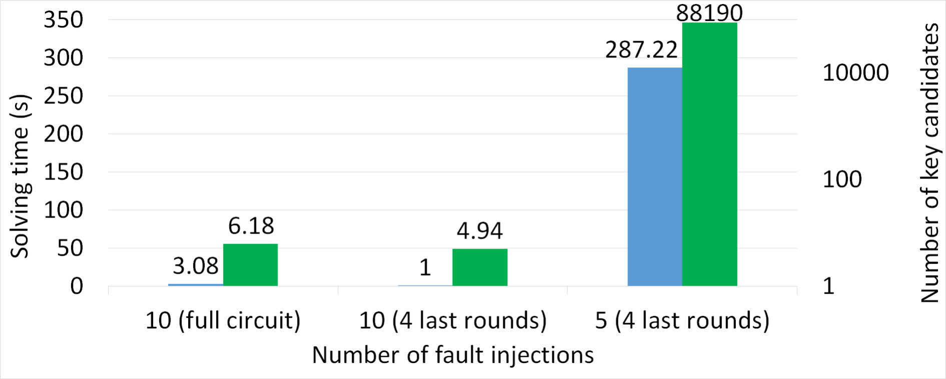
Multiple Fault Impact on LED Cipher



Average solving times and number of key candidates
LED64 using CMS (4 cores)

- Two fault injections is the most efficient, as more faults lead to larger solving times

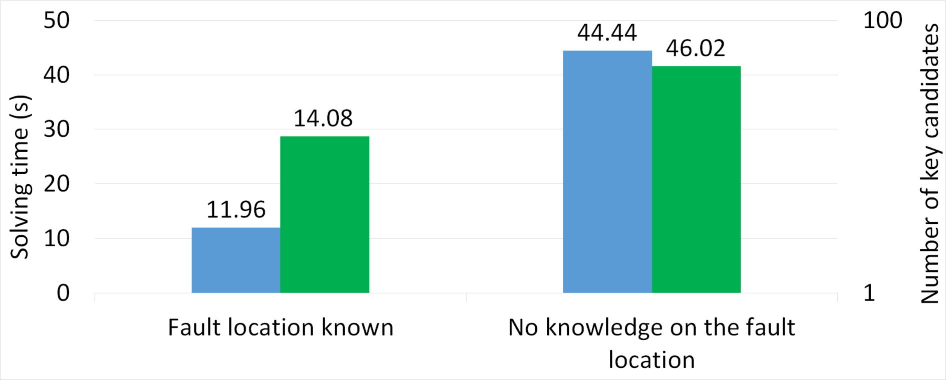
PRESENT Cipher



Average solving times and number of key candidates
PRESENT using CMS (4 cores)

- Successful attack with multiple faults
- More efficient to use only the relevant truncated circuit with multiple faults

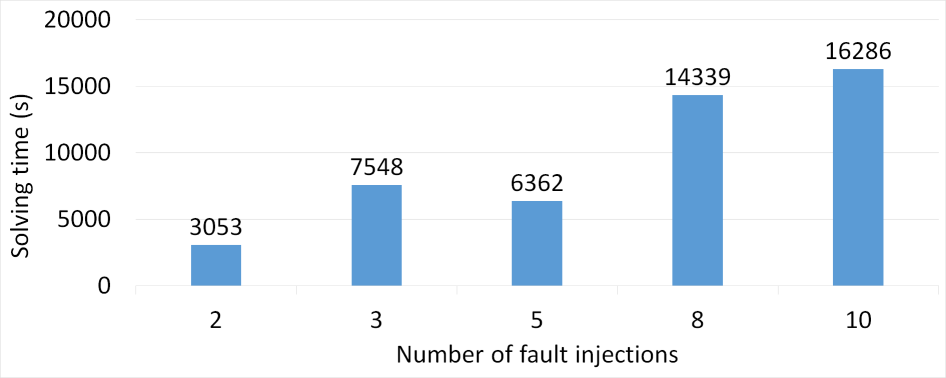
Unknown Nibble Attack Scenario on Small Scale AES



Average solving times and number of key candidates
Small Scale AES 224 using CMS (4 cores)

- AutoFault is able to solve without any fault location knowledge (longer runtime)

Application of AutoFault to a Full Scale AES



Average solving times
Full Scale AES using CMS (4 cores)

- With a single fault: 1 successful run (16 days)
- The support of multiple faults allows to solve for the full scale AES

Conclusion

4

Comparison with Other AFA Frameworks

AFA solver	Cipher description	Multiple faults support reported	Results for ciphers
XFC	Functional	no	AES, CLEFIA, SMS4
Saha et al.	Functional	no	AES, PRESENT
Zhang et al.	Functional	yes	Piccolo, AES, DES, MIBS-64 LED, PRESENT, Twofish
Zhao et al.	Functional	no	LED
AutoFault (2017)	hardware-oriented	no	Small-scale AES, LED
AutoFault (2019)	**hardware-oriented**	**yes**	**AES, LED, PRESENT**

Conclusion

- Evaluation of cryptographic implementation at multiple stages of the design
- Various fault model supported
- Support for multiple faults (attacks on AES & PRESENT)
- Successful attack on full scale AES

- Future work:
  - Impact of different countermeasures
  - Combine with side-channel analysis
  - Different solvers (algebraic)
  - Different class of cryptosystem (ECC, Post Quantum)

Universität Stuttgart

Maël Gay
Institut für Technische Informatik (ITI), Universität Stuttgart

eMail mael.gay@informatik.uni-stuttgart.de
Telefon +49 711 685 88290
Fax +49 711 685 88288