מכון ויצמן למדע
WEIZMANN INSTITUTE OF SCIENCE

RUHR UNIVERSITÄT BOCHUM
RUB

Experimentally Verifying a Complex Algebraic Attack on the Grain-128 Cipher Using Dedicated Reconfigurable Hardware

SHARCS 2012 – Washington D.C.

Itai Dinur[1], Tim Güneysu[2], Christof Paar[2], Adi Shamir[1], and Ralf Zimmermann[2]

[1] Computer Science Dept., The Weizmann Institute, Israel
[2] Horst Görtz Institute for IT Security, Ruhr-University Bochum

18.03.2012

Outline

RUHR
UNIVERSITÄT
BOCHUM
RUB
מכון ויצמן למדע
WEIZMANN INSTITUTE OF SCIENCE

- **Introduction**
- Implementation
- Problems and Solutions
- Results and Conclusion

Introduction

Experimentally Explaining The Title

Experimentally Verifying a Complex Algebraic Attack on the Grain-128 Cipher Using Dedicated Reconfigurable Hardware

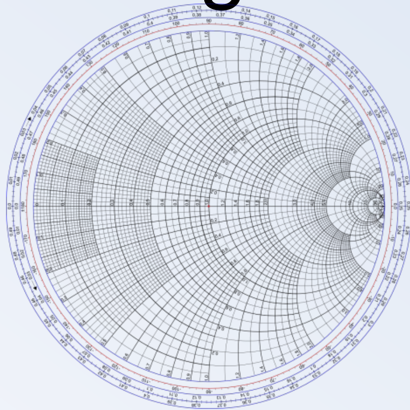
Introduction

Experimentally Explaining The Title

Experimentally Verifying a Complex Algebraic Attack on the Grain-128 Cipher Using Dedicated Reconfigurable Hardware

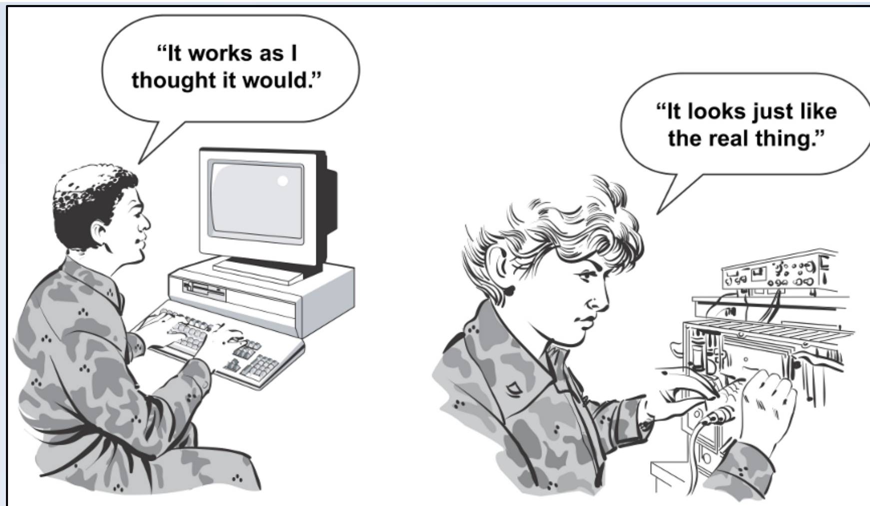
Introduction
Experimentally Explaining The Title

Experimentally Verifying a Complex Algebraic Attack on the Grain-128 Cipher Using Dedicated Reconfigurable Hardware

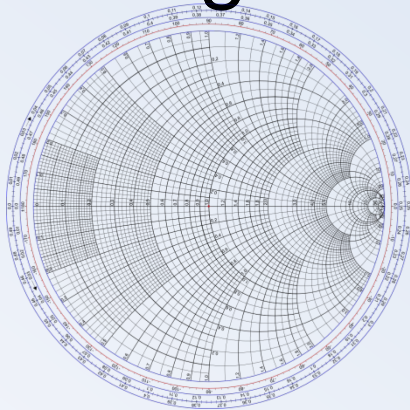


Introduction

Experimentally Explaining The Title



Experimentally Verifying a Complex Algebraic Attack on the Grain-128 Cipher Using Dedicated Reconfigurable Hardware

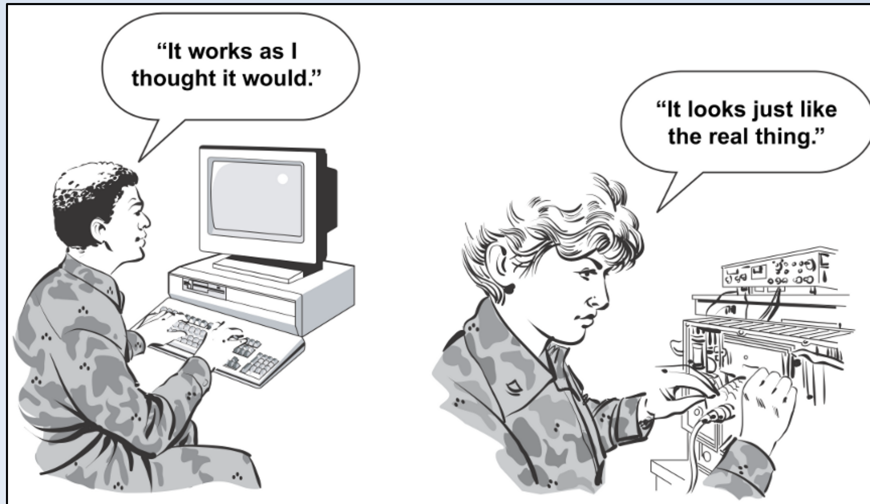


Introduction

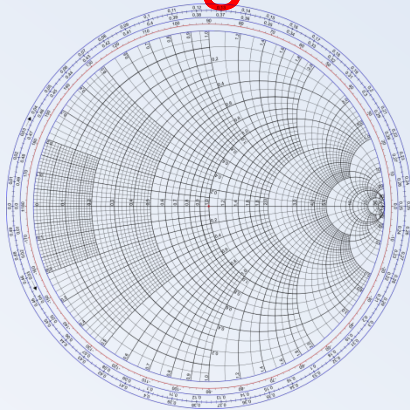
Experimentally Explaining The Title

RUHR UNIVERSITÄT BOCHUM

מכון ויצמן למדע
WEIZMANN INSTITUTE OF SCIENCE



Experimentally Verifying a Complex Algebraic Attack on the Grain-128 Cipher Using Dedicated Reconfigurable Hardware



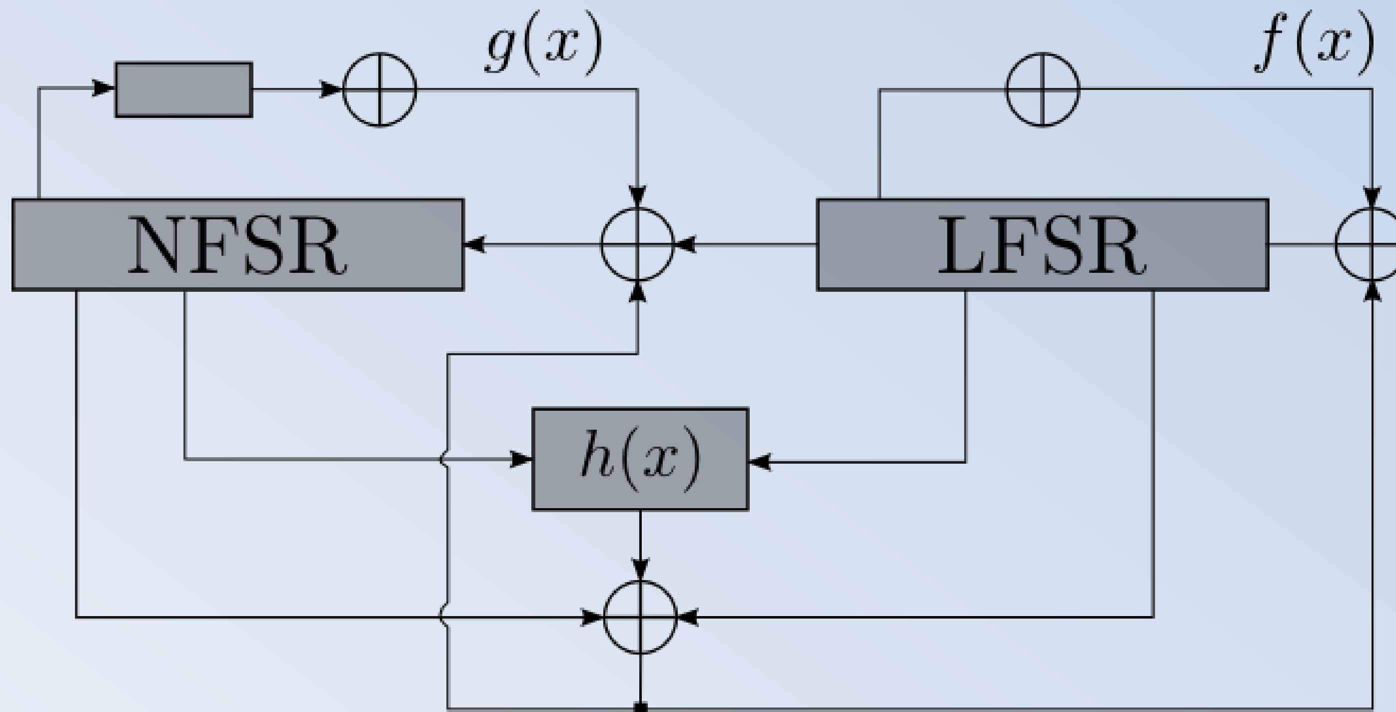
Introduction

Grain-128

RUHR UNIVERSITÄT BOCHUM

מכון ויצמן למדע

WEIZMANN INSTITUTE OF SCIENCE



- 128-bit key, 96-bit IV
- Boolean functions
- 256 clock cycles

Introduction

Cube Attack (very brief ☺)

- Algebraic Attack
  - Dinur/Shamir (FSE 2011), improved (Asiacrypt 2011)
  - Complexity $d*2^{d+e-10}$ (d = 50, e = 39)
  - **Implication: $2^{128} \rightarrow 2^{85}$**

Introduction
Cube Attack (very brief ☺)

- Algebraic Attack
  - Dinur/Shamir (FSE 2011), improved (Asiacrypt 2011)
  - Complexity $d*2^{d+e-10}$ (d = 50, e = 39)
  - **Implication: $2^{128} \rightarrow 2^{85}$**
- Uses CubeTesters
  - Aumasson/Dinur/Meier/Shamir (FSE 2009)
  - Related to higher order differential attacks
  - Distinguishes (special) polynomials from random functions

Introduction
Cube Attack (very brief ☺)

- Algebraic Attack
  - Dinur/Shamir (FSE 2011), improved (Asiacrypt 2011)
  - Complexity $d*2^{d+e-10}$ (d = 50, e = 39)
  - **Implication: $2^{128} \rightarrow 2^{85}$**
- Uses CubeTesters
  - Aumasson/Dinur/Meier/Shamir (FSE 2009)
  - Related to higher order differential attacks
  - Distinguishes (special) polynomials from random functions
- Multiple Steps
  - Guess and generate scores
  - Determine most likely values of secret expression
  - Recover the key

Introduction

Cube Attack - Partial Simulation

- **Motivation**:
  - Attack complexity only estimated
  - Theoretical success probability realistic?

Introduction
Cube Attack - Partial Simulation

- **Motivation**:
  - Attack complexity only estimated
  - Theoretical success probability realistic?

- Simulate *correct guess for known* key
  1. Compute cube summations
  2. Compute score of correct guess
  3. Estimate position in sorted guess list

Introduction

Cube Attack - Partial Simulation

RUHR UNIVERSITÄT BOCHUM

RUB

מכון ויצמן למדע

WEIZMANN INSTITUTE OF SCIENCE

- **Motivation**:
  - Attack complexity only estimated
  - Theoretical success probability realistic?

- Simulate *correct guess for known* key
  1. Compute cube summations
  2. Compute score of correct guess
  3. Estimate position in sorted guess list

Details: Dinur et al. (Asiacrypt 2011)

Outline

- Introduction
- **Implementation**
- Problems and Solutions
- Results and Conclusion

RUHR UNIVERSITÄT BOCHUM

RUB

מכון ויצמן למדע

WEIZMANN INSTITUTE OF SCIENCE

- **Considerations**

- **Flexibility**

- **Operability**

Implementation

Hardware Design Goals

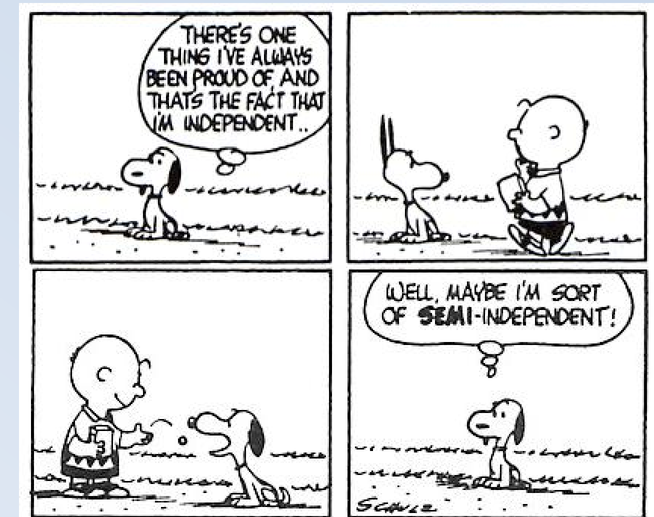
- **Considerations**
  - Needs high performance!
  - Data complexity?
  - Bottlenecks?

- **Flexibility**

- **Operability**

Implementation
Hardware Design Goals

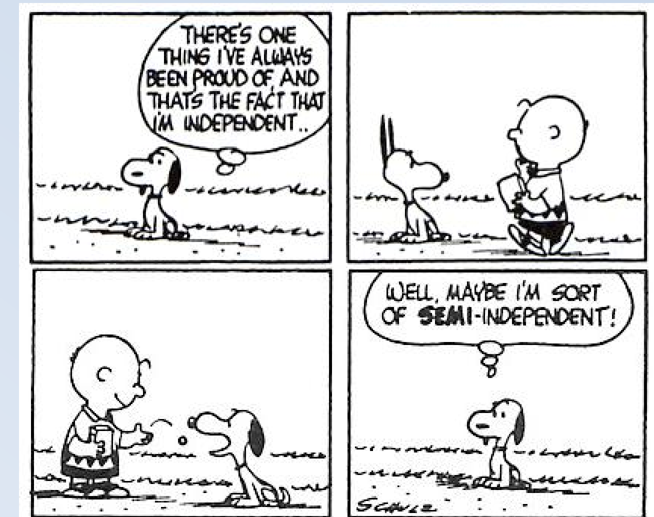
- **Considerations**
  - Needs high performance!
  - Data complexity?
  - Bottlenecks?
- **Flexibility**
  - Adaptable to modified cube attacks
  - Adaptable to modified parameter sets
- **Operability**



Implementation
Hardware Design Goals

- **Considerations**
  - Needs high performance!
  - Data complexity?
  - Bottlenecks?

- **Flexibility**
  - Adaptable to modified cube attacks
  - Adaptable to modified parameter sets



- **Operability**
  - Fully working post-place and route design
  - Fully working on RIVYERA FPGA Cluster

Implementation
RIVYERA Architecture

RUHR
UNIVERSITÄT
BOCHUM

RUB

מכון ויצמן למדע
WEIZMANN INSTITUTE OF SCIENCE

- 8 Spartan-3 5000 FPGAs

Implementation
RIVYERA Architecture

- 8 Spartan-3 5000 FPGAs
- 16 Boards

Implementation
RIVYERA Architecture

RUHR UNIVERSITÄT BOCHUM
RUB
מכון ויצמן למדע
WEIZMANN INSTITUTE OF SCIENCE

- 8 Spartan-3 5000 FPGAs
- 16 Boards
- i7 Processor

Implementation

The Algorithm - Hands-On

RUHR UNIVERSITÄT BOCHUM

RUB

מכון ויצמן למדע

WEIZMANN INSTITUTE OF SCIENCE

Algorithm 2 The Dynamic Cube Attack Simulation

Input: 128-bit key K.
Input: Expressions $e_1, ..., e_{13}$ and the corresponding indexes of the dynamic variable $i_1, ..., i_{13}$.
Input: Big cube $C = (c_1, ..., c_{50})$ containing the indexes of the 50 cube variables.
Output: The score of K.

1: $S \leftarrow (0, ..., 0)$ ▷ the 51 cube boolean sums, where $S[51]$ is the sum of the big cube
2: $IV \leftarrow (0, ..., 0)$ ▷ as the initial 96-bit IV
3: **for** $j \leftarrow 1$ **to** 13 **do**
4: $e_j \leftarrow eval(e_j, K)$ ▷ Plug the value of the secret key into the expression
5: **end for**
6: **for all** cube indexes CV **from** 0 to 2^{50} **do**
7: **for** $j \leftarrow 1$ **to** 50 **do**
8: $IV[c_j] \leftarrow CV[j]$ ▷ Update IV with the value of the cube variable
9: **end for**
10: **for** $j \leftarrow 1$ **to** 13 **do**
11: $IV[i_j] \leftarrow eval(e_j, IV)$ ▷ Update IV with the evaluation of the dynamic variable
12: **end for**
13: $b \leftarrow$ Grain-128(IV, K) ▷ Calculate the first output bit of Grain-128
14: **for** $j \leftarrow 1$ **to** 50 **do**
15: **if** $CV[j] = 0$ **then**
16: $S[j] \leftarrow S[j] + b \pmod 2$ ▷ Update cube sum
17: **end if**
18: **end for**
19: $S[51] \leftarrow S[51] + b \pmod 2$
20: **end for**
21: $HW \leftarrow 0$
22: **for** $j \leftarrow 1$ **to** 51 **do**
23: **if** $S[j] = 0$ **then**
24: $HW \leftarrow HW + 1.$
25: **end if**
26: **end for**
27: **return** $HW/51$

Implementation

The Algorithm - Hands-On

RUHR UNIVERSITÄT BOCHUM

RUB

מכון ויצמן למדע

WEIZMANN INSTITUTE OF SCIENCE

Algorithm 2 The Dynamic Cube Attack Simulation

Input: 128-bit key K.
Input: Expressions $e_1, ..., e_{13}$ and the corresponding indexes of the dynamic variable $i_1, ..., i_{13}$.
Input: Big cube $C = (c_1, ..., c_{50})$ containing the indexes of the 50 cube variables.
Output: The score of K.

1: $S \leftarrow (0, ..., 0)$ ▷ the 51 cube boolean sums, where $S[51]$ is the sum of the big cube
2: $IV \leftarrow (0, ..., 0)$ ▷ as the initial 96-bit IV
3: **for** $j \leftarrow 1$ **to** 13 **do**
4: $e_j \leftarrow eval(e_j, K)$ ▷ Plug the value of the secret key into the expression
5: **end for**
6: **for all** cube indexes CV **from** 0 to 2^{50}
7: **for** $j \leftarrow 1$ **to** 50 **do**
8: $IV[c_j] \leftarrow CV[j]$ pdate IV with the value of the cube variable
9: **end for**
10: **for** $j \leftarrow 1$ **to** 13 **do**
11: $IV[i_j] \leftarrow eval(e_j, IV)$ e IV with the evaluation of the dynamic variable
12: **end for**
13: $b \leftarrow$ Grain-128(IV, K) ▷ Calculate the first output bit of Grain-128
14: **for** $j \leftarrow 1$ **to** 50 **do**
15: **if** $CV[j] = 0$ **then**
16: $S[j] \leftarrow S[j] + b \pmod 2$ ▷ Update cube sum
17: **end if**
18: **end for**
19: $S[51] \leftarrow S[51] + b \pmod 2$
20: **end for**
21: $HW \leftarrow 0$
22: **for** $j \leftarrow 1$ **to** 51 **do**
23: **if** $S[j] = 0$ **then**
24: $HW \leftarrow HW + 1.$
25: **end if**
26: **end for**
27: **return** $HW/51$

Implementation

The Algorithm - Hands-On

Algorithm 1 Dynamic Cube Attack Simulation (Algorithm 2), Optimized for Implementation

Input: 96 bit integer $baseIV$, cube dimension d, cube $C = \{C_0, \ldots, C_d\}$ with $0 \leq C_i < 96\ \forall C_i \in C$, number of polynomials m, dynamic variable indices $D = \{D_0, \ldots, D_m\}$ with $0 \leq C_i < 96\ \forall D_i \in D$, state bit indices $S = \{S_0, \ldots, S_m\}$ with $0 \leq S_i < 96\ \forall S_i \in S$.

Output: $(d+1)$ bit cubesum s

```
 1: IV ← baseIV
 2: s ← 0.

    Key Selection
 3: Choose random 128 bit key K.
 4: Choose key-dependent polynomials P_j(X) nullifying state bits S_j.

    Computation
 5: for i ← 0 to 2^d − 1 do
 6:     for j ← 0 to d − 1 do
 7:         setBit(IV, C_j, getBit(i, j))
 8:     end for
 9:     for j ← 0 to m − 1 do
10:         setBit(IV, D_j, P_j(i))
11:     end for
12:     ks ← first bit of Grain-128(IV, K) keystream
13:     if ks = 1 then
14:         s ← s ⊕ (1|not(i))
15:     end if
16: end for
17: return s.
```

Implementation

The Algorithm - Hands-On

Algorithm 1 Dynamic Cube Attack Simulation (Algorithm 2), Optimized for Implementation

Input: 96 bit integer $baseIV$, cube dimension d, cube $C = \{C_0, \ldots, C_d\}$ with $0 \leq C_i < 96\ \forall C_i \in C$, number of polynomials m, dynamic variable indices $D = \{D_0, \ldots, D_m\}$ with $0 \leq C_i < 96\ \forall D_i \in D$, state bit indices $S = \{S_0, \ldots, S_m\}$ with $0 \leq S_i < 96\ \forall S_i \in S$.

Output: $(d+1)$ bit cubesum s

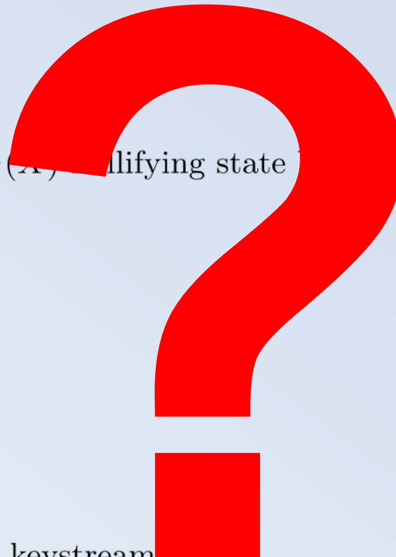
1: IV $\leftarrow$ $baseIV$
2: $s \leftarrow 0$.

Key Selection

3: Choose random 128 bit key K.
4: Choose key-dependent polynomials P_j ... llifying state

Computation

5: **for** $i \leftarrow 0$ **to** $2^d - 1$ **do**
6: **for** $j \leftarrow 0$ **to** $d-1$ **do**
7: SETBIT(IV, C_j, GETBIT(i, j))
8: **end for**
9: **for** $j \leftarrow 0$ **to** $m-1$ **do**
10: SETBIT(IV, D_j, $P_j(i)$)
11: **end for**
12: $ks \leftarrow$ first bit of Grain-128(IV, K) keystream
13: **if** $ks = 1$ **then**
14: $s \leftarrow s \oplus (1|\text{not}(i))$
15: **end if**
16: **end for**
17: **return** s.



Implementation

The Algorithm - Hands-On

RUHR UNIVERSITÄT BOCHUM

RUB

מכון ויצמן למדע

WEIZMANN INSTITUTE OF SCIENCE

- **Focus on time consuming steps**
  1. Chose random key
  2. Generate boolean functions (polynomials to evaluate)
  3. Compute 2^{50} times the first output bit (Grain-128 Initialization)
  4. XOR the results in some way

Implementation
The Algorithm - Hands-On

- **Focus on time consuming steps**
  1. Chose random key
  2. Generate boolean functions (polynomials to evaluate)
  3. Compute 2^{50} times the first output bit (Grain-128 Initialization)
  4. XOR the results in some way

Sounds easy! Let's try it in Software…

Implementation
Software View

- Prepare IV in an array
  - Unfilled: initial IV (unchanged)
  - Red: cube indices
  - Blue: dynamic variables

Example

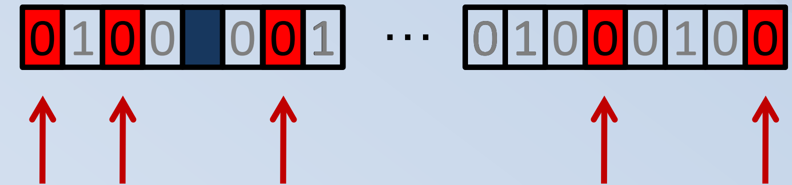
Implementation

Software View

- Prepare IV in an array
  - Unfilled: initial IV (unchanged)
  - Red: cube indices
  - Blue: dynamic variables

- Update the IV:
  - Increment cube indices by 1

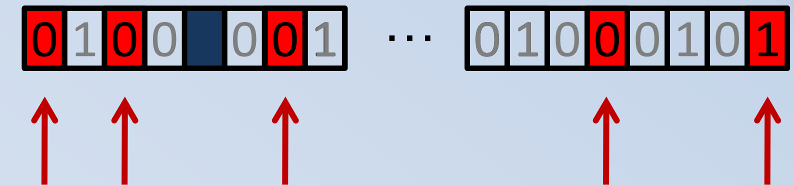
Example



Implementation
Software View

- Prepare IV in an array
  - Unfilled: initial IV (unchanged)
  - Red: cube indices
  - Blue: dynamic variables
- Update the IV:
  - Increment cube indices by 1

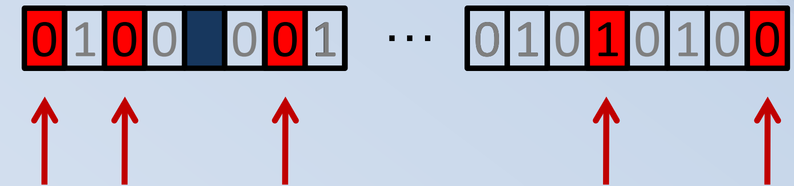
Example



Implementation
Software View

- Prepare IV in an array
  - Unfilled: initial IV (unchanged)
  - Red: cube indices
  - Blue: dynamic variables
- Update the IV:
  - Increment cube indices by 1

Example

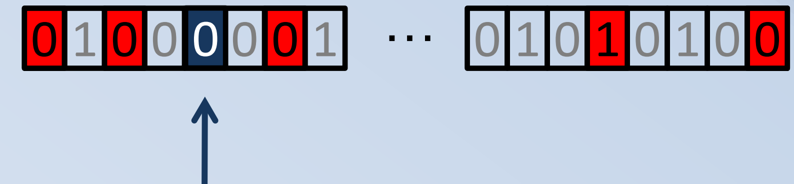


Implementation

Software View

- Prepare IV in an array
  - Unfilled: initial IV (unchanged)
  - Red: cube indices
  - Blue: dynamic variables
- Update the IV:
  - Increment cube indices by 1
  - Evaluate polynomials
- Polynomial Evaluation
  - Loop over all Monomials
  - Simple Array-Lookup

Example



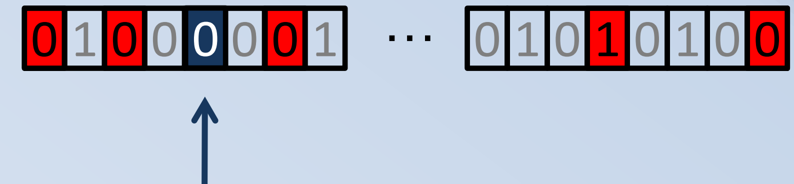
0 and 1 xor 0 and 1 and 1

Implementation

Software View

- Prepare IV in an array
  - Unfilled: initial IV (unchanged)
  - Red: cube indices
  - Blue: dynamic variables
- Update the IV:
  - Increment cube indices by 1
  - Evaluate polynomials
- Polynomial Evaluation
  - Loop over all Monomials
  - Simple Array-Lookup
- But: Very slow in Software (2^{50} Grain iterations per key)

Example



0 and 1 xor 0 and 1 and 1

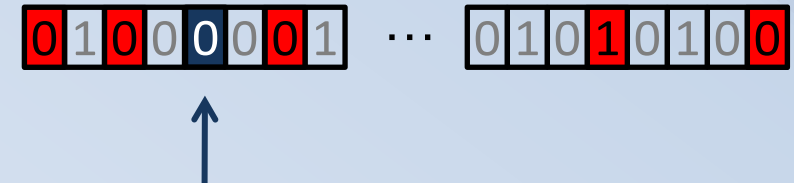
Implementation

Software View

RUHR UNIVERSITÄT BOCHUM
RUB
מכון ויצמן למדע
WEIZMANN INSTITUTE OF SCIENCE

- Prepare IV in an array
  - Unfilled: initial IV (unchanged)
  - Red: cube indices
  - Blue: dynamic variables

Example



- Update the IV:
  - Increment cube indices by 1
  - Evaluate polynomials

0 and 1 xor 0 and 1 and 1

Let's try hardware!

- Polynomial Evaluation
  - Loop over all Monomials
  - Simple Array-Lookup

- But: Very slow in Software (2^{50} Grain iterations per key)

Outline

RUHR UNIVERSITÄT BOCHUM RUB

מכון ויצמן למדע
WEIZMANN INSTITUTE OF SCIENCE

- Introduction
- Implementation
- **Problems and Solutions**
- Results and Conclusion

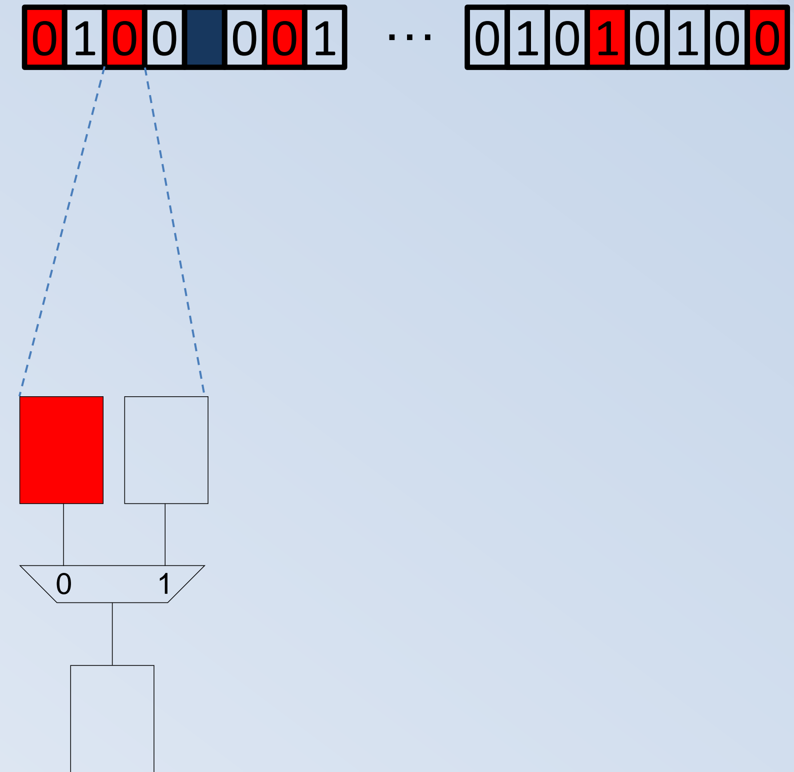
Problems

Flexibility → Feasibility ?

RUHR UNIVERSITÄT BOCHUM

WEIZMANN INSTITUTE OF SCIENCE

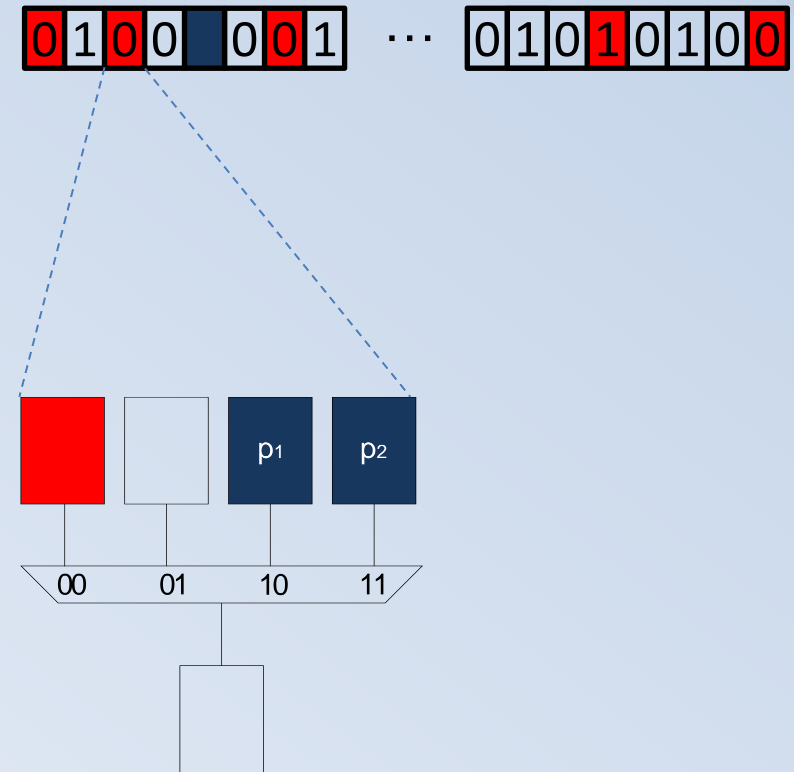
- Multiplex Signals to Register Input
  - Unfilled: initial IV (unchanged)
  - Red: cube indices



Problems

Flexibility → Feasibility ?

- Multiplex Signals to Register Input
  - Unfilled: initial IV (unchanged)
  - Red: cube indices
  - Blue: dynamic variables

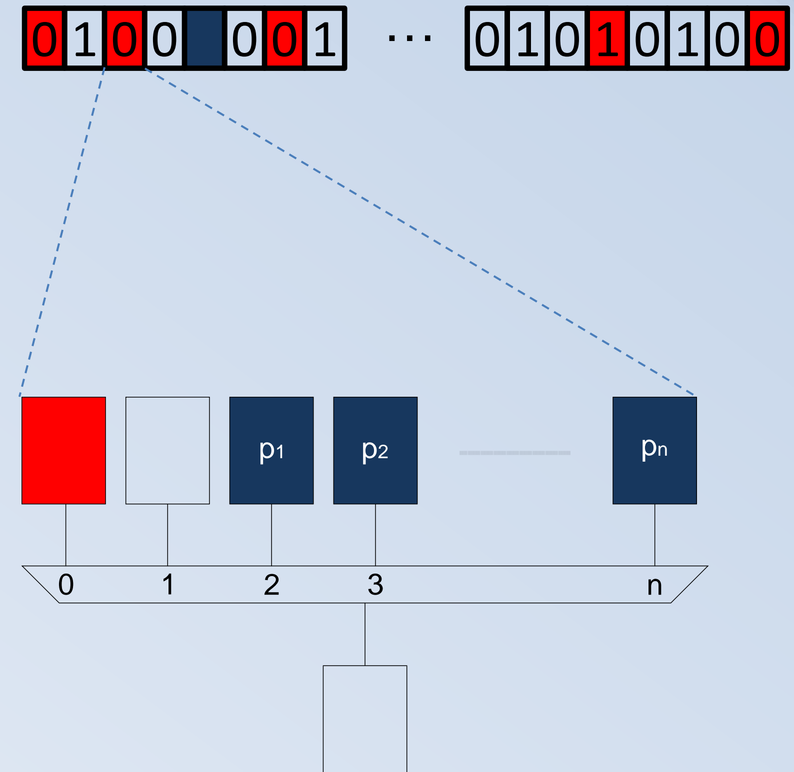


Problems

Flexibility → Feasibility ?

RUHR UNIVERSITÄT BOCHUM

- Multiplex Signals to Register Input
  - Unfilled: initial IV (unchanged)
  - Red: cube indices
  - Blue: dynamic variables

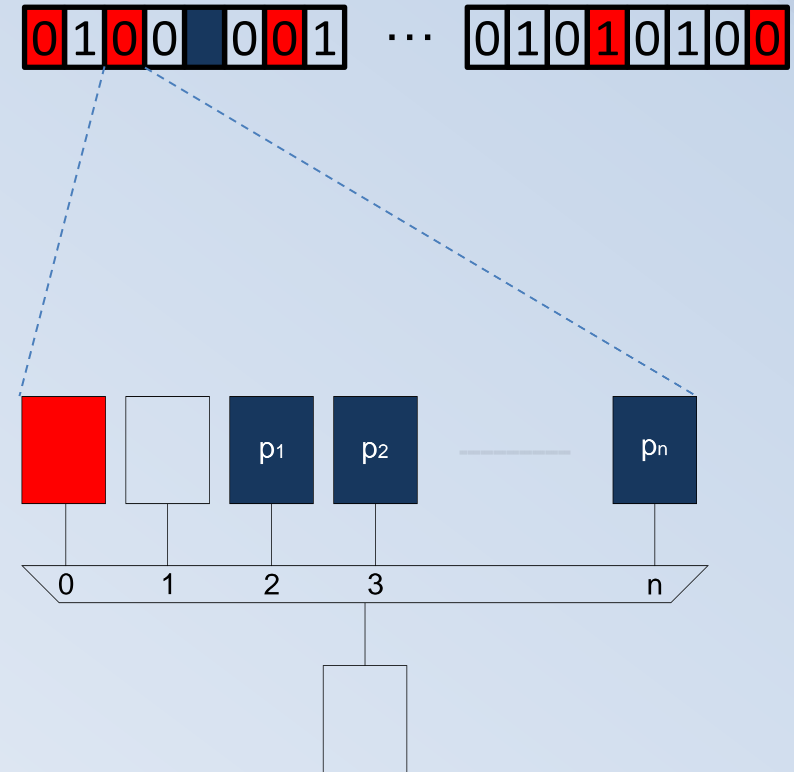


Problems

Flexibility → Feasibility ?

- Multiplex Signals to Register Input
  - Unfilled: initial IV (unchanged)
  - Red: cube indices
  - Blue: dynamic variables

- Updating the IV:
  - HW adder to increment on cube indices
    - 96-bit adder
    - Addition constant depends on positions
  - Evaluate polynomials somehow (?)



Problems

Flexibility → Impossibility

RUHR UNIVERSITÄT BOCHUM

RUB

מכון ויצמן למדע

WEIZMANN INSTITUTE OF SCIENCE

Problems come with polynomials…

- All possible monomials over **d** positions

  → $\sum_{k=1}^{d} \binom{d}{k}$ (with d = 50 → 10^{15})

Problems

Flexibility → Impossibility

RUHR UNIVERSITÄT BOCHUM

מכון ויצמן למדע
WEIZMANN INSTITUTE OF SCIENCE

Problems come with polynomials…

- All possible monomials over **d** positions
  → $\sum_{k=1}^{d} \binom{d}{k}$ (with d = 50 → 10^{15})
- And the d positions are not fixed

Problems

Flexibility → Impossibility

Problems come with polynomials…

- All possible monomials over **d** positions
  → $\sum_{k=1}^{d} \binom{d}{k}$ (with d = 50 → 10^{15})
- And the d positions are not fixed
- Polynomials connect multiple monomials

Problems

Flexibility → Impossibility

RUHR
UNIVERSITÄT
BOCHUM
RUB

מכון ויצמן למדע
WEIZMANN INSTITUTE OF SCIENCE

- We need up to **n** different polynomials…

RUHR UNIVERSITÄT BOCHUM

RUB

מכון ויצמן למדע

WEIZMANN INSTITUTE OF SCIENCE

Solution

- *Locally* fix the **d** positions and **n** polynomials
  - → only needed monomials are computed
  - → no space wasted
  - → no additional multiplexing

Solution

- *Locally* fix the **d** positions and **n** polynomials
  - → only needed monomials are computed
  - → no space wasted
  - → no additional multiplexing

- But: FPGA must be reconfigured for each
  - Parameter-Set
  - Random-Key

Solutions

Flexibility and Feasibility

Solution

- *Locally* fix the **d** positions and **n** polynomials
  - → only needed monomials are computed
  - → no space wasted
  - → no additional multiplexing

- But: FPGA must be reconfigured for each
  - Parameter-Set
  - Random-Key

Solutions
Flexibility and Feasibility

Solution

- *Locally* fix the **d** positions and **n** polynomials
  - → only needed monomials are computed
  - → no space wasted
  - → no additional multiplexing

- But: FPGA must be reconfigured for each
  - Parameter-Set
  - Random-Key

Solutions

Flexibility and Feasibility

Solution

- *Locally* fix the **d** positions and **n** polynomials
  - → only needed monomials are computed
  - → no space wasted
  - → no additional multiplexing

- But: FPGA must be reconfigured for each
  - Parameter-Set
  - Random-Key

Solutions

Top-Level Design

RUHR UNIVERSITÄT BOCHUM

RUB

מכון ויצמן למדע

WEIZMANN INSTITUTE OF SCIENCE

- Multiple workers per FPGA
- Each worker generates IVs locally

Cube Attack System

Worker

Worker

Worker

Worker

d-11

d-11

d-11

7

FPGA number

d

Partial cubesum (System)

FPGA number

IV generator

Grain-128

Grain-128

Grain-128

worksum control

partial cube sum

Solutions

Hardware / Software Process

RUHR UNIVERSITÄT BOCHUM

RUB

מכון ויצמן למדע

WEIZMANN INSTITUTE OF SCIENCE

- Hardware
  - Divides search space
  - Computes 2^{50} **Grain** iterations

Read parameters

Choose random key and generate VHDL constants

Generate FPGA bitstream

Queue bitstream for RIVYERA

Try different strategies

Wait for next bitstream

Program RIVYERA and wait for results

Store results

Solutions

Hardware / Software Process

RUHR UNIVERSITÄT BOCHUM RUB

מכון ויצמן למדע

WEIZMANN INSTITUTE OF SCIENCE

- Hardware
  - Divides search space
  - Computes 2^{50} **Grain** iterations
- Software
  - Generates key-specific VHDL
  - Generates FPGA configuration
  - Reconfigures cluster
  - Fetches results from cluster
  - Computes post-processing

Read parameters

Choose random key and generate VHDL constants

Generate FPGA bitstream

Queue bitstream for RIVYERA

Try different strategies

Wait for next bitstream

Program RIVYERA and wait for results

Store results

Outline

- Introduction
- Implementation
- Problems and Solutions
- **Results and Conclusion**

Results and Conclusion

Getting back to the Cube Attack

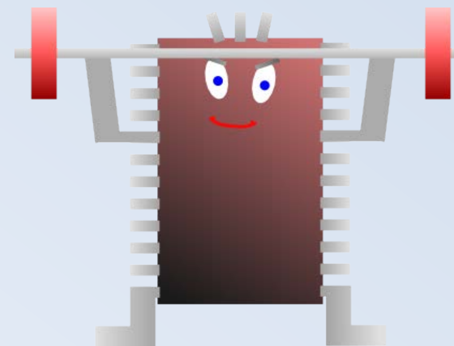
- **Experimentally verified** the main components of the attack
  - More than 150 tests with randomly-chosen keys
  - Each test requires running the initialization process 2^{50} times

Results and Conclusion

Getting back to the Cube Attack

- **Experimentally verified** the main components of the attack
  - More than 150 tests with randomly-chosen keys
  - Each test requires running the initialization process 2^{50} times

- **Software vs Hardware**

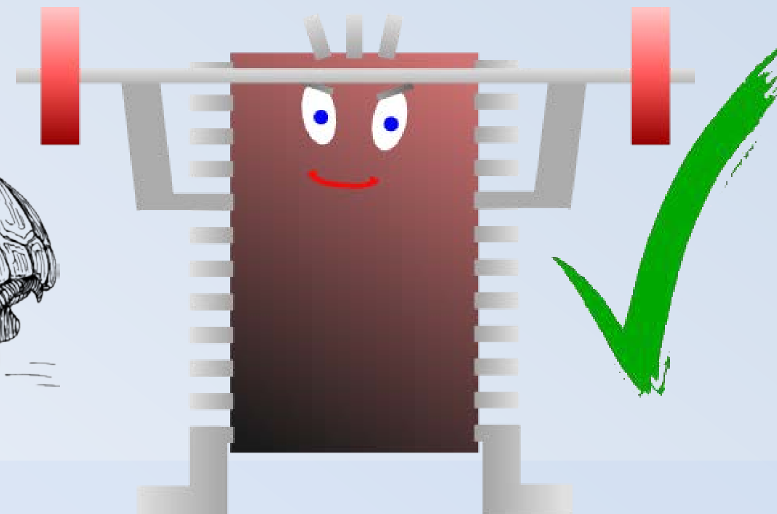


Results and Conclusion

Getting back to the Cube Attack

- **Experimentally verified** the main components of the attack
  - More than 150 tests with randomly-chosen keys
  - Each test requires running the initialization process 2^{50} times

- **Software vs Hardware**



מכון ויצמן למדע
WEIZMANN INSTITUTE OF SCIENCE

RUHR UNIVERSITÄT BOCHUM
RUB

Experimentally Verifying a Complex Algebraic Attack on the Grain-128 Cipher Using Dedicated Reconfigurable Hardware

SHARCS 2012 – Washington D.C.

Itai Dinur[1], Tim Güneysu[2], Christof Paar[2], Adi Shamir[1], and Ralf Zimmermann[2]

[1] Computer Science Dept., The Weizmann Institute, Israel
[2] Horst Görtz Institute for IT Security, Ruhr-University Bochum

18.03.2012

Thank you for your attention!

Any Questions?

Results and Conclusion

Hardware Results

Global Settings	Worker Clock (MHz)
2.4× Input Clk	120
2.2× Input Clk	110
2.0× Input Clk	100
RIVYERA Input Clk	50

Map Settings	Placer Effort	Placer Extra Effort	Register Duplication	Cover Mode
Speed	Normal	None	On	Speed
Area	High	Normal	Off	Area

Place and Route Settings	Overall Effort	Placer Effort	Router Effort	Extra Effort
Fast Build	High	Normal	Normal	None
High Effort	High	High	High	Normal

Cube Dimension d	46			47	50	
Clock Frequency (MHz)	100	110	120	120	110	120
Configurations Built	1	7	8	6	60	93
Percentage	6.25	43.75	50	100	39.2	60.8
Online Phase Duration	17.2 min	15.6 min	14.3 min	28.6 min	4h 10 min	3h 49 min

- Polynomials have high impact
- Building Time up to 8 hours