A Practical Fault Attack on ARX-like Ciphers

With a Case Study on ChaCha20

S V Dilip Kumar[1], Sikhar Patranabis[1], **Jakub Breier**[2], Debdeep Mukhopadhyay[1], Shivam Bhasin[2], Anupam Chattopadhyay[3], and Anubhab Baksi[3]

[1] SEAL, Dept. of Computer Science and Engineering, IIT Kharagpur, India
[2]PACE, Temasek Laboratories, NTU, Singapore
[3]School of Computer Science and Engineering, NTU, Singapore

25 September 2017

Table of Contents

Table of Contents

ChaCha Overview

- Stream cipher published by D. Bernstein in 2008[1].
- Popular thanks to adoption by Google in their TLS implementation.
- It is a variant of Salsa20, with increased level of diffusion.
- Based on ARX (Addition-Rotation-Xor).
- No practical fault analysis published yet.

[1] D. J. Bernstein: ChaCha, a variant of Salsa20, New Stream Cipher Designs - The eSTREAM Finalists, 2008.

ChaCha20 Overview

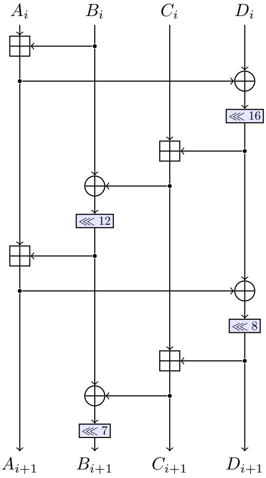


Figure: `Quaterround` function.

$$X = \begin{pmatrix} x_0 & x_1 & x_2 & x_3 \\ x_4 & x_5 & x_6 & x_7 \\ x_8 & x_9 & x_{10} & x_{11} \\ x_{12} & x_{13} & x_{14} & x_{15} \end{pmatrix} = \begin{pmatrix} c_0 & c_1 & c_2 & c_3 \\ k_0 & k_1 & k_2 & k_3 \\ k_4 & k_5 & k_6 & k_7 \\ t_0 & t_1 & v_0 & v_1 \end{pmatrix}$$

$$\begin{aligned} b_0 &= x_0 + x_1, b_3 = (x_3 \oplus b_0) \lll 16 \\ b_2 &= x_2 + b_3, b_1 = (x_1 \oplus b_2) \lll 12 \\ y_0 &= b_0 + b_1, y_3 = (b_3 \oplus y_0) \lll 8 \\ y_2 &= b_2 + y_3, y_1 = (b_1 \oplus y_2) \lll 7 \end{aligned}$$

32-bit words:

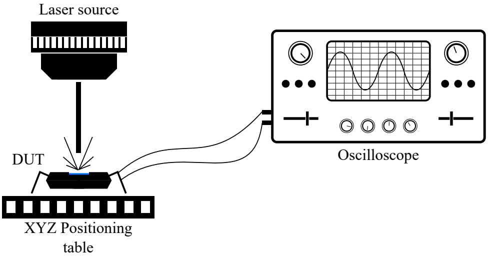
- c_i – constants
- k_i – key
- t_i – counter
- v_i – nonce

ChaCha20 Algorithm

Inputs: Constant C; Key K; Counter T; Nonce V;

X ← InitialState(C,K,T,V)
Y ← X
for $i = 0$ to 9 **do**
    {Column Round}
    $(x_0, x_4, x_8, x_{12}) \leftarrow Quaterround(x_0, x_4, x_8, x_{12})$
    $(x_1, x_5, x_9, x_{13}) \leftarrow Quaterround(x_1, x_5, x_9, x_{13})$
    $(x_2, x_6, x_{10}, x_{14}) \leftarrow Quaterround(x_2, x_6, x_{10}, x_{14})$
    $(x_3, x_7, x_{11}, x_{15}) \leftarrow Quaterround(x_3, x_7, x_{11}, x_{15})$
    {Diagonal Round}
    $(x_0, x_5, x_{10}, x_{15}) \leftarrow Quaterround(x_0, x_5, x_{10}, x_{15})$
    $(x_1, x_6, x_{11}, x_{12}) \leftarrow Quaterround(x_1, x_6, x_{11}, x_{12})$
    $(x_2, x_7, x_8, x_{13}) \leftarrow Quaterround(x_2, x_7, x_8, x_{13})$
    $(x_3, x_4, x_9, x_{14}) \leftarrow Quaterround(x_3, x_4, x_9, x_{14})$
end for
Z ← X + Y
return Z

Experimental Setup



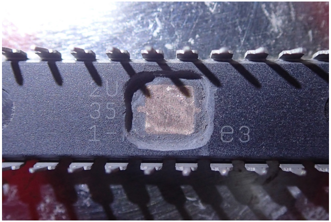
- 1064 nm diode pulse laser with 8 W pulse power.
- $20\times$ magnification lens, resulting to $15 \times 3.5\ \mu m^2$ spot size.
- DUT: 8-bit ATmega328P mounted on Arduino UNO board.

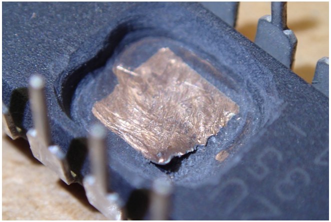
Contributions

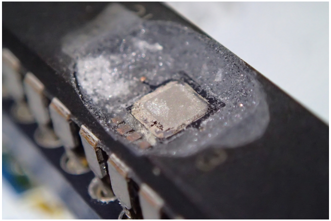
- We demonstrate the first practical attack on ChaCha.
- We target the following operations of the cipher:
  - Final addition operation between X and Y at the end of the keystream generation.
  - 12-bit rotation during `Quaterround` in round 20.
  - Branch-not-equal during `Quaterround` in round 20.

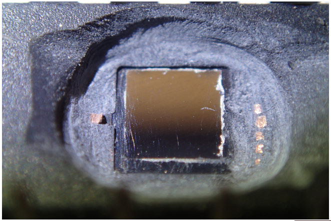
Table of Contents

Mechanical Decapsulation

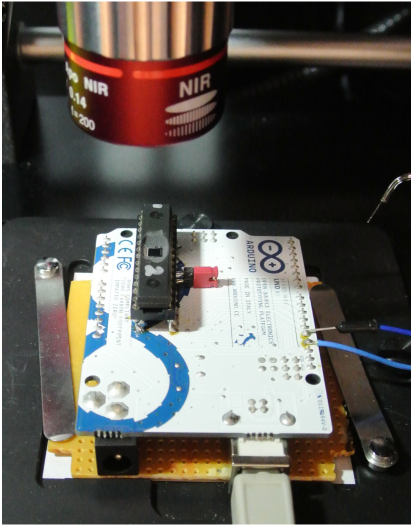








DUT - Arduino Board



Spatial Profiling

- Total die area: $\approx 3 \times 3$ mm^2.
- Step size in each direction: 15 μm, resulting to 40.000 scans.
- Sensitive area of the chip: $\approx 20 \times 55$ μm^2.

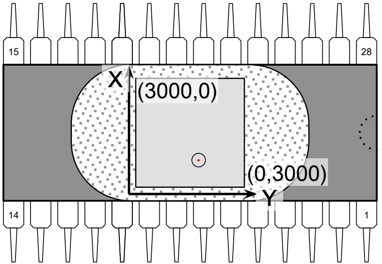


Figure: Sensitive area of ATmega328P.

Temporal Profiling

- One instruction takes 62.5 ns.
- We targeted EOR instructions during the profiling phase.
- We could precisely disturb a single instruction.

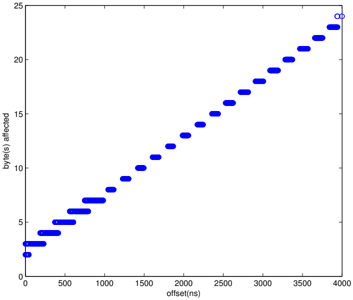
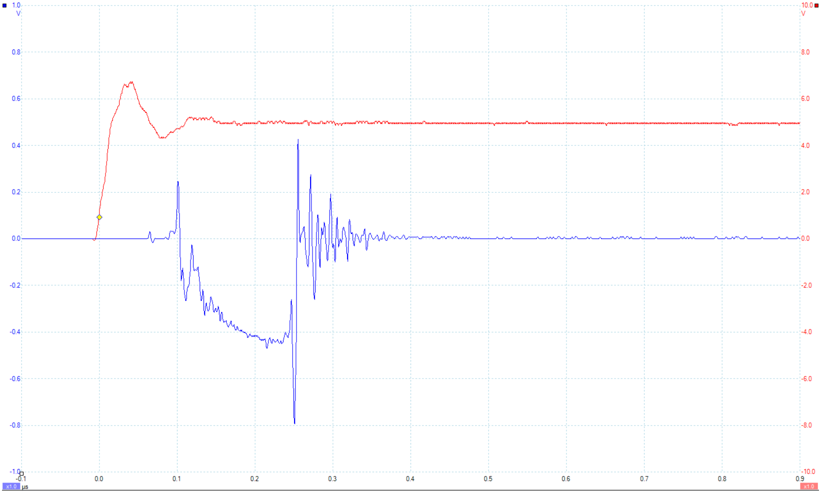


Figure: Disturbance of different instructions w.r.t. time.

Trigger and Laser Response

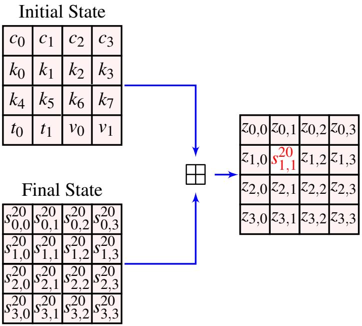


Instruction Changes and “Skips”

- Fault injection changes the instruction opcode on the bus.
- This can result into instruction change (`ADD` $\rightarrow$ `SUB`, `ADC` $\rightarrow$ `SBC`).
- In case the opcode is invalid, instruction decoder will not recognize it – effectively resulting into `NOP`.

Table of Contents

Attack on Final Addition

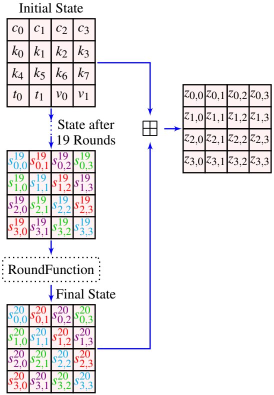


Fault injection skips the function call to 32-bit addition subroutine.

$$z_{1,1} = k_1 + s^{20}_{1,1}$$
$$z'_{1,1} = s^{20}_{1,1}$$
$$k_1 = z_{1,1} - z'_{1,1}$$

Key Words Revealed	Fault Injection Timing (in clock cycles)
k_0	t_0 + 153
k_1	t_0 + 191
k_2	t_0 + 230
k_3	t_0 + 281
k_4	t_0 + 319
k_5	t_0 + 359
k_6	t_0 + 398
k_7	t_0 + 436

Attack on Rotation



- We target 12-bit rotation in each `Quaterround` in round 20.
- Shifted values of the diagonals and knowledge of the first and the last row of the initial matrix helps to reduce the key search space from 2^{256} to 2^{12} with just 4 faults and recovers the key with 8 faults.
- Similarly to previous attack, function calls (`RCALL`) to 32-bit shifts were skipped.

Diagonal Fault Attack

```
ARX%=:
cpi r19, 0x00
brne 40f
...
40:
cpi r19, 0x00
brne 80f
...
80:
cpi r19,0x01
...
83:
cpi r19, 0x04
brne 44f
...
44:
cpi r19, 0x04
brne 84f
...
84:
cpi r19, 0x05
brne 45f
...
45:
cpi r19, 0x05
brne 85f
...
85:
cpi r19, 0x06
brne 46f
...
46:
cpi r19, 0x06
brne 86f
...
86:
cpi r19, 0x07
brne 47f
...
47:
cpi r19, 0x07
brne 87f
...
87:
rjmp ARXExit%=
```

Column 1 : (0,4,8,12)

Diagonal 1 : (0,5,10,15)

Diagonal 2 : (1,6,11,12)

Diagonal 3 : (2,7,8,13)

Diagonal 4 : (3,4,9,14)

- Attack takes advantage of the fact that operation on any diagonal involves a series of `CPI - BRNE` instructions.
- By carefully skipping certain branch instructions, we can force some operations on diagonals to be executed twice.
- 5 fault injections can recover the whole 256-bit key.

Instruction Replacement Attack

- Targets the four 8-bit additions after the last round (final addition).
- Platform dependent, taking advantage of the fact that:
  - `ADD` + fault → `SUB`
  - `ADC` + fault → `SBC`
- For each 32-bit key word, we need 4 faults, resulting to 32 fault in total to recover the whole key.

Table of Contents

Conclusion

Table: Summary of Fault Attack Results on ChaCha.

Attack Type	# of Fault Injections	Key Space
Attack on final Addition	8	1
Attack on Rotation	8	1
Diagonal Fault Attack	3	2^{64}
	5	1
Instruction Replacement	32	1

- While some faults models can be made harder by unrolling the implementation, it would only increase the required number of faults.
- Final addition is the main reason these attacks work.

Thank you!

Any questions?