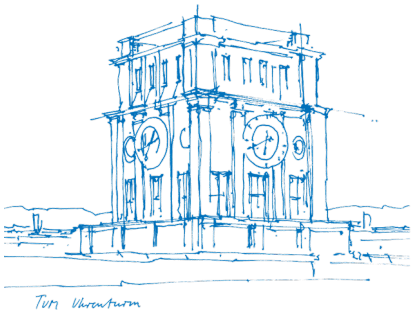
Algebraic Fault Analysis of Subterranean 2.0

Michael Gruber, Patrick Karl, and Georg Sigl

Technical University of Munich,
Department of Electrical and Computer Engineering,
Chair of Security in Information Technology

FDTC 2021 - virtual, 17.10.2021



Overview

Motivation

- Demand for lightweight ciphers, e.g. for IoT
  - → Low algebraic degree
  - → Susceptible to Algebraic Fault Analysis

- Algebraic Fault Analysis (AFA)
  - → Analytical/Differential type of Fault Analysis
  - → Model a fault attack as an equation system
  - → Automated solving

→ This work: Breaking Subteranean 2.0 with AFA

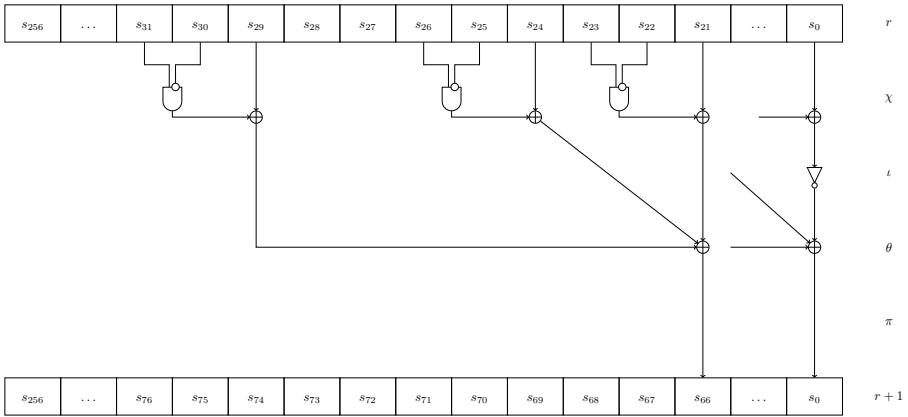
Overview

Subterranean 2.0

- Permutation reused from J. Daemens PhD thesis [3]
- Optimized for hardware implementations
- 2nd round NIST LWC candidate [4]
- Unfortunately not a finalist
- Stream cipher
  - → Low degree ($GF(2)$)
  - → 257-bit state
  - → 128-bit key
  - → non-linear feedback (round function)
- Sponge-like duplex:
  - → Absorb: 33-bit input rate σ
  - → Squeeze: 32-bit output rate (keystream)

$$R = \pi \circ \theta \circ \iota \circ \chi \tag{1}$$

Round Function



Authenticated Encryption Scheme

Algorithm Subterranean SAE

function SUBTERRANEAN-SAE(K, N, AD, PT)

$s \leftarrow 0^{257}$ ▷ Initialization
s.absorb(K)
s.absorb(N)
s.blank(8)

s.absorb(AD) ▷ Processing
$CT \leftarrow$ s.absorb(PT)

s.blank(8) ▷ Finalization
$T \leftarrow$ s.squeeze_tag()

return (CT, T)
end function

Overview

1. Collecting Faulty Outputs

Fault model:

- Fault position/clock cycle p
- Fault location/state index l
- Fault width w
- Bitflip / random bitflip

$\rightarrow$ Adding a w-bit mask to state index l at end of cycle p

$\rightarrow$ Extended Python reference implementation

2. Generate Cipher Equations

Initialization:

$$s_i = 0$$

Data Absorption:

$$s_i^{t_0} = s_i^p + s_{i+1}^p s_{i+2}^p + s_{i+2}^p + \delta_i$$
$$s_i^{t_1} = s_{12i}^{t_0} + s_{12i+3}^{t_0} + s_{12i+8}^{t_0}$$

$$s_i^{p+1} = \begin{cases} s_i^{t_1} + \sigma_j & \text{, if } i \text{ input position} \\ s_i^{t_1} & \text{, otherwise} \end{cases}$$

Encryption of 32-bit word of PT:

$$ct_i = pt_i + s_{12^4 i} + s_{-12^4 i}$$

Squeezing 32-bit word of Tag:

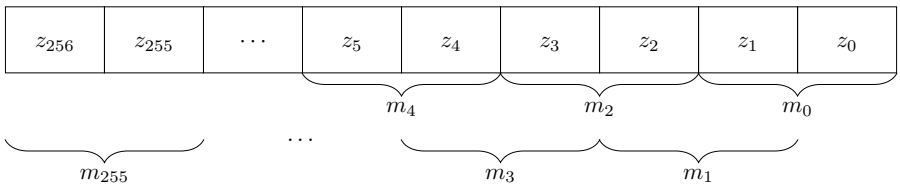
$$t_i = s_{12^4 i} + s_{-12^4 i}$$

3. Generate Fault Equations

- Zhang et al.: Generalized framework [7]
  - → Intermediate state difference
  - → Information on fault model
  - → Equations from fault injection to end of computation

- State difference: $z = s + s^*$

- Split up z into blocks of width w



4. Solve the Equation System

- BOSPHORUS [2] for preprocessing
  - → ANF-CNF conversion
  - → iterative simplification

- CryptoMiniSat5 [6] for solving
  - → conflict driven SAT solver
  - → native XOR support

Overview

Results

1. **Fault Model Evaluation**:
   - Choosing fault location l
   - Optimal fault position p
   - Influence of fault width w

2. **Equation System Optimization**

3. **Measurement Assumptions**

   - Empty message considered
   - 20 instances per measurement, 12 hours
   - Results consider average over all instances

4. **Comparison with Trivium**

Results

Tool	Parameter	CNF	
		non-optimized	optimized
BOSPHORUS	Variables	$70,942$	$71,849$
	Clauses	$440,229$	$178,000$
	Time	2.10 s	$\approx$ 4.5 h
CryptoMiniSat [6]	Time	$>$ 48 h	3.83 s
	Total Time	$>$ 48 h	$\approx$ 4.5 h

Table: Comparison of CNFs with 20 fault injections

$\rightarrow$ Significant benefit when optimizing during preprocessing

Comparison with Trivium

- Trivium [1]: lightweight, hardware-centric stream cipher
- Mohamed et al. [5]
- Careful comparison due to different structure/toolchain/CPU

	Trivium [5]	**Subterranean 2.0**
State size [bit]	288	257
Key size [bit]	80	128
	AFA Results	
No. Faults	2	5
No. Key bits	$420 - 800$	128
Solving time [s]	$0.127 - 138.653$	3.45

Table: Comparison of Trivium and Subterranean 2.0

Thank you for your attention!

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

References

[1] C. D. Cannière. “Trivium: A Stream Cipher Construction Inspired by Block Cipher Design Principles”. In: *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2006, pp. 171–186.

[2] D. Choo et al. “BOSPHORUS: Bridging ANF and CNF Solvers”. In: (Dec. 11, 2018). arXiv: http://arxiv.org/abs/1812.04580v1 [cs.LO].

[3] J. Daemen. “Cipher and hash function design strategies based on linear and differential cryptanalysis”. PhD thesis. Doctoral Dissertation, March 1995, KU Leuven, 1995.

[4] J. Daemen, P. M. C. Massolino, and Y. Rotella. *The Subterranean 2.0 cipher suite*. https://csrc.nist.gov/Projects/lightweight-cryptography/round-2-candidates. 2019.

[5] M. S. E. Mohamed, S. Bulygin, and J. Buchmann. “Using SAT Solving to Improve Differential Fault Analysis of Trivium”. In: *Communications in Computer and Information Science*. Springer Berlin Heidelberg, 2011, pp. 62–71.

[6] M. Soos, K. Nohl, and C. Castelluccia. “Extending SAT Solvers to Cryptographic Problems”. In: *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2009, pp. 244–257.

[7] F. Zhang et al. “A Framework for the Analysis and Evaluation of Algebraic Fault Attacks on Lightweight Block Ciphers”. In: *IEEE Transactions on Information Forensics and Security* 11.5 (2016), pp. 1039–1054.