The Efficient Randomness Testing using Boolean Functions

BoolTest

M. Sýs, D. Klinec, P. Švenda
{syso, svenda,klinec} @fi.muni.cz
Faculty of Informatics, Masaryk University, Brno, CZ

CROCS
Centre for Research on Cryptography and Security

Problem description

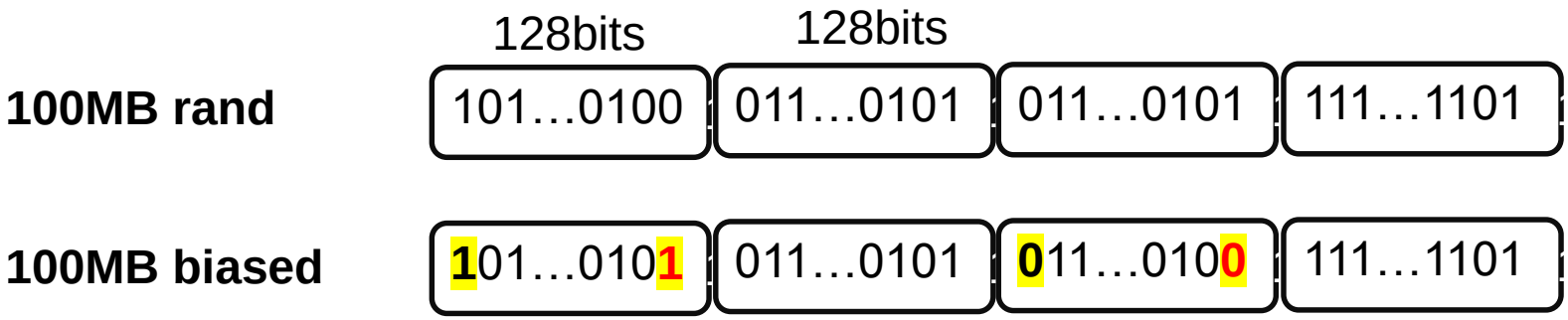
Test design of cryptographic primitive (hash, block, stream cipher) for information leakage. Identify the problem of design if info is leaking.

Approaches:

1. Cryptanalysis - linear, differential, algebraic
2. Randomness analysis – fully automated but weaker
   - batteries NIST STS, Dieharder, **TestU01**

Why new test is needed?

Extremely simple pattern can pass standard tests



Tested by (NIST, Diehard, TestU01)

- 1% blocks changed (Pr = 10^{-545}): **one** test failed
- 0.1% blocks changed (Pr = 10^{-6}): **pass** all (~200) tests

Limitations of standard tests

General purpose: not suitable for crypto-primitives

Interpretation: statistic ⇒ pass/fail

Hard to identify **concrete** correlated bits.

Simple tests: fast, small data but weak

Complex tests: strong but big data needed and slow

Our goal: reasonably strong, fast, small data, interpretable

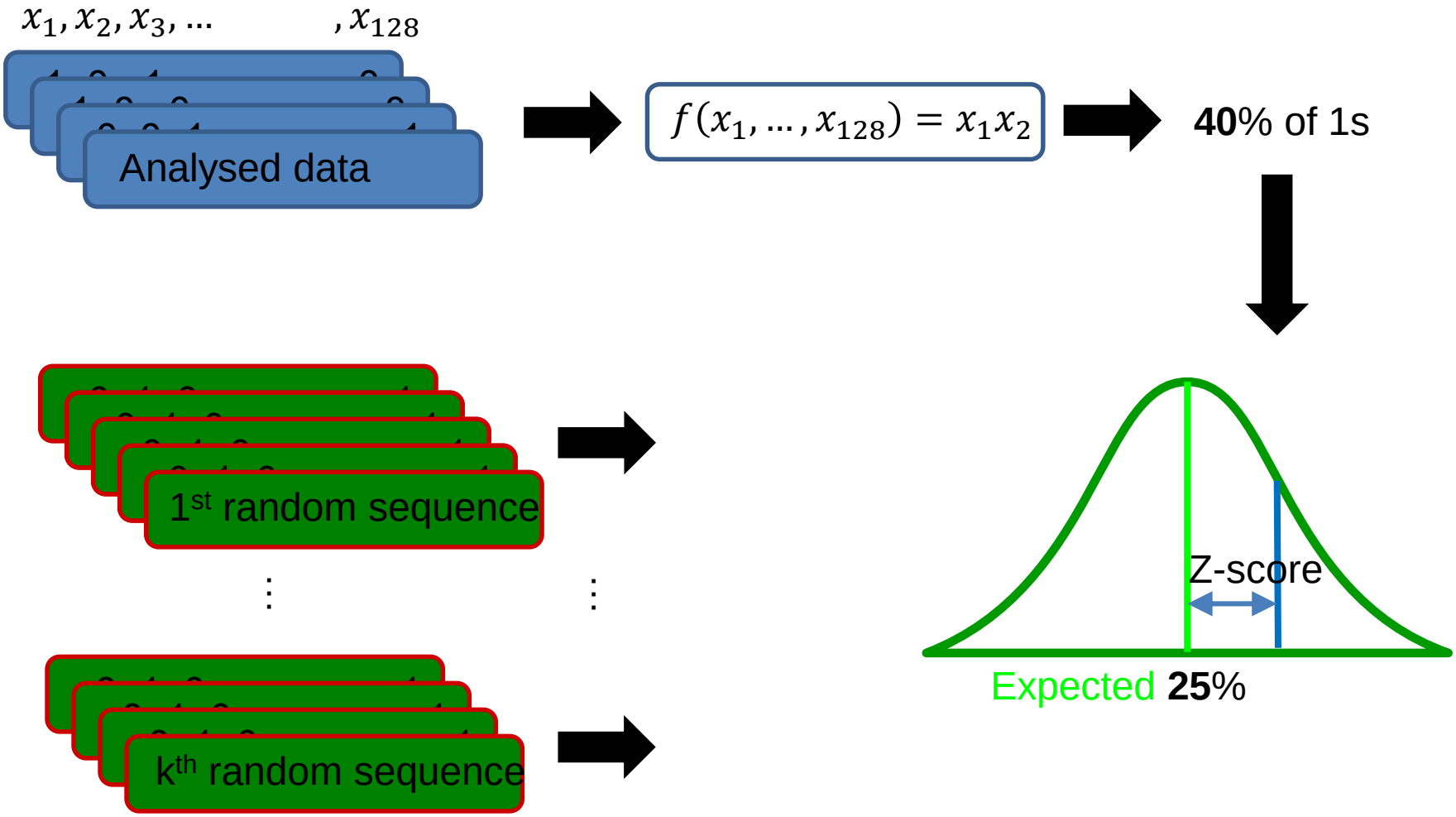
New BoolTest

1. Looks for function with some bits of a block as input and {0,1} as output
2. Function is applied to all blocks and #1 is counted
3. Looks for function with **maximal** bias from expected

   if bias is large data are probably not random

- But how to find such a function?
  - combination of **brute force** & **heuristics**

Boolean function evaluation



Distinguisher construction

Construction (brute force) of b.f. of the form

$$f(x_1, \dots, x_m) = b_1 + b_2 + \cdots + b_k\ ,\ b_i \in S$$

1. Phase find set S of **t** best monomials of degree **deg**

   S = $\{x_1x_2,\ x_{32}x_{56},\ x_{45}x_{56}\}$, (deg = 2, t = 3)

2. Phase – combine **k** best monomials

$$x_1x_2 + x_{32}x_{56},\ \ x_1x_2 + x_{45}x_{56},\ \ x_{32}x_{56} + x_{45}x_{56}$$

Result: b.f. with **maximal** Z-score

e.g. $f(x_1, \dots, x_m) = x_1x_2 + x_{45}x_{56}$ with Z-score=16

Comparison of BoolTest and std. batteries

Experiments on real crypto-primitives

1. PRNGs – C rand, Java rand.util, Mersenne Twister
2. Common crypto-primitives (with limited number of rounds): SHA256, AES, Keccak, …

- Evaluation criteria:
  - amount of data needed to detect bias
  - execution speed
  - significance of bias for fixed amount of data

Results PRNG

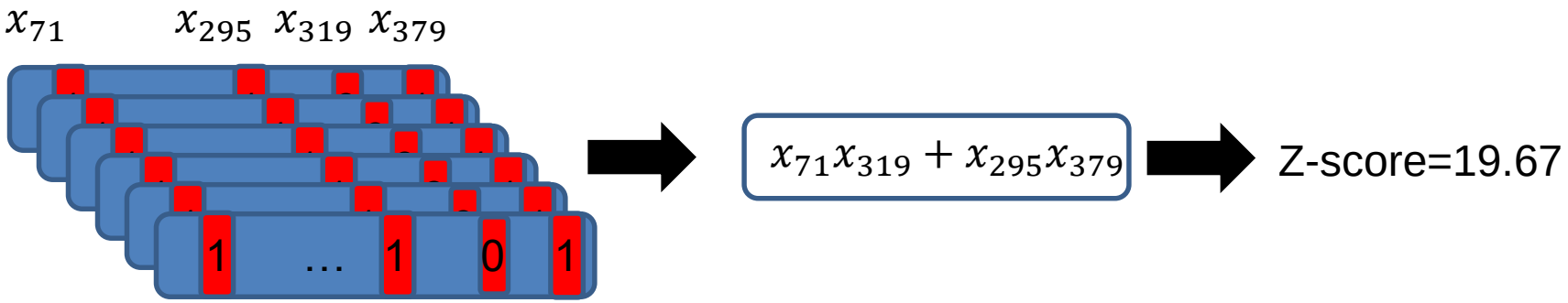
Distinguishers for *C rand* & *Java.util.Random*

size	func	NI	Di	U01	*BoolTest*
1MB	c	-	∀	∀	19.67
	java	-	∀	∀	17.78
10MB	c	∀	∀	∀	60.92
	java	∀	∀	∀	55.98
100MB	c	∀	22/23	∀	191.37
	java	∀	∀	15/16	176.62

Z-score > 8 bias detected

Distinguisher interpretation

Rand C: $f(x_1, \ldots, x_{384}) = x_{71}x_{319} + x_{295}x_{379}$



Z-score = 19.67 (1MB) $\Rightarrow$ Pr = 10^{-90}

Results – primitives (25 tested in paper)

size	func	NI	Di	U01	Bool3	Bool4
	AES (3)	∀	18	15	8.6	6.7
	TEA (4)	∀	20	∀	20.6	11.5
	Keccak (3)	∀	∀	15	3.7	5.3
10MB	MD6 (9)	∀	∀	∀	3.9	13.3
	SHA256 (3)	0	0	6	88.7	242
	AES (3)	∀	16	15	8.9	15.0
	TEA (4)	14	21	∀	73.6	5.2
	Keccak (3)	14	22	15	3.8	9.2
100MB	MD6 (9)	∀	∀	∀	3.7	26.4
	SHA256 (3)	0	0	4	50.7	828
	AES (3)	9	18	14	12.8	41.2
	TEA (4)	13	24	∀	127	4.3
	Keccak (3)	∀	26	15	3.5	32.0
1GB	MD6 (9)	13	25	15	4.1	26.4
	SHA256 (3)	0	1	3	78.0	3043

BoolTest summary

1. Strong test – surprisingly strong given its simplicity
2. Interpretable – identify concrete correlated bits
3. Distinguisher for C/Java rand where batteries failed
4. Usually need significantly less data than batteries
   - order of 10MBs or even less
5. Fast – depends on degree of b.f.
   - degree 1,2 – 10 MB per 10s (batteries in minutes)
   - degree 3 – 10 MB per 200s

Thank you for your attention!

Questions 

BoolTest https://github.com/crocs-muni/booltest