Towards cryptographic function distinguishers with evolutionary circuits

Statistical testing of cryptographic function output based on genetic programming

Petr Švenda, Martin Ukrop, Vashek Matyáš
{svenda,xukrop,matyas}@fi.muni.cz
Overview

1. Randomness testing with STS NIST & Dieharder
   – Can we beat traditional approach? (Speed, input length.)

2. Random distinguisher based on software circuit
   – Our approach based on genetic programming

3. Results for selected eStream/SHA-3 candidates
   – How good is it?

4. Discussion, interesting observations
Why to test randomness of function output?

1. Building block for pseudorandom generator
2. Common requirement
   - AES, SHA-3 competition, FIPS-140
3. Significant deviances from uniform distribution and unpredictability indicate function defects
   - but no proof in opposite case

- Manual approach: human cryptanalysis
- Automated approach: statistical testing
Workflow with STS NIST/Dieharder

Tests

Count the 1s
Overlapping permutations
Runs tests

“null hypothesis”
⇒ p-values

p-value < α ⇒ fail

10^5-10^9 B
Hypothesis: *If function output is somehow defective, we should be able to distinguish between the data produced by a function and truly random data.*
Proposed idea – software circuit

• Design test(s) automatically
  – test is algorithm ⇒ hardware-like circuit (next slide)

• Several issues:
  – Who will define null hypothesis? (*random distinguisher*)
  – Who will design the circuit? (*genetic programming*)
  – How to compare quality of candidates? (*test vectors*)
Software circuit (EACirc)

https://github.com/petrs/EACirc/
Genetic programming of circuits

Population

Test vectors \((10^2 - 10^5)\) 

\([\text{input}_i, \text{exp.output}_i]\)

Comparator

\(\text{exp.output}_i == \text{output}\)

Circuit emulator

fitness

\% correct answers
HW(10110111) > 4 => QRNG
Methodology

• Limit number of algorithm rounds
  – tested on 7 eStream and 18 SHA-3 candidates
• Generate & run STS NIST and Dieharder tests
• Prepare input data for EACirc
  – generate ½ test vectors from function (key change freq.)
  – generate ½ test vectors from truly random source
    (QRBGS http://random.irb.hr/)
• Generate & test software circuits (repeat, EA)
Were we successful?

- Definition of success?
- Better than random guessing?
- Better or at least as good as human-made batteries?
- Other advantages against statistical batteries?
Salsa20 – limited to two rounds

(0.87 success rate)
Test vectors – key change frequency

Key fixed for whole run (all generations)

Key fixed only for one test set (e.g., 500 test vectors)

Key per every test vector (e.g., every 16 bytes)
Table 2: Results for Decim.

<table>
<thead>
<tr>
<th># of rounds</th>
<th>IV and key reinitialization once for run</th>
<th>for each test set</th>
<th>for each test vector</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Dieharder (x/20)</td>
<td>STS NIST (x/162)</td>
<td>EACirc</td>
</tr>
<tr>
<td>1</td>
<td>0.0</td>
<td>0</td>
<td>(0.54)</td>
</tr>
<tr>
<td>2</td>
<td>0.5</td>
<td>0</td>
<td>(0.54)</td>
</tr>
<tr>
<td>3</td>
<td>1.0</td>
<td>0</td>
<td>(0.53)</td>
</tr>
<tr>
<td>4</td>
<td>3.5</td>
<td>79</td>
<td>(0.52)</td>
</tr>
<tr>
<td>5</td>
<td>4.5</td>
<td>79</td>
<td>(0.52)</td>
</tr>
<tr>
<td>6</td>
<td>19.0</td>
<td>158</td>
<td>(0.52)</td>
</tr>
<tr>
<td>7</td>
<td>18.5</td>
<td>162</td>
<td>(0.52)</td>
</tr>
<tr>
<td>8</td>
<td>20.0</td>
<td>162</td>
<td>(0.52)</td>
</tr>
</tbody>
</table>
Decim – 6 out of 8 rounds (preliminary)

χ² difference between random/fnc histograms of categories
What is a function test then?

• One particular circuit?
  – circuit was evolved for particular function and key
  – sometimes, circuit works even when key is changed
  – (most probably) not useful for a different function

• Test = whole process with evolution of circuits!
  – Is evolution able to design a distinguisher in limited number of generations?
  – If so, then function output is defective!
Comparison to statistical batteries

• **Advantages**
  – new approach, no need for predefined pattern
  – dynamic construction of test for particular function
  – works on very short sequences (16 bytes only)

• **Disadvantages**
  – no proof of test quality or coverage (random search)
  – possibly hard to analyze the result (possibly automatic)
  – possibly longer test run time (learning period)

Questions
Thank you for your attention!

Questions