Tests for Randomness

Cryptographic Hardware for Embedded Systems ECE 3170

Fall 2025

Assoc. Prof. Vincent John Mooney III

Georgia Institute of Technology

National Institute of Standards and Technology

- This lecture is based on NIST Special Publication 800-22 Revision 1a, "A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications," Rukhin et al., April 2010, https://nvlpubs.nist.gov/nistpubs/legacy/sp/nistspecialpublication800-22r1a.pdf

Abstract

"This paper discusses some aspects of selecting and testing random and pseudorandom number generators. The outputs of such generators may be used in many cryptographic applications, such as the generation of key material. Generators suitable for use in cryptographic applications may need to meet stronger requirements than for other applications. In particular, their outputs must be unpredictable in the absence of knowledge of the inputs. Some criteria for characterizing and selecting appropriate generators are discussed in this document. The subject of statistical testing and its relation to cryptanalysis is also discussed, and some recommended statistical tests are provided. These tests may be useful as a first step in determining whether or not a generator is suitable for a particular cryptographic application. **However, no set of statistical tests can absolutely certify a generator as appropriate for usage in a particular application, i.e., statistical testing cannot serve as a substitute for cryptanalysis.** The design and cryptanalysis of generators is outside the scope of this paper."

Random Number Testing

- Aim to detect if a binary sequence deviates from randomness
- Random Number Generator (RNG)
  - Result of a sequence of unbiased (i.e., “fair”) coin tosses
    - Each flip has a probability of exactly ½ of producing a “0” or a “1”
    - Each flip is independent of each other
  - Typically not used directly for most applications due to cost
  - NOTE: also known as a *True Random Number Generator* (TRNG)
- Pseudo-Random Number Generator (PRNG)
  - Starting point based on a *seed*
  - Typically deterministic and known; hence *pseudo*-random
  - For most use cases, session is short enough to appear to be truly random (i.e., an RNG) given that the seed is unknown to the adversary
    - May use an RNG to produce a list of seeds which are distributed accordingly

Random Number Testing (continued)

- "*Randomness is a probabilistic property*" (page 1-2)
- Key idea: test for any "pattern" which would not be random
- No finite set of tests is considered "complete"
- **null hypothesis** (H_0)
  - The sequence being tested is random, i.e., there is no correlation between the bit sequence and any known predictive technique
- alternate hypothesis (H_a)
  - The sequence being tested is not random

TRUE SITUATION	CONCLUSION	
	Accept H_0	Accept H_a (reject H_0)
Data is random (H_0 is true)	No error	Type I error
Data is not random (H_a is true)	Type II error	No error

Main Idea

- For each test, determine if non-randomness appears to be occurring with what probability
- For example, can say that a particular test result indicates that there is less than a 1% chance that a random bitstring would have exhibited the test pattern observed
- For this class (Cryptographic Hardware for Embedded Systems or CHES), we will not explain the statistics used for each test to indicate the percentage chance that a random bitstring would have exhibited the test pattern under consideration
- Conclusion: a truly random bitstring should pass all or nearly all tests

Type I and Type II Errors

- Different use-case scenarios may prioritize Type I and Type II errors differently
  - An inexpensive consumer electronics game may be very concerned about Type I errors due to increased cost of delay & bitstring regeneration (the bitstring is in fact random but is misidentified as nonrandom, hence the game is delayed while a new bitstring is computed)
  - A highly sensitive company or government communication may be very concerned about Type II errors resulting in release of the sensitive information
- Statistical test parameters can be set to minimize Type I errors or Type II errors (and also may provide some possible tradeoffs)

Test 1: Frequency (Monobit) Test

- Focus: test the number of zeros and ones in the entire sequence
- If the sequence is truly random, the fraction should approach ½
- Test: add the bits where zero is converted to -1

Test 2: Frequency Test within a Block

- Test the number of zeros and ones within subsets of the sequence
- *M* = block length, *n* = sequence length, *M* < *n*
- Let $N = \left\lfloor \frac{n}{M} \right\rfloor$
- Determine ratio of ones in each block; discard any leftover bits
- Example: sequence ε = 0110011010, *M* = 3
  - Three blocks: 011, 001 and 101 (the final 0 is discarded)
  - Ratios: π_1 = 2/3, π_2 = 1/3 and π_3 = 2/3

Test 3: Runs Test

- Count number of runs in a sequence
  - Where a "run" of length *k* is defined as a sequence of ones or zeros (not both) bounded by the other number
- Various statistics are computed, e.g., the total number of zero-runs and the total number of one-runs
- Example: sequence ε = 1001101011
  - The runs test first executes the frequency test: result here is π = 6/10
  - If frequency test passed, count number of runs
  - Here we have 7 runs: 1 00 11 0 1 0 11
  - Apply statistical test to determine if the result is statistically unlikely

Test 4: Longest-Run-of-Ones in a Block

- Find longest run of ones within an M-bit block from the sequence ε
- Statistically, an irregularity in the expected length of the longest run of ones would also typically result in a similar irregularity in the expected length of the longest run of zeros
  - Therefore, only test for the longest run of ones
- The reference distribution for this test is known & is compared against
  - Can say, for example, that there is less than a 1% chance that a truly random bit sequence fails a particular longest-run-of-ones test for a particular choice of M

Test 5: Binary Matrix Rank Test

- Test for rank (independent rows or columns) of disjoint sub-matrices
- Let *M* = # matrix rows, *Q* = # matrix columns
  - E.g., *M* = 32 and *Q* = 32
  - Test $N = \left\lfloor \frac{n}{MQ} \right\rfloor$ matrices, discarding any unused bits
- Example: ε = 01011001001010101101, *M* = 3 and *Q* = 3
- Result: two matrices, the first with rank 2 and the second with rank 3
- Note: test code provided by NIST is optimized for *M* = 32 and *Q* = 32

All 15 Random Number Generator Tests

1. The Frequency (Monobit) Test
2. Frequency Test within a Block
3. The Runs Test
4. Tests for the Longest-Run-of-Ones in a Block
5. The Binary Matrix Rank Test
6. The Discrete Fourier Transform (Spectral) Test
7. The Non-overlapping Template Matching Test
8. The Overlapping Template Matching Test
9. Maurer's "Universal Statistical" Test
10. The Linear Complexity Test
11. The Serial Test
12. The Approximate Entropy Test
13. The Cumulative Sums (Cusums) Test
14. The Random Excursions Test
15. The Random Excursions Variant Test