Repetition Count Tests

Overview and Recommendations

Michael Powers and Jason Tseng
Cryptographic & Security Testing Laboratory (CSTL)
6841 Benjamin Franklin Drive
Columbia, MD 21046
NVLAP Lab Code: 200427-0

leidos

Introductions

Who are we?

- **Jason Tseng**
- Leidos CSTL Lab Manager
  - Responsible for all FIPS 140-2 and TWIC related activities
- B.S in Computer Engineering in 2011 from University of Maryland College Park

Who are we? (Continued)

- **Michael Powers**
- Currently working for the Leidos AT&E labs as a Security Assurance Engineer
  - Responsible for FIPS 140, FIPS 201, and SCAP-related activities
- Previous work experience includes penetration testing, reverse engineering, server administration, and product development.
- Has a B.S in Mathematics and a Minor in Computer Science from UMBC

The current state of continuous testing (for random number generators) with FIPS 140-2

leidos

Overview of Continuous Testing on Random Number Generators (RNGs)

- Continuous testing is intended to detect catastrophic errors in an RNG. Examples include:
  - A non-deterministic RNG "**flat-lining**", (e.g.: It starts repeatedly outputting the sequence 10101010…).
  - Internal errors such as memory depletion, buffer overflows, etc. that may otherwise cause an RNG to malfunction.
  - The RNG does not appear to be providing as much entropy as you've estimated it to provide.
    - **Example**: Internal testing estimates that the entropy source is independent and identically distributed with a min-entropy estimate of 7.8 bits of entropy per byte of output – but continuous testing indicates that the RNG is behaving like a source that you'd only expect to provide 1 bit of entropy per byte)

leidos

Current state of FIPS 140-2

- Currently, the FIPS 140-2 standard mandates a Continuous Random Number Generator Test (**CRNGT**) for all RNGs
- The general gist of a CRNGT is:
  - Use the RNG to produce 1 block of data. Do not output/use this first block of data – but rather store it off internally.
  - For subsequent blocks produced by the RNG, numbered '**n**', compare the block '**n**' to the previous block '**n-1**'. If the blocks are equal, then the CRNGT fails and you must enter an error state. If the blocks are not equal, store off the current block '**n**' for future comparison.

leidos

Pros and Cons of CRNGTs

- Pros
  - Minimal overhead – Per block, you're adding one memory comparison and one memory copy. The storage required to store one block is negligible
  - Can detect some forms of catastrophic failures (primarily "flat-lining") in an RNG
- Cons
  - *Can* introduce bias in an RNG depending on your error state implementation (e.g.: An attacker might be able to deduce that for your RNG, it always have the property that block 1 != block 2, block 2 != block 3, … block N != block N + 1, etc.)
  - Has no meaningful statistical significance, as it's purely based on the block size of the RNG
  - Doesn't make much sense when applied to a pseudo-random number generator (e.g.: It's very easy to design a PRNG that never has two consecutive, equal blocks, like AES in CTR mode)
  - Can't detect failures like ABABABAB or ABCABCABC
  - Can't always detect patterns that are smaller/larger than the RNGs block size:
    - For example, if the block size was 16 bits, a 3-bit pattern of "101" repeated would not be detected as **1011011011011011** does not equal **0110110110110110**

leidos

“Traditional” interpretation of Error States

- Regarding error states, the FIPS 140-2 standard states:
  - *“If a cryptographic module fails a self-test, the module shall enter an error state and output an error indicator via the status output interface. The cryptographic module shall not perform any cryptographic operations while in an error state. All data output via the data output interface shall be inhibited when an error state exists”*
- In the past, regarding CRNGTs in particular, many vendors took the following two approaches:
  - When the CRNGT fails, **discard the block** and output some sort of error indicator, but otherwise continue normal operation
  - When the CRNGT fails, have the module output an error indicator and then proceed to **terminate execution**.
- With both of these “traditional” interpretations, you end up effectively **throwing away bits**. Due to this undesirable outcome, we reached out to the NIST CMVP for clarification.

Clarification on Error States

- For the CRNGT, the NIST CMVP has provided the following guidance to the Leidos CSTL lab regarding error states:
  - *"The module can enter the error state, inhibit all data output via the data output interface while there exists an error state, and then exit the error state and do nothing."*
  - *"**The module is not required to throw away any bits** that caused the test to fail."*
- This seems to indicate that a valid error state could look something like this:
  - **Function CRNGT_error_state:**
    1. Inhibit data output
    2. Perform error-state operations (e.g.: logging the failure via the status output interface, setting a flag to determine the program's future behavior, or perhaps not doing anything)
    3. Un-inhibit data output and exit the error state
    4. (Optionally) return block of data that caused the test to fail

Clarification on Error States (Continued)

- We recommend that in order to avoid the situation mentioned before where your RNG has the property *"block 1 != block 2, block 2 != block 3, … block N != block N + 1, etc."* that you **don't discard blocks as a result of a failed CRNGT**.
  - Instead, an alternative might be to simply not count that block as entropy during run-time (but still use it):
    - For example, if your NDRNG normally produces 64-bit blocks with full entropy, and you need 256-bits of entropy to properly seed your DRBG, you'd normally need **four** blocks of output from the NDRNG to fulfil this.
    - If the CRNGT on the NDRNG fails once, however, we recommend that you produce **five** blocks of output from the NDRNG to seed your DRBG for that particular instantiation/reseed, and provide all five blocks as input to the DRBG.

leidos

Repetition Count Tests and DRAFT Implementation Guidance 9.8

leidos

Potential changes to FIPS 140-2 with RCTs

- DRAFT implementation guidance 9.8 for FIPS 140-2 standard adds an alternative test called a Repetition Count Test (**RCT**)
- The general gist of an RCT is:
  - Use the RNG to produce 1 block of data. Do not output/use this first block of data – but rather store it off internally in a variable '**A**'. Set a counter variable '**B**' to a value of **1**.
  - For subsequent blocks produced by the RNG, numbered '**n**', compare the block '**n'** to the current value stored in '**A**'.
    - If the blocks are equal, then increase a counter **'B'.** If '**B**' now equals the cutoff threshold value '**C**', then enter an error state.
    - If the blocks are not equal, store off the current block '**n**' in the variable '**A**' for future comparisons and reset the counter value '**B**' to **1**.

leidos

Repetition Count Test Parameters

- From the previous slide, the cutoff threshold '**C**' is calculated in the following manner:
  - Determine a desired false positive rate '**W**'. The recommendation provided by NIST is $\mathbf{2^{-30}}$ (approximately 1 in 1,000,000,000)
  - Determine your entropy requirements '**H**'. For example, if you are using the output of the RNG as entropy to seed a DRBG and you are assuming that the RNG is providing 64 bits of entropy per block, then **H = 64**.
  - Calculate '**C**' as 1 + (($-\log_2(W)$) / H), rounded up
    - For our example:
      - C = 1 + (($-\log_2(\mathbf{2^{-30}})$) / **64**)
      - C = 1 + ((30)) / 64)
      - C = 1.46875, rounded up **C = 2**

leidos

RCT benefits over the CRNGT

- The RCT now has some statistical significance, as it is based around trying to confirm an entropy estimate '**H**' within bounds.
  - **Note** that due to the fact that 'C' is calculated by rounding up, the "statistical significance" is reduced.
  - For example, in an extreme case where C was calculated as $C = 1 + ((- \log(2^{-30})) / 3000) =$ **1.01**, you would round up to **2**, which is significantly (~98%) higher.
- In some cases (where **C > 2**) the RCT reduces the chances of unintentionally introducing bias into the RNG output.

leidos

Other potential changes with DRAFT Implementation Guidance 9.8

- Draft IG 9.8 also proposes the removal of the CRNGT/ RCT requirements for the SP800-90A DRBGs.
  - The SP800-90A DRBGs have extensive health-testing built into them that would otherwise detect catastrophic errors.
  - This addresses the previous point where we stated that the CRNGT (and RCT) "*Doesn't make much sense when applied to a pseudo-random number generator*"

Impacts and looking forward

Retroactive Impact of the RCT

- The RCT does not appear to have any retroactive impact on current modules.
  - When C = 2, the RCT is equivalent to a CRNGT; so current implementations of CRNGTs would not need modification.
    - CRNGTs are a subset of the RCTs, and will always be **either equally as stringent or more stringent** than an RCT
  - Regardless of whether a CRNGT can be classified as a subset of RCTs, the current DRAFT of implementation guidance 9.8 allows for either the CRNGT or the RCT to be implemented.

Difficulties for NIST

- Continuous testing on entropy sources in particular can be next to impossible to do effectively:
  - For example, detecting that a proprietary (and not publically documented) hardware entropy source is providing adequate entropy is hard:
    - If it performs it's own conditioning, it could end up **removing bias and other indicators** from it's output such that no form of entropy testing would be effective at detecting a fault.
  - There is also the potential for a **malicious entropy source**, such that it produces seemingly random data, but it's really fully deterministic/predictable (see: Dual_EC_DRBG). With elevated access, even a software source like /dev/random could be replaced/manipulated.
- Creating a generic test that can be applied to all RNGs is a difficult task as well
  - Current tests (CRNGT, RCT) are based on the block size of the RNG. There could be cases like a hardware RNG that may produce 256-bit blocks, but due to the nature of the hardware RNG a failure may manifest itself as a repeating 7-bit pattern.

leidos

Hope for a Brighter Future?

- ISO/IEC 19790:2012 does not mandate any continuous testing for random number generators
- Standard is set to become next version of FIPS 140 i.e. “FIPS 140-3”
- Public commenting period ended on September 28, 2015
- No ETA on when ISO/IEC 19790:2012 will become official

Recommendations for vendors

- We recommend **against** implementing the RCT unless you have a compelling reason to. Compelling reasons may include:
  - Small block sizes (like 16 bits) that could end up triggering many false positives in a traditional CRNGT.
- We recommend that if you do not like the "traditional" error state behavior for CRNGTs and RCTs (effectively discarding repeated blocks), that you **modify your error behavior such that blocks are no longer discarded**.

leidos

Questions?

Leidos CSTL Contact Information

- Daun-Marie Sniegowski – AT&E Laboratory Director
  - Daun-Marie.C.Sniegowski@leidos.com
  - (443) 367-7629
- Amit Sharma – CSTL Laboratory Director
  - Amit.Sharma@leidos.com
  - (443) 367-7733
- Jason Tseng – CSTL Laboratory Manager
  - Jason.K.Tseng@leidos.com
  - (443) 367-7808
- www.leidos.com/infosec/testing-accreditation