BZA02 - MNG

Model 2019

Bezpečná zařízení

Část 2

Generátory náhodných čísel

Post 18/19

Souhrnné materiály

Ver 0.1

BZA

New 2018/19

BZA 02
Generátory náhodných čísel

Petr Hanáček
Faculty of Information Technology
Technical University of Brno
Božetěchova 2
612 66 Brno
tel. (05) 4114 1216
e-mail: hanacek@fit.vutbr.cz

Netscape Navigator

Netscape Navigator, známý též jako Netscape, byl proprietární webový prohlížeč, který byl populární v průběhu 90. let. Jednalo se o vlajkový produkt společnosti Netscape Communications Corporation a o dominantní webový prohlížeč na trhu do té doby, než jej společnost Microsoft vytlačila integrací svého prohlížeče Internet Explorer do svého operačního systému Windows. Druhým důvodem byla malá inovace, se kterou prohlížeč koncem 90. let přicházel.

Navigator byl nahrazen balíkem Netscape Communicator.

Historie verzí [editovat | editovat zdroj]

Verze	Datum vydání
0.9	13. října, 1994
1.0	15. prosince, 1994
1.1	Březen, 1995
1.22	Srpen, 1995
2.0	Březen, 1996
3.0	19. srpen, 1996
4.0	Červen, 1997
4.08	19. listopad, 1998

Randomness and the Netscape Browser

January 1996 *Dr. Dobb's Journal*

How secure is the World Wide Web?

by Ian Goldberg and David Wagner

Ian and David are PhD students in the computer science department at the University of California,

As the World Wide Web gains broad public appeal, companies are becoming interested in using the Since ordering a product online requires the customer to transmit payment information (such as a cr Internet, there's need for cryptographic protection. By encrypting payment information before transr purchasing can decode that sensitive data.

Netscape Communications has been at the forefront of the effort to integrate cryptographic techniqu Layer (SSL), a cryptographic protocol developed by Netscape to provide secure Internet transaction cryptographic protocol on the Internet, we decided to study Netscape's SSL implementation in detai

Figure 3 shows the key-generation algorithm, also reverse-engineered from Netscape's browser. An attacker who can guess the PRNG seed value can easily determine the encryption keys used in Netscape's secure transactions.

```
RNG_GenerateRandomBytes()
    x = MD5(seed);
    seed = seed + 1;
    return x;

global variable challenge, secret_key;

create_key()
    RNG_CreateContext();
    tmp = RNG_GenerateRandomBytes();
    tmp = RNG_GenerateRandomBytes();
    challenge = RNG_GenerateRandomBytes();
    secret_key = RNG_GenerateRandomBytes();
```

Figure 3: *The Netscape v1.1 key-generation process: pseudocode.*

In Figure 2, it's important to note that mklcpr() and MD5() are fixed, unkeyed algorithms that will presumably be known by an adversary. The seed generated depends only on the values of a and b, which in turn depend on just three quantities: the time of day, the process ID, and the parent process ID. Thus, an adversary who can predict these three values can apply the well-known MD5 algorithm to compute the exact seed generated.

```
global variable seed;

RNG_CreateContext()
    (seconds, microseconds) = time of day; /* Time elapsed since 1970 */
    pid = process ID;  ppid = parent process ID;
    a = mklcpr(microseconds);
    b = mklcpr(pid + seconds + (ppid << 12));
     seed = MD5(a, b);

mklcpr(x) /* not cryptographically significant; shown for completeness */
    return ((0xDEECE66D * x + 0x2BBB62DC) >> 1);

MD5() /* a very good standard mixing function, source omitted */
```

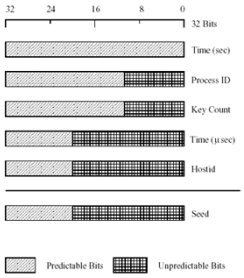
Figure 2: *The Netscape 1.1 seeding process: pseudocode.*

Kerberos v4 RNG (1988)

- 1. Time-of-day seconds since UTC 0:00 Jan. 1, 1970
- 2. Process ID of the Kerberos server process
- 3. Cumulative count of session keys generated
- 4. Fractional part of time-of-day seconds since UTC 0:00 Jan. 1, 1970 in microseconds
- 5. Hostid of the machine on which the server is running
- Kerberos V4 uses the UNIX random function to produce the random DES keys.
- Kerberos first seeds the random number generator with a 32 bit seed chosen as outlined in above, then it makes two calls to the random function to get 64 pseudo-random bits. This 64-bit block has every eighth bit set as a parity bit, leaving a 56-bit DES key.
- The random function relies on a 32-bit seed value to determine the state of the linear feedback shift register used for generating the pseudo random numbers. Thus, any sequence of numbers created by the random function, no matter how long, relies solely on the 32-bit seed value. The entropy of any number sequence produced by random() has an entropy of only 32 bits.

BZA

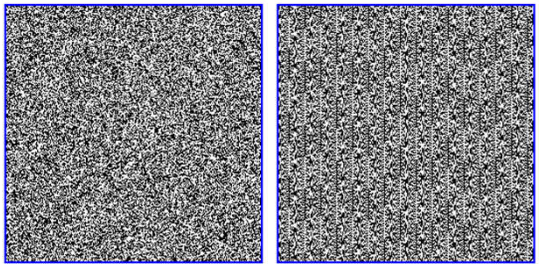
Kerberos v4



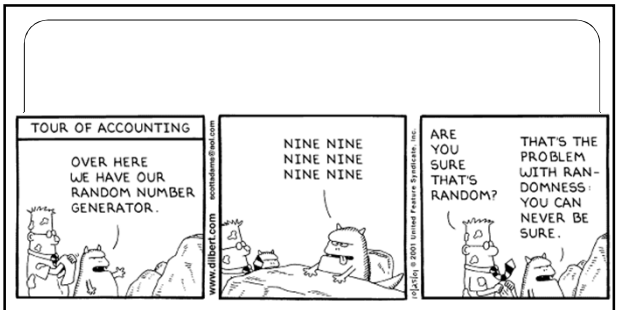
Seed Component	Insider	Outsider
time (seconds)	0	0
process ID	0	≤ 10
number of keys	0	≤ 10
time (microseconds)	20	20
hostid	0	20
Total Unknown bits	20	20

Table 1. Entropy of seed components.

PHP rand()



RANDOM.ORG PHP rand() on Microsoft Windows

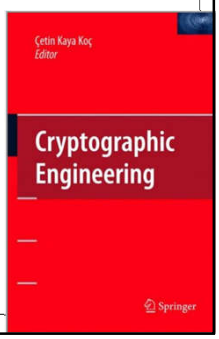


Kategorie RNG

Cryptographic Engineering

- Cetin Kaya Koc: Cryptographic Engineering
- Springer Publishing Company2008
- ISBN: 0387718168 9780387718163

Chapter 2
Random Number Generators for Cryptographic Applications

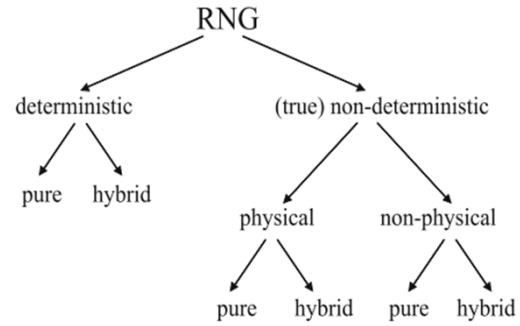


Functionality classes for random number generators

- Werner Schindler: A proposal for: Functionality classes for random number generators
- Bundesamt für Sicherheit in d Informationstechnik (BSI), Bo
- Volně dostupné

A proposal for:
Functionality classes for random number generators

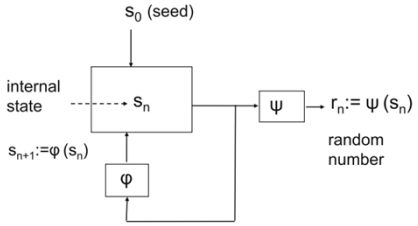
BZA



Požadavky

- **(R1) The random numbers should have good statistical properties.**
- **(R2) The knowledge of subsequences of random numbers shall not allow one to practically compute predecessors or successors or to guess these numbers with non-negligibly larger probability than without knowledge of these subsequences.**

Deterministic Random Number Generators (DRNGs)



Pure DRNG

$$r_1, r_2, \ldots, r_{n-1} \in R$$

have been generated, the internal state of the DRNG attains the value $s_n \in S$. The finite sets S and R are called the *state space* and the *output space* of the DRNG. The *output transition function* $\psi: S \to R$ computes the next random number r_n from the current internal state s_n. Then s_n is updated to s_{n+1} with the *state transition function* ϕ, i.e., $s_{n+1} := \phi(s_n)$. The first internal state s_1 is derived from the *seed* s_0, e.g., simply $s_1 = \phi(s_0)$, or a more complicated mechanism may be used. Clearly, the seed s_0 determines all internal states $s_1, s_2, \ldots$ and all random numbers $r_1, r_2, \ldots$. In order to fulfil requirement (R2) the seed must be selected randomly. A pure DRNG can be described by a 5-tuple

$$(S, R, \phi, \psi, p_S) \tag{2.1}$$

where p_S defines the probability distribution of the random seed. Note, however, that the seed generation is performed outside the DRNG boundaries. Usually the seed is generated by a TRNG.

Požadavky

- **Backward Secrecy**
- **(R3) The knowledge of the internal state shall not allow one to practically compute 'old' random numbers or even a previous internal state or to guess these values with non-negligibly larger probability than without knowledge of the internal state.**

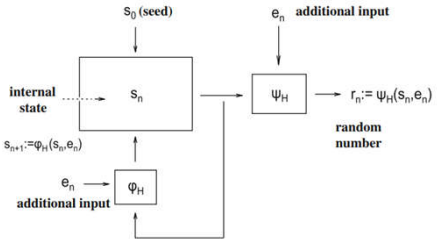
Požadavky

- **Forward Secrecy**
- **(R4) Even the knowledge of the internal state shall not allow one to practically compute the next random numbers or to guess these values with non-negligibly larger probability than without the knowledge of the internal state.**

Remark 2.2. (i) Requirements (R3) and (R4) are specific DRNG requirements. For TRNGs, (R3) and (R4) are usually 'automatically' fulfilled if (R2) is valid.

BZA

Hybrid DRNG



Of course, a hybrid DRNG cannot be described by a 5-tuple (S, R, ϕ, ψ, p_S) (cf. (2.1)). Instead, we use a 7-tuple

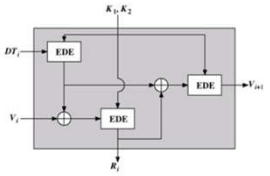
$$(S, R, E, \phi_H, \psi_H, p_S, (q_n)_{n\in N}). \qquad (2.2)$$

The set E and the sequence $(q_n)_{n\in N}$ denote the set of additional input data and the probability distributions of the additional data.

Koc: Cryptographic Engineering, Kap 2

Example: DRNG ANSI X9.17

- ANSI X9.17 PRNG
  - uses date-time + seed inputs and 3 triple-DES encryptions to generate new seed & random
- Input: two pseudorandom inputs:
  - DT_i : a 64-bit representation of the current date/time
  - a 64-bit seed V_i generated at the beginning of i^{th} stage
- Keys (K_1, K_2): all 3DES modules use the same pair of 56-bit keys
- Output: 64-bit pseudorandom number (R_i) and 64-bit seed value (V_{i+1})
  - $R_i = EDE_{K1,K2}[V_i \oplus EDE_{K1,K2}[DT_i]]$
  - $V_{i+1} = EDE_{K1,K2}[R_i \oplus EDE_{K1,K2}[DT_i]]$

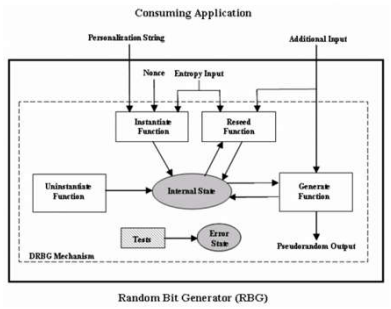


78

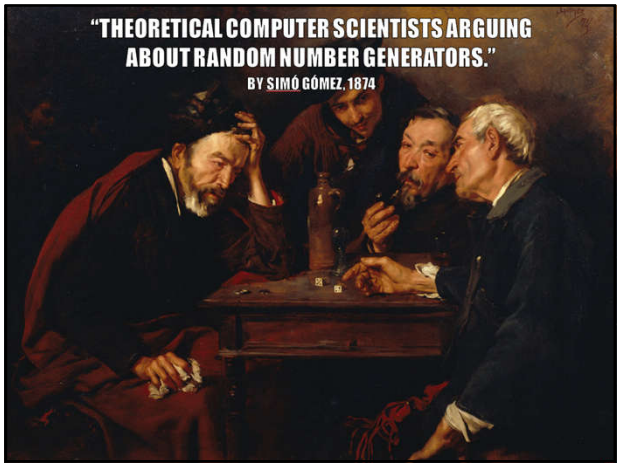
https://slideplayer.com/slide/4805921/

Example DRNG

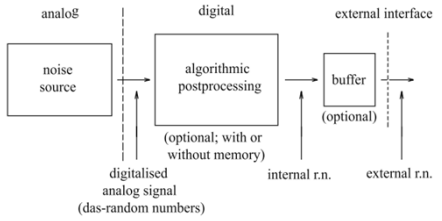
- **RGB Functional model defined in [NIST800-90]**



01_AIS31_Functionality_classes_for_random_number_generators



Physical True Random Number Generators (PTRNGs)



Koc: Cryptographic Engineering, Kap 2

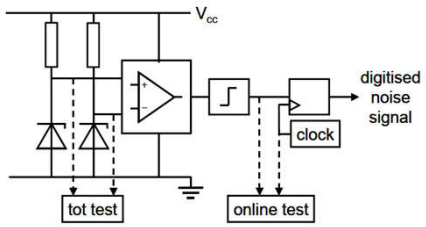
Příklady realizací

- **Radioaktivní rozpad**
- **Šumové diody**
- **Pomaluběžné oscilátory**
- **PUF**

BZA

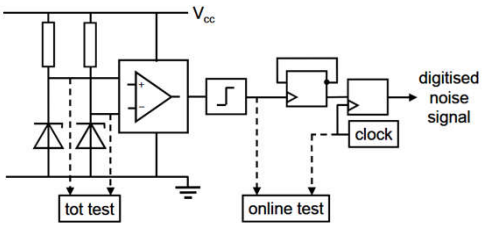
PTRNG with two noisy diodes

- Basic design of RNG with noisy diodes



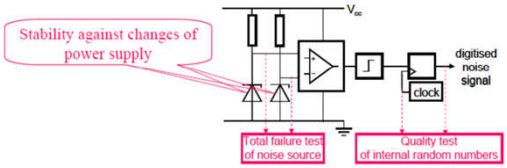
PTRNG with two noisy diodes

- Variant of the basic design of RNG with noisy diodes
- The advanced variant of the basic design outputs the number of Schmitt trigger impulses
-

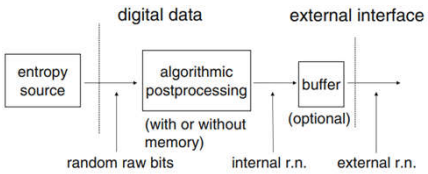


Examples of self-protection in PTRNG based on noise diodes

- Examples of self-protection in PTRNG based on noise diodes

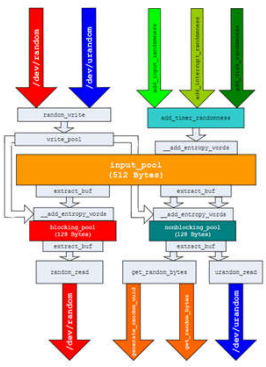


Non-physical True Random Number Generators (NPTRNGs)



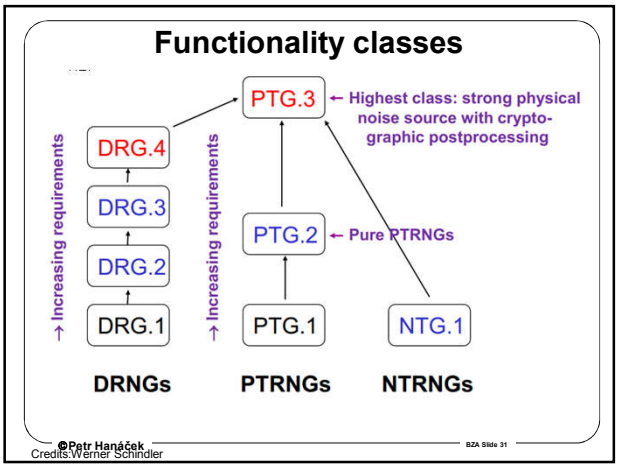
Example of NPTRNG

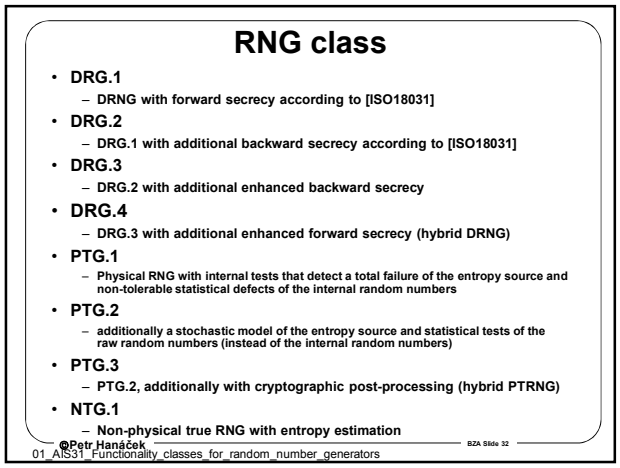
- Example: Linux NPTRNG

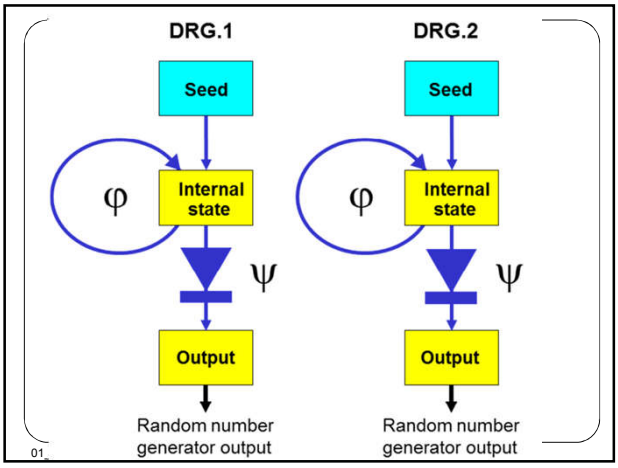


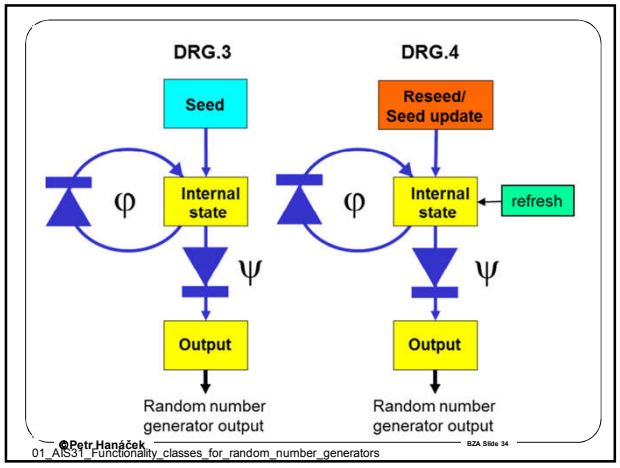
Funkční třídy RNG

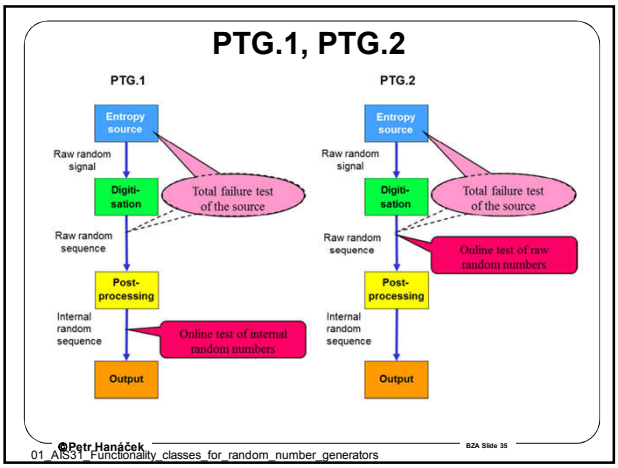
BZA

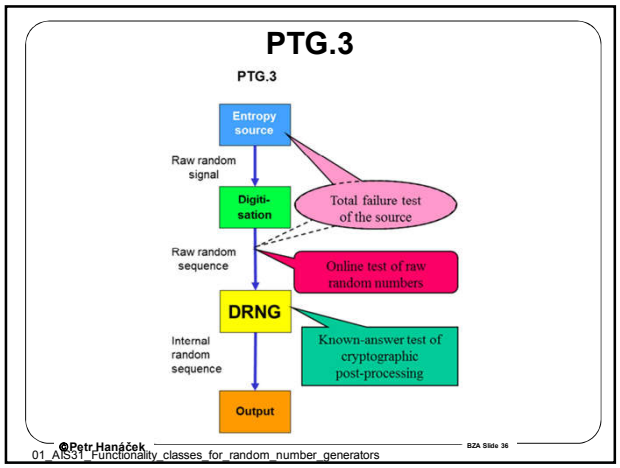

Functionality classes
PTG.3
← Highest class: strong physical noise source with crypto-graphic postprocessing
DRG.4
DRG.3
DRG.2
DRG.1
PTG.2
← Pure PTRNGs
PTG.1
NTG.1
→ Increasing requirements
→ Increasing requirements
DRNGs
PTRNGs
NTRNGs
©Petr Hanáček
BZA Slide 31
Credits:Werner Schindler


RNG class
DRG.1
– DRNG with forward secrecy according to [ISO18031]
DRG.2
– DRG.1 with additional backward secrecy according to [ISO18031]
DRG.3
– DRG.2 with additional enhanced backward secrecy
DRG.4
– DRG.3 with additional enhanced forward secrecy (hybrid DRNG)
PTG.1
– Physical RNG with internal tests that detect a total failure of the entropy source and non-tolerable statistical defects of the internal random numbers
PTG.2
– additionally a stochastic model of the entropy source and statistical tests of the raw random numbers (instead of the internal random numbers)
PTG.3
– PTG.2, additionally with cryptographic post-processing (hybrid PTRNG)
NTG.1
– Non-physical true RNG with entropy estimation
©Petr Hanáček
BZA Slide 32
01_AIS31_Functionality_classes_for_random_number_generators


DRG.1
DRG.2
Seed
Seed
φ
Internal state
φ
Internal state
ψ
ψ
Output
Output
Random number generator output
Random number generator output
01_

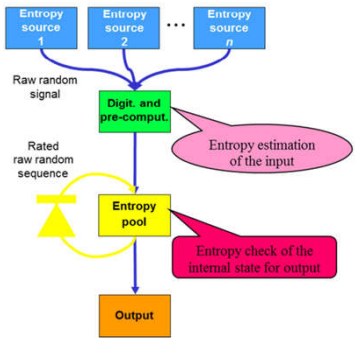

DRG.3
DRG.4
Seed
Reseed/ Seed update
φ
Internal state
φ
Internal state
refresh
ψ
ψ
Output
Output
Random number generator output
Random number generator output
©Petr Hanáček
BZA Slide 34
01_AIS31_Functionality_classes_for_random_number_generators


PTG.1, PTG.2
PTG.1
PTG.2
Entropy source
Entropy source
Raw random signal
Raw random signal
Digiti-sation
Digiti-sation
Total failure test of the source
Total failure test of the source
Raw random sequence
Raw random sequence
Online test of raw random numbers
Post-processing
Post-processing
Internal random sequence
Internal random sequence
Online test of internal random numbers
Output
Output
©Petr Hanáček
BZA Slide 35
01_AIS31_Functionality_classes_for_random_number_generators


PTG.3
PTG.3
Entropy source
Raw random signal
Digiti-sation
Total failure test of the source
Raw random sequence
Online test of raw random numbers
DRNG
Internal random sequence
Known-answer test of cryptographic post-processing
Output
©Petr Hanáček
BZA Slide 36
01_AIS31_Functionality_classes_for_random_number_generators

BZA

NTG.1



01_AIS31_Functionality_classes_for_random_number_generators

NIST Test Suite

The NIST Test Suite consists of 16 statistical tests that were developed to test the randomness of **binary sequences** .

1. The Frequency (Monobit) Test
2. Frequency Test within a Block
3. The Runs Tests
4. Test 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 Lempel-Ziv Compression Test
11. The linear Complexity Test
12. The Serial Test
13. The approximate Entropy Test
14. The Cumulative Sums (Cusums) Test
15. The Random Excursion Test
16. The Random Excursion Variant Test

Source: NIST

NIST Test Suite

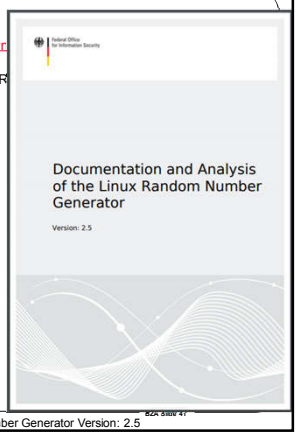
The NIST Statistical Test Suite

Strings Viewed As Random Walks	Look for Patterns		**Complexity/ Compression**
Frequency	Runs	Long Runs	**Rank**
Block Frequency	Aperiodic Templates	Universal Statistical	**Spectral**
Cumulative Sum	Periodic Templates	Serial	**Lempel-Ziv Complexity**
Random Excursions (Variant)	Approximate Entropy		**Linear Complexity**

Source: NIST

NPTRNG Linux

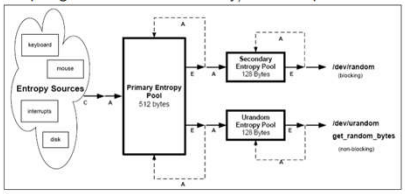
- https://www.bsi.bund.de/SharedDocs/Downl... xRNG/LinuxRNG_EN.pdf
- Documentation and Analysis of the Linux R...



Documentation and Analysis of the Linux Random Number Generator Version: 2.5

Pools and counters

- Internal state kept in three entropy pools:
  - Primary (512 B), secondary (128 B), and urandom (128 B)
  - Entropy sources add data to the primary (or secondary) pool
  - Output generated from secondary/urandom pool

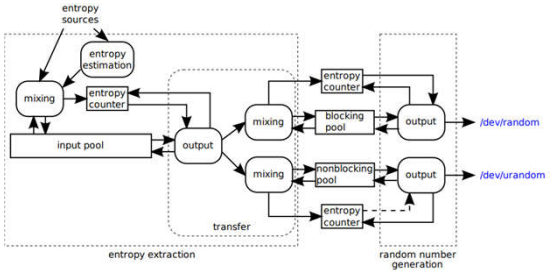


  - Entropy extraction/transfer => feedback (hash of extracted bits)
  - Each pool has its own entropy estimation counter
    - ⋆ Important especially for secondary pool

https://www.fi.muni.cz/~xkrhovj/lectures/2006_PA168_LinuxRNG_slides.pdf

BZA

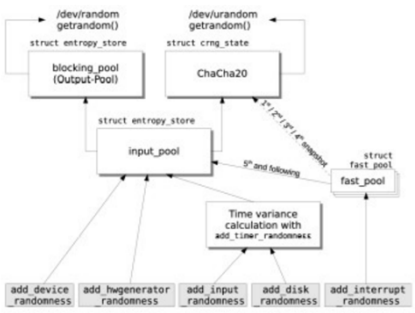
Pools and counters II



http://users.ics.aalto.fi/~arock/slides/slides_devrand.pdf

Relationship of entropy pools, ChaCha20-DRNG and entropy sources

- URL
-



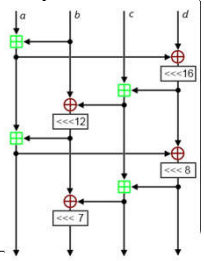
Documentation and Analysis of the Linux Random Number Generator Version: 2.5

ChaCha20

- **ChaCha 20 is a stream cipher developed by Daniel Bernstein.**
- **It is a refinement of the Salsa 20 cipher.**
- **ChaCha 20 works on 4 byte words, takes 16 words of plaintext and outputs 16 words of ciphertext**

Initial state of ChaCha

Cons	Cons	Cons	Cons
Key	Key	Key	Key
Key	Key	Key	Key
Pos	Pos	Nonce	Nonce



https://en.wikipedia.org/wiki/Salsa20

Noise sources

- Device drivers may provide data that the device driver author believes to contain some randomness via the **add_device_randomness** API.
- The Linux kernel implements device drivers for hardware random number generators. They may provide true random data via the **add_hwgenerator_randomness** API. Such hardware random number generators are available in specialized hardware only.
- HID such as keyboard or mice form the next noise source used by the Linux-RNG and may provide entropy via the **add_input_randomness** API. The data obtained by HID events such as a pressed key or mouse movement is supplemented with a time stamp that the Linux-RNG obtains when an event arrives using the add_timer_randomness function.
- Hardware events pertaining to any kind of block devices such as hard disks are obtained by the Linux-RNG with the **add_disk_randomness** API forming another noise source. Events cover read and write operations of a hard disk. The Linux-RNG adds a time stamp to each disk event by invoking the add_timer_randomness function.
- When an interrupt arrives, the Linux-RNG is triggered with the **add_interrupt_randomness** API. For each received interrupt, the Linux-RNG obtains a time stamp and supplemental data which is fed into a fast_pool instance that is local to the CPU on which the interrupt is processed. The use of fast_pool instead of injecting the data directly into

Documentation and Analysis of the Linux Random Number Generator Version: 2.5

add_input_randomness

- **The add_input_randomness function uses the following values as event value that will eventually be mixed into the input_pool:**
- **low 4 bits of the event type ⊕ event code ⊕ high 4 bits of the event code ⊕ event value**

Documentation and Analysis of the Linux Random Number Generator Version: 2.5

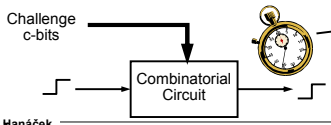
Physical Unclonable Functions (PUFs)

Hanáček, Švenda – Cryptography for WSN

BZA

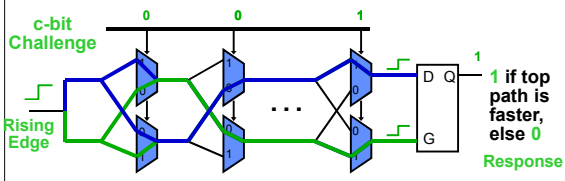
Physical Unclonable Functions (PUFs)

- Extract secrets from a complex physical system
- Because of random process variations, no two integrated circuits even with the same layouts are identical
  - Variation is inherent in fabrication process
  - Hard to remove or predict
  - Relative variation increases as the fabrication process advances
- Delay-Based Silicon PUF concept
  - Generate secret keys from unique delay characteristics of each processor chip



Credits: Steve Trimberger

An Arbiter-Based Silicon PUF



- Compare two paths with an identical delay in design
  - Random process variation determines which path is faster
  - An arbiter outputs 1-bit digital response
- Multiple bits can be obtained by use different challenges
  - Each challenge selects a unique pair of delay paths

Credits: Steve Trimberger

Implementation of security mechanisms

- PUF can be used in practice for four different security mechanisms - for creating unique identifiers for a particular hardware chip (chip signature), for cryptographic authentication, as a source of cryptographic keys for the specific device and as a random number generator. Implementation of cryptographic mechanisms uses three basic properties of PUF:
  - • Inter-chip variation - when generating a bit string using PUF on two different chips, we get two different bit strings.
  - • Intra-chip variation - when generating multiple bit strings using the same PUF chip, we get different bit strings.
  - • Challenge-response property - when generating a bit string using the PUF by entering different challenges we get different responses.

KONEC

4. Pre-defined RNG Classes

256 This chapter defines classes of PTRNG, NPTRNG and DRNG for typical use cases. These classes are hierarchically organized. The definition of these classes is accompanied by application notes explaining their security capabilities and quality metrics.

257 This chapter also identifies the minimum information expected from the technical point of view to fulfil the assurance requirements addressed in the PP or ST. The following paragraphs identify the lowest assurance elements describing the relevant evidence. If the ST requires a component hierarchical to the mentioned assurance component, the equivalent element of the hierarchically-higher component will require analogue information.

4.1. Overview of pre-defined RNG classes

258 This section defines pre-defined RNG classes based on the component FCS_RNG.1. It describes the specific evidence necessary for the evaluation of an RNG class based on a chosen EAL according to the CEM.

259 The security functional requirements of the RNG class are described by means of the component FCS_RNG.1 where

- the RNG type is selected,
- the security capabilities are assigned,
- the quality metric is assigned.

260 The developer shall provide specific information describing how the RNG meets the assurance requirements expressed in a PP or ST for the security functional requirements described in FCS_RNG.1.

261 If the ST defines in FCS_RNG.1 that the TOE supports one of the pre-defined RNG classes, the developer is expected to provide specific information and evidence for the assigned security capabilities and quality of the random numbers according to the content and presentation of elements for the assurance components selected in the ST. If the ST requires a component hierarchical to the mentioned assurance component, the equivalent element of the hierarchically-higher component will require analogue information.

262 The pre-defined RNG classes relate to those described in [AIS20] and [AIS31] as follows (coarse comparisons):

RNG class	Comparable to [AIS20] or [AIS31] class	Comment
PTG.1	AIS31, P1	Physical RNG with internal tests that detect a total failure of the entropy source and non-tolerable statistical defects of the internal random numbers
PTG.2	AIS31, P2	PTG.1, additionally a stochastic model of the entropy source and statistical tests of the raw random numbers

RNG class	Comparable to [AIS20] or [AIS31] class	Comment
		(instead of the internal random numbers)
PTG.3	No counterpart	PTG.2, additionally with cryptographic post-processing (hybrid PTRNG)
DRG.1	AIS20, K2, partly K3	DRNG with forward secrecy according to [ISO18031]
DRG.2	AIS20, K3	DRG.1 with additional backward secrecy according to [ISO18031]
DRG.3	AIS20, K4	DRG.2 with additional enhanced backward secrecy
DRG.4	No counterpart	DRG.3 with additional enhanced forward secrecy (hybrid DRNG)
NTG.1	No counterpart	Non-physical true RNG with entropy estimation

263 The following figures illustrate different classes of RNGs. We point out that other realizations of these classes are possible. The pictures show in pink the total failure tests, in red the online tests and in dark green known-answer-tests.

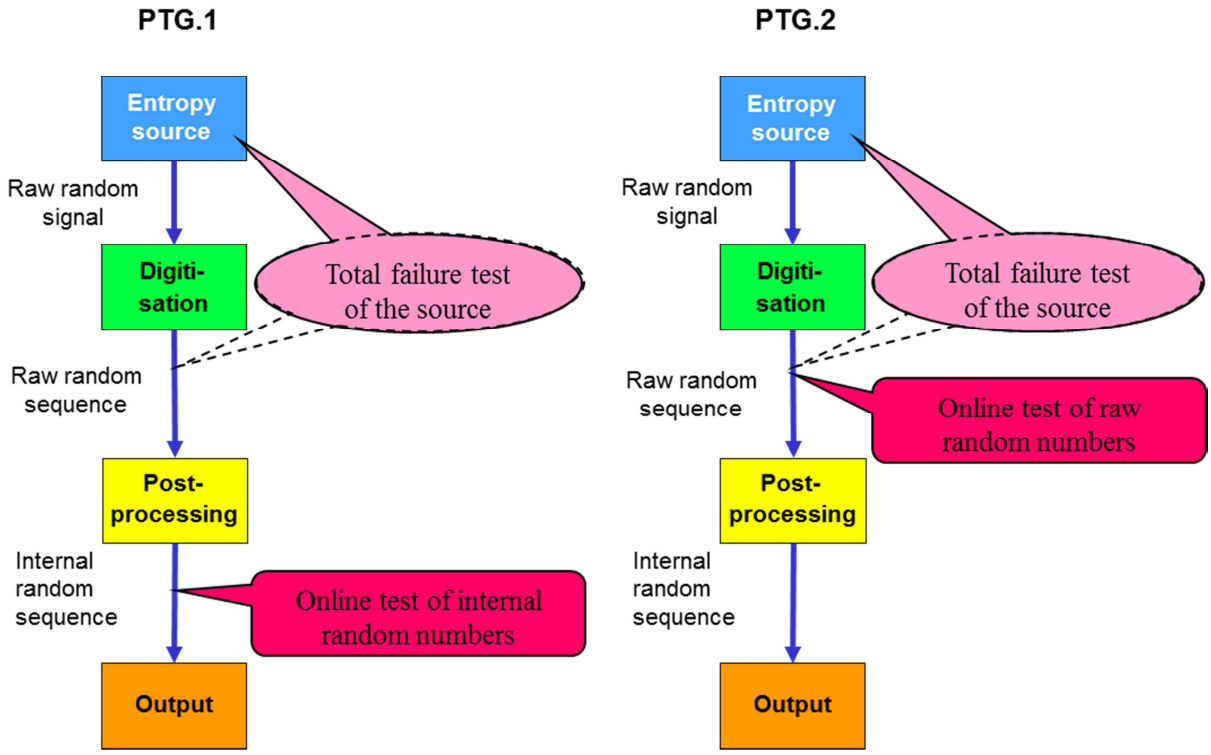


Figure 3: Example of PTRNGs that belong to the pre-defined classes PTG.1 and PTG.2

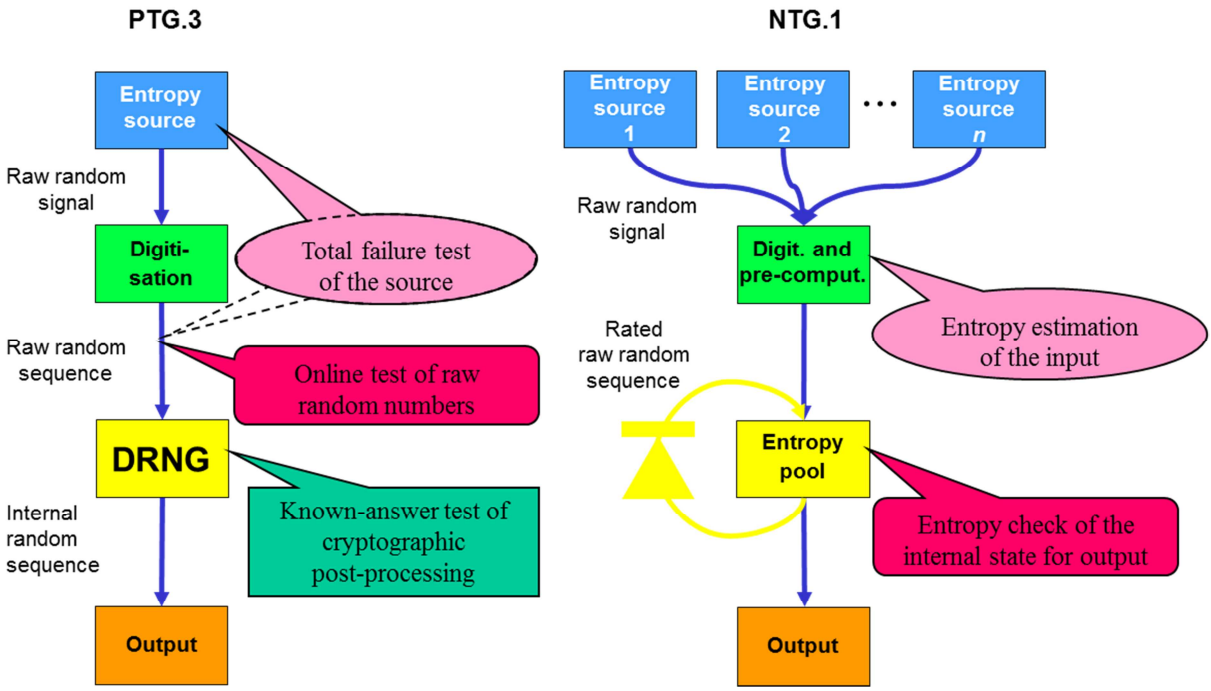


Figure 4: Example of a PTG.3 and NTG.1 that belongs to the pre-defined class PTG.3 and NTG.1

264 The DRNG classes are illustrated by the following figures (φ - state transition function, ψ - output function, A–|–B - symbol for a one-way function for the state transition function φ and the extended output function ψ^*):

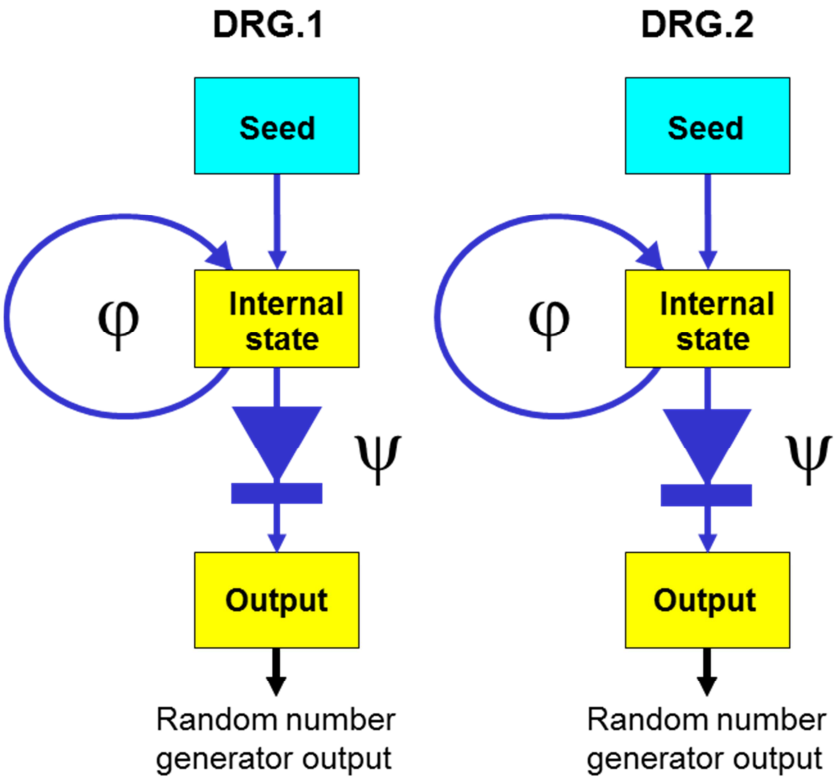


Figure 5: Examples of DRNGs that belong to the pre-defined classes DRG.1 and DRG.2

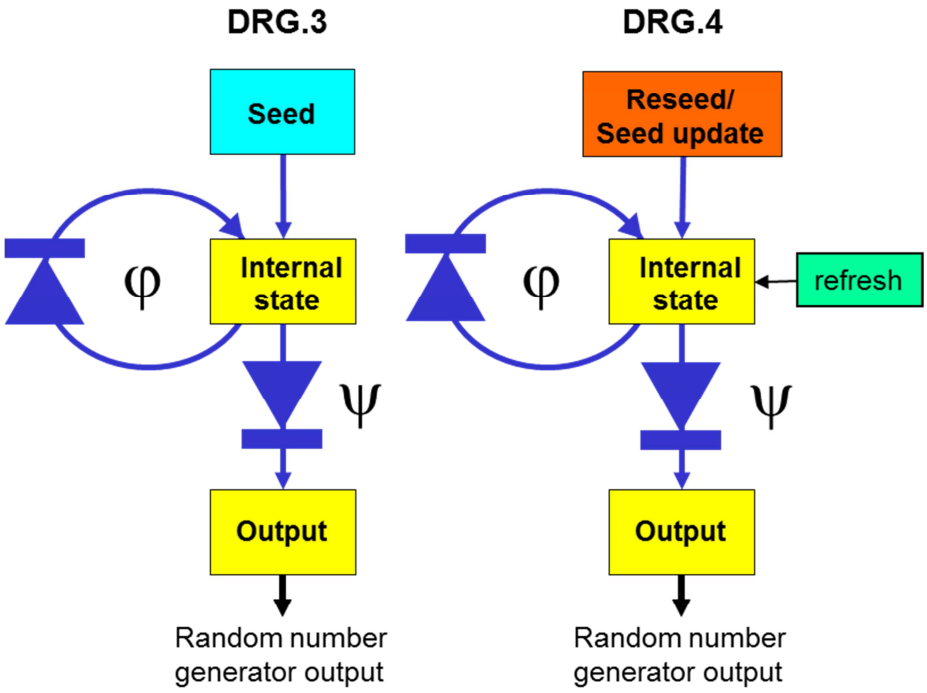


Figure 6: Examples of DRNGs that belong to the pre-defined classes DRG.3 and DRG.4

4.2. General Remarks (Exemplary applications, side-channel attacks, fault attacks)

265 In the description of the particular pre-defined RNG classes below possible (exemplary) cryptographic applications are mentioned that shall motivate the class definitions. We point out that these examples are informative only. In such or in related real-world applications there might be special (additional) requirements on the random numbers, which make it necessary or at least advisable to select an RNG from a higher class. Similarly, possible progress in cryptanalysis might implicate the selection of higher RNG classes in the future. The designer of an application is responsible for the choice of an appropriate RNG.

266 Implementation attacks, in particular side-channel attacks and fault attacks, constitute serious threats against cryptographic implementations. Principally, also RNGs might be concerned.

267 (DRNGs): Although irrelevant from a cryptanalytic (logical) point of view in the light of implementation attacks (in particular, of side-channel attacks) we recommend to use different instantiations of a DRNG for different applications, and maybe even for different tasks within one application (e.g., random numbers remain secret, random numbers remain unknown apart from legitimate users, random numbers might be disclosed later, random numbers are open). Otherwise the attacker might try to perform a side-channel attack, which exploits the known random numbers to recover the internal state, which determines at least all future random numbers.

268 (DRNG): To prevent side-channel attacks it might be recommendable not to keep secret parts of the internal state constant. This would be the case, for instance, for a DRNG that uses a block cipher with constant secret key in OFB mode.

269 (PTG.3) + (DRG.4): Hybrid RNGs combine security properties of both PTRNGs and DRNGs. One may hope that the combination of both an analogue part and algorithmic post-processing might also help to harden RNG implementations against side-channel attacks and fault attacks.

270 From a logical point of view statistical tests on the output data of a cryptographic post-processing algorithm are pointless. However, online tests might serve as additional security measure that might detect fault attacks.

4.3. Class PTG.1

271 The class PTG.1 defines requirements for PTRNGs, which might be used to generate random numbers for cryptographic applications, where the random numbers need not meet any unpredictability properties, neither in the past nor in the future.

272 The required quality metric of PTG.1 does not prevent that the random numbers might be guessed.

4.3.1. Security functional requirements for the RNG class PTG.1

273 The functional security requirements of the class PTG.1 are defined by component FCS_RNG.1 with specific operations as given below.

FCS_RNG.1 Random number generation (Class PTG.1)

FCS_RNG.1.1 The TSF shall provide a *physical*[20] random number generator that implements:

(PTG.1.1) A total failure test detects a total failure of the entropy source immediately when the RNG has started. When a total failure is detected, no random numbers will be output.

(PTG.1.2) If a total failure of the entropy source occurs while the RNG is being operated, the RNG [selection: prevents the output of any internal random numbers that depend on raw random numbers generated after the total failure occurred, generates the internal random numbers with a post-processing algorithm of class DRG.1 until the output of further internal random numbers is prevented].

(PTG.1.3) The online test detects non-tolerable statistical defects of the internal random numbers. The online test is [selection: triggered externally, applied after regular intervals, applied continuously, applied upon specified internal events]. When a defect is detected, the output of further random numbers is prevented.

(PTG.1.4) Within one year of typical use, the probability that an online alarm occurs is in the order of 10^{-6} *or larger if the RNG works properly.*[21]

FCS_RNG.1.2 The TSF shall provide [selection: *bits, octets of bits, numbers [assignment: format of the numbers]*] that meet:

(PTG.1.5) Test procedure A [assignment: additional standard test suites] does not distinguish the internal random numbers from output sequences of an ideal RNG.[22]

4.3.2. Application notes

274 An RNG of class PTG.1 shall generate true random numbers based on an entropy source. A total breakdown of the entropy source causes zero entropy for future raw random numbers. Moreover, a failure of the digitization of the analogue signal of the post-processing algorithm might cause defects of the internal random numbers. The total failure test and the online test shall detect those types of errors, and the TOE shall prevent output of random numbers of poor quality.

275 The total failure test may detect

(i) the breakdown of the entropy source by analyzing the raw random signal, or

(ii) the total failure of the entropy source including the digitization by analyzing the raw random number sequence.

[20] [selection: *physical, non-physical true, deterministic, hybrid physical, hybrid deterministic*]

[21] [assignment: *list of security capabilities*]

[22] [assignment: *a defined quality metric*]

The total failure test must consider the physical principle of the entropy source. In case (i), the total failure test may measure the physical entropy effect. In case (ii), the breakdown analysis of the entropy source usually identifies characteristic patterns in the raw random signal (e.g., constant sequences, meanders) that can be detected quickly by a suitable test.

276 If a total failure of the entropy source occurs the total failure test might need some time to detect this failure, and the post-processing mechanism might delay the effect of the non-random raw random numbers on the output random numbers. The FCS_RNG.1.1 clause (PTG.1) explains the response of the RNG when a total failure of the entropy source has been detected: preventing the output of random numbers. FCS_RNG.1.1 clause (PTG.1.2) addresses the time between the occurrence and the detection of a total failure. The selection covers two types of post-processing algorithms:

277 Case (i): The post-processing mechanism generates internal random numbers that depend only on some fixed number of raw random signals (e.g., the internal random number might be calculated from a raw random number sequence that is stored in a first-in, first-out memory). After a total failure, the raw random signal loses any randomness, and the generated raw random numbers affect the generated internal random numbers in a deterministic way. After some time, the internal random numbers are generated only from non-random raw random numbers. No later than this point the RNG shall prevent the output of internal random numbers because they have *completely been generated after a total failure of the entropy source*.

278 Case (ii): The internal random numbers might principally depend on an unlimited number of raw random numbers (memory!), although the internal state is finite (e.g., the internal random numbers are calculated from raw random numbers that are stored in a feedback shift register). Therefore, the internal random numbers still depend on “truly random” raw random numbers, which have been generated before the entropy source has broken down.. The RNG *generates output with a post-processing mechanism of class DRG.1 until the RNG prevents the output of internal random numbers*, which ensures the quality of the internal random numbers during the time between occurrence and detection of a total failure, as required by element FCS_RNG.1.1 clause (PTG.1.2).

279 The online test shall detect non-tolerable statistical defects of the internal random numbers. As distinct from a total failure, these defects usually can be detected only by statistical tests. The FCS_RNG.1.1 clause (PTG.1.3) addresses the conditions under which the online tests shall be executed. The FCS_RNG.1.1 clause (PTG.1.4) ensures that the online test is sharp enough to detect statistical defects. The *course of one year of typical use of the RNG* is defined by the use of the TOE or by additional evidence provided by the developer. If the internal random numbers pass the online test, the TOE design shall ensure that the output random numbers meet the quality described in (PTG.1.5).

280 If the tot test and / or the online test are not part of the TOE but to be implemented later as an external security measure the applicant must submit a specification of the test(s), a justification for effectiveness and a reference implementation. The suitability of the tot test and the online test shall be verified based on the reference implementation. In the positive case the RNG is said to be PTG.1 compliant under the condition that the final implementation meets the specification in the user manual (to be checked in a composite evaluation).

281 The developer may or may not assign additional standard test suites (i.e. the assignment may be empty) in the element FCS_RNG.1.2 clause (PTG.1.5). The element FCS_RNG.1.2 clause (PTG.1.5) demands that the application of test procedure A and - if assigned – of additional

standard test suites does not reject the null hypothesis “the internal random numbers were generated by an ideal random generator”. The same requirement is demanded for classes PTG.2 and DRG.1. The efforts of testing depend on the claimed resistance (in the ST) against attacks (cf. selected component of the family AVA_VAN). The evaluator may apply additional statistical tests as penetration tests. Note that this requirement does not necessarily imply that the rejection probability for the internal random numbers equals the rejection probability of sequences from ideal RNGs. Moreover, even this enhanced property is weaker than Requirements PTG.3.8, DRG.2.5, DRG.3.5 and DRG.4.7.

4.4. Class PTG.2

282 The class PTG.2 defines requirements for RNGs intended to generate, for example:

- cryptographic keys (e.g., for block ciphers or for RSA),
- random padding bits,
- seeds for DRNGs of the classes DRG.1, DRG.2, DRG.3 or DRG.4, or
- cryptographic applications with similar requirements (in particular, secrecy of the random numbers)

(cf. also par. 265). Roughly speaking, PTG.2 conformant RNGs generate high-entropy random numbers. These random numbers may not be practically indistinguishable from independent uniformly distributed random numbers (output from an ideal RNG). The entropy shall in particular prevent successful guessing attacks. In particular, class PTG.2 includes the applications for class PTG.1.

283 The PTG.2 class specification allows that the internal random numbers may have a small entropy defect, e.g. due to a small bias. When used for the generation of ephemeral keys for DSA signatures or ECDSA signatures, for example, a potential attacker might try to combine information from many signatures, resp. on several ephemeral keys. Although no concrete attack is known to date to stay on the safe side it might be favourable to use a class PTG.3 RNG as a measure of precaution. Similar considerations hold for any other applications, too, where an attacker might be able to collect information on many internal random numbers. The practical relevance of this potential vulnerability clearly depends on the concrete application.

284 The PTG.2 class specification does not require a post-processing algorithm if the raw random numbers are already good enough. However, even then it might be reasonable to apply a post-processing algorithm with memory. The post-processing algorithm might smooth a bias or short-term dependencies. Even if it is not data-compressing the entropy of its internal state might compensate entropy defects of the raw random numbers provided that in the course of the time more random raw bits are fed into the post-processing algorithm than outputted by the PTRNG.

285 For a PTG.2 RNG the post-processing algorithm (if it exists) may not be cryptographic. If the post-processing algorithm belongs to class DRG.2, resp. even to DRG.3, (viewed as a free-running DRNG) this extends the reaction time upon a total failure of the entropy source (PTG.2.2), resp. the PTRNG may even belong to class PTG.3 (cf. PTG.3.6).

4.4.1. Security functional requirements for the RNG class PTG.2

286 Functional security requirements of the class PTG.2 are defined by component FCS_RNG.1 with specific operations as given below.

FCS_RNG.1 Random number generation (Class PTG.2)

FCS_RNG.1.1 The TSF shall provide a *physical*[23] random number generator that implements:

(PTG.2.1) A total failure test detects a total failure of entropy source immediately when the RNG has started. When a total failure is detected, no random numbers will be output.

(PTG.2.2) If a total failure of the entropy source occurs while the RNG is being operated, the RNG [selection: prevents the output of any internal random number that depends on some raw random numbers that have been generated after the total failure of the entropy source, generates the internal random numbers with a post-processing algorithm of class DRG.2 as long as its internal state entropy guarantees the claimed output entropy].

(PTG.2.3) The online test shall detect non-tolerable statistical defects of the raw random number sequence (i) immediately when the RNG has started, and (ii) while the RNG is being operated. The TSF must not output any random numbers before the power-up online test has finished successfully or when a defect has been detected.

(PTG.2.4) The online test procedure shall be effective to detect non-tolerable weaknesses of the random numbers soon.

(PTG.2.5) The online test procedure checks the quality of the raw random number sequence. It is triggered [selection: externally, at regular intervals, continuously, applied upon specified internal events]. The online test is suitable for detecting non-tolerable statistical defects of the statistical properties of the raw random numbers within an acceptable period of time.[24]

FCS_RNG.1.2 The TSF shall provide [selection: *bits, octets of bits, numbers [assignment: format of the numbers]*] that meet:

(PTG.2.6) Test procedure A [assignment: additional standard test suites] does not distinguish the internal random numbers from output sequences of an ideal RNG.

(PTG.2.7) The average Shannon entropy per internal random bit exceeds 0.997.

4.4.2. Application notes

287 The class PTG.2 includes the requirements of class PTG.1 for the security capabilities (PTG.1.1) (PTG.1.3) and (PTG.1.4) as defined in the element FCS_RNG.1.1. It reformulates (PTG.1.2) in the form of (PTG.2.2). The security capability of an online test of the internal

[23] [selection: *physical, non-physical true, deterministic, hybrid physical, hybrid deterministic*]

[24] [assignment: *list of security capabilities*]

random numbers is replaced by online tests of the raw random number sequence, i.e., tests are oriented towards the entropy source. It adds requirements for the minimum entropy of random numbers.

288 The element FCS_RNG.1.1 clause (PTG.2.2) addresses security capabilities that shall ensure the quality of outputted internal random numbers between the occurrence and the detection of a total failure of the entropy source. It allows the output of some internal random numbers that depend on some raw random numbers with zero entropy *if the post-processing algorithm can be viewed as a DRNG of class DRG.2*. If the PTRNG had worked properly before the total breakdown of the entropy source, the entropy of the internal state should be maximal, which limits the maximal number of internal random numbers that may be outputted (PTG.2.7). Let *t* denote the ratio of the bit length of the internal state of the DRNG and the bit size of the internal random numbers. Under the assumption that the DRNG has appropriate mixing properties the RNG may output at most *t* internal random numbers that depend on some raw random numbers that have been generated after the total failure of the entropy source in order to ensure sufficient entropy of the internal random numbers. (Recall that PTG.1.2 only demands a post-processing algorithm of the weaker class DRG.1, and there is no upper bound for the outputted internal random numbers.)

289 The entropy source might be affected, for instance, by aging, tolerances of components, environmental stress, or events like the failure of particular components (e.g., if the PTRNG comprises more than one entropy source). These factors might decrease the entropy of the raw random signal, but might not result in total failure of the (overall) entropy source. Such defects shall be detected by the online test of the raw random number sequence required in the element FCS_RNG.1.1 clause (PTG.2.3).

290 The element FCS_RNG.1.1 requires in clause (PTG.2.3) an online test that detects non-tolerable statistical defects of the *raw random numbers*. The online test of the raw random number sequence in (PTG.2.3) analyses the digitized random signal before post-processing while online tests of the internal random numbers in (PTG.1.3) consider the post-processed random numbers. Note that the online tests for the raw random numbers shall be selected such that the tested statistical properties correspond to possible entropy defects. This is usually not the case for blackbox tests. The rationale for the online tests of the raw random number sequence must be based on stochastic models of the entropy source. The online test procedure shall consider the statistical properties of the raw random numbers. Thus the online tests are usually applied to the raw random numbers. Note, however, that this requirement does not categorically exclude that (in exceptional cases) the online test might operate on analogue values (e.g., to estimate the jitter) or on the internal random numbers. Example: Assume that the RNG generates iid (maybe strongly) biased das random bits (← stochastic model), and that the post-processing algorithm xors non-overlapping pairs of das random bits. Then the internal random numbers are also iid, and the online test could be applied to the internal random numbers as well, indirectly verifying the quality of raw random numbers. In this scenario information on statistical properties of the raw random numbers (here: bias) can easily be translated into information on the raw random numbers, and this essentially corresponds with entropy. However, even then it might be reasonable to test the raw random numbers since a large deviation of the distribution of the raw random numbers implies only a significantly smaller deviation in the internal random numbers, which might be more difficult to detect.

291 If the tot test and / or the online test are not part of the TOE but to be implemented later as an external security measure the applicant must submit a specification of the test(s), a justification for effectiveness and a reference implementation. The suitability of the tot test and the online

test shall be verified based on the reference implementation. In the positive case the RNG is said to be PTG.2 compliant under the condition that the final implementation meets the specification in the user manual (to be checked in a composite evaluation).

292 If the post-processing algorithm does not reduce the average entropy per bit, the average entropy per internal random bit equals at least the average entropy per raw random bit.

293 The term “power-up test” in clause (PTG.2.3) addresses the online test of the raw random number sequence when the RNG is started after the TOE has been powered up (after being powered off), reset, rebooted etc., or after the operation of the RNG has been stopped (e.g., to reduce the power consumption of the TOE). If the PTRNG does not apply a post-processing algorithm (or formally: if it applies the identity mapping), the internal random numbers coincide with the raw random signals. In this case, the clauses (PTG.2.3) and (PTG.2.5) will cover (PTG.1.3) and (PTG.1.4). If the RNG applies a post-processing algorithm, the raw random signals and the internal random numbers usually have different statistical properties. If the raw random number sequence passes the online test and if the post-processing algorithm works correctly, the internal random numbers will have appropriate properties (in particular, sufficient entropy). A temporary or permanent failure in the implementation of the post-processing algorithm might result in non-tolerable entropy defects of the internal random numbers (at least the post-processing algorithm does not work as expected). For many post-processing algorithms, it seems hardly possible to implement effective statistical tests on the internal random numbers, e.g., because post-processing induces complicated dependencies between the internal random numbers. The correctness of the (deterministic) post-processing may be checked while the PTRNG is in operation, e.g., by a known answer test (cf. FPT_TST.1 TSF test).

294 The element FCS_RNG.1.2 clause (PTG.2.7) demands that the average Shannon entropy is at least 0.997 per internal random bit. If the raw random numbers are binary-valued the entropy may be checked by the statistical test procedure B as described above. (Note that if the raw random numbers are Markovian, i.e. if there are no higher dependencies than 1-step dependencies, Step 1 and Step 2 of test procedure B indicate an entropy defect per das random bit of less than 0.002. If the das bits are iid the entropy defect per bit does not exceed 0.0002.) Note that the Min-entropy is the most generally-applicable entropy measure that can be used to estimate the guesswork in vulnerability analysis, but the min-entropy is difficult to quantify.

295 The developer may or may not assign additional test suites (i.e. the assignment may be empty) in the element FCS_RNG.1.2 clause (PTG.2.6). The element FCS_RNG.1.2 clause (PTG.2.6) demands that the application of test procedure A and - if assigned – of additional standard test suites does not reject the null hypothesis “the internal random numbers were generated by an ideal random number generator”. The same requirement is demanded for classes PTG.1 and DRG.1. The efforts of testing depend on the claimed resistance (in the ST) against attacks (cf. selected component of the family AVA_VAN). The evaluator may apply additional statistical tests as penetration tests. Note that this requirement does not necessarily imply that the rejection probability for the internal random numbers equals the rejection probability for sequences from ideal RNGs. Moreover, even this enhanced property is weaker than Requirements PTG.3.8, DRG.2.5, DRG.3.5 and DRG.4.7.

4.4.3. Further aspects

296 The developer *shall* provide evidence that the entropy of the internal random numbers is sufficiently large. The evidence comprises a stochastic model (cf. subsection 2.4.1 on pp. 39)

tailored to the TOE design and substantiated by statistical tests. There is only one level of detail in the description of the stochastic model, irrespective of the chosen EAL. Below this guidance describes explicitly two evaluation methods Method A and Method B, which may be applied to PTRNGs that generate 1-bit raw random numbers.

Method A
A.1 On the basis of the stochastic model, the developer shows that the raw random numbers are stationary distributed, and that there are no significant (long-step) dependencies, which are not covered by the statistical tests from test procedure B.
A.2 The raw random numbers pass the statistical test procedure B under all relevant environmental conditions.
A.3 The developer verifies that the post-processing algorithm does not reduce the entropy per bit. Alternatively, the developer provides evidence that the average entropy per internal random number remains sufficiently large.
A.4 The internal random numbers pass the statistical test procedure A (and other statistical standard test suites if applied) under all relevant environmental conditions.

Method B
B.1 On the basis of the stochastic model, the developer shows that the raw random numbers are stationary distributed, and that there are no significant (long-step) dependencies that are not covered by the statistical tests from test procedure B.
B.2 The developer verifies on the basis of the stochastic model that due to the post-processing algorithm the entropy per internal random number is sufficiently large. Under suitable conditions test procedure B might support this goal.
B.3 The internal random numbers pass the statistical test procedures A (and other statistical standard test suites if applied) and test procedures B under all relevant environmental conditions.

297 In Evaluation Method A, the raw random numbers pass the statistical test suite B under all relevant environmental conditions. In particular, the average entropy per raw random number is sufficiently large. Hence, the post-processing algorithm need not increase the entropy per bit. The identity mapping is allowed, which means ‘no post-processing’.

298 If Evaluation Method B is applied, three cases are possible: (i) test suite B does implicitly verify the entropy of the raw random numbers, and test suite B is passed; (ii) test suite B does implicitly verify the entropy of the raw random numbers, and test suite B fails; or (iii) test suite B cannot be applied or is not applicable to the raw random numbers, e.g. because there is no access to the raw random numbers or the raw random numbers are not binary-valued.

299 Evaluation Method A and Evaluation Method B consider the case that the PTRNG generates a single raw random bit per time unit. If the entropy source generates ($k \geq 1$) – bit raw random numbers, additional problems might occur (e.g., dependencies between the bits of each k-bit raw random number, different statistical behaviour of the particular bit traces) and thus must be considered. The evaluation may follow the line of Evaluation Method A or of Evaluation Method B described above. Depending on the concrete PTRNG design, this might require the specification of a new test suite B’, which shall be at least as effective as test suite B under the conditions of Evaluation Method A, or of Evaluation Method B, respectively. The effectiveness of the chosen test suite B’ shall be verified.

300 In the definition of the different evaluation methods, environmental conditions are viewed as relevant if they either (i) belong to the specified range of admissible working conditions, or (ii)

lie outside that range but cannot reliably be detected by anti-tamper measures (e.g., by sensors) although they may affect the behaviour of the entropy source.

301 Every statistical test considers only particular statistical properties. In particular, there are no generally-applicable blackbox tests that provide reliable entropy estimates for random numbers. To be recognized as effective for a positive verification of security properties, a statistical test must be based on a stochastic model.

4.5. Class PTG.3

302 The class PTG.3 defines requirements for RNGs that shall be appropriate for any cryptographic applications, in particular including those for PTG.2. Unlike PTG.2 - PTRNGs the security of PTG.3 - PTRNGs does not only rely on one security anchor but on two security anchors: information-theoretical security ensured on the physical part of the RNG *and* computational security ensured on the properties of the cryptographic post-processing algorithm. In particular, the internal random numbers will not show any bias nor short term dependencies.

303 PTG.3 is the strongest class that is defined in this document. PTG.3 conformant PTRNGs may be used for any cryptographic application. Typical PTG.3 applications are the generation of ephemeral keys for DSA signatures and for ECDSA signatures, for instance.

304 Class PTG.3 demands a post-processing algorithm with memory that (interpreted as a DRNG) is DRG.3-conformant (cf. chapter 4.8) even if its input data are known at some point in time. In particular, the state transition function φ and the extended output function ψ^* of this DRNG are cryptographic one-way functions.

4.5.1. Security functional requirements for the RNG class PTG.3

305 Functional security requirements of the class PTG.3 are defined by component FCS_RNG.1 with the specific operations given below.

FCS_RNG.1 Random number generation (Class PTG.3)

FCS_RNG.1.1 The TSF shall provide a *hybrid physical* [25] random number generator that implements:

(PTG.3.1) A total failure test detects a total failure of entropy source immediately when the RNG has started. When a total failure has been detected no random numbers will be output.

(PTG.3.2) If a total failure of the entropy source occurs while the RNG is being operated, the RNG [selection: prevents the output of any internal random number that depends on some raw random numbers that have been generated after the total failure of the entropy source, generates the internal random numbers with a post-processing algorithm of class DRG.3 as long as its internal state entropy guarantees the claimed output entropy].

(PTG.3.3) The online test shall detect non-tolerable statistical defects of the raw random number sequence (i) immediately when the RNG is started, and (ii)

[25] [selection: *physical, non-physical true, deterministic, hybrid physical, hybrid deterministic*]

while the RNG is being operated. The TSF must not output any random numbers before the power-up online test and the seeding of the DRG.3 post-processing algorithm have been finished successfully or when a defect has been detected.

(PTG.3.4) The online test procedure shall be effective to detect non-tolerable weaknesses of the random numbers soon.

(PTG.3.5) The online test procedure checks the raw random number sequence. It is triggered [selection: externally, at regular intervals, continuously, upon specified internal events]. The online test is suitable for detecting non-tolerable statistical defects of the statistical properties of the raw random numbers within an acceptable period of time.

(PTG.3.6) The algorithmic post-processing algorithm belongs to Class DRG.3 with cryptographic state transition function and cryptographic output function, and the output data rate of the post-processing algorithm shall not exceed its input data rate.

FCS_RNG.1.2 The TSF shall provide [selection: *bits, octets of bits, numbers [assignment: format of the numbers]*] that meet:

(PTG.3.7) Statistical test suites cannot practically distinguish the internal random numbers from output sequences of an ideal RNG. The internal random numbers must pass test procedure A [assignment: additional test suites].

(PTG.3.8) The internal random numbers shall [selection: use PTRNG of class PTG.2 as random source for the post-processing, have [assignment: work factor], require [assignment: guess work]].

4.5.2. Application notes

306 The security capabilities in element FCS_RNG.1.1 clause (PTG.3.2) ensure the quality of the output in the time period between the occurrence and the detection of a total failure of the entropy source. The internal state of the post-processing algorithm shall ensure that the outputted internal random numbers contain sufficient entropy in this time period. Clause (PTG.3.6) ensures enhanced backward secrecy (cf. (DRG.3.3)) even if the entropy source has broken down and if the internal state is compromised.

307 Clauses (PTG.3.5) and (PTG.3.8) shall ensure that the quality of the internal random numbers is sufficiently large unless a noise alarm occurs.

308 The security capability (PTG.3.8) separates PTG.3-conformant PTRNGs from DRG.4-conformant DRNGs. Essentially, clauses (PTG.3.6) and (PTG.3.8) demand that the average entropy (over the time) of the input data of the algorithmic post-processing algorithm should not be smaller than the average number of internal random bits in the same time period; a small entropy defect might be tolerable. Since the bit length of the internal random numbers is usually much larger than the bit size of the input data of the post-processing algorithm, this requirement might not be fulfilled in short time intervals. However, the entropy of the internal state shall compensate such time-local effects for any time interval, i.e., the entropy of the input data shall not be smaller than the number of internal random bits minus the bit length of the internal state

of the algorithmic post-processing algorithm. The security capabilities (PTG.3.3) and (PTG.3.5) ensure that the internal random numbers contain enough entropy while the PTRNG is in operation.

309 A PTG.3-conformant PTRNG may be viewed as a composition of an "inner" PTRNG with a DRNG post-processing where the output data of the PTRNG serve as input data for the DRNG, which updates / refreshes its internal state. The "inner" PTRNG itself may comprise an 'inner' algorithmic post-processing algorithm. In particular, the output data of the inner PTRNG need not necessarily be raw random numbers but may already be algorithmically post-processed.

310 If we view a PTG.3-conformant PTRNG as a composition of an inner 'PTRNG' part and a DRNG part, we may distinguish two cases: (i) the inner PTRNG is PTG.2-conformant, or (ii) the inner PTRNG is not PTG.2-conformant. For case (i), the RNG may principally be operated in three modes: (a) as a PTG.3-PTRNG, (b) as a PTG.2-PTRNG if the output data of the physical part are used directly, or (c) as a DRG.4-DRNG if the input sequence of the algorithmic post-processing algorithm is 'extended'. For case (ii), only options (a) and (c) remain. Security requirements and functional requirements of particular cryptographic applications might make such a diversification meaningful. Conformity to the particular classes must be verified in separate evaluation processes. Evaluation results clearly may be used for all these evaluations.

311 The post-processing algorithm belongs to class DRG.3 even if the PTRNG random source has totally broken down and an attacker knows or is able to guess its output (i.e., the input of the post-processing algorithm). Of course, this demands that the internal state of the post-processing algorithm was unpredictable at the time of the breakdown, which is the case if the PTRNG had worked properly for at least a short time. In the case of a total breakdown, of course, the PTRNG must not output more internal random bits than the size of the internal state in bits. Otherwise, the RNG belongs to class DRG.3 after this instant.

312 Unlike for PTG.3-conformant PTRNGs, DRG.4-conformant DRNGs may 'extend' the input data, i.e., DRG.4 conformant DRNGs may compute large output sequences from short input sequences. In particular, there is no minimum entropy bound per internal random bit (cf. chapter 4.6 for details).

313 The element FCS_RNG.1.2 clause (PTG.3.7) requires that statistical tests cannot practically distinguish the internal random numbers from output sequences of an ideal RNG. The feature that statistical test suites cannot **practically** distinguish the RNG output from uniformly distributed random bit sequences depends on the claimed in the ST resistance against attacks (cf. selected component of the family AVA_VAN). The effort of testing is defined by the used test suites and the amount of test data. The developer will provide functional tests with test suite A and maybe other appropriate tests suites. The evaluator may additionally apply further statistical tests as penetration tests. These tests may be tailored to the RNG design. Requirement (PTG.3.7) is stronger than (PTG.1.5) and (PTG.2.6)

314 The clause (PTG.3.8) provides three methods describing the quality of the output. The work factor and the guess work may be used directly in the vulnerability analysis of the application using the random number output. The selection "use PTRNG of class PTG.2 as random source" together with (PTG.3.6) allows an indirect verification of the output quality.

315 If the DRNG post-processing algorithm maps the input data from the inner PTRNG bijectively onto the output space, its entropy remains constant. If the inner PTRNG is PTG.2-conformant,

the (Shannon) entropy per random bit is sufficiently large. Since PTG.2-conformant PTRNGs generate stationary random raw sequences, the Shannon entropy provides an appropriate estimate for the work factor unless the sequences are too short, which is not the case for DRG.3-conformant DRNGs. An example of a bijective DRG.3-conformant post-processing algorithm is a block cipher that is operated in OFB mode where the internal random number is given by the whole ciphertext block. Before a new internal random number is output, a fresh random bit string from the inner PTRNG is XORed to that part of the inner state of the DRNG that stores the previous internal random number (= last ciphertext). Then the updated part of the internal state is encrypted and output (internal random number), and then a one-way function is applied to the updated internal state.

316 DRG.3-conformity requires a one-way state transition function. One-way functions usually reduce the entropy per bit unless the length of the input data significantly exceeds the length of the output. One usually models one-way functions as realisations of random mappings. Section 5.4.4 investigates the effect of random mappings on uniformly distributed input data. Output sequences of PTG.2-conformant real-world PTRNGs are not ideal, but should be close enough to the uniform distribution so that it appears reasonable to assume that the figures from section 5.4.4 are also valid for input data from PTG.2-conformant PTRNGs (see section 5.4.4 for further details). An example of this type of post-processing algorithm is the following: Whenever an internal random number shall be output, a fresh n-bit string from the inner PTRNG is XORed to the internal state. Then an m-bit hash value $(m < n)$ of the internal state is output, and the internal state is updated by applying (another) one-way function.

317 If the tot test and / or the online test are not part of the TOE but to be implemented later as an external security measure the applicant must submit a specification of the test(s), a justification for effectiveness and a reference implementation. The suitability of the tot test and the online test shall be verified based on the reference implementation. In the positive case the RNG is said to be PTG.3 compliant under the condition that the final implementation meets the specification in the user manual (to be checked in a composite evaluation).

318 Under certain conditions class PTG.3 allows a composite evaluation. For example, a software developer might use the output of a PTG.2 RNG, which is implemented in hardware on the device, as input for a DRG.3 RNG with memory. In the composite evaluation it has to be checked whether all requirements that concern the post-processing algorithm itself and its interaction with the PTG.2 output are fulfilled. If this is the case the composite RNG (PTG.2 + DRG.3 conformant post-processing) is PTG.3 conformant.

4.5.3. Further aspects

319 The developer *shall* provide the evidence required in ATE_FUN.{1,2}.PTG.3.2 clause (PTG.3.8), i.e., the developer shall provide evidence that the entropy of the internal random numbers is sufficiently large. The evidence comprises a stochastic model (cf. subsection 2.4.1 on pp. 39) tailored to the TOE design and substantiated by statistical tests. There is only one level of detail in the description of the stochastic model, irrespective of the chosen EAL. The stochastic model shall consider the situation before the application of the DRG.3 post-processing algorithm, i.e., the input data of the post-processing algorithm (as for PTG.2-conformant PTRNGs) and the effect of this post-processing.

320 For the non-post-processed data, this guidance describes Method A* and Method B*, which may be applied if 1-bit raw random numbers are generated. These evaluation methods are related to Method A and Method B for PTG.2-PRNGs.

321 **Method A***

A.1 On the basis of the stochastic model the developer shows that the raw random numbers are stationary distributed, and that there are no significant (long-step) dependencies, which are not covered by the statistical tests from test suite B.

A.2 The raw random numbers pass the statistical test suite B under all relevant environmental conditions.

A.3 The developer verifies that the inner post-processing algorithm (if one exists, resp. if it is different from the identity mapping) does not reduce the entropy per bit. Alternatively, the developer provides evidence that the average entropy per internal random number remains sufficiently large.

A.4 The internal random numbers pass the statistical test procedure A (and other statistical standard test suites if applied) under all relevant environmental conditions.

322 **Method B***

B.1 On the basis of the stochastic model the developer shows that the raw random numbers are stationary distributed, and that there are no significant (long-step) dependencies that are not covered by the statistical tests from test suite B.

B.2 The developer verifies on the basis of the stochastic model and the inner post-processing algorithm (if it exists, resp. if it is different from the identity mapping) that the average entropy per input bit of the DRG.3-post-processing exceeds a certain entropy bound (to be specified).

B.3 The internal random numbers pass the statistical test procedure A (and other statistical standard test suites if applied) under all relevant environmental conditions.

323 Method A* guarantees average of 0.997 bit Shannon entropy per input bit (i.e., 7,976 bit Shannon entropy per input octet) of the post-processing algorithm. To estimate the average entropy per output bit, one may apply results on random mappings or random permutations.

324 Evaluation Method A* and Evaluation Method B* require that the PTRNG generates a single random raw bit per time unit. If the entropy source generates k-bit raw random numbers ($k > 1$), additional problems (e.g., dependencies between the bits of each k-bit raw random numbers, different statistical behaviour of the particular bit traces) might occur and thus must be considered. The evaluation may follow the line of Evaluation Method A* or of Evaluation B* described above. Depending on the concrete PTRNG design, this might require the specification of a new test procedure B', which shall be at least as effective as test procedure B under the conditions of Evaluation Method A*, or of Evaluation Method B*, respectively. The effectiveness of the chosen test procedure B' shall be explained.

325 In the definition of the different evaluation methods, environmental conditions are viewed as relevant if they either (i) belong to the specified range of admissible working conditions, or (ii) lie outside that range but cannot reliably be detected by anti-tamper measures (e.g., by sensors) although they may affect the behaviour of the entropy source.

326 Each statistical test considers only particular statistical properties. In particular, there are no generally-applicable blackbox tests that provide reliable entropy estimates for random numbers. To be recognized as effective for a positive verification of security properties, a statistical test must be based on a stochastic model.

327 Statistical tests, which estimate the entropy of a random sequence, (tacitly) assume that the sequence has specific properties. The rationale behind the evaluation methods A* and B* is that

statistical tests cannot effectively be applied to the internal random numbers because the DRG.3 post-processing algorithm causes complicated dependencies within the internal random number sequence.

4.6. Class DRG.1

328 The class DRG.1 defines requirements for deterministic RNGs. It shall not be possible to distinguish the generated random numbers from output sequences from an ideal RNG by simple statistical blackbox tests. DRG.1 conformant DRNGs provide forward secrecy.

329 An RNG of class DRG.1 might be used for applications that need fresh data that are distinct from previously-generated data with high probability, e.g., to generate challenges in cryptographic protocols or initialization vectors for block ciphers in special modes of operation, provided that previous random numbers need not be protected. DRG.1-conformant DRNGs may be used for zero-knowledge proofs (cf. par. 265).

4.6.1. Security functional requirements for the RNG class DRG.1

330 Functional security requirements of class DRG.1 are defined by component FCS_RNG.1 with specific operations as given below.

FCS_RNG.1 Random number generation (Class DRG.1)

FCS_RNG.1.1 The TSF shall provide a *deterministic*[26] random number generator that implements:

(DRG.1.1) If initialized with a random seed [selection: using a PTRNG of class PTG.2 as random source, using a PTRNG of class PTG.3 as random source, using an NPTRNG of class NTG.1 [assignment: other requirements for seeding]], the internal state of the RNG shall [selection: have [assignment: amount of entropy], have [assignment: work factor], require [assignment: guess work]].

(DRG.1.2) The RNG provides forward secrecy.[27]

FCS_RNG.1.2 The TSF shall provide random numbers that meet:

(DRG.1.3) The RNG, initialized with a random seed [assignment: requirements for seeding], generates output for which [assignment: number of strings] strings of bit length 128 are mutually different with probability [assignment: probability].

(DRG.1.4) Test procedure A [assignment: additional standard test suites] does not practically distinguish the random numbers from output sequences of ideal RNGs.[28]

4.6.2. Application notes

[26] [selection: *physical, non-physical true, deterministic, hybrid physical, hybrid deterministic*]

[27] [assignment: *list of security capabilities*]

[28] [assignment: *a defined quality metric*]

331 The vulnerability analysis shall show that an attacker is not able to guess the internal state of the DRNG with the attack potential that is claimed in the ST. The value assigned in clause (DRG.1.1) ***shall*** meet the attack potential identified in the vulnerability analysis component. The entropy of the initial internal state is an upper bound of the entropy of the generated random number sequence. The entropy of the internal state may decrease over the lifetime of the DRNG instantiation. The internal state of the DRNG ***shall*** contain sufficient entropy to prevent successful guessing attacks within the lifetime of the DRNG instantiation. Table 12 gives lower entropy bounds for the internal state. Its bit length must at least equal the minimal entropy bound.

332 It is natural to generate the seed of a DRNG with a PTG.2- or PTG.3-conformant PTRNG. If the internal state of the DRNG is initialized in this way, and if the internal state is at least 25% larger (in bits) than the Min-entropy bounds given in Table 12, an explicit assessment of the Min-entropy is not necessary. This is justified by the fact that PTG.2-conformant PTRNGs generate stationary output sequences with large Shannon entropy, which ensures a large work factor. Similarly, PTG.3-conformant PTRNGs also generate high-entropy random numbers (see also the application notes for class PTG.3). We point out that tighter entropy bounds for PTG.2 and PTG.3 than the generic 25% margin (allowing smaller internal states) should be possible in most cases but require justification (cf. stochastic model).

Table 12: Attack potential, Min-entropy, and recommended length of the internal state

Component of the vulnerability analysis			Required min-entropy of the internal state	Recommended length of the internal state
CC version 2.3		CC version 3.1		
		AVA_VAN.1, 2 (basic)	≥ 40 bit	≥ 80 bit
AVA_SOF.1, low	AVA_VLA.2 (low)	AVA_VAN.3 (enhanced basic)	≥ 48 bit	≥ 96 bit
AVA_SOF.1, medium	AVA_VLA.3 (moderate)	AVA_VAN.4 (moderate)	≥ 64 bit	≥ 128 bit
AVA_SOF.1, high	AVA_VLA.4 (high)	AVA_VAN.5 (high)	≥ 100 bit	≥ 200 bit

333 The (Min-)entropy of the internal state clearly depends on the initialization procedure with a random seed, but also on the state transition function φ and possibly publicly known input. If the state transition function is bijective (i.e., $S = \varphi(S,i)$, for each publicly known input i), it maintains the entropy of the internal state. If the state transition function behaves like a randomly-selected mapping, it will reduce the number of possible internal states for some observed public input data i (i.e. $|\varphi(S,i)| < |S|$), thus reducing the entropy of the internal state (cf. [FlOd89] for details about statistics of random mappings). The publicly-known input does not increase the overall entropy of the system, but it might influence the process of entropy reduction of the internal state over time and make attacks more difficult.

334 The security capability (DRG.1.2) forward secrecy means the following: subsequent (future) values cannot be determined or guessed with non-negligible probability from current or

previous output values [ISO18031]. In particular, the design of the DRNG shall prevent a successful guess the internal state, allowing the calculation of future output. The forward secrecy capability requires that the internal state has sufficient entropy to prevent guessing as well as the confidentiality of the current internal state is protected by the design of the DRNG (e.g., one-way extended output function) and the security architecture of the TSF (cf. self-protection accessed in ADV_ARC).

335 The security capability (DRG.1.3) requires the DRNG to generate mutually different pseudo-random numbers. The first assignment in the element FCS_RNG.1.2 describes the requirements on the seeding procedure (e.g., the entropy of the seed, how many random numbers can be generated between two seeding procedures). Under this assumption, clause (DRG.1.3) defines the capability of the DRNG to generate an assigned number (let's say k) of fresh random strings with given length of 128 bit being mutually different with at least the defined probability (let's say $1-\varepsilon$, i.e. ε is the probability of at least one coincidence). If the DRNG generates shorter output values (random numbers) several consecutive output values are concatenated and, if necessary, these joint random bit strings are cut off after 128 bits. The selection of the parameters k and ε depends on the intended application of the DRNG described in the guidance documentation:

- the requirements for seeding shall fit to the intended use cases, and
- during the lifetime of the DRNG instantiation (i.e., during the time between two seeding processes) the DRNG must not produce more random bits than the product of the assigned number k of output strings by their bit length.

The assigned number of strings, string length in bits, and probability shall allow to provide evidence demonstrating that this requirement is fulfilled. The parameters assigned in the element FCS_RNG.1.1 shall meet the attack potential identified in the vulnerability analysis component.

336 Table 13 provides necessary conditions to resist attacks, which are based on the repetition of random strings generated by the RNG and are exploitable in the intended environment with the identified attack potential. The TOE might be vulnerable if the RNG is used for purposes that require other properties of the RNG. In this case, the developer shall consider an RNG with additional security features, like class DRG.2 and higher.

Table 13: Requirements for the parameters in (DRG.1.3) depending on claimed attack potential

Component of the vulnerability analysis			Parameter k denotes the number of output strings that shall be mutually different with probability $\geq 1 - \varepsilon$
CC version 2.3		CC version 3.1	
		AVA_VAN.{1, 2} (basic)	$k > 2^{14}$ and $\varepsilon < 2^{-8}$
AVA_SOF.1, low	AVA_VLA.2 (low)	AVA_VAN.3 (enhanced basic)	$k > 2^{19}$ and $\varepsilon < 2^{-10}$
AVA_SOF.1, medium	AVA_VLA.3 (moderate)	AVA_VAN.4 (moderate)	$k > 2^{26}$ and $\varepsilon < 2^{-12}$
AVA_SOF.1, high	AVA_VLA.4 (high)	AVA_VAN.5 (high)	$k > 2^{34}$ and $\varepsilon < 2^{-16}$

337 For an ideal RNG the probability for at least one collision within the first k 128 bit output strings is approximately $1-\exp\left(-k^2 / 2^{129}\right)$. The expected number of bit strings until the first collision can be approximated by $\sqrt{\pi}2^{63,5}$ (cf. formulae (30) and (31)).

338 The developer may or may not assign additional standard test suites (i.e. the assignment is empty) in the element FCS_RNG.1.2 clause (DRG.1.4). The effort of testing to demonstrate that test procedure A and the assigned test suites do not practically distinguish the RNG output from uniformly distributed random bit sequences depends on the resistance against attacks as claimed in the ST (cf. selected component of the family AVA_VAN). The quality metric (DRG.1.4) is different from (PTG.1.5):

- the class PTG.1 generates **true random** numbers, which cannot be distinguished from ideal random numbers by tests with the test procedure A and - if assigned in clause (PTG.1.5) – with the additional standard test suites,

- the DRG.1 generates **deterministic random** numbers, but they cannot be distinguished from ideal random numbers by tests with the test procedure A and - if assigned in clause (DRG.1.4) - with the additional standard test suites.

The DRNG gets its initial random state from a randomly selected seed. The most straight-forward methods are to use the seed as the initial internal state or to apply the state transition function to the seed. The output string from the initial internal state is part of the deterministically-generated output sequence. The entropy of the output string (and, therefore, of the random numbers generated) cannot be greater than the entropy of the seed. Depending on the seeding procedure of the DRNG, the tests are applied to one or more output strings.

4.6.3. Further aspects

339 In many cases it may be practically infeasible to specify the distribution p_A of the first internal state. It suffices to specify a set of distributions that contains p_A if all elements of this set fulfil

the requirements of class DRG.1. Example: The seed entropy exceeds a certain lower entropy bound, e.g. because the seed has been generated with an RNG that is conformant to class PTG.2, PTG.3 or NTG.1. The security architecture description shall describe the secure initialization process of the RNG.

4.7. Class DRG.2

340 The class DRG.2 defines requirements for deterministic RNGs. It shall not be possible to distinguish the generated random numbers from output sequences from an ideal RNG by statistical tests, and the generated random numbers sequence shall have at least some minimum amount of Min-entropy (contained in the seed), and backward secrecy is ensured. The class DRG.2 includes the properties of class DRG.1.

341 RNGs of class DRG.2 may be used for the generation of cryptographic keys and parameters, pseudo-random padding bits, etc. (cf. par. 265). The TSF protects the internal state of the RNG from being compromised.

4.7.1. Security functional requirements for the RNG class DRG.2

342 Functional security requirements of the class DRG.2 are defined by component FCS_RNG.1 with specific operations as given below.

FCS_RNG.1 Random number generation (Class DRG.2)

FCS_RNG.1.1 The TSF shall provide a *deterministic*[29] random number generator that implements:

(DRG.2.1) If initialized with a random seed [selection: using a PTRNG of class PTG.2 as random source, using a PTRNG of class PTG.3 as random source, using an NPTRNG of class NTG.1 [assignment: other requirements for seeding]], the internal state of the RNG shall [selection: have [assignment: amount of entropy], have [assignment: work factor], require [assignment: guess work]].

(DRG.2.2) The RNG provides forward secrecy.

(DRG.2.3) The RNG provides backward secrecy. [30]

FCS_RNG.1.2 The TSF shall provide random numbers that meet:

(DRG.2.4) The RNG, initialized with a random seed [assignment: requirements for seeding], generates output for which [assignment: number of strings] strings of bit length 128 are mutually different with probability [assignment: probability].

(DRG.2.5) Statistical test suites cannot practically distinguish the random numbers from output sequences of an ideal RNG. The random numbers must pass test procedure A [assignment: additional test suites]. [31]

[29] [selection: *physical, non-physical true, deterministic, hybrid physical, hybrid deterministic*]

[30] [assignment: *list of security capabilities*]

4.7.2. Application notes

343 Class DRG.2 includes the requirements of class DRG.1 for the security capability (DRG1.1) and the quality of the random numbers, (DRG.1.2) and (DRG.1.3). The application notes for class DRG.1 are valid for class DRG.2, as well. The (DRG.2.4) parameters must meet the conditions in Table 13.

344 The class DRG.2 requires assigned quality of internal state in (DRG.2.1). The clauses (DRG.2.2) and (DRG.2.3) require forward and backward secrecy according to [ISO18031], i.e., that unknown previous or future output values cannot be determined from known output values (random numbers). The developer should select a cryptographic function for the one-way extended output function and should consider (but is not requested to choose) a cryptographic one-way function for the state transition function, as well. The whole internal state is viewed as input, including cryptographic keys. Hence, keyed bijections, typically coming from strong block ciphers, also count as one-way functions (cf. section 5.3 for details).

345 If the DRNG is intended for the generation of cryptographic keys, the entropy in the element FCS_RNG.1.1 clause (DRG.2.1), should meet the security level of the cryptographic algorithm. If no cryptographic weaknesses are known, the security level of a symmetric cryptographic algorithm is assumed to be equal to its key length. For example, if the assignment in the element FCS_RNG.1.1 clause (DRG.2.1) assigns ≥ 128 bit Min-entropy to its internal state, the DRNG may be used to generate AES-128 bits. If the internal state contains less entropy, the AES key generation by means of such a DRNG might be viewed as a potential vulnerability of the cryptosystem.

346 The element FCS_RNG.1.2, clause (DRG.2.5), requires that statistical tests cannot **practically** distinguish the random numbers from output sequences of an ideal RNG. The effort of testing in order to demonstrate that the statistical test suites cannot practically distinguish the RNG output from uniformly distributed random bit sequences depends on the resistance against attacks as claimed in the ST (cf. selected component of the family AVA_VAN). The effort of testing is defined by the used test suites and the amount of test data. The developer shall provide functional tests with test procedure A and maybe other appropriate tests suites or specific tests that are tailored to the particular DRNG. The evaluator may provide additional statistical test as penetration tests. Of course, clause (DRG.2.5) excludes 'unfair' tests that exploit the knowledge of the given internal state.

347 Note that the one-way property of the output function is a necessary condition for forward secrecy, but not a sufficient condition for good statistical properties of the DRNG output. For example, if the DRNG outputs the hash value of the internal state, the output is expected to be indistinguishable from the output of an ideal RNG. If the DRNG output function concatenates statistically weak strings (e.g., a sequence number of the output) to this hash values, this might no longer be true.

4.7.3. Further aspects

348 In many cases it may be practically infeasible to specify the distribution p_A of the first internal state. It suffices to specify a set of distributions that contains p_A if all elements of this set fulfil the requirements of class DRG.2. Example: The seed entropy exceeds a certain lower entropy bound, e.g. because the seed has been generated with an RNG that is conformant to class

[31] [assignment: *a defined quality metric*]

PTG.2, PTG.3 or NTG.1. The security architecture description shall describe the secure initialization process of the RNG.

4.8. Class DRG.3

349 The class DRG.3 defines requirements for deterministic RNGs. It shall not be possible to distinguish the generated random numbers from output sequences from an ideal RNG by statistical tests, and the generated random numbers sequence shall have at least some minimum amount of Min-entropy (contained in the seed), and enhanced backward secrecy is ensured. The class DRG.3 includes the requirements of class DRG.2.

350 RNGs of class DRG.3 might be used for the generation of cryptographic keys and parameters, pseudo-random padding bits, etc. (cf. par. 265). Any compromise of the internal state of the DRNG shall be detected, and re-seeding shall be enforced before further use of the RNG.

4.8.1. Security functional requirements for the RNG class DRG.3

351 Functional security requirements of the class DRG.3 are defined by component FCS_RNG.1 with specific operations as given below.

FCS_RNG.1 Random number generation (Class DRG.3)

FCS_RNG.1.1 The TSF shall provide a *deterministic*[32] random number generator that implements:

(DRG.3.1) If initialized with a random seed [selection: using a PTRNG of class PTG.2 as random source, using a PTRNG of class PTG.3 as random source, using an NPTRNG of class NTG.1 [assignment: other requirements for seeding]], the internal state of the RNG shall [selection: have [assignment: amount of entropy], have [assignment: work factor], require [assignment: guess work]].

(DRG.3.2) The RNG provides forward secrecy.

(DRG.3.3) The RNG provides backward secrecy even if the current internal state is known. [33]

FCS_RNG.1.2 The TSF shall provide random numbers that meet:

(DRG.3.4) The RNG, initialized with a random seed [assignment: requirements for seeding], generates output for which [assignment: number of strings] strings of bit length 128 are mutually different with probability [assignment: probability].

(DRG.3.5) Statistical test suites cannot practically distinguish the random numbers from output sequences of an ideal RNG. The random numbers must pass test procedure A [assignment: additional test suites].[34]

[32] [selection: *physical, non-physical true, deterministic, hybrid physical, hybrid deterministic*]

[33] [assignment: *list of security capabilities*]

4.8.2. Application notes

352 The class DRG.3 includes the requirements of class DRG.2 for the security capabilities (DRG.2.1), (DRG.2.2), and (DRG.2.3), and the quality metrics in (DRG.2.4) and (DRG.2.5). The application notes for class DRG.2 are valid for the class DRG.3 as well. It adds requirements for security capabilities referring to enhanced backward secrecy in (DRG.3.3).

353 While (DRG.2.2) and (DRG.2.3) require forward and backward secrecy (i.e., unknown output value cannot be determined from known output values), the security capabilities (DRG.3.2) and (DRG.3.3) additionally require enhanced backward secrecy. This means that previous output values cannot even be determined with knowledge of the current internal state and current and future output values. Enhanced backward secrecy might be relevant, for instance, for software implementations of a DRNG when the internal state has been compromised while all random numbers generated in the past shall remain secret (e.g., cryptographic keys).

354 Clause (DRG.3.3) essentially requires a cryptographic state transition function. DRG.3-conformant designs with non-cryptographic output functions may exist. However, it is recommended to apply a cryptographic output function.

4.8.3. Further aspects

355 In many cases it may be practically infeasible to specify the distribution p_A of the first internal state. It suffices to specify a set of distributions that contains p_A if all elements of this set fulfil the requirements of class DRG.3. Example: The seed entropy exceeds a certain lower entropy bound, e.g. because the seed has been generated with an RNG that is conformant to class PTG.2, PTG.3 or NTG.1. The security architecture description shall describe the secure initialization process of the RNG.

4.9. Class DRG.4

356 Class DRG.4 defines requirements for hybrid deterministic RNGs that primarily rely on the security imposed by computational-complexity, which is 'enhanced' by additional entropy from a physical true RNG. RNGs of class DRG.4 clearly may be used for the same cryptographic applications as DRG.3-conformant DRNGs, and additionally for applications that require enhanced forward secrecy.

357 Class DRG.4 is based on class DRG.3 but may not use an external source of randomness for the seeding process. RNGs of class DRG.4 contain an internal source of randomness for seeding and reseeding, resp. seed-update (to ensure forward secrecy).

4.9.1. Security functional requirements for the RNG class DRG.4

358 Functional security requirements of the class DRG.4 are defined by component FCS_RNG.1 with specific operations as given below.

FCS_RNG.1 Random number generation (Class DRG.4)

[34] [assignment: *a defined quality metric*]

FCS_RNG.1.1 The TSF shall provide a *hybrid deterministic*[35] random number generator that implements:

(DRG.4.1) The internal state of the RNG shall [selection: use PTRNG of class PTG.2 as random source, have [assignment: work factor], require [assignment: guess work]].

(DRG.4.2) The RNG provides forward secrecy.

(DRG.4.3) The RNG provides backward secrecy even if the current internal state is known.

(DRG.4.4) The RNG provides enhanced forward secrecy [selection: on demand, on condition [assignment: condition], after [assignment: time]].

(DRG.4.5) The internal state of the RNG is seeded by an [selection: internal entropy source, PTRNG of class PTG.2, PTRNG of class PTG.3, [other selection]].[36]

FCS_RNG.1.2 The TSF shall provide random numbers that meet:

(DRG.4.6) The RNG generates output for which [assignment: number of strings] strings of bit length 128 are mutually different with probability [assignment: probability].

(DRG.4.7) Statistical test suites cannot practically distinguish the random numbers from output sequences of an ideal RNG. The random numbers must pass test procedure A [assignment: additional test suites].[37]

4.9.2. Application notes

359 Class DRG.4 includes the requirements of class DRG.3 for the security capabilities (DRG.3.1) and (DRG.3.3) and the quality metrics of (DRG.3.4) and (DRG.3.5). The assignment in clause (DRG.4.1) should meet the conditions presented in Table 13 (cf. also the application notes for DRG.2). The assignment in clause (DRG.4.6) should meet the conditions of Table 13. (DRG.4.1) and (DRG.4.6) do not depend on an external entropy source because the RNG is seeded by the internal random source identified in (DRG.4.5). Under this consideration, the application notes for class DRG.3 are applicable for the class DRG.4.

360 DRG.4 includes the forward secrecy according to [ISO18031], i.e., subsequent (future) values cannot be determined from current or previous output values, and adds requirements for enhanced forward secrecy (DRG.4.4), i.e., after the identified event or time, the subsequent (future) output values cannot be determined from current or previous output values, even if the current internal state is compromised.

361 The selection in FCS_RNG.1.1 clause (DRG.4.4) depends on the implementation of the reseeding process, resp. of seed update process. The TOE may provide forward secrecy on demand, e.g., if the RNG is used for the generation of sensitive cryptographic keys like a

[35] [selection: *physical, non-physical true, deterministic, hybrid physical, hybrid deterministic*]

[36] [assignment: *list of security capabilities*]

[37] [assignment: *a defined quality metric*]

signature-creation key in a smart card. The TOE may provide forward secrecy on condition or after time, e.g., if the RNG gets continuously fresh entropy from the internal entropy source. The assignments shall consider the seeding procedure and the entropy, which is provided by the internal physical true RNG.

362 The security capability of forward secrecy in (DRG.4.4) requires fresh entropy that is provided by the internal source of randomness for reseeding or seed-updating the internal state. If "*internal entropy source"* is selected in clause (DRG.4.5), the RNG shall implement mechanisms (entropy estimator) to ensure that the internal entropy source has provided sufficient entropy to ensure forward secrecy. A combination of an online test of the internal entropy source and the condition selected in (DRG.4.4) may ensure a (suitably large) lower entropy bound. If "*physical true RNG of class PTG.2*" or "*physical true RNG of class PTG.3*" is selected, these tests are required as security capabilities of the PTG class (cf. to class definition above). Note that the selection in clauses (PTG.4.1) and (PTG.4.5) shall be consistent if an internal entropy source is used for seeding.

363 The (first) seeding of the internal state might be done within a personalization process with an external entropy source. This ensures that the internal state is unknown from the beginning even if forward secrecy is assured only on demand and if the first application does not apply for forward secrecy.

4.9.3. Further aspects

364 In many cases it may be practically infeasible to specify the distribution p_A of the first internal state. It suffices to specify a set of distributions that contains p_A if all elements of this set fulfil the requirements of class DRG.4. Example: The seed entropy exceeds a certain lower entropy bound, e.g. because the seed has been generated with an RNG that is conformant to class PTG.2 or PTG.3. The security architecture description shall describe the secure initialization process of the RNG.

365 The security architecture must protect the internal state of a DRNG as one aspect of self-protection. If the internal state has been compromised backward secrecy DRG.4.3 ensures the secrecy of all previous random numbers while enhanced forward secrecy DRG.4.4 ensures the secrecy of the random numbers that will be generated after the next reseeding, resp. the next seed update. However, if an attacker knows the current internal state he may calculate all output values that are generated before the next reseeding, resp. the next seed update.

4.10. Class NTG.1

366 The class NTG.1 defines requirements for non-physical true RNGs that rely on information-theoretical security (similar as physical RNGs) but use external input signals as entropy source. Additionally, a suitable cryptographic post-processing algorithm shall provide a second security anchor.

4.10.1. Security functional requirements for the NPTRNG class NTG.1

367 Functional security requirements of the class NTG.1 are defined by the component FCS_RNG.1 with specific operations as given below.

FCS_RNG.1 Random number generation (Class NTG.1)

FCS_RNG.1.1 The TSF shall provide a *non-physical true*[38] random number generator that implements:

(NTG.1.1) The RNG shall test the external input data provided by a non-physical entropy source in order to estimate the entropy and to detect non-tolerable statistical defects under the condition [assignment: requirements for NPTRNG operation].

(NTG.1.2) The internal state of the RNG shall have at least [assignment: Min-entropy]. The RNG shall prevent any output of random numbers until the conditions for seeding are fulfilled.

(NTG.1.3) The RNG provides backward secrecy even if the current internal state and the previously used data for reseeding, resp. for seed-update, are known. [39]

FCS_RNG.1.2 The TSF shall provide random numbers that meet:

(NTG.1.4) The RNG generates output for which [assignment: number of strings] strings of bit length 128 are mutually different with probability [assignment: probability].

(NTG.1.5) Statistical test suites cannot practically distinguish the internal random numbers from output sequences of an ideal RNG. The internal random numbers must pass test procedure A [assignment: additional test suites].

(NTG.1.6) The average Shannon entropy per internal random bit exceeds 0.997.[40]

4.10.2. Application notes

368 A non-physical true RNG comprises three parts:

- the input pre-computation block, which computes the input for the internal DRNG from several external input signals provided by (usually several) entropy sources,
- the entropy pool, which collects entropy and computes the output,
- the control block, which prevents the output of random numbers until the RNG has sufficient entropy to ensure the randomness of the output.

369 The class NTG.1 combines security capabilities of deterministic RNGs and security capabilities similar to those of physical true RNGs. By clause (NTG.1.3) the entropy pool with its updating mechanism and output function (viewed as a DRNG) is DRG.3-conformant.

370 The security capability (NTG.1.1) checks the external input signals from the entropy sources with regard to total failure and non-tolerable weaknesses. Usually, an 'entropy counter' (applying heuristic rules) is kept to provide plausibility that enough fresh entropy in mixed up with the current internal state. The entropy counter reduces the (estimated) entropy of the internal state by *m* whenever *m* bits are output. If the value of the entropy counter is smaller

[38] [selection: *physical, non-physical true, deterministic, hybrid physical, hybrid deterministic*]

[39] [assignment: *list of security capabilities*]

[40] [assignment: *a defined quality metric*]

than m the output of an m-bit string is prohibited. One says the input is “rated for entropy estimation”.

371 Online tests for NPTRNGs, however, will usually be very different from online tests for classes PTG.2 or PTG.3 because locally the input data for NPTRNGs may provide only low entropy; they might be biased or strongly dependent. It is usually impossible to formulate a precise stochastic model for input data of NPTRNGs.

372 The security capability (NTG.1.2) is the same as (DRG.2.4). The security capability (NTG.1.3) of enhanced backward secrecy is the same as (DRG.3.4). The class NTG.1 includes the requirements for the quality of the random numbers (DRG.1.3), (DRG.2.5), and (PTG.2.7).