ICS 35.030
CCS L 80

中华人民共和国密码行业标准

GM/T 0105—2021

软件随机数发生器设计指南

Design guide for software-based random number generators

2021-10-18 发布　　2022-05-01 实施

国家密码管理局　　发布

目　　次

前　言

本文件按照GB/T 1.1—2020《标准化工作导则　第1部分：标准化文件的结构和起草规则》的规定起草。

请注意本文件的某些内容可能涉及专利。本文件的发布机构不承担识别专利的责任。

本文件由密码行业标准化技术委员会提出并归口。

本文件起草单位：中国科学院数据与通信保护研究教育中心、中国科学院软件研究所、浙江大学、深圳技术大学、深圳市纽创信安科技开发有限公司、成都卫士通信息产业股份有限公司、中国科学技术大学网络空间安全学院、成都信息工程大学、中国金融认证中心、北京宏思电子技术有限公司、北京智芯微电子科技有限公司、智巡密码（上海）检测技术有限公司。

本文件主要起草人：马原、吕娜、陈华、沈海斌、郑昉昱、陈天宇、张翌维、樊俊锋、林璟锵、刘攀、吴鑫莹、张立廷、吴震、王飞宇、张文婧、胡晓波、范丽敏、韩玮。

引　言

随机数的质量直接影响到密钥生成、数字签名以及其他密码算法和协议的实际安全性。随着软件密码模块使用越来越广泛，其中的随机数发生器设计备受关注。

本文件为软件随机数发生器的设计提供了通用的基本模型，描述了其基本部件的设计指导和建议，以指导软件随机数发生器的设计者、开发者和测试者。

软件随机数发生器设计指南

1 范围

本文件给出了软件随机数发生器设计的基本模型、基本部件的设计指南以及安全分级方法，并在附录中给出了基于SM3算法和基于SM4算法的设计实例。

本文件适用于软件随机数发生器的设计、开发、检测和评估。

2 规范性引用文件

下列文件中的内容通过文中的规范性引用而构成本文件必不可少的条款。其中，注日期的引用文件，仅该日期对应的版本适用于本文件；不注日期的引用文件，其最新版本(包括所有的修改单)适用于本文件。

GB/T 15852.1 信息技术 安全技术 消息鉴别码 第1部分：采用分组密码的机制

GB/T 17964 信息安全技术 分组密码算法的工作模式

GB/T 32905 信息安全技术 SM3密码杂凑算法

GB/T 32907 信息安全技术 SM4分组密码算法

GB/T 32915—2016 信息安全技术 二元序列随机性检测方法

GB/T 37092—2018 信息安全技术 密码模块安全要求

GM/Z 4001 密码术语

3 术语和定义

GB/T 32915—2016、GB/T 37092—2018和GM/Z 4001界定的以及下列术语和定义适用于本文件。

3.1

熵 entropy

对一个封闭系统的无序性、随机性或变化性等状态的度量。

注：随机变量 X 的熵是对通过观测 X 所获得信息量的一个数学度量。

3.2

熵源 entropy source

产生输出的部件、设备或事件。当该输出以某种方法捕获和处理时，产生包含熵的比特串。

3.3

已知答案测试 known-answer test

一种测试确定性机制的方法，即通过该机制处理给定的输入，然后将所得到的输出与已知值进行比较。

3.4

熵池 entropy pool

临时保存熵的存储区域。

3.5

最小熵　min-entropy

熵的下界，是对确定样本熵最坏情况的估值。

注：如果 k 为最大值，使得 $P(X=x)\leqslant 2^{-k}$，那么比特串 X（或更准确地说，是形成此类随机比特串的相应的随机变量）的最小熵为 k。也就是说，X 至少包含 k 比特的熵或随机性。

3.6

随机数发生器　random number generator；RNG

产生随机二元序列的器件或程序。

[来源：GB/T 32915—2016，2.2]

3.7

种子　seed

用作随机数发生器的输入的比特串。

3.8

重播种函数　reseed function

一种特定的内部状态转移函数，该函数在提供新种子值的情况下更新内部状态。

3.9

关键安全参数　critical security parameter

与安全相关的秘密信息，这些信息被泄露或被修改后会危及密码模块的安全性。

注：关键安全参数可以是明文形式的也可以是经过加密的。

[来源：GB/T 37092—2018，3.3]

3.10

确定性随机数发生器　deterministic random number generator；DRNG

一种随机数生成器，通过将确定性算法应用于适当的随机初始值（称为“种子”）产生随机样式的比特序列。

3.11

熵率　entropy rate

平均每比特数据包含的熵的大小。

3.12

密码边界　cryptographic boundary

明确定义的边线，该边线建立了密码模块的物理和/或逻辑边界，并包括了密码模块的所有硬件、软件和/或固件部件。

[来源：GB/T 37092—2018，3.4]

3.13

软件随机数发生器　software-based RNG

软件密码模块（或混合密码模块的软件部件）中的随机数发生器部件，可以单独作为软件密码模块，也可以作为软件密码模块（或混合密码模块的软件部件）的一部分。

3.14

密码模块　cryptographic module

实现了安全功能的硬件、软件和/或固件的集合，并且被包含在密码边界内。

注：密码模块根据其组成，可分为硬件密码模块、固件密码模块、软件密码模块以及混合密码模块。

[来源：GB/T 37092—2018，3.5]

3.15

公开安全参数　public security parameter

与安全性相关的公开信息，一旦被修改，会威胁到密码模块安全。

注：例如，公钥、公钥证书、自签名证书、信任锚、与计数器和内部保持的日期和时间相关联的一次性口令。公开安全参数如果不能被修改或者修改后能够被密码模块发现，此时可以认为该公开安全参数是受保护的。

[来源：GB/T 37092—2018，3.14]

3.16

敏感安全参数　sensitive security parameter

包括关键安全参数和公开安全参数。

[来源：GB/T 37092—2018，3.17]

3.17

重播种计数器值　reseed counter

一种内部状态计数器变量，表明自初始化或重播种期间获得新的熵输入以来，请求随机数生成的次数。

3.18

上次重播种时间值　time value of last reseed

软件随机数发生器上一次重播种的时间值，单位为秒。

3.19

重播种计数器阈值　reseed counter threshold

在软件随机数发生器重播种前，能够产生随机数的最大请求次数。

3.20

重播种时间阈值　reseed time threshold

距离软件随机数发生器上一次重播种的最大时间间隔，单位为秒。

4　缩略语

下列缩略语适用于本文件。

CPU：中央处理器(Central Processing Unit)

RNG：随机数发生器(Random Number Generator)

SRNG：软件随机数发生器(Software-based RNG)

DRNG：确定性随机数发生器(Deterministic Random Number Generator)

IV：初始向量(Initial Vector)

5　软件随机数发生器设计

5.1　基本模型

本文件给出的软件随机数发生器基本模型见图1，主要包含系统熵源、熵估计、熵池、DRNG和健康测试等部分。其中，DRNG主要由内部状态、初始化函数、重播种函数、输出函数、自测试等基本部件组成。

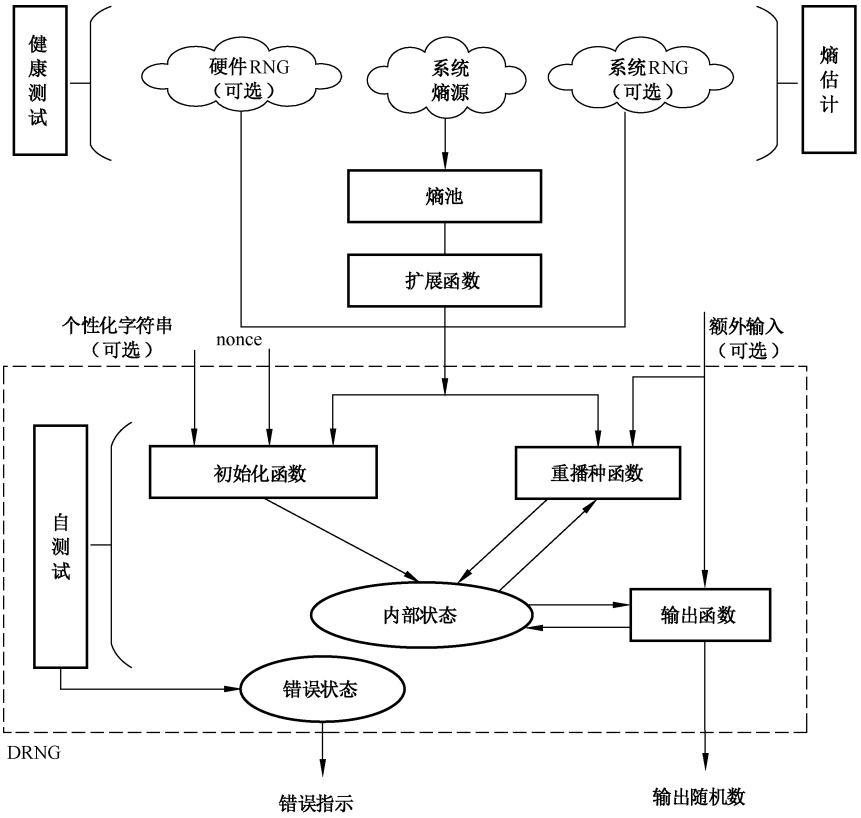

图 1　软件随机数发生器的基本模型

软件随机数发生器收集系统熵源的随机性作为其随机性的来源(随机性的来源也可以来自系统或硬件随机数发生器)。系统熵源的数据进入熵池进行熵累积,等待收集足够的熵源数据,在对熵池中的数据经扩展函数压缩后,与系统或硬件随机数发生器的输出数据(可选)经数据拼接后一同作为 DRNG 的输入产生种子。DRNG 经过初始化,开始内部状态的迭代以及必要的重播种操作,以产生所需数量的随机数。除熵输入外,DRNG 输入还包括 nonce(必选)、个性化字符串(可选)和额外输入(可选)。

为保证软件随机数发生器的安全性,需要对熵源的熵进行熵估计,以及对熵源的状态进行必要的健康测试。对于一个关键安全参数,其收集的最小熵值应符合 GB/T 37092—2018 的要求,本文件以 256 比特的最小熵值为基准来给出软件随机数发生器的参数信息。

5.2　熵源

熵源是软件随机数发生器随机性的来源。由于软件密码模块运行在通用操作系统上,因此其随机性的来源也一般依赖操作系统提供,包括系统时间、特定的系统中断事件、磁盘状态、人机交互输入事件(如按键、鼠标移动等动作)等。为了保障软件随机数发生器的可靠性,建议随机性来源不少于 3 种。A.2 以操作系统中的中断事件为例,给出如何从中断事件中提取随机性。

此外,在特定的软硬件平台上,熵源也可以是平台已有的随机数发生器,包括硬件随机数发生器和系统随机数发生器(见图 1),如基于 CPU 抖动的随机数发生器、CPU 内置的硬件随机数发生器、外部硬件随机数发生器部件等。软件随机数发生器也可以用这些特定软硬件平台上的熵源作为熵输入来增大随机性,但仍然需要保证在没有这些熵源时,输入的熵仍然是充足的。不同的是,为了考虑效率和安全性,图 1 中的系统熵源的输出数据需要在熵池中累积并经过扩展函数输出后再作为 DRNG 的熵输入。

5.3 熵池

熵池作为软件随机数发生器中保证熵采集的重要部件，在空闲时，RNG 可以把熵源输出收集起来；当需要时，可以及时从熵池中读取所需的数据，从而减少产生随机数的延迟。此外，为增加熵源数据混淆、节省熵池空间，熵池内可采用迭代压缩函数来增加熵率，如通过循环移位寄存器实现。如果熵池太小会导致累积的熵不足，如果熵池太大则浪费内存空间。出于以上考虑，熵池大小应大于或等于 512 字节，但不宜超过 4 096 字节。A.3 给出了一种基于循环移位寄存器的熵池结构示例。

在收到 DRNG 初始化或重播种请求时，在熵池中的熵值大于 256 比特后（根据 5.4 的熵估计结果），所有熵池中的数据进行压缩后与图 1 中的系统 RNG（可选）或硬件 RNG（可选）的输出数据经数据拼接后一同作为 DRNG 的输入。同时，扩展函数压缩后的结果也反馈回熵池中，以保证后向安全性。为保证熵池中的熵能够最大限度的保持，可使用密钥扩展函数作为压缩算法，仅使用密码杂凑算法无法保证压缩结果是满熵（full-entropy）的。一种基于 SM3 密码杂凑算法的密钥扩展函数（SM3_df）按附录 B，输出长度可设置为 440 比特。有关 SM3 密码杂凑算法的计算方法按 GB/T 32905。

5.4 熵估计

熵估计是保证随机数发生器安全性的关键。系统熵源一般属于非物理熵源，因此难以建立一个可靠的数学模型对熵源的随机行为进行刻画，也难以进一步通过理论建模的方法对熵源的熵进行估计。本文件给出了一种离线统计熵估计的方法，对系统熵源的最小熵进行估计（见附录 C），其中包含了一种较为通用的基于马尔可夫预测器的熵估计器，更多的熵估计器可以参考文献[2][5]。此外，对使用外部随机数发生器作为熵输入的情况（如系统 RNG 或硬件 RNG），统计熵估计也适用于对这些随机数发生器输出的熵估计。

5.5 健康测试

健康测试用于监测熵源的状态，以保证随机数发生器运行时的安全。健康测试包括上电健康测试、连续健康测试和按需健康测试三种。

a） 上电健康测试：

- 在软件随机数发生器或其所在的密码模块上电或重启，以及首次使用熵源之前执行；
- 上电健康测试可以确保熵源在正常运行条件下使用前可以按预期工作，并且自上次上电健康测试以来没有发生任何故障；
- 上电健康测试需要对至少 1 024 个连续样本执行连续健康测试；
- 在上电健康测试期间，熵源的输出不能用于其他操作；
- 在测试完成后且没有发现任何故障或错误，那么可以考虑使用测试期间的熵源的输出。

b） 连续健康测试：

- 连续健康测试关注熵源的行为，在熵源工作时，持续地检测从熵源得到的所有数字化的样本，目的是当熵源输出时能够及时发现熵源潜在的多种故障，执行该测试时无需禁止熵源的输出；
- 在熵源正常运行时，该测试的误警率应非常低，在许多系统中，一个合理的误警率策略可以保证即便在很长的使用时间内也几乎不会发生故障报警；
- 连续健康测试会受到资源限制，这个限制将影响检测到熵源故障的能力，因此，连续健康测试通常被设计成检测严重故障。

c） 按需健康测试：

- 可以在任何时候被执行；
- 在按需健康测试的过程中，从熵源收集的数据在测试完成前不应使用，可以随时丢弃，也

可以在测试完成后且未发生任何错误的情况下使用。

上述三种健康测试方法中，上电健康测试和连续健康测试是应做的。附录 D 给出了连续健康测试方法，包含 2 个测试算法：重复计数测试和自适应比例测试。除了附录 D 中的两种算法外，设计者也可以遵循如下两个要求，根据具体设计提出其他的连续健康测试方法。

a) 如果一个样本在熵源采样序列中连续出现超过$\left\lceil \frac{100}{H} \right\rceil$次（$H$ 为通过附录 C 估计的样本最小熵），那么连续健康测试方法应至少以 99%的概率检测到该值；

b) 令 $P=2^{-H}$，如果熵源的行为发生了变化，使得观察到特定样本值的概率至少增加到 $P^{*}=2^{-H}/2$，那么当检查来自该熵源的 5 万个连续样本时，连续健康测试方法应至少以 50%的概率检测到这种变化。

5.6 DRNG

5.6.1 概述

本文件给出的 DRNG 架构见图 1，对外接口包括初始化函数、重播种函数、输出函数。DRNG 还在内部维持了内部状态，同时利用自测试模块对自身进行测试，确保 DRNG 在运行过程中能够及时检测出故障。本文件给出了两种 DRNG 的设计实例，包括基于 SM3 算法（见 GB/T 32905）的 DRNG 和基于 SM4 算法（见 GB/T 32907）的 DRNG，分别见附录 B 和附录 E。根据使用的密码算法不同，其内部状态组成、函数计算过程以及参数要求等不同，下面给出了 DRNG 中的内部状态以及相关函数的总体设计说明，关于具体参数要求、赋值要求等按附录 B 和附录 E。

5.6.2 内部状态

DRNG 的内部状态可视为软件随机数发生器的内部（临时）存储，由 DRNG 所使用或执行的所有参数、变量以及其他存储的值组成。DRNG 的内部状态包括：

a) 敏感状态信息：DRNG 的熵输入数据直接决定了 DRNG 的敏感状态信息的初始值，DRNG 根据敏感状态信息借助伪随机数生成算法直接产生随机数，并在每次生成随机数时按照一定规则不断迭代更新敏感状态信息，由于敏感内部状态直接决定了当前以及下一次重播种之前的所有随机数，它应看作 GB/T 37092—2018 中的关键安全参数受到保护，不能被非授权地访问、使用、泄露、修改和替换；

b) 管理状态信息：即与 DRNG 运行相关的管理状态信息，包括重播种计数器值、上次重播种时间值，管理状态信息应看作 GB/T 37092—2018 中的公开安全参数受到保护，不能被非授权地修改和替换；

c) 常量值：重播种计数器阈值和重播种时间阈值，该常量值与安全等级有关，不同安全等级的 SRNG，重播种计数器阈值和重播种时间阈值有所不同（见 6.2 和 6.3）。

5.6.3 初始化函数

初始化函数利用熵源数据、nonce 和个性化字符串对初始状态进行赋值。nonce 可以提供除熵源数据之外的额外安全保障，在软件 RNG 的熵源存在故障时提供额外的安全防护，降低 DRNG 在多次实例化过程中可能存在的风险。例如，一个存在故障的系统每次开机后在熵池中总是累积相同的熵源数据，此时如果不使用 nonce 值，那么每次初始化后将会产生相同的随机数，nonce 的使用可以在一定程度上缓解这一问题。个性化字符串的使用，可以降低不同平台上同一个软件 RNG 机制产生随机数过程中的安全风险。例如，存在故障的多个系统可能在熵池中累积了相同的熵源数据，如果没有使用个性化字符串，它们将会产生相同的随机数。

DRNG 自身的安全性应不依赖于 nonce 和个性化字符串的保密性。对 nonce 和个性化字符串的设

计建议如下。

a) nonce(必选输入):

- nonce 应至少具有 128 比特熵,或者预期重复概率不大于 2^{-128};
- nonce 可以是随机值、时间戳、单调递增的计数器值或者它们的组合。

b) 个性化字符串(可选输入):

- 如果使用个性化字符串,则个性化字符串与熵源数据以及 nonce 一起生成 DRNG 的初始种子;
- 对于相同 DRNG 机制在不同平台的各个实例,个性化字符串需要具有唯一性;
- 个性化字符串可包含的内容包括但不限于:设备序列号、公钥信息、用户标识、时间戳、网络地址、应用程序标识符、协议版本标识符。

5.6.4 重播种函数

重播种函数利用熵源数据、额外输入(可选)对内部状态进行更新。

额外输入可以是秘密的,也可以是公开的,但 DRNG 自身的安全性应不依赖于额外输入的保密性。额外输入可以为 DRNG 提供更多熵,如果额外输入是非公开的并且有充足的熵,可以为 DRNG 提供更多的安全保证。

除了可以在调用重播种函数时注入额外输入外,也可以在每次调用输出函数时注入额外输入。

5.6.5 输出函数

输出函数利用 DRNG 的内部状态和额外输入(可选)通过特定的密码计算产生随机数据,同时更新 DRNG 的内部状态。可选的额外输入的要求与重播种函数中的额外输入要求一致。

本文件建议输出函数每次调用只能输出一组随机数据,其中基于 SM3 算法的 DRNG 输出一组随机数据的长度为 256 比特,基于 SM4 算法的 DRNG 输出一组随机数据的长度为 128 比特。

5.6.6 DRNG 自测试

类似于 GB/T 37092—2018 中密码模块的自测试,DRNG 也需要执行自测试以确保设计的 DRNG 能够持续按照所设计和实现的方式正确运行。

本文件给出的 DRNG 采用已知答案测试,即 DRNG 实例利用已知的输入调用初始化函数、重播种函数和输出函数,并确认输出与预期答案是否一致,如果一致,则通过自测试;否则 DRNG 需要进入错误状态,并返回错误指示信号。

对 DRNG 的自测试可以作为其所在软件密码模块自测试的一部分,也可以单独执行。但无论以何种形式进行,对 DRNG 的自测试需要满足与所在软件密码模块的安全等级相对应的自测试要求(按 GB/T 37092—2018 中 7.10)。在执行自测试时,应禁止从 DRNG 边界内输出数据;在 DRNG 正常运行期间,已知答案测试的结果不能作为随机数据输出。

6 安全分级方法

6.1 概述

由于不同级别的密码模块的安全防护要求不相同,本文件针对 DRNG 重播种机制,从种子的使用时长和使用次数两方面考虑,对满足 GB/T 37092—2018 安全等级一级和安全等级二级密码模块中的软件随机数发生器给出分级方法。

6.2 GB/T 37092 安全等级一级

在 GB/T 37092—2018 安全等级一级的密码模块中，对于 DRNG 每一组新输入的熵（通过初始化或种子更新获得），当以下任意条件满足时，需要对 DRNG 执行重播种操作：

a) 距离上一次 DRNG 重播种时间超过 600 s（即，重播种时间阈值为 600 s）；

b) 输出函数已被调用 2^{20} 次（即，重播种计数器阈值为 2^{20}）。

6.3 GB/T 37092 安全等级二级

在 GB/T 37092—2018 安全等级二级的密码模块中，对于 DRNG 每一组新输入的熵（通过初始化或种子更新获得），当以下任意条件满足时，需要对 DRNG 执行重播种操作：

a) 距离上一次 DRNG 重播种时间超过了 60 s（即，重播种时间阈值为 60 s）；

b) 输出函数已被调用 2^{10} 次（即，重播种计数器阈值为 2^{10}）。

7 实现

7.1 通用

软件随机数发生器既可以是单独的软件密码模块，也可以作为软件密码模块（或混合密码模块的软件部件）的一部分，在实现时应符合 GB/T 37092—2018 的所有适用指标。

7.2 关键安全参数定义

软件随机数发生器相关部件的关键安全参数见表 1，包括熵源、熵池和 DRNG 所涉及的关键安全参数。

表 1 软件随机数发生器的关键安全参数

软件随机数发生器部件	关键安全参数
熵源	熵源产生的熵
熵池	循环移位寄存器的状态
DRNG	内部状态中的敏感状态信息
	熵输入
	输出的随机数

7.3 熵源独占性

熵源同一时刻只能被一个软件随机数发生器独占访问，以保证软件随机数发生器独占过程中所读取的熵源数据无法被其他实体（例如，同一运行环境下的其他进程）获取，可以通过如下机制来保证熵源独占性：

——软件随机数发生器本身的机制，例如，熵源本身是软件随机数发生器的一部分，软件随机数发生器自身的机制保证熵源访问的独占性；

——熵源本身的机制，例如，熵源同一时刻只能为一个实例提供熵，当熵源被占用时，将拒绝其他实体的访问；

——运行环境的安全机制，例如，操作系统通过互斥锁的机制，保证只有一个实体可以访问该熵源。

附 录 A
（资料性）
熵源和熵池结构示例

A.1 概述

本附录给出了一种基于操作系统中断事件的熵源示例，以及一种基于循环移位寄存器实现熵池更新的方案，供软件随机数发生器的设计者作为参考。

A.2 熵源示例

这里选择中断事件发生的时间作为熵源，可以采用如下两种模式对中断事件发生时间的随机性进行数字化。

——获取中断事件发生时的系统时间作为熵源数据：任何一个中断事件的发生都会促使操作系统生成两个字符串，其中第一个字符串表示事件发生的时间（自从系统启动以来所经过的毫秒数），第二个字符串表示所发生事件的类型，可以将第一个字符串或同时将这两个字符串输入熵池作为随机数据，在计算中断事件熵值时，只考虑中断事件的发生时间，不关注事件的类型。

——通过 CPU 获取中断事件发生时的高精度时间值作为熵源数据：现代 CPU 一般会有高精度计时部件，该部件以 64 位无符号整型数的格式，记录了自 CPU 上电以来所经过的时钟周期数。由于目前的 CPU 主频都非常高，因此这个部件可以达到纳秒级的计时精度，根据熵提取的方式不同，有时可能仅处理高精度时间值的最低有效位而丢弃最高有效位，因为最低有效位是快速变化的位。

A.3 熵池示例

一种基于循环移位寄存器实现熵池更新的方案如下：

随机数发生器的熵池更新基于一个本原多项式，本原多项式的阶数为熵池的大小（以字为单位）。假设熵池大小为 128 个字（512 字节），其本原多项式为 128 阶，即 $x^{128}+x^{103}+x^{76}+x^{51}+x^{25}+x+1$。熵池更新的过程可以看作使用循环移位寄存器来加密输入的熵源数据，熵池结构见图 A.1，其中，Pool 代表熵池，j 为熵池中当前位置的索引（本示例中 j 为 0～127）。假设 g 为被添加到熵池中的新的熵数据（大小为 1 个字），则对熵池的更新操作如下：

temp＝g ⊕ Pool[j]
temp＝temp ⊕ Pool[(j＋1) mod 128]
temp＝temp ⊕ Pool[(j＋25) mod 128]
temp＝temp ⊕ Pool[(j＋51) mod 128]
temp＝temp ⊕ Pool[(j＋76) mod 128]
temp＝temp ⊕ Pool[(j＋103) mod 128]
temp＝(temp＞＞3)⊕ table[temp & 7]
Pool[j]＝temp

其中，table 为自定义的 8 个字的数组，示例如下：

table[0]＝0x0
table[1]＝0x3b6e20c8
table[2]＝0x76dc4190
table[3]＝0x4db26158

table[4]=0xedb88320
table[5]=0xd6d6a3e8
table[6]=0x9b64c2b0
table[7]=0xa00ae278

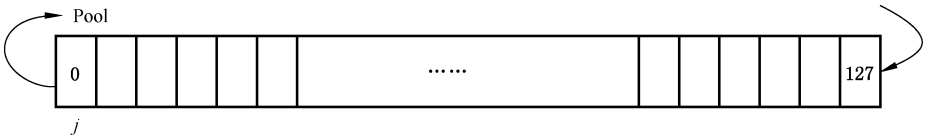

图 A.1　一种基于循环移位寄存器的熵池结构示例

附 录 B
（规范性）
基于 SM3 算法的 RNG 设计

B.1 参数函数说明

本附录给出了基于 SM3 算法的 DRNG 设计实例的伪代码，该发生器的实例为 SM3_RNG，包括内部状态、初始化函数、重播种函数和输出函数等。SM3_RNG 使用的符号及参数含义如下：

||：数据拼接

V：比特串 V，为随机数发生器的内部状态变量，在每次调用 DRNG 时更新值

C：长度为 seedlen 比特的常量 C，为随机数发生器的内部状态变量，在初始化和重播种时更新值

seedlen：种子的比特长度

reseed_counter：重播种计数器值，是一种内部状态计数器变量，表明自初始化或重播种期间获得新的熵输入以来，请求随机数生成的次数

reseed_interval_in_counter：重播种计数器阈值，在软件随机数发生器重播种前，能够产生随机数的最大请求次数

last_reseed_time：重播种时间值，指 DRNG 上一次重播种时间值，单位为秒

reseed_interval_in_time：重播种时间阈值，距离上一次 DRNG 重播种的最大时间间隔，单位为秒

entropy_input：由熵输入源获得的比特串，用于确定种子材料和生成种子

personalization_string：个性化字符串

additional_input：可选的额外输入的比特串

additional_input_length：可选的额外输入的比特串长度

min_entropy：最小熵

min_entropy_input_length：最小的熵输入长度

max_entropy_input_length：最大的熵输入长度

nonce：随机比特串

outlen：输出函数输出的比特长度

seed：种子

seed_material：种子材料

working_state：当前内部状态

new_working_state：新的内部状态

status：函数的返回状态

temp：一个临时变量

input_string：SM3 派生函数(SM3_df)的输入字符串

number_of_bits_to_return：SM3 派生函数(SM3_df)返回的比特长度

requested_bits：SM3 派生函数(SM3_df)返回的比特串

requested_number_of_bits：输出函数(SM3_RNG_Generate)返回的随机比特串的长度

returned_bits：每调用 1 次输出函数(SM3_RNG_Generate)，返回的随机比特串

表 B.1、表 B.2 分别列出了 SM3_RNG 使用的参数要求和函数说明。

表 B.1　SM3_RNG 使用的参数要求

参数名称	参数类型	取值要求
V	比特串	长度为 seedlen
C	比特串	长度为 seedlen
outlen	整型	256 比特
seedlen	整型	440 比特
reseed_interval_in_time	整型	安全等级一级：600 s 安全等级二级：60 s
reseed_interval_in_counter	整型	安全等级一级：2^{20} 安全等级二级：2^{10}
last_reseed_time	整型	取值为以秒为单位的时间值
reseed_counter	整型	初始值为 1，每调用一次输出函数自动加 1，最大值不超过 reseed_interval_in_counter
entropy_input	比特串	entropy_input 应至少具有 256 比特熵，其长度范围为[256，2^{35})，单位为比特
min_entropy	整型	256 比特
min_entropy_input_length	整型	256 比特
max_entropy_input_length	整型	2^{35} 比特
nonce	—	nonce 应至少具有 128 比特熵或者预期重复概率不大于 2^{-128}，nonce 可以是随机值、时间戳、单调递增的计数器值或者它们的组合，长度范围为[128，2^{34})
additional_input	比特串	长度不大于 2^{35} 比特
additional_input_length	整型	不大于 2^{35} 比特
personalization_string	比特串	长度不大于 2^{35} 比特
requested_number_of_bits	整型	不大于 256 比特
number_of_bits_to_return	整型	长度为 seedlen

表 B.2　SM3_RNG 使用的函数说明

函数名称	描述
leftmost (V，a)	比特串 V 从左边起的 a 个比特
Get_entropy(min_entropy， min_entropy_input_length， max_ entropy_input_length)	从熵源获取一串比特的函数，要求返回的比特串(entropy_input)，至少含有 min_entropy 比特熵，长度大于或等于 min_ entropy_input_length 比特，小于或等于 max_ entropy_input_length 比特
SM3(a)	使用 SM3 对数据进行杂凑运算
SM3_df(input_string，number_of_bits_to_return)	SM3 派生函数，对输入字符串进行杂凑运算，返回长度为 number_of_bits_to_return 的比特串

表 B.2　SM3_RNG 使用的函数说明（续）

函数名称	描述
SM3_RNG_Instantiate(personalization_string,nonce)	初始化函数:利用个性化字符串、nonce,对内部状态进行设置
SM3_RNG_Reseed(working_state,entropy_input,additional_input)	重播种函数:在提供新的熵输入的情况下更新内部状态
SM3_RNG_Generate(working_state,requested_number_of_bits,additional_input)	输出函数:利用当前的内部状态和可能的额外输入返回随机数,并对内部状态进行更新
Get_current_time_in_second()	获取当前的时间值,单位为秒

B.2　内部状态

内部状态组成为{V,C,reseed_counter,last_reseed_time,reseed_interval_in_counter,reseed_interval_in_time},定义如下:

——比特串 V,长度为 seedlen,每次随机数生成时会进行更新;

——比特串 C,长度为 seedlen,每次重播种时会进行更新;

——reseed_counter,在初始化或重播种之后,请求随机数的次数,每次执行随机数生成操作时会自动加 1;

——last_reseed_time,上次 DRNG 重播种的时间值,单位为秒;

——reseed_interval_in_counter,重播种计数器阈值,该值为常量值;

——reseed_interval_in_time,重播种时间阈值,单位为秒,该值为常量值。

B.3　初始化函数

函数定义:SM3_RNG_Instantiate(personalization_string,nonce)

输入:

——personalization_string(可选);

——nonce。

输出:

——初始内部状态,即 V,C,reseed_counter,last_reseed_time 的初始值。

SM3_RNG_Instantiate 流程:

a)　min_entropy = min_entropy_input_length;

b)　(status,entropy_input)=Get_entropy(min_entropy,min_entropy_input_length,max_entropy_input_length);

c)　If(status≠"Success"),返回错误信息;

d)　seed_material=entropy_input||nonce||personalization_string;

e)　生成种子:seed=SM3_df(seed_material,seedlen);

f)　初始化内部状态变量:

1)V=seed;

2)C=SM3_df(0x00||V,seedlen);

3)reseed_counter=1;

4)last_reseed_time=Get_current_time_in_second();

g) 返回 V,C,reseed_counter,last_reseed_time,作为初始内部状态变量。

函数定义:Get_entropy(min_entropy,min_entropy_input_length,max_entropy_input_length),下同。

输入:

——min_entropy,min_entropy_input_length,max_entropy_input_length,均为整型。

输出:

——status,entropy_input(含有熵的比特串),均为 string 类型。

Get_entropy 流程:

a) 从合适的熵源获取熵输入 entropy_input,entropy_input 的熵值≥min_entropy,min_entropy_input_length≤entropy_input 长度≤max_entropy_input_length;

b) 返回(status,entropy_input),status 为 Success 或错误信息。

B.4 SM3 派生函数

SM3 派生函数 SM3_df 在软件随机数发生器的初始化和重播种阶段使用,对输入字符串进行杂凑运算,并返回请求的随机比特串。定义如下:

函数定义:SM3_df(input_string,number_of_bits_to_return)

输入:

——input_string:输入字符串;

——number_of_bits_to_return:SM3_df 函数返回的比特长度。

输出:

requested_bits:SM3_df 函数返回的结果。

SM3_df 流程:

a) temp=NULL;

b) $len=\left\lceil \frac{number\_of\_bits\_to\_return}{outlen} \right\rceil$;

c) counter=0x01(计数器变量,初始值为 1);

d) For i=1 to len do

1) temp=temp||SM3(counter||number_of_bits_to_return||input_string);

注:number_of_bits_to_return 用 32 比特位整数表示,采用十六进制的 big-endian 表示。

2) counter=counter+1;

e) requested_bits=leftmost(temp,number_of_bits_to_return);

f) 返回(Success,requested_bits)。

B.5 重播种函数

利用熵输入及额外输入(可选)更新种子,同时对内部状态进行更新。SM3_RNG 的重播种操作函数如下:

函数定义:SM3_RNG_Reseed(working_state,entropy_input,additional_input)

输入:

——当前内部状态 working_state:V,C,reseed_counter 和 last_reseed_time 的当前值;

——熵输入 entropy_input:由熵输入源获得的比特串;

——额外输入 additional_input:来自应用的比特串,其长度可能为 0。

输出:

——新的内部状态 new_working_state。

SM3_RNG_Reseed 流程：

a) seed_material＝0x01||entropy_input||V||additional_input；

b) 更新种子：seed＝SM3_df(seed_material，seedlen)；

c) 更新内部状态变量：

1) V＝seed；

2) C＝SM3_df(0x00||V，seedlen)；

3) reseed_counter＝1；

4) last_reseed_time＝Get_current_time_in_second()；

d) 返回 V，C，reseed_counter，last_reseed_time 作为新的内部状态，即 new_working_state。

B.6 输出函数

函数定义：SM3_RNG_Generate(working_state，requested_number_of_bits，additional_input)

输入：

——当前内部状态 working_state：V，C，reseed_counter 和 last_reseed_time 的当前值；

——requested_number_of_bits：返回的随机数的长度，最大为 256 比特；

——额外输入 additional_input：来自应用的比特串，其长度可能为 0。

输出：

——状态 status：Success 或者需要执行种子更新的标识；

——返回的随机数 returned_bits：每调用 1 次输出函数，返回的随机比特；

——新的内部状态 new_working_state：V，C，reseed_counter，last_reseed_time 的新值。

SM3_RNG_Generate 流程：

a) 如果 reseed_counter＞reseed_interval_in_counter，
或者 Get_current_time_in_second()-last_reseed_time＞reseed_interval_in_time，则执行 SM3_RNG_Reseed，否则继续执行下述步骤；

b) 如果 additional_input 非空，则：

1) W＝SM3(0x02||V||additional_input)；

2) V＝(V＋W)mod $2^{seedlen}$；

c) temp＝NULL；

d) temp＝SM3(V)；

e) returned_bits＝leftmost(temp，requested_number_of_bits)；

f) H＝SM3(0x03||V)；

g) V＝(V＋H＋C＋reseed_counter)mod $2^{seedlen}$；

h) reseed_counter＝reseed_counter＋1；

i) 返回(Success，returned_bits，V，C，reseed_counter，last_reseed_time)。

注：如果实现时不需要额外输入，那么省略额外输入参数并且 SM3_RNG_Generate 函数中的第 b)步可以省略。

附 录 C
（资料性）
熵估计方法

C.1 熵估计方法设计指导

假设熵源输出元素 s 的取值空间为 $A=\{x_1,\cdots,x_k\}$，且取值概率分别是 $P(s=x_i)=p_i$，其中 $i=1,2,\cdots,k$。

最常用的熵度量方式是香农熵，其在信息论等领域应用广泛，香农熵的计算方式：$H_s=-\sum_{i=1}^{k}p_i*\log_2 p_i$。但是在密码学领域，最小熵更为常用，最小熵的计算方式：$H_{\min}=-\log_2[\max_{1\leqslant i\leqslant k}(p_i)]$。

香农熵指熵源的平均熵值，而最小熵给出熵源的熵值的下界，即熵源具有的随机性的下界。在密码学应用中，在熵源随机性最差的情况下仍需保证安全，因此本文件选择最小熵作为熵估计计算方法。此外，最小熵和攻击者猜测熵源输出的最大正确概率密切相关。假设某熵源输出的序列具有最小熵 H，则攻击者猜对该序列的最大概率是 2^{-H}，如果该攻击者拥有 2^w 次猜测机会，那么他猜测正确的最大概率是 2^{w-H}。

设计者应根据设计的随机数发生器提供对熵源最大可能输出概率 $\max p_i$ 的估计 $\hat{p}$，进而得到最小熵估计 $\hat{H}=-\log_2\hat{p}$。

如果对熵源的分布有较好的了解，且熵源输出服从独立同分布，则可以根据熵源输出的最大频数计算 $\hat{p}$。例如，熵源是投掷一枚硬币，该熵源有两个输出，分别是正面和反面，可以记作 0 和 1。虽然不知道投掷该硬币得到正面或反面的理论概率 p_0 和 p_1，但是可以通过多次投掷的实验结果得到 p_0 和 p_1 的估计 $\hat{p}_0$ 和 $\hat{p}_1$。假设，投掷 n 次得到 h 次正面，则 $\hat{p}_0=\frac{h}{n}$，对于任意的 $\varepsilon>0$，存在足够大的 n 使得 $P\left(\left|\frac{h}{n}-p_0\right|<\varepsilon\right)\to 1$。进而可计算出该熵源的最小熵估计 $\hat{H}=-\log_2\max(\hat{p}_0,\hat{p}_1)$。

如果熵源的概率分布复杂，设计者也可以自行设计基于预测器的熵估计方法。根据已知熵源输出序列预测熵源的下一输出，预测器预测正确的概率即可作为估计 $\hat{p}$，进而得到最小熵估计。一种基于马尔可夫预测器的熵估计方法参见 C.3，更多熵估计方法参见文献[2][5]。

C.2 熵估计流程

为了确保软件随机数发生器熵源的熵率与其设计者所声称的熵率一致，需要对熵源进行测试验证，图 C.1 给出了熵源测试的一般流程。这里假设设计者声称的熵估计值为 $H_{\text{submitter}}$。

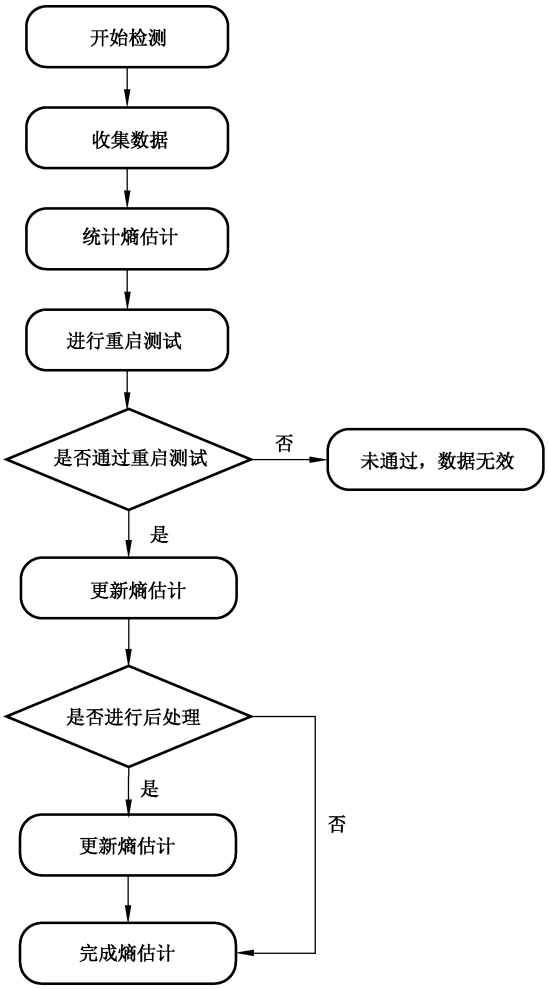

图 C.1　软件随机数发生器熵源测试流程示例

熵源测试步骤如下：

第一步：数据收集。软件随机数发生器的设计者需要提供采集的数据用于熵估计计算，数据收集方法和收集量如下：需要直接从熵源采集至少 100 万个样本；如果无法连续采集 100 万个样本，那么可以拼接几个较小的连续样本集（来自同一熵源），每个样本集至少包含 1 000 个样本，拼接后的数据至少包含 100 万个样本。

第二步：统计熵估计。选择合适的统计熵估计方法，对收集的数据进行熵估计，熵估计结果记为 H_{original}。此时，熵源的初始熵估计值为 $H_{\text{I}}=\min(H_{\text{original}},H_{\text{submitter}})$。

第三步：重启测试。重启独立性测试是检查重新启动随机数发生器后产生序列之间的关系，并将结果与初始熵估计值 H_{I}进行比较，以确保随机数发生器在每次重启之后产生的随机数都是随机的，即多次重启后产生的多组数据不应具有相关性。对于不同类型熵源的软件随机数发生器而言，重启操作可能是各不相同的，例如所在设备重新上电、重新启动程序、延迟一段时间后再采样等。因此，软件随机数发生器的设计者需要明确指出所设计的发生器的重启操作。

本附录给出如下软件随机数发生器重启独立性测试方法供参考。对软件随机数发生器进行1 000次重启操作，并将随机数发生器输出的前1 000个数据存入矩阵 $M[i][j]$。i 表示重启序数，j 表示每组数据中的元素位置，i 和 j 均为 1 到1 000的整数。重启独立性测试步骤如下：

a）　记 $p=2^{-H_{\text{I}}}$，$\alpha=0.000\ 005$；

b）　对于矩阵 M 每一行的1 000个数据，求出相同元素出现的最大频数记为 X_{R}；

c) 对于矩阵 M 每一列的1 000个数据，求出相同元素出现的最大频数记为 X_C；

d) 记 $X_{max}=\max(X_R,X_C)$；

e) 计算 $c=P(X\geqslant X_{max})=\sum_{j=X_{max}}^{1\,000}\binom{1\,000}{j}p^j(1-p)^{1\,000-j}$；

f) 如果 c 小于 α，则重启独立性测试失败，表明熵源测试失败；

g) 如果 c 大于 α，则对矩阵 M 的每一行和每一列数据分别采用上述第二步中选用的统计熵估计方法进行熵估计，熵估计结果分别记为 H_R 和 H_C。理想情况下，我们期望重启测试中所采集数据矩阵 M 的每一行和每一列数据的熵估计值与 H_I 接近。如果 $\min(H_R,H_C)<H_I/2$，则熵源测试失败；否则，熵源的最终熵估计结果为 $\min(H_R,H_C,H_I)$。

C.3 基于马尔可夫预测器的熵估计方法示例

假设待测样本数据为 $S=\{s_1,\cdots,s_L\}$，共包含 L 个样本，其中 $L=10^6$，每个样本 $s_i\in A=\{x_1,\cdots,x_k\}$。其中，$A$ 为样本空间，k 为样本空间的大小，如对于二元序列，k 为 2，$A=\{0,1\}$；对于三元序列，k 为 8，$A=\{0,1,\cdots,7\}$。对于 D 阶马尔可夫模型，基于马尔可夫预测器的熵估计计算过程如下。

a) 初始化参数 $D=16$，$N=L-2$，$winner=1$。分别创建包含 D 个值的列表 subpredict，scoreboard，entries 来记录 D 个子预测器的预测值、预测正确的次数以及已经记录的不同模板的个数，并分别初始化为 Null、0、0。其中，entries 的最大值为 maxEntries=100 000。创建数组 correct 包含 N 个值，初始化为全 0，用来记录马尔可夫预测器 N 次预测的结果。

b) 将 d 的值从 1 取到 D 分别代表 16 个不同的子预测器。设置 M_d 是一组计数器，$M_d[x,y]$ 记录模板为 x，其后一个输出为 y 的元组(x,y)的出现次数。

c) For $i=3$ to L：

1) For $d=1$ to D：

If $d<i-1$：

i) If $[(s_{i-d-1},\cdots,s_{i-2}),s_{i-1}]$在计数器 M_d 中，则 $M_d[(s_{i-d-1},\cdots,s_{i-2}),s_{i-1}]$加 1；

ii) Else if $entries_d<$maxEntries，为元组$[(s_{i-d-1},\cdots,s_{i-2}),s_{i-1}]$ 增加一个新计数器，初始化为 $M_d[(s_{i-d-1},\cdots,s_{i-2}),s_{i-1}]=1$，并且给 $entries_d$ 的值加 1。

2) For $d=1$ to D：

If $d<i$，找到元素 y 使得 $M_d[(s_{i-d},\cdots,s_{i-1}),y]$最大，将该元素记为 y_{max}。如果 y_{max}存在多种选择，选取其中最大值。然后更新 $subpredict_d=y_{max}$。如果所有模板的 $M_d[(s_{i-d},\cdots,s_{i-1}),y]$都是 0，那么 $subpredict_d=$Null。

3) 记 prediction=$subpredict_{winner}$。

4) If (prediction=s_i)，记 $correct_{i-2}=1$。

5) 更新列表 scoreboard，For $d=1$ to D：

If ($subpredict_d=s_i$)

i) 记 $scoreboard_d=scoreboard_d+1$；

ii) If $scoreboard_d\geqslant scoreboard_{winner}$，记 winner=$d$。

d) 记 C 为 *correct* 中 1 的个数。

e) 计算整体预测正确率：对于一个预测器估计器来说，首先需要对整体预测正确率进行一个直接的估计，基础整体预测正确率 $P_{global}=\frac{C}{N}$，其中 N 是总共的预测次数，C 是预测正确的个数。接着计算基础整体预测正确率的 99%置信区间的上界，作为修正后的整体预测正确率，记作 P'_{global}，计算公式如下：

$$P'_{\text{global}}=\begin{cases}1-0.01^{\frac{1}{N}}, & \text{如果 } P_{\text{global}}=0\\ \min\left(1,P_{\text{global}}+2.576\sqrt{\dfrac{P_{\text{global}}(1-P_{\text{global}})}{N-1}}\right), & \text{其他情况}\end{cases}$$

f) 计算局部预测正确率：局部预测正确率根据局部出现的最长连续正确预测子序列的长度 r 计算。局部预测正确率 P_{local} 满足以下方程式：

$$0.99=\frac{1-P_{\text{local}}x}{(r+1-rx)q}\times\frac{1}{x^{N+1}}$$

其中，$q=1-P_{\text{local}}$，N 是预测的次数，x 是方程 $1-x+qP_{\text{local}}^{r}x^{r+1}=0$ 的正实数解[4]。

g) 计算最小熵估计 $\hat{H}=-\log_2\max\left(P'_{\text{global}},P_{\text{local}},\frac{1}{k}\right)$。

附 录 D
（规范性）
连续健康测试方法

D.1 概述

本附录给出了两种连续健康测试方法：重复计数测试和自适应比例测试。这两种连续健康测试方法都被设计为需要最少的资源，在熵源输出样本时进行实时计算。如果熵源的连续健康测试中包括这两项健康测试，则不需要其他测试。但是，建议设计者、开发者在开发中包含针对熵源量身定制的其他连续健康测试。

和所有统计测试一样，这两种测试都具有假阳性概率，即正常运行的熵源的输出无法通过测试的概率。在许多应用中，对于Ⅰ类错误概率（即假阳性概率）的合理选择是 $\alpha=2^{-20}$。该值将在本附录其余部分的所有计算中使用。熵源的开发人员应根据熵源及其使用方法的详细信息，确定Ⅰ类错误的合理概率和相应的临界值。

D.2 重复计数测试

重复计数测试的目的是快速检测出灾难性故障，这些故障会导致熵源长时间输出相同的值。

给定一个熵源的估计最小熵 H，该熵源连续生成 n 个相同样本的概率[1)]最大为 $2^{-H(n-1)}$。如果样本重复 C 次或更多次，该测试将给出一个错误指示。临界值 C 由可接受的假阳性概率 α 和熵估计值 H 按照如下公式计算：

$$C=1+\left\lceil\frac{-\log_2\alpha}{H}\right\rceil$$

C 是满足不等式 $\alpha\geqslant 2^{-H(C-1)}$ 的最小整数，这可以确保从 C 个连续熵源样本获得相同值的序列的概率最大为 α。例如，对于 $\alpha=2^{-20}$，每个样本 $H=2.0$ 比特的熵源的重复计数测试临界值 C 为 $1+\lceil 20/2.0\rceil=11$。

假设 next()为熵源输出的下一个样本，给定连续的熵源采样序列和临界值，重复计数测试执行过程如下：

a) A=next();
b) B=1;
c) X=next();
d) If(X=A)
   B=B+1;
   If(B≥C)
   错误提示;
   else
   A=X;
   B=1;

1) 该概率由以下方法计算得到，令随机变量取值的概率为 p_i，对于 $i=1,\cdots,k$，其中 $p_1\geqslant p_2\geqslant\cdots\geqslant p_k$，则产生 C 个连续相同样本的概率为 $\sum {p_i}^C$，因为 $\sum {p_i}^C\leqslant p_1{p_1}^{C-1}+p_2{p_1}^{C-1}+\cdots+p_k{p_1}^{C-1}=(p_1+p_2+\cdots p_k){p_1}^{C-1}={p_1}^{C-1}=2^{-H(C-1)}$。

e) 重复步骤 c)。

该测试的临界值 C 可以适用于任何熵估计值 H，包括非常小和非常大的熵估计值。需要注意的是，重复计数测试作用有限，它只能检测到熵源的灾难性故障。例如，假设样本最小熵估计值为 8 比特的熵源的临界值 C 为 6(即重复 6 次)，以确保每生成 10^{12} 个样本的假阳性率约为 1 次。如果该熵源某种程度上未能使每个样本有 1/16 的概率与先前样本相同，从而每个样本仅提供 4 比特的最小熵，那么在重复计数测试发现这个问题之前，仍然需要抽取大约 100 万个样本。

D.3 自适应比例测试

自适应比例测试的目的是检测由于某些物理故障或环境变化影响熵源而可能发生的大量熵损失。该测试连续测量一系列熵源样本中样本值的局部出现频率，以确定样本是否出现得过于频繁。因此，给定每个样本的评估熵，该测试能够检测出何时一些值比预期更频繁地出现。相比于重复计数测试，该测试可以检测熵源的更多的细微故障，而不像重复计数测试那样检测熵源的整体故障。

该测试从熵源中抽取一个样本，然后统计该抽取样本与熵源接下来输出的 $W-1$ 个样本中值相同出现的次数。如果统计数值达到临界值 C，该测试将给出一个错误指示。窗口大小 W 根据样本空间大小进行设定。如果熵源的输出序列是二元序列(即熵源仅产生两个不同的值)，那么 W 为 1 024；如果熵源输出不是二元序列(即，熵源会产生两个以上的不同值)，那么 W 为 512。

假设 next()为熵源输出的下一个样本，给定连续的熵源采样序列、临界值 C 和窗口大小 W，自适应比例测试执行过程如下：

a) A=next()；

b) B=1；

c) For i=1 to W−1
    If(A=next())
        B=B+1；
    If(B ≥ C)
        错误提示；

d) 返回步骤 a)。

选择临界值 C，使得在 W 的窗口大小中观察到 C 个或更多个相同样本的概率最大为 α。C 满足以下关系式：

$$P_r(B \geqslant C) \leqslant \alpha$$

对于二进制的熵源，开发人员可以通过检查 $W-B \geqslant C$ 来扩展测试，这可以确保出现频率太高的二进制值在第一个测试窗口中被捕获。

对于样本空间较大的熵源(例如，样本空间大于 256)，设计者可以使用合理的方法将样本空间减小为一个较小的值。

表 D.1 给出了当 $\alpha=2^{-20}$ 时，给定窗口大小、样本的不同最小熵估计值对应的自适应比例测试的临界值 C。

表 D.1　自适应比例测试的临界值示例

二进制数据 $W=1\ 024$		非二进制数据 $W=512$	
熵	临界值 C	熵	临界值 C
0.2	941	0.5	410
0.4	840	1	311
0.6	748	2	177
0.8	664	4	62
1	589	8	13

附 录 E
（规范性）
基于 SM4 算法的 RNG 设计

E.1 参数函数说明

本附录给出了基于 SM4 算法的 DRNG 设计实例的伪代码，该发生器的实例为 SM4_RNG，包括内部状态、初始化函数、重播种函数和输出函数等。SM4_RNG 使用的符号及参数含义如下：

||：数据拼接

V：比特串 V，为随机数发生器的内部状态变量，在每次调用 DRNG 时更新值

Key：内部状态变量

keylen：分组密码算法使用的密钥的长度

reseed_counter：重播种计数器值，是一种内部状态计数器变量，表明自初始化或重播种期间获得新的熵输入以来，请求随机数生成的次数

reseed_interval_in_counter：重播种计数器阈值，在软件随机数发生器重播种前，能够产生随机数的最大请求次数

last_reseed_time：重播种时间值，指 DRNG 上一次重播种时间值，单位为秒

reseed_interval_in_time：重播种时间阈值，距离上一次 DRNG 重播种的最大时间间隔，单位为秒

entropy_input：由熵输入源获得的比特串，用于确定种子材料和生成种子

personalization_string：个性化字符串

additional_input：可选的额外输入的比特串

min_entropy_input_length：最小的熵输入长度

max_entropy_input_length：最大的熵输入长度

nonce：随机比特串

seedlen：种子的比特长度

outlen：输出函数输出的比特长度

blocklen：分组密码算法的输出分组的长度

seed_material：种子材料

min_entropy：从熵源获得并在种子中提供的最小熵值

working_state：当前内部状态

new_working_state：新的内部状态

status：函数的返回状态

input_string：SM4 派生函数(SM4_df)操作的比特串，长度为 8 的倍数

number_of_bits_to_return：由 SM4 派生函数(SM4_df)返回的比特数

requested_bits：SM4 派生函数(SM4_df)返回的比特串

output_block：SM4 算法返回的比特串

requested_number_of_bits：输出函数(SM4_RNG_Generate)返回的随机比特串的长度

returned_bits：输出函数(SM4_RNG_Generate)返回的随机比特串

表 E.1、表 E.2 分别列出了 SM4_RNG 使用的参数要求和函数说明。

表 E.1　SM4_RNG 使用的参数要求

参数名称	参数类型	要求
V	比特串	长度为 blocklen
Key	比特串	长度为 keylen
outlen	整型	128 比特
keylen	整型	128 比特
blocklen	整型	128 比特
seedlen=outlen+ keylen	整型	256 比特
entropy_input	比特串	entropy_input 应至少具有 256 比特熵，其长度范围为[256，2^{35})，单位为比特
min_entropy_input_length	整型	256 比特
max_entropy_input_length	整型	2^{35} 比特
nonce	—	nonce 应至少具有 128 比特熵或者预期重复概率不大于 2^{-128}，nonce 可以是随机值、时间戳、单调递增的计数器值或者它们的组合，长度范围为[128，2^{34})
additional_input	比特串	长度不大于 2^{35} 比特
additional_input_length	整型	[seenlen，2^{35})，单位为比特
personalization_string	比特串	长度不大于 2^{35} 比特
requested_number_of_bits	整型	不大于 128 比特
reseed_interval_in_time	整型	安全等级一级：600 s 安全等级二级：60 s
reseed_interval_in_counter	整型	安全等级一级：2^{20} 安全等级二级：2^{10}
last_reseed_time	整型	取值为以秒为单位的时间值
reseed_counter	整型	初始值为 1，每调用一次输出函数自动加 1，最大值不超过 reseed_interval_in_counter

表 E.2　SM4_RNG 使用的函数说明

函数名称	描述
leftmost(V，a)	比特串 V 从左边起的 a 个比特
rightmost (V，a)	比特串 V 从右边起的 a 个比特
len (a)	比特串 a 的长度
Get_ entropy (min _ entropy，min _ entropy _ input_length，max_entropy_input_length)	从熵源获取一串比特的函数，要求返回的比特串(entropy_input)，至少含有 min_entropy 比特熵，长度大于或等于 min_entropy_input_length 比特，小于或等于 max_entropy_input_length 比特
select(V，a，b)	比特串 V 从比特位置 a 到 b 的子串
SM4(K，V)	使用 SM4 算法 ECB 工作模式加密 V(ECB 工作模式按 GB/T 17964)
SM4 _ RNG _ Instantiate （personalization _ string，nonce)	初始化函数：利用个性化字符串、nonce，对内部状态进行设置

表 E.2　SM4_RNG 使用的函数说明（续）

函数名称	描述
SM4_RNG_Update (seed_material,Key,V)	内部状态更新函数
SM4_df(input_string,number_of_bits_to_return)	SM4 派生函数,对输入字符串进行密码运算,返回长度为 number_of_bits_to_return 的比特串
CBC_MAC(Key,data_to_MAC)	一种基于 SM4 算法的 CBC 工作模式计算消息鉴别码的方法,按 GB/T 15852.1
SM4_RNG_Reseed (working_state, additional_input)	重播种函数:在提供新的熵输入的情况下更新内部状态
SM4_RNG_Generate (working_state, requested_number_of_bits,additional_input)	输出函数:用当前的内部状态和可能的额外输入返回随机数,并对内部状态进行更新
Get_current_time_in_second()	获取当前的时间值,单位为秒

E.2　内部状态

内部状态组成为 working_state{V,Key,reseed_counter,last_reseed_time,reseed_interval_in_counter,reseed_interval_in_time },定义如下:

——比特串 V,长度为 blocklen,每次一组长度为 blocklen 随机数生成时会进行更新;

——比特串 Key,长度为 keylen,当生成预定组数的输出时,会进行更新;

——reseed_counter,在初始化或重播种之后,请求随机数的次数,每次执行随机数生成操作时会自动加 1;

——last_reseed_time,上次 DRNG 重播种的时间值,单位为秒。

——reseed_interval_in_counter,重播种计数器阈值,该值为常量值;

——reseed_interval_in_time,重播种时间阈值,单位为秒,该值为常量值。

E.3　初始化函数

SM4_RNG 的初始化函数如下。

函数定义:SM4_RNG_Instantiate(personalization_string,nonce)

输入:

——personalization_string(可选);

——nonce。

输出:

——初始内部状态,即 V,Key,reseed_counter,last_reseed_time 的初始值。

SM4_RNG Instantiate 流程:

a)　min_entropy=min_entropy_input_length;

b)　(status,entropy_input)=Get_entropy(min_entropy,min_entropy_input_length,max_entropy_input_length);

c)　如果(status≠"Success"),返回错误信息;

d)　seed_material=entropy_input||nonce||personalization_string;

注:确保 seed_material 的长度为 seedlen 比特。

e)　seed_material=SM4_df (seed_material,seedlen);

f)　Key=0^{keylen}(keylen 位的零比特串);

g) V=$0^{blocklen}$(blocklen 位的零比特串)；

h) (Key,V)=SM4_RNG_Update(seed_material,Key,V)；

i) reseed_counter=1；

j) last_reseed_time=Get_current_time_in_second()；

k) Return(V,Key,reseed_counter,last_reseed_time)。

E.4 更新函数

更新函数 SM4_RNG_Update 使用 seed_material 更新 SM4_RNG 的内部状态。

函数定义：SM4_RNG_Update(seed_material,Key,V)

输入：

——seed_material：在初始化，重播种和输出函数中产生的比特串，长度为 seedlen；

——Key：当前 Key 的值；

——V：当前 V 的值。

输出：

——Key：Key 的更新值；

——V：V 的更新值。

SM4_RNG_Update 流程：

a) temp=Null；

b) While (len(temp)<seedlen) do：

1) V=(V+1) mod $2^{blocklen}$；

2) output_block=SM4(Key,V)；

3) temp=temp||ouput_block；

c) temp=leftmost(temp,seedlen)；

d) temp=temp ⊕ seed_material；

e) Key=leftmost(temp,keylen)；

f) V=rightmost(temp,blocklen)；

g) Return(Key,V)。

E.5 SM4 派生函数

函数定义：SM4_df(input_string,number_of_bits_to_return)

输入：

——input_string：函数操作的比特串，长度为 8 的倍数；

——number_of_bits_to_return：由 SM4_df 返回的比特数。

输出：

——requested_bits：执行 SM4_df 的结果。

SM4_df 流程：

a) L=len(input_string)/8；

注：L 是由 len(input_string)/8 产生的整数的位串，L 应该表示为一个 32 位整数。

b) N=number_of_bits_to_return/8；

注：N 是由 number_of_bits_to_return/8 产生的整数的位串表示，N 应表示为一个 32 位整数。

c) S=L||N||input_string||0x80；

注：将字符串长度和请求的输出长度添加到 input_string 中。如果需要，在 S 后面加 0。

d) While (len(S) mod outlen≠0)，S=S||0x00；

e) temp=Null;

f) i=0;

注：i为一个32位整数，如len (i) = 32。

g) K=leftmost (0x00010203...1F,keylen).

h) While len(temp)<keylen+outlen,do

1) IV=i||$0^{outlen-len\ (i)}$;

注：i的32位整数表示形式由0填充到outlen位。

2) temp=temp||CBC_MAC(K,(IV||S));

3) i=i + 1;

i) K=leftmost(temp,keylen);

j) X=select(temp,keylen+1,keylen+outlen);

k) temp=Null;

l) While len(temp)<number_of_bits_to_return,do;

1) X=SM4(K,X);

2) temp=temp||X;

m) requested_bits=leftmost(temp,number_of_bits_to_return);

n) Return(Success,requested_bits)。

E.6 CBC_MAC函数

CBC_MAC函数是一种计算消息鉴别码的方法，使用SM4算法的CBC模式。

函数定义：CBC_MAC(Key,data_to_MAC)

输入：

——Key：SM4的加密密钥；

——data_to_MAC：要操作的数据(数据长度应是outlen的整数倍)。

输出：

——output_block：CBC_MAC函数返回的结果。

CBC_MAC流程：

a) chaining_value=0^{outlen};

注：将第一个链接值设置为长度为outlen个0的比特串。

b) n=len(data_to_MAC)/outlen;

c) 从最左边的数据位开始，将data_to_MAC分割成n个outlen位块，形成块$block_1$到块$block_n$;

d) For i=1 to n:

1) input_block=chaining_value $\oplus$ $block_i$;

2) chaining_value=SM4 (Key,input_block);

e) output_block=chaining_value;

f) Return(output_block)。

E.7 重播种函数

函数定义：SM4_RNG_Reseed(working_state,additional_input)

输入：

——working_state：当前内部状态，即V,Key,reseed_counter和last_reseed_time的当前值；

——additional_input：额外输入(可选)。

输出：

——new_working_state：新的内部状态，即 V，Key，reseed_counter 和 last_reseed_time 的更新值。

SM4_RNG_Reseed 流程：

a) min_entropy＝min_entropy_input_length；

b) (status，entropy_input)＝Get_entropy(min_entropy，min_entropy_input_length，max_entropy_input_length)；

c) If (status≠"Success")，返回错误信息；

d) seed_material＝entropy_input||additional_input；

注：确保 seed_material 的长度为 seedlen 比特。

seed_material＝SM4_df(seed_material，seedlen)；

(Key，V)＝SM4_RNG_Update(seed_material，Key，V)；

e) reseed_counter＝1；

f) last_reseed_time＝Get_current_time_in_second()；

g) Return(V，Key，reseed_counter，last_reseed_time)。

E.8 输出函数

函数定义：SM4_RNG_Generate(working_state，requested_number_of_bits，additional_input)

输入：

——working_state：当前内部状态，即 V，Key，reseed_counter 和 last_reseed_time 的当前值；

——requested_number_of_bits：返回的随机数的长度，最大为 128 比特；

——additional_input：额外输入(可选)。

输出：

——new_working_state：新的内部状态，即 V，Key，reseed_counter 和 last_reseed_time 的新值；

——returned_bits：每调用 1 次输出函数，返回的随机数。

SM4_RNG_Generate 流程：

a) 如果 reseed_counter＞reseed_interval_in_counter，或者 Get_current_time_in_second()-last_reseed_time＞reseed_interval_in_time，则执行 SM4_RNG_Reseed，否则继续执行下述步骤；

b) If (additional_input≠Null)，then：

1) additional_input＝SM4_df(additional_input，seedlen)；

2) (Key，V)＝SM4_RNG_Update(additional_input，Key，V)；

Else additional_input＝$0^{seedlen}$；

c) V＝(V＋1) mod $2^{blocklen}$；

d) output_block＝SM4(Key，V)；

e) returned_bits＝leftmost(output_block，requested_number_of_bits)；

f) (Key，V)＝SM4_RNG_Update(additional_input，Key，V)；

g) reseed_counter＝reseed_counter＋1；

h) 返回(Success，returned_bits，Key，V，reseed_counter，last_reseed_time)。

参 考 文 献

[1] NIST SP 800-90A Revision 1,Recommendation for Random Number Generation Using Deterministic Random Bit Generators,June 2015

[2] NIST SP 800-90B,Recommendation for the Entropy Sources Used for Random Bit Generation,January 2018

[3] ISO/IEC 18031, Information technology—Security techniques—Random bit generation,November 2011

[4] Feller W.An introduction to probability theory and its applications: volume 1 [M].New York: Wiley,1950

[5] Zhu S, Ma Y, Li X, et al. On the analysis and improvement of min-entropy estimation on time-varying data. IEEE Transactions on Information Forensics and Security, 2020, 15: 1696-1708