随机数发生器的设计与检测

马　原[1,2]　陈天宇[1,2]　吴鑫莹[1,2,3]　杨　静[1,2,3]　林璟锵[1,2,3]　荆继武[1,2,3]

[1](中国科学院数据与通信保护研究教育中心　北京　100093)

[2](信息安全国家重点实验室(中国科学院信息工程研究所)　北京　100093)

[3](中国科学院大学网络空间安全学院　北京　100049)

(yma@is.ac.cn)

Design, Implementation and Testing of Random Number Generators

Ma Yuan[1,2], Chen Tianyu[1,2], Wu Xinying[1,2,3], Yang Jing[1,2,3], Lin Jingqiang[1,2,3], and Jing Jiwu[1,2,3]

[1](*Data Assurance and Communication Security Research Center*, *Chinese Academy of Sciences*, *Beijing* 100093)

[2](*State Key Laboratory of Information Security* (*Institute of Information Engineering*, *Chinese Academy of Sciences*), *Beijing* 100093)

[3](*School of Cyber Security*, *University of Chinese Academy of Sciences*, *Beijing* 100049)

Abstract　Random number generator (RNG) is indispensable for modern cryptography. The unpredictability of the generated random number provides basic security for cryptographic applications, such as cryptographic algorithms and security protocols. Once the quality of the random number cannot satisfy the security requirements as expected, it may lead to the existing of serious security risks in these applications. In this paper, it gives a systematic investigation and summary for the studies of RNGs from the view of design, and testing. On the design and implementation aspect, we introduce the researches on the hardware TRNGs and software TRNGs. On the testing aspect, it includes the research progress of RNG (black-box) statistical tests, entropy estimation and online tests.

Key words　random number generator (RNG); design; implementation; entropy estimation; statistical test

摘　要　随机数发生器(random number generator, RNG)在现代密码学中发挥着不可替代的作用,其生成随机数的不可预测性为密码算法和安全协议等密码应用提供基本的安全保障.一旦随机数的质量无法满足预期,则会导致密码应用存在严重的安全性风险.从RNG设计和检测的角度,对RNG的研究工作进行了全面系统的调研总结.在设计和实现方面,介绍了硬件和软件形式的TRNG研究工作;在检测方面,介绍了黑盒统计检测、熵估计方法和在线测试等RNG检测技术的研究进展.

关键词　随机数发生器;设计;实现;熵评估;统计测试

中图法分类号　T309.2

收稿日期:2018-11-15

基金项目:国家自然科学基金项目(61602476,61772518,61872357,61802396);"十三五"国家密码发展基金项目(MMJJ20170205,MMJJ20180113)

随机数发生器是密码学领域的安全基石，为众多密码技术提供根本性的安全保障. RNG 的输出被称为随机数，广泛用于对称或非对称算法的密钥产生、挑战-响应方案中的挑战值、数字签名方案中的秘密信息，以及抵抗侧信道分析攻击等方面.

虽然在密码体制的设计时，通常会理想地假设所用到的随机数是连续不断且不可预测的. 然而近年来关于随机数和 RNG 的安全问题频出，例如，RSA 算法在实现时由于使用了弱随机数，曾导致通过计算公因子将用到的大整数进行了分解[1]. 美国国家标准和技术研究院(National Institute of Standards and Technology, NIST)曾推荐使用的双椭圆曲线(Dual EC)随机数产生算法被广泛猜测设计上有后门[2].

按随机性来源和产生方式的不同，RNG 主要分为真随机数发生器(TRNG)和伪随机数发生器(PRNG). 对于 TRNG，它的不可预测性一般来自于物理或非物理现象的随机性，如电子噪声、电磁辐射、进程中断. 而 PRNG 则是采用确定性算法将一段随机序列(种子)扩展成任意长度的序列. 因此，TRNG 的质量更能决定随机数的安全，也更具研究价值. TRNG 又可细分为 2 类：基于物理源的硬件 RNG 和基于外部非物理源的软件 RNG.

本文，我们结合当前 TRNG 设计和检测研究工作，探讨 TRNG 在密码应用中需要考虑的几个方面，包括基于物理源的硬件 TRNG 设计和实现、基于非物理源的软 TRNG 设计和实现，以及 TRNG 的统计检测、熵估计方法和在线测试等检测技术. 以上几个方面，有的研究目前已经相对成熟，有的还在探索前行，有的关注尚少. 现有的大量研究表明，对 TRNG 的研究虽已有一定成果，但仍有诸多挑战有待解决，而在任何一个方面出现问题都会影响到 TRNG 的安全性、吞吐率等指标，导致提供的随机数服务无法在密码应用中达到预期效果，使得相应的密码应用存在严重的安全风险.

1 硬件 RNG 的设计与实现

在本节我们重点关注在硬件平台(如 AISC，FPGA)上的 RNG 设计与实现技术. 以硬件方式实现 RNG 是一种普遍的设计方式，硬件 RNG 也具有集成方便、安全性高等特点. 但是，需要说明的是，对硬件 RNG 的设计技术进行全面的调研非常困难，因为这方面的文献非常分散(不同学科领域的都会研究 RNG 的设计技术)，而且很多设计是以专利形式进行保护，并没有公开发表. 下面我们将重点梳理主要学术文献中的设计结构，并对产生原理和后处理方式进行介绍. 关于对硬件 RNG 进行安全性评估的内容参见第 3 节.

1.1 基本架构

硬件 RNG 通常利用电路中的随机性经过采样/数字化处理产生随机序列，若输出序列存在一些统计上的缺陷还可经过后处理电路处理再行输出. 硬件 RNG 的一般结构如图 1 所示：

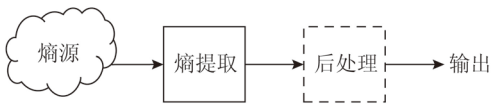

图 1 RNG 硬件设计的基本结构

RNG 在硬件设计时一般可分为熵源、熵提取以及后处理 3 个部分. 熵源又称随机源，即 RNG 所利用的物理现象中的随机性，例如电路中的热噪声和散射噪声、电路中信号的抖动和亚稳态、布朗运动、大气无线电噪声以及放射性衰变等. 熵源的质量直接关系着硬件 RNG 生成的随机数质量，因此熵源是硬件 RNG 设计中最核心的组成部分. 熵提取电路的设计是 RNG 硬件设计中的关键技术，目的是根据熵源和硬件 RNG 的基本原理，使用一些技巧或方法实现从熵源中收集尽可能多的随机性. 收集随机性的过程也可认为是将物理现象中的随机性数字化的过程. 后处理算法是在熵源受到一些因素的干扰或由于采用的熵提取方法不恰当，导致产生的比特序列存在偏差或相关性时，通过特定的算法对使用熵提取方法得到的比特序列进行处理，以提升序列的统计性质或增强硬件 RNG 在运行时的健壮性.

根据熵源类型的不同，目前主流的硬件 RNG 包括以下几种典型设计类型：基于时间抖动的硬件 RNG、基于亚稳态的硬件 RNG、基于混沌现象的硬件 RNG 以及基于量子的 RNG. 由于基于量子 RNG 的设计原理和实现平台与前 3 种有着明显区分，本文主要对可以用集成电路形式实现的

前 3 种产生方式进行研究，并对常用的后处理算法进行介绍.

1.2 基于时间抖动的硬件 RNG 设计

基于时间抖动的硬件 RNG 的随机性来源于电路噪声，噪声使得振荡信号的电平转换位置与理想位置存在偏差. 典型的设计结构如图 2 所示，低频振荡信号对高频振荡信号进行采样，由于低频或高频振荡信号中相位噪声的影响，采样位置存在不确定性，产生的随机数则具有随机性.

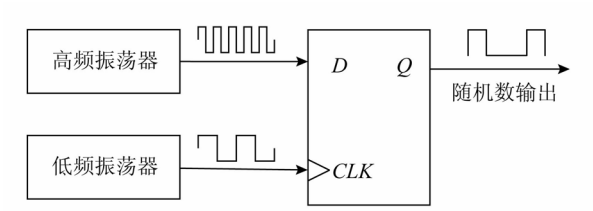

图 2 基于时间抖动的 RNG 典型结构

这种结构的设计原理清晰，实现方便，成为许多 RNG 设计的基础. 同时，许多对 RNG 进行理论建模和熵估计的工作也是针对这种结构的. 根据我们[3]在 CHES 2014 上的研究结论，由于抖动比例太小且采样方式的粒度较粗，在这种采样方式下，需要抖动累积很长时间才能保证采样得到的熵是充足的，这就导致了这种基础架构的随机数产生速率受限（只有 Kbps 量级）.

因此，许多基于时间抖动的 RNG 设计工作都在研究如何提升随机数产生速率. 这方面的设计技术可以分为 2 类：一类是改进振荡器结构，让随机性更多地累积到振荡信号中；另一类是改进采样方式，让抖动能尽快体现在采样出的随机数中.

在改进振荡器结构方面，Sunar 等人和 Wold 等人[4-5]提出多环结构，将多个振荡环异或再采样以增加抖动量来提高吞吐率，最后输出能达到 100 Mbps. 但是，这种多环结构也容易受到外界攻击的干扰，如 Markettos 等人[6]提到了对多环设计的频率注入攻击，受攻击的振荡环会出现频率互锁现象，使得输出的随机性明显降低. Cherkaoui 等人[7-8]设计了一款新的振荡环（self-timed ring，STR），使得相邻级振荡信号跳变沿的时间差小于抖动，利用时钟信号采样各级输出的振荡信号，再进行异或作为随机数输出，这样能保证每一次的采样都在某一级振荡信号的抖动区间.

在改进采样方式方面，Kohlbrenner 等人[9]和 Yang 等人[10]提到利用相干采样技术提高抖动分辨率达到提升吞吐率，速率最后提升到 4 Mbps. 以及 Rozic 等人[11]将振荡环中每级振荡信号用进位链引出，再进行采样输出，提高抖动率分辨率，速率可至 14.3 Mbps.

1.3 基于亚稳态的硬件 RNG 设计

基于亚稳态的硬件 RNG 设计是指触发器捕捉到的信号位置，并非 2 个确定的逻辑状态：0 或 1，而是 2 个逻辑状态转换过程的沿上这一平衡态（亚稳态）. 典型设计[12]如图 3 所示，在时钟 CLK 的上升沿 RS 锁存器的输出出现亚稳态（$(Q,\bar{Q})=(1,0)$或$(0,1)$）.

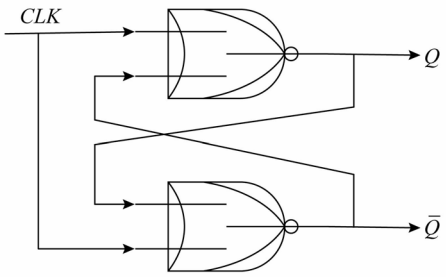

图 3 基于 RS 锁存器的亚稳态现象示例

由于所采的区域在 2 个确定状态之间，因此输出变得随机. 这种状态的维持依赖于电路噪声的性质[13]. 一般来说亚稳态设计的 TRNG 产生速率比较理想，但容易受到周围环境的影响致使输出特性不够稳定. 如 Danger 等人[14]的设计，将 1 个时钟信号的多个小延时信号异或再用同一个时钟信号直接采样输出，要求小延时比抖动更小. 再如 Majzoobi 等人[15]的设计，一个振荡信号经 2 条延时线分别产生 2 个振荡信号，让其中一个去采样另一个作为输出. 为了提高采到亚稳态的概率，利用了 FPGA 中查找表（LUT）中悬空（未用）输入的赋值不同来细粒度调整信号的传播路径长度.

1.4 基于混沌现象的硬件 RNG 设计

基于混沌现象的硬件 RNG 设计是指利用一个确定性的低维动力系统对初始条件非常敏感所展现的随机性行为. 利用混沌函数的特性设计混沌系统，是将随机性噪声作为这个混沌系统的微小扰动，由于系统的输出受系统中随机噪声的影响，使系统输出序列不可预测，产生随机序列.

混沌系统包括离散混沌和连续混沌 2 种. 一种典型的基于离散混沌系统的 RNG 结构如图 4

所示，以电路方式实现的混沌函数称为混沌电路，包括内部状态和反馈电路 2 个部分. 由于混沌电路的初态不能完全确定，因此外部采样时无法确定采样的状态，从而产生了随机性. 基于混沌现象设计的随机数发生器具体结构可参考文献[16-19].

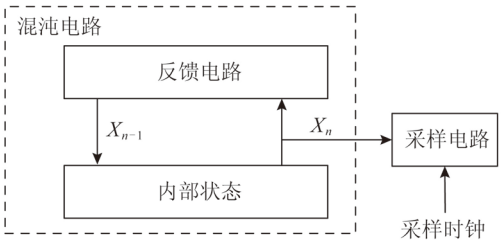

图 4　基于离散混沌系统的 RNG 基本结构

混沌电路在实现时还存在很多不稳定因素，如上电或重启时初始电压有可能被锁死在低电平或高电平，影响随机数输出的质量. 因此，在设计混沌电路时要充分考虑相关不稳定因素，使电路能稳定工作在混沌区.

1.5　后处理算法

在随机数发生器的硬件设计与实现中，由于电路内部相关噪声、外部环境变化以及器件本身的缺陷等因素，有熵源直接产生的随机数存在一定的偏差或相关性，这就需要采用后处理算法对其进行修正. 常用的后处理方法包括：1）较为简单的异或后处理（XOR corrector）[20]. 将多个不重叠的原始序列异或后作为输出，该方法以降低吞吐率来提升序列的统计特性，产生稳定速率的输出. 2）冯·诺依曼校正（Von Neumann corrector）[21]. 将不重叠的连续 2 b 相同的输入（00 或 11）舍弃，将不同的输入保留（01 输出 1，10 输出 0），该方法对统计独立但不均衡的序列具有很好的效果. 除此之外，还有较为复杂的 H 函数（H post-processing function）[22-23]、弹性函数（resilient funtcion）[24-25]等.

2　基于外部熵源的软件 RNG 设计与分析

随着随机数使用范围扩大，传统基于硬件的随机数发生器已不能完全满足当前的技术发展和用户需求，对于便携式物联网设备、智能移动终端、路由器等，由于其难以嵌入专用的硬件随机数发生器芯片或者模块，因此，只能通过采集外部熵源信息，并使用软件处理的方式产生随机数. 由于基于外部熵源的软件 RNG 中的熵源不可控，对处理算法设计和实现的安全性要求较高，因此出现了许多针对软件平台上 RNG 实现的分析和攻击.

2.1　软件 RNG 设计

如何设计一个随机性好的软件随机数发生器一直是密码学研究的重点，一些工作给出了随机数发生器的设计建议[25-26]：Barak 等人[26]提出稳健的随机生成模型和架构；Gutmann[27]提供了设计和实现一个随机数发生器的全面指南；Shamir[26]首次给出从短随机种子中生成一长串伪随机数的实例. 此外，软件随机数发生器目前已有许多成熟的结构：Kelsey 等人和 Murray 等人设计和实现了 Yarrow 结构[28-29]，iOS 和 Mac OS X 系统使用 Yarrow 发生器，FreeBSD 之前的版本也使用该发生器提供随机数；Schneier 等人[30]对 Yarrow 算法进行了改进，提出了 Fortuna 结构，FreeBSD 系统使用 Fortuna 发生器取代了 Yarrow 发生器；Gutterman 等人[31]分析了 Linux 内核中的随机数发生器（LRNG）的结构. Dorrendorf 等人[32]分析了 Windows 内核的随机数发生器 CryptGenRandom 的结构.

一般来说，基于外部熵源的软件随机数发生器由熵源、处理函数、中间状态、输出函数组成. 其结构如图 5 所示. 熵源主要分为 2 类：基于系统时钟生成种子和外部随机源提供种子. 外部随机源包括：鼠标键盘的操作、网络中断、磁盘读写或真随机数发生器提供的随机数. 处理函数用于熵提取和更新中间状态，包括熵提取函数、播种（seed）函数和重播种（reseed）函数. 熵源通过熵提取和播种函数产生中间状态，通过条件控制（熵值达到阈值，数据长度达到阈值等）执行种子更新（重播种函数），来更新中间状态. 中间状态用来存储发生器使用或产生作用的所有参数、变量和其他存储值，包含管理数据（如安全强度），以及在工作状态期间产生作用或修改的数据. 用输出函数对中间

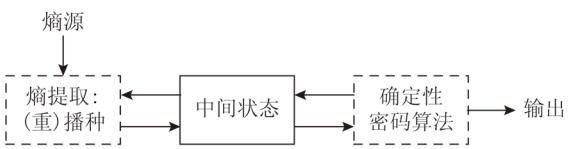

图 5　基于外部熵源的软件 RNG 基本结构

状态处理生成随机数，目前的发生器中使用的输出函数有 CTR 模式的分组密码、杂凑算法等.

下面主要介绍在通用操作系统上的软件随机数发生器的结构，包括 Mac，Linux，Android 和 Windows 内核所带的随机数发生器.

iOS 和 Mac OS X 系统使用 Yarrow 发生器，Yarrow 设计者在设计时考虑可能的攻击（选择输入攻击、迭代猜测攻击、回溯攻击等），是面向攻击的设计，设计中尽量避免了攻击隐患. Yarrow 熵源为鼠标键盘的操作. 通过杂凑算法播种和重播种，采用 CTR 模式的分组密码输出随机数. Yarrow 根据累积的熵值大小决定是否执行重播种操作，因此，熵估计的准确性直接影响重播种的效果. 但是 Yarrow 的熵估计算法并不准确，会影响生成随机数的质量. Fortuna 发生器通过多熵池（32 个熵池）和重播种策略解决熵估计不准确的问题，去掉了熵估计模块. Fortuna 在调用输出单元时检查是否需要重播种，通过熵池数据长度和更新间隔时间来控制是否需要重播种.

Linux 随机数发生器（即 LRNG）的熵源为鼠标键盘的操作、硬盘读写操作以及特殊的中断. 有 3 个熵池：主熵池、阻塞熵池和非阻塞熵池. 熵池存储中间状态，熵池更新的机制基于 TGFSR（扭环广义反馈移位寄存器），提取函数为 SHA-1 和一个称为 folding 函数的结合.

Android 系统的随机数发生器（ARNG）基于 LRNG. ARNG 熵源为进程 ID、用户 ID、系统时间和 Linux 内核提供的随机数等，其中 Linux 内核提供的随机数为主要的熵源. 初始化 ARNG 状态和内部状态更新，通过 Rand_add() 函数，使用多轮杂凑算法将熵源数据混淆到内部状态中. 输出随机数调用 Rand_byte() 函数，多次杂凑产生随机数.

Windows 内核的随机数发生器为 CryptGenRandom，熵源为当前进程 ID、线程 ID、本地时间、用户名、硬盘空间等. 从熵源提取熵和更新内部状态使用的是杂凑和 RC4 算法结合的函数，输出函数使用基于 SHA-1 的算法，从内部状态产生随机数同时更新状态.

2.2 软件 RNG 安全性分析

对软件 RNG 进行安全性分析十分必要. 其安全性应能抵抗住外部和内部的攻击. 假定攻击者已知发生器的代码，并且可能对内部状态的更新信息有一定程度的掌握，软件随机数发生器在设计，检测和使用时应保证满足以下的安全要求[26]：

1）伪随机性. 对于一个对发生器内部状态一无所知的观测者而言，发生器的输出应该是看上去随机的. 即便观测者完全控制了用于更新内部状态的数据，上述结论应仍保持不变.

2）前向安全性. 对于一个攻击者而言，即便他知道发生器在某一特定时刻的状态，也不能由此获知任意之前时刻的输出.

3）后向安全性（入侵恢复性）. 对于一个攻击者而言，即便他知道了发生器在某一特定时刻的状态，也不能由此获知任意之后时刻的输出.

对于软件随机数发生器的上述要求，敌手对发生器状态完全不可知是一个充分条件. 在很多的场景中，敌手也许能够获取到发生器的状态，例如通过旁路绕过操作系统的限制，或是将状态从内存或硬盘中读取出来. 前向安全性的要求确保了敌手在已知某一时刻状态的条件下，也无法获取之前时刻发生器输出的相关信息. 后向安全性的要求确保了在已知某一时刻状态的条件下，敌手预测之后时刻的发生器输出的困难度也不会降低.

现有软件 RNG 中仍然存在一些局限性，即使在最广泛使用的 Windows 和 Linux 系统中，它们也会受到 Gutterman 等人[31-32]描述的攻击. 目前对软件 RNG 的攻击可分为以下 2 类：

1）发生器不满足前向安全性的要求. Gutterman 等人[31]发现 Linux 内核中的随机数发生器无法完全满足前向安全性的要求，利用熵更新函数 refresh() 启动机制的缺陷，可以在已知当前发生器状态下恢复出之前时刻的状态. Windows 内核随机数发生器 CryptGenRandom 的熵源为当前进程 ID、线程 ID、本地时间、用户名、硬盘空间等. Dorrendorf 等人[32]指出 Windows 内核中的随机数发生器 CryptGenRandom 也存在无法满足前向安全性的问题，根据发生器的中间状态可以在 2^{23} 步内计算出发生器之前的状态，实现了一个缓冲区溢出攻击得到发生器的一个状态，从而预测所有的随机数.

2）攻击是由于熵源熵不足，主要为基于 Linux 的系统. Linux 系统内核的随机数发生器熵源为鼠

标、键盘、磁盘读写和网络中断. 基于嵌入式 Linux 系统的开源无线路由固件 OpenWRT 没有鼠标键盘和硬盘，因此存在熵源提供的熵不足的问题. Yoo 等人[33]监听 OpenWRT 从初始化起的网络信息流量，恢复出 OpenWRT 中发生器的内部状态. Android 系统的随机数发生器基于 Linux 的随机数发生器. 手机端操作没有鼠标键盘的操作以及中断，因此存在种子熵不足的问题. Kim 等人[34]揭示 Android 上的 OpenSSL PRNG 漏洞，使用 OpenSSL 的应用程序从相同的初始状态开始生成随机数据，利用此漏洞和熵源熵不足的问题，可以恢复出第 1 个 SSL 会话的 PreMasterSecret，计算复杂度为 $O(2^{58})$. 同样由于 OpenSSL 发生器的种子熵不足原因，Strenzke[35]提供了一种恢复方法用于说明可以借助当前的输出数据来得到 OpenSSL 发生器之前时刻的状态. 在特定场景下，作者还发现这种熵不足的问题会导致通过输出数据甚至可以恢复出发生器的种子. 针对熵源熵不足的问题，在发生器设计时一方面考虑加入新的真随机种子来增加熵，另一方面，优化发生器熵估计的准确性，从而保证熵足时再产生随机数.

3 RNG 的安全性检测

RNG 的安全性评估和检测贯穿 RNG 的整个生命周期，包括 RNG 的设计、使用等环节，目的是确保 RNG 的安全性和预期的相符. RNG 在设计时需要根据其结构和工作原理，给出严格的安全性论证，使得随机数的产生原理和发生器的安全机制能够满足所预期的安全需求，采取的方法如基于随机模型的熵估计和在线测试. RNG 在（离线）检测和使用环节，需要验证所设计的发生器与其声称的安全性是一致的. 一方面，可以采用重新推演或其他验证手段，判断相应的随机模型、在线测试等方法的正确性和有效性；另一方面，可以采用较为通用的黑盒统计测试等方法，检查 RNG 熵源或输出的质量是否存在明显缺陷.

下面，我们将从 4 个方面介绍 RNG 安全性评估和检测的相关工作，包括：1）黑盒统计检测技术；2）理论熵估计方法；3）统计熵估计方法；4）RNG 在线测试技术.

3.1 黑盒统计检测技术

黑盒统计检测是对 RNG 输出的序列进行统计性质的核验，判断被测序列是否存在（明显的）统计缺陷. 由于这类检测技术属于对 RNG 的黑盒测试，所以具有良好的通用性，适用于各类型 RNG 的安全性评估，但也因此很难或无法检查出 RNG 内部（如熵源、熵提取方法）问题，例如确定性序列经过严格后处理算法（如杂凑算法）后，输出序列依然可以顺利通过统计检测.

在实际使用黑盒统计检测时，一般是将若干个针对于不同方面的统计性质的检测项合并到一起，形成一套统计检测套件. 使用者利用检测套件检查被测序列，并根据检测结果评估 RNG 的质量. 个人和研究机构对统计检测标准或要求的制定，以及配套检测套件的实现均作出过贡献. 其中，由研究机构发布的有：美国 NIST 的 FIPS 140 系列[36-37]、SP 800-22[38] 和 SP 800-90B[39]；德国 BSI 的 AIS 20/AIS 31[40]；我国的密码行业标准 GM/T 0005—2012《随机性检测规范》[41]（现已升级为国家标准 GB/T 32915—2016《信息安全技术 二元序列随机性检测方法》）和 GM/T 0062—2018《密码产品随机数检测要求》. 由个人提出的有：George 的 DiehardBattery[42]；L'Ecuyer 和 Simard 提出的 TestU01[43]等.

NIST SP 800-22 是目前具有代表性的一款统计检测套件，对随机序列的统计性质具有较全面的覆盖. 其研究工作主要集中在 3 个方面：1）对检测项和检测结果评判方法的改进[44-49]，使得套件的检测结果可以更准确、可靠地反映被测序列真实的随机性水平；2）探究检测项之间的关系[50-52]，分析检测套件对序列统计性质的覆盖程度，以及检测项之间是否存在关联[51]使得检测结果有所重叠；3）提升检测效率[53-55]，结合运行环境特点改进检测套件执行效率，使其可以充分利用环境资源或满足环境需要，所指环境如并行计算、在线测试. 我们的研究一方面改进了 SP 800-22 中检测结果评判方法的不足，增强了套件的检测强度；另一方面针对 SP 800-22 提出了适用于 on-the-fly 测试的效率优化方案，更高效地检查出不合格序列.

3.2 理论熵估计方法

理论熵估计方法属于对 TRNG 的白盒测试，其思想是从熵的角度出发，根据 TRNG 的工作原

理，从理论上计算出 RNG 输出的真随机性大小，分为对基于物理源的硬件 TRNG 和基于非物理源的软件 TRNG 熵估计方法 2 类.

3.2.1 对基于物理源的 TRNG 熵估计方法

基于物理源的 TRNG 熵估计方法是从 TRNG 的结构和工作原理出发形成的一套白盒测试. 相比于基于统计性质的黑盒检测，这种熵估计方法首先对特定 TRNG 建立数学模型，然后通过熵计算来量化 TRNG 的输出所含有的真随机性大小.

目前，基于物理源的 TRNG 熵估计方法的研究还集中在对基础或通用结构上，近年来研究人员普遍关注对振荡采样结构的熵估计. 对该结构的熵估计主要有 2 个研究角度：基于时间演变和基于相位演变，二者只是衡量角度不同，本质上是可以等价的. 对于振荡采样结构，熵估计的难点在于 n 比特概率的计算，和进一步熵求解时面临的空间爆炸. 为了避免这个问题，Killmann 等人[56]从时域角度对一款通用的 TRNG 结构进行建模，并通过计算条件熵给出了比特率熵的一个下界，但并未获得精确的熵. 为此，Baudet 等人[57]从相位角度给出了比特率熵的精确计算. 在 CHES 2014 上，我们的研究[3]也从时域角度建模，并通过迭代方式给出了一套计算比特率熵的方法. 除了上述基础架构外，Sunar 等人[4]提出的多环结构也得到广泛应用，但由于还没有数学方法对频率互锁问题建模，所以多环结构的熵估计研究一直以来是一个开放问题.

3.2.2 对基于非物理源的 TRNG 熵估计方法

由于基于非物理源的 TRNG 熵源部分由1 种或多种带有随机性成分但非物理现象的事件（如敲击键盘、磁盘写入/写出操作）组成，而这些行为一般复杂多变，其随机过程不具有平稳性，无法给出相应的合理假设，因此不能采用建立随机模型的方式得到熵估计结果. 通常，这类 RNG 的熵估计包括以下 2 部分：1）熵收集模块. 从事件中收集并累积熵的数量，等待用户调用请求，提供所需数量的熵. 2）熵估计策略. 根据 RNG 实际采用的带有随机性成分的事件，制定的估计熵值的方案，用于量化熵收集模块中熵值的变化.

目前，这部分研究工作的典型是 Linux 内核中的 LRNG 所使用的熵估计方法，其主要思想是根据同一非物理事件发生的时间间隔来计算熵[31].

3.3 统计熵估计方法

统计熵估计是在随机数统计检测的基础上，从“熵”的角度评估 RNG 的安全性，典型实例如 NIST 于 2018 年初发布的标准 NIST SP 800-90B[39]给出的统计熵估计方法. 该标准并非要求设计者对发生器的工作原理进行建模，而是提供了一套通用的统计检测方法，从最小熵的角度评估熵源质量.

标准 SP 800-90B 的第 1 版草案中采纳了由 Hagerty 等人[58]提出的基于最大频数、碰撞、压缩、马尔科夫、重复模板估计、最长重复子序列等熵估计方法，这类方法将待测序列当作整体来考虑，用统计方法研究熵源不同方面的性质. 第 2 版草案在 2016 年发布，其中重要的更新是添加了由 NIST 的 Kelsey 等人[59]在 CHES 2015 上提出的基于预测模型的熵估计方法. 不过，在 Kelsey 等人修订标准的同时，也发现了 SP 800-90B 检测套件中存在熵估计不准确的问题. 例如，他们通过比较 SP 800-90B 中众多熵估计方法得到的结果，发现最小值频繁出自于碰撞估计和压缩估计，而且明显低于其他方法的估计值，因此猜测这 2 种方法本身可能存在不足. 这一疑问在我们的工作[60]中得到证实，该研究首次分析了上述缺陷存在的根本原因，同时给出了一种新的熵估计器，能够在较小开销情况下完成对随机数熵的准确估计，并且能够替代现有标准中的多个熵估计器.

3.4 RNG 在线测试技术

在线测试的目的旨在 RNG 运行期间，通过对其安全性的监控以确保输出序列的质量. RNG 的在线测试技术分为 2 类：1）自检测. 利用一些在线检测项，将其内嵌在 RNG 结构中或留出测试接口以供第三方检测. 一般地，若未通过检测则应立即停止 RNG 或报警禁止输出，以防止未通过检测的序列输出造成安全风险. 2）自校验. 根据 RNG 设计和随机模型，确定可以反映 RNG 安全性的关键安全参数. 通过在 RNG 运行时对该关键安全参数测量、校验（与模型中理论值比较），将校验结果反馈给 RNG，作为调整设计参数或压缩比特数的依据，起到监控运行时 RNG 安全性并保证 RNG 可以连续输出高质量随机序列的作用.

早期的在线测试技术以自检测为主，例如 Schindler[61]在 CHES 2001 上提出了高效的在线

检测方法，以供设计者和第三方进行RNG的安全性自检测. 近年来，在线测试技术研究则更趋向于自校验的方法设计和实现，因为这不仅可以监测RNG运行是否正常，而且可以及时纠正设计参数，确保了RNG在恶劣的环境下依然可以输出高质量的序列，从而提升了RNG的健壮性. 在CHES 2005上，Bucci和Luzzi[62]建议对未处理序列进行熵检测，并根据熵检测结果对输出前的数据进行压缩以保证RNG输出的质量. 另外，他们提到：噪声源质量的评估还应该包括其在攻击、故障和制造缺陷方面的健壮性表现.

随着技术和标准的推进，一种内部嵌入式的熵测量技术成为研究的焦点，该测量技术是在噪声源内部设计一块检测电路，通过计算一些和噪声源安全性直接相关的参数，实现对噪声源的熵进行实时监测的目的. 在CHES 2014上，Fischer和Lubicz[63]采用蒙特卡洛随机抽样方法，设计了一款可以精确测量抖动的电路. 而文献[3]为验证模型的正确性，提出了一种双计数器的抖动测量电路. 与文献[63]不同的是，我们的电路是直接对安全性参数（质量因子）测量，而无需进行从抖动到质量因子的转换. 在DATE 2014上，Haddad等人[64]修正了过去随机模型的假设，即加入了相关噪声影响下的相关抖动部分，并给出了一种周期抖动的离线测量方法. 在2015年IEEE Trans on Computers期刊上，Lubicz和Bochard[65]基于相干（欠）采样技术设计抖动测量方法. 2017年，Yang等人[66]采用快速进位链，给出了一款针对振荡采样结构的简洁高效的测量方法，并利用差分采样技术避免了确定性干扰等不良因素的影响.

4 结束语

随机数发生器在现代密码学中占据非常重要的地位，许多安全事件的发生都是其中使用的随机数或随机数发生器遭到破解，因此对随机数发生器的设计和检测技术的研究一直是研究的热点话题. 本文系统调研并分析了软硬件平台上的随机数发生器设计方法和检测技术，指出了以熵为评价指标的随机数发生器设计与检测理念，可以为我国随机数发生器相关标准的制定提供参考.

参 考 文 献

[1] Bernstein D J, Chang Y A, Cheng C M, et al. Factoring RSA keys from certified smart cards: Coppersmith in the wild [G] //LNCS 8270: Advances in Cryptology - ASIACRYPT 2013. Berlin: Springer, 2013: 341-360

[2] Barak B, Shaltiel R, Tromer E. True random number generators secure in a changing environment [G] //LNCS 2779: Cryptographic Hardware and Embedded Systems CHES 2003. Berlin: Springer, 2003: 166-180

[3] Ma Y, Lin J, Chen T, et al. Entropy evaluation for oscillator-based true random number generators [G] // LNCS 8731: Cryptographic Hardware and Embedded Systems CHES 2014. Berlin: Springer, 2014: 544-561

[4] Sunar B, Martin W J, Stinson D R. A provably secure true random number generator with built-in tolerance to active attacks [J]. IEEE Trans on Computers, 2007, 56(1): 109-119

[5] Wold K, Tan C H. Analysis and enhancement of random number generator in FPGA based on oscillator rings [C] // Proc of Int Conf Reconfigurable Computing and FPGAs. Piscataway, NJ: IEEE, 2008: 385-390

[6] Markettos A T, Moore S W. The frequency injection attack on ring-oscillator-based true random number generators [G] //LNCS 5747: Cryptographic Hardware and Embedded Systems CHES 2009. Berlin: Springer, 2009: 317-331

[7] Cherkaoui A, Fischer V, Aubert A, et al. A self-timed ring based true random number generator [C] //Proc of the 19th IEEE Int Symp on Asynchronous Circuits and Systems. Piscataway, NJ: IEEE, 2013: 99-106

[8] Cherkaoui A, Fischer V, Fesquet L, et al. A very high speed true random number generator with entropy assessment [G] //LNCS 8086: Cryptographic Hardware and Embedded Systems CHES 2013. Berlin: Springer, 2013: 179-196

[9] Kohlbrenner P, Gaj K. An embedded true random number generator for FPGAs [C] //Proc of the 2004 ACM/SIGDA 12th Int Symp on Field programmable gate arrays. New York: ACM, 2004: 71-78

[10] Yang J, Ma Y, Chen T, et al. Extracting more entropy for TRNGs based on coherent sampling [C] //Proc of Security and Privacy in Communication Systems. Berlin: Springer, 2016: 694-709

[11] Rozic V, Yang B, Dehaene W, et al. Highly efficient entropy extraction for true random number generators on FPGAs [C] // Proc of the 52nd ACM/EDAC/IEEE Design Automation Conference. Piscataway, NJ: IEEE, 2015: 1-6

[12] Hata H, Ichikawa S. FPGA implementation of metastability-based true random number generator [J]. IEICE Trans on Information and Systems, 2012, 95(2): 426-436

[13] Varchola M. FPGA based true random number generators for embedded cryptographic applications [D]. Kosice: Technical University of Kosice, 2008: 74-76

[14] Danger J L, Guilley S, Hoogvorst P. High speed true random number generator based on open loop structures in FPGAs [J]. Microelectronics Journal, 2009, 40(11): 1650-1656

[15] Majzoobi M, Koushanfar F, Devadas S. FPGA-based true random number generation using circuit metastability with adaptive feedback control [G] //LNCS 6917: Cryptographic Hardware and Embedded Systems CHES 2011. Berlin: Springer, 2011: 17-32

[16] Ergün S. A high-speed truly random number generator based on an autonomous chaotic oscillator [C] //Proc of Asia Pacific Conf on Circuits and Systems 2014. Piscataway, NJ: IEEE, 2014: 217-220

[17] Ergün S, Özoğğuz S. A truly random number generator based on a continuous-time chaotic oscillator for applications in cryptography [G] //LNCS 3733: Computer and Information Sciences ISCIS 2005. Berlin: Springer, 2005: 205-214

[18] Koyuncu I, Ozcerit A T, Pehlivan I, et al. Design and implementation of chaos based true random number generator on FPGA [C] //Proc of the 22nd Signal Processing and Communications Applications Conf (SIU). Piscataway, NJ: IEEE, 2014: 236-239

[19] Zhou T, Zhou Z, Yu M, et al. Design of a low power high entropy chaos-based truly random number generator [C] // Proc of 2006 IEEE Asia Pacific Conf on Circuits and Systems. Piscataway, NJ: IEEE, 2006: 1955-1958

[20] Davies R B. Exclusive OR (XOR) and hardware random number generators [J/OL]. [2018-11-15]. http://www.robertnz.net

[21] Von Neumann J. Various techniques used in connection with random digits [J]. National Bureau of Standards, Applied Math Series, 1951, 12(1): 36-38

[22] Dichtl M. Bad and good ways of post-processing biased physical random numbers [G] //LNCS 4593: Fast Software Encryption 2007. Berlin: Springer, 2007: 137-152

[23] Lacharme P. Post-processing functions for a biased physical random number generator [G] //LNCS 5086: Fast Software Encryption 2008. Berlin: Springer, 2008: 334-342

[24] Colbourn C J, Dinitz J H, Stinson D R. Applications of combinatorial designs to communications, cryptography, and networking [J]. Surveys in Combinatorics, 1993m 187 (1): 37-100

[25] Barak B, Halevi S. An architecture for robust pseudo-random generation and applications to/dev/random [C] // Proc of ACM Conf on Computer and Communications Security. New York: ACM, 2005: 203-212

[26] Shamir A. On the generation of cryptographically strong pseudorandom sequences [J]. ACM Trans on Computer Systems (TOCS), 1983, 1(1): 38-44

[27] Gutmann P. Software generation of practically strong random numbers [C] //Proc of the 7th USENIX Security Symp. Berkeley, CA: USENIX Association, 1998: 243-257

[28] Kelsey J, Schneier B, Ferguson N. Yarrow-160: Notes on the design and analysis of the yarrow cryptographic pseudorandom number generator [G] //LNCS 1758: Selected Areas in Cryptography 1999. Berlin: Springer, 1999: 13-33

[29] Murray M R V. An Implementation of the Yarrow PRNG for FreeBSD [C] //Proc of BSDCon 2002. Berkeley, CA: USENIX Association, 2002: 47-53

[30] Ferguson N, Schneier B. Practical cryptography [M]. New York: Wiley, 2003

[31] Gutterman Z, Pinkas B, Reinman T. Analysis of the Linux random number generator [C] //Proc of 2006 IEEE Symp on Security and Privacy (S&P'06). Piscataway, NJ: IEEE, 2006: 371-385

[32] Dorrendorf L, Gutterman Z, Pinkas B. Cryptanalysis of the random number generator of the windows operating system [J]. ACM Trans on Information and System Security (TISSEC), 2009, 13(1): 1-32

[33] Yoo D, Yeom Y. The OpenWRT's random number generator designed like/dev/urandom and its vulnerability [G] //Advances in Computer Science and Ubiquitous Computing CSA-CUTE2016. Berlin: Springer, 2016: 825-830

[34] Kim S H, Han D, Lee D H. Predictability of Android OpenSSL's pseudo random number generator [C] //Proc of the 2013 ACM SIGSAC Conf on Computer & Communications Security. New York: ACM, 2013: 659-668

[35] Strenzke F. An analysis of OpenSSL's random number generator [G] //LNCS 9665: Advances in Cryptology EUROCRYPT 2016. Berlin: Springer, 2016: 644-669

[36] National Institute of Standards and Technology. Federal information processing standards publication fips pub 140-1 [S]. Gaithersburg: National Institute of Standards and Technology, 1994

[37] National Institute of Standards and Technology. Federal information processing standards publication fips pub 140-2 [S]. Gaithersburg: National Institute of Standards and Technology, 2001

[38] Rukhin A, Soto J, Nechvatal J, et al. A statistical test suite for random and pseudorandom number generators for cryptographic applications [R]. Mclean Va: Booz-Allen and Hamilton Inc, 2001

[39] Barker E B, Kelsey J M. Recommendation for Random Number Generation Using Deterministic Random Bit Generators (Revised)[M]. Gaithersburg: US Department of Commerce, Technology Administration, National Institute of Standards and Technology, Computer Security Division, Information Technology Laboratory, 2007

[40] Killmann W, Schindler W. A proposal for: Functionality classes for random number generators [S]. Bonn: Bundesamt fur Sicherheit in der Informationstechnik, 2011

[41] 国家密码管理局. GM/T 0005—2012 随机性检测规范[S]. 北京：国家密码管理局，2012

[42] Marsaglia G. The Marsaglia random number CDROM including the diehard battery of tests of randomness [J/OL]. [2018-11-15]. http://www. stat. fsu. edu/pub/diehard/

[43] L'Ecuyer P, Simard R. TestU01: AC library for empirical testing of random number generators [J]. ACM Trans on Mathematical Software (TOMS), 2007, 33(4): 1-40

[44] Kim S J, Umeno K, Hasegawa A. Corrections of the NIST statistical test suite for randomness [J/OL]. IACR Cryptology ePrint Archive, (2004) [2018-11-15]. https://arxiv. org/abs/nlin/0401040v1

[45] Hamano K. The distribution of the spectrum for the discrete Fourier transform test included in SP800-22 [J]. IEICE Trans on Fundamentals of Electronics, Communications and Computer Sciences, 2005, 88(1): 67-73

[46] Hamano K, Kaneko T. Correction of overlapping template matching test included in NIST randomness test suite [J]. IEICE Trans on Fundamentals of Electronics, Communications and Computer Sciences, 2007, 90(9): 1788-1792

[47] Sulak F, Doğanaksoy A, Ege B, et al. Evaluation of randomness test results for short sequences [G] //LNCS 6338: Sequences and Their Applications SETA 2010. Berlin: Springer, 2010: 309-319

[48] Pareschi F, Rovatti R, Setti G. Second-level NIST randomness tests for improving test reliability [C] //Proc of 2007 IEEE Int Symp on Circuits and Systems. Piscataway, NJ: IEEE, 2007: 1437-1440

[49] Pareschi F, Rovatti R, Setti G. On statistical tests for randomness included in the NIST SP800-22 test suite and based on the binomial distribution [J]. IEEE Trans on Information Forensics and Security, 2012, 7(2): 491-505

[50] Turan M S, DoĞanaksoy A, Boztaş S. On independence and sensitivity of statistical randomness tests [G] //LNCS 5203: Sequences and Their Applications SETA 2008. Berlin: Springer, 2008: 18-29

[51] 范丽敏，冯登国，陈华. 基于熵的随机性检测相关性研究[J]. 软件学报，2009，20(7)：1967-1976

[52] Sulak F, UĞUZ M, Kocak O, et al. On the independence of statistical randomness tests included in the NIST test suite [J]. Turkish Journal of Electrical Engineering & Computer Sciences, 2017, 25(5): 3673-3683

[53] Chen M, Chen H, Fan L, et al. Templates selection in non-overlapping template matching test [J]. Electronics Letters, 2016, 52(18): 1533-1535

[54] Sýs M, Říha Z, Matyáš V. Algorithm 970: Optimizing the NIST statistical test suite and the berlekamp-massey algorithm [J]. ACM Trans on Mathematical Software, 2017, 43(3): 1-11

[55] Huang J, Lai X. Measuring random tests by conditional entropy and optimal execution order [G] //LNCS 6802: Trusted Systems 2010. Berlin: Springer, 2010: 148-159

[56] Killmann W, Schindler W. A design for a physical RNG with robust entropy estimators [G] //LNCS 5154: Cryptographic Hardware and Embedded Systems CHES 2008. Berlin: Springer, 2008: 146-163

[57] Baudet M, Lubicz D, Micolod J, et al. On the security of oscillator-based random number generators [J]. Journal of Cryptology, 2011, 24(2): 398-425

[58] Hagerty P, Draper T. Entropy bounds and statistical tests [C] //Proc of NIST Random Bit Generation Workshop. Gaithersburg: NIST, 2012: 1319-1327

[59] Kelsey J, McKay K A, Turan M S. Predictive models for min-entropy estimation [G] //LNCS 9293: Cryptographic Hardware and Embedded Systems CHES 2015. Berlin: Springer, 2015: 373-392

[60] Zhu S, Ma Y, Chen T, et al. Analysis and improvement of entropy estimators in NIST SP 800-90B for Non-IID entropy sources [J]. IACR Trans on Symmetric Cryptology, 2017, 2017(3): 151-168

[61] Schindler W. Efficient online tests for true random number generators [G] //LNCS 2162: Cryptographic Hardware and Embedded Systems CHES 2001. Berlin: Springer, 2001: 103-117

[62] Bucci M, Luzzi R. Design of testable random bit generators [G] //LNCS 3659: Cryptographic Hardware and Embedded Systems CHES 2005. Berlin: Springer, 2005: 147-156

[63] Fischer V, Lubicz D. Embedded evaluation of randomness in oscillator based elementary TRNG [G] //LNCS 8731: Cryptographic Hardware and Embedded Systems CHES 2014. Berlin: Springer, 2014: 527-543

[64] Haddad P, Teglia Y, Bernard F, et al. On the assumption of mutual independence of jitter realizations in P-TRNG stochastic models [C] //Proc of the conf on Design, Automation & Test in Europe. Belgium: European Design and Automation Association, 2014: 1-6

[65] Lubicz D, Bochard N. Towards an oscillator based TRNG with a certified entropy rate [J]. IEEE Trans on Computers, 2015, 64(4): 1191-1200

[66] Yang B, Rozic V, Grujic M, et al. On-chip jitter measurement for true random number generators [C] //Proc of 2017 Asian Hardware Oriented Security and Trust Symposium (AsianHOST). Piscataway, NJ: IEEE, 2017: 91-96

马　原

副研究员，主要研究方向为随机数发生器设计与检测、密码算法高速实现.

yma@is. ac. cn

陈天宇

助理研究员，主要研究方向为随机数发生器设计与检测.

tychen@is. ac. cn

吴鑫莹

博士研究生，主要研究方向为随机数发生器设计与检测.

wuxinying@is. ac. cn

杨　静

博士研究生，主要研究方向为随机数发生器设计与评估.

yangjing@is. ac. cn

林璟锵

研究员，主要研究方向为应用密码学、数据安全与隐私、网络与系统安全.

linjq@is. ac. cn

荆继武

研究员，主要研究方向为网络空间安全、身份管理与网络信任技术、系统安全理论与技术.

jing@is. ac. cn