素数与偏见:对抗性下的素数检验条件

Martin R. Albrecht1, Jake Massimo1, Kenneth G. Paterson1, Juraj Somorovsky2

[1] 伦敦大学皇家霍洛威学院

[2] 德国波鸿鲁尔大学

martin.albrecht@rhul.ac.uk, jake.massimo.2015@rhul.ac.uk, kenny.paterson@rhul.ac.uk, juraj.somorovsky@rub.de

摘要这项工作提供了对抗性条件下素数测试的系统分析，在对抗性条件下，被测试素数的数字不是随机生成的，而是由可能的恶意方提供的。这种情况可能出现在服务器向对等方提供 Diffie-Hellman 参数的安全消息协议中，或者在像 TLS 这样的安全通信协议中，开发人员可以插入这样的数字，以便稍后被动地监视客户端-服务器数据。我们研究了广泛的加密库，并评估了它们在这种对抗性环境中的性能。作为我们发现的例子，我们能够通过 OpenSSL 在其默认配置中的素数测试来构建 2048 位复合材料，这些复合材料被声明为素数，概率为 1/16;所宣称的性能为 2^{-80}。我们还可以构造 1024 位的复合材料，当配置为推荐的最小轮数时，总是通过 GNU GMP 中的质数测试例程。并且，对于一些库(Cryptlib, LibTomCrypt, JavaScript Big number, WolfSSL)，我们可以构建总是通过提供的素数测试的复合材料。我们探索了应用程序中这些安全故障的含义，重点是恶意 Diffie-Hellman 参数的构造。我们表明，除非进行仔细的素数测试，否则敌手可以提供表面上看起来安全的参数(p, q, g)，但其中由 g 生成的 q 阶子群中的离散对数问题很容易。最后，我们为用户和开发人员提出了建议。特别是，我们推广了 Baillie-PSW 素性测试，它既高效，而且被推测为即使在对抗性设置中，对高达几千位的数字也具有鲁棒性。

1 介绍

许多密码基元依赖于素数，RSA 是最著名的例子。然而，即使在不依赖于将整数分解为素数因子的困难的构造中，素数也经常被用于防止敌手应用分而求之的方法(例如在 pohligi - hellman 算法或在 Lim-Lee 小子群攻击[VAS+17]中)或防止存在诸如零因子之类的退化情况(这可能使安全性证明复杂化或减少输出熵)。

在这些密码基元的实例化中获得质数的一种方法是在任何需要它们的设备上产生所需的这些数字。这是通过对随机整数进行采样并检查质数来完成的。这个过程的计算量可能会大到令人望而却步的地步。产生质数的高成本导致实现寻求降低这种成本的方法，正如[NSS+17]中所展示的那样，这些性能改进可能会导致毁灭性的攻击。

如果所需的质数是公开的，那么另一种方法是可能的:(低功耗)从服务器或标准中为设备提供质数。例如，流行的 Telegram messenger [LLC18]使用服务器提供的 Diffie-Hellman (DH)参数在对等体之间建立端到端的加密。如果对等体不验证所提供的 DH 参数的正确性，[3] 则 Telegram 服务器可以提供具有复合组序的恶意 DH 参数，从而被动地获得所建立的秘密。作为一个具体的和

[3] 我们强调它们在默认实现中执行验证。

与此密切相关的例子是 Bleichenbacher [Ble05]，他展示了 GNU Crypto 1.1.0 中 Miller-Rabin 素性测试对一组小而固定的基的依赖如何被利用来欺骗 GNU Crypto 接受恶意的 DH 参数。特别是，这导致了对 SRP 的 GNU Crypto 实现的攻击，使用户的密码可以被恢复。

另一个例子是传输层安全协议[DR08]，它可以在握手协议中使用 Diffie-Hellman 密钥交换来建立主密钥。DH 参数由 TLS 服务器生成，并在每次 TLS 握手时发送给客户端。[4] 很明显，由于 TLS 服务器提供商知道已建立的主密钥，因此向客户端发送恶意的 DH 参数并没有获得任何优势。然而，我们可以考虑一个对抗性开发人员，他实现了一个恶意服务器，通过后门 DH 参数生成，cf. [Won16,FGHT17]。如果这些参数被 TLS 客户端接受并用于 DH 密钥交换，那么被动攻击者就可以观察流量并获得主密钥。在这里，仍然通过可信工具测试的弱 DH 参数提供了一种合理的可推诿感。此外，如果一个应用程序只是默默地拒绝了不好的参数，那么任何对策都可以通过反复发送恶意的参数集来克服，这些参数集具有愚弄这些对策的合理概率，直到目标客户端接受它们。

近年来，我们已经在加密实现中看到了几个后门。例如，NIST 标准化了双 EC 伪随机数发生器(PRNG)，如果敌手可以选择一个离散对数已知的生成器点 Q，并收集足够的 PRNG 输出[CNE+14]，则允许敌手预测生成的随机值。2016 年，Juniper 已经实现了这种 PRNG，使得敌手能够被动地解密 VPN 会话[CMG+16]。

一个值得注意的涉及合数的潜在后门的例子是命令行数据传输实用程序 socat 推送的安全咨询[Rie16]，它受到渗透测试等安全专业人员的欢迎。在那里，DH 素数 p 参数被替换为新的 2048 位值，因为“硬编码的 1024 位 DH p 参数不是素数”。该建议继续指出，“由于没有迹象表明这些参数是如何选择的，因此不能排除存在一个陷门，使窃听者有可能从使用它们的密钥交换中恢复共享密钥”，这突出了这种攻击模型在现实世界中的应用。同样，RFC5114 [LK08]的第 23 组给出的用于 DH 密钥交换的素数群参数 p 被发现部分易受小的子群攻击[VAS+17]。看起来，代码审查和严格质数测试的可用性(例如，在数学软件包中，cf.附录 L)对代码或标准中的恶意参数集施加了很高的可检测率，但正如这些例子所强调的，这样的集在实践中仍然会发生。

考虑到这些事件，我们可以假设一个有动机的对手能够实现软件服务于恶意生成的质数和/或 DH 参数。因此，需要依赖第三方质数来执行质数测试的加密应用程序。事实上，许多密码库都包含素数测试功能，因此看起来这个需求很容易满足。然而，这些测试的主要应用是在素数生成期间检查本地生成的、随机输入的素数(或者更准确地说，是组合性)。因此，很自然地要问这些库是否对恶意输入具有鲁棒性，即设计用于欺骗库以接受合数为质数的输入。我们将这种设置称为对抗性条件下的质数测试。

1.1 素性测试概述

使用最广泛的素性检验之一是 Miller-Rabin [Mil75,Rab80]检验。Miller-Rabin 基于重复平方的模幂运算，是一种高效的多项式时间算法，其复杂度为 $O(t \log^3 n)$，其中 t 为所执行的试验次数。然而，由于其概率性，众所周知，t-trial Miller-Rabin 检验只有在声明给定的复合 t 时才准确

以概率至少为 $1-(1/4)$的复合数。Arnault [Arn95]、Pomerance [PSW80]和 Narayanan [Nar14]都探索了产生米勒-拉宾伪质数的方法，即:

[4] 直至协议的 1.2 版(含)。

表 1。我们对密码库的分析结果。这显示了 Miller-Rabin 使用的轮数是如何确定的，是否实施了 Baillie-PSW 测试，记录的素数测试的失败率(即它错误地将一个合数声明为素数的概率)，以及我们对合数输入实现的最高失败率。

MR 测试的库轮数	Baillie-PSW 吗？	记录失败率	我们的最高失败率
OpenSSL 1.1.1-pre6 默认的字节大小	没有	$< 2^{-80}$	1/16
GNU GMP 6.1.2 用户自定义 t .	没有	$(1/4)^t$	t≤15 时 100%
GNU Mini-GMP 6.1.2 用户自定义 t	没有	$(1/4)^t$	t≤101 时 100%
Java 10 自定义 t	是(≥100 位)	$< (1/2)^t$	0%≥100 位
JSBN 1.4 自定义 t	没有	$< (1/2)^t$	100%
Libgcrypt 1.8.2 用户自定义 t	没有	没有给	1/1024[†]
Cryptlib 3.4.4 用户自定义 t≤100	没有	没有给	100%
Apple corecrypto 自定义 t≤256	没有	$(1/4)^t$	100%
Apple CommonCrypto 16	没有	$< 2^{-32}$	100%
LibTomMath 1.0.1 自定义 t≤256	没有	$(1/4)^t$	100%
LibTomCrypt 1.18.1 自定义 t≤256	没有	$(1/4)^t$	100%
WolfSSL 3.13.0 自定义 t≤256	没有	$(1/4)^t$	100%
Bouncy Castle c# 1.8.2 用户自定义 t	没有	$(1/4)^t$	$(1/4)^t$
博坦 2.6.0 用户自定义 t	没有	≤(1/2)	$(1/4)^t$
Crypto++ 7.0 2 或 12	是的	没有给	0%
GoLang 1.10.3 用户自定义 t	是的	$< (1/4)^t$	0%
GoLang 1.8 之前自定义 t	没有	$< (1/4)^t$	t≤13 时 100%

[†] 当调用 check prime 函数而不是 gcry prime check(或调用 gcry prime check 在 1.3.0 之前的版本中)。

合成数，在米勒-拉宾测试时，达到 t 的(1/4)的最高概率被错误地归类为“可能是质数”。

另一种常见的选择是 Lucas test [BW80]，其更严格的变体是强 Lucas probable prime test。与米勒-拉宾检验类似，强卢卡斯检验的 t 次试验将宣布给定的合数为合数，其概率至少为 $1-(4/15)^t$，是质数，其概率最多为(4/15)[Arn97]。与米勒-拉宾检验一样，有 t 已知构造强 Lucas 伪素数的方法[Arn95]。

Lucas 检验(强或标准)可以与单一的 Miller-Rabin 检验(base 2)相结合，形成所谓的 Baillie-PSW 检验[Pom84]。由于运行时间稍长，这种测试通常只在数学软件包中使用，在密码库中很少见到。与 Miller-Rabin 和 Lucas 单独执行时的测试不同，Baillie-PSW 测试没有已知的伪质数(然而也没有证据表明它们不存在)。

显然，在进行 Miller-Rabin 或 Lucas 测试时，参数 t(试验次数)的选择至关重要。许多密码学库，例如 OpenSSL [OP18b]，使用的测试参数源自[DLP93]，这些参数在《应用密码学手册》[MVOV96]中得到了普及。这些给出了在测试随机输入 n 时错误率小于 2^{-80} 所需要的米勒-拉宾迭代次数。[DLP93]的一个主要结果是，如果 n 是随机选择的 b 位奇数，那么 t 轮独立的米勒-拉宾测试来给出错误概率:

$$P(X|Y_t) < b^{3/2}2^t t^{-1/2}4^{2-\sqrt{tb}} \quad \text{for } 3 \le t \le b/9 \text{ and } b \ge 21,$$

其中 X 表示 n 是复合的事件，y_t 表示 t 轮 Miller-Rabin 将 n 声明为质数的事件。这一界限使计算所需的最小值 t 获得 $P(X|Y_t)^{-80} \leqslant 2$ 的比特长度范围 b;见表 2。

然而，这些误差估计是用于 Miller-Rabin 对随机生成的 n 进行质数测试。在对抗性设置中，我们实际上关心的是 Miller-Rabin 的 t 次试验(或其他一些测试)声明给定的 n 为质数的概率，前提是它是合成的。这个概率

3.

与位大小无关，如果在 Miller-Rabin 测试中使用随机基，则最多为$(1/4)^t$。类似的备注适用于 Lucas 测试的两个变体。

加密机制 BSI tr - 02101 - 1 [fSidI17]等技术指导文件和公开公布的标准，如数字签名标准(DSS) FIPS 186-4 C.3.2 [Pub13]和素数生成国际标准 ISO/IEC 18032 [Sta05]，对参数选择提供了正式指导和建议。BSI TR-02102-1 建议，在最坏情况下，必须进行 50 轮随机基选择 Miller-Rabin，在平均情况下，如 ISO/IEC 18032，参考上述 Damg˙ [DLP93]提出的方法和《应用密码学手册》[MVOV96]。BSI tr - 02101 - 1 还参考了 FIPS 184 - 4 给出的指导，该指导对 DSA 参数生成给出了更为保守的迭代轮(1024 位 t = 40, 2048 位 n t = 56)，并给出了详细的论证。FIPS 186-4 主张使用额外的 Lucas 素数检验(参见 2.3 节)，并在其附录 F.2 中详细阐述了上述两种条件概率之间的区别。标准 ISO/IEC 18032 正确地规定了最坏情况下的失败概率确实是$(1/4)^t$，但并没有把这两种条件概率的区别做得那么清楚。

许多库，例如 GNU GMP [Gt18]，提供了质数测试函数，可以部署在需要任意精度运算的数学软件包等应用程序中。这些函数通常要求用户选择所执行的质数测试的“确定性”或准确性。由于这些参数通常对终端用户隐藏，这将迫使使用库的应用程序的开发人员承担选择合适参数的责任。然后，从标准中过滤出来的唯一结果指导可以在库的文档中找到，这些文档通常是简短和非正式的。

1.2 贡献和大纲

本文研究了密码库和数学软件包中素数测试的实现景观，并衡量了在对抗式环境中实现鲁棒素数测试的广泛失败的安全影响。

我们在第 2 节中回顾了素性测试。然后在第三节中，我们回顾了构造伪素数的已知技术，并考虑到我们的目标应用对其进行扩展。在第 4 节中，我们接着调查了密码库和数学软件中的素性测试，评估了它们在对抗性环境中的性能。我们提出技术来击败他们的测试，我们可以。总的来说，我们的发现是，大多数库在对抗性环境中并不健壮。我们在这个方向上的主要结果总结在表 1 中。

作为我们结果的一个重点，我们发现 OpenSSL 的默认素数测试例程将声明加密大小的某些组合 n 为素数，概率为 1/16，而记录的失败率为 2^{-80}。这是由于 OpenSSL 依赖于表 2 来计算所需的 Miller-Rabin 测试轮数，并且这个数字随着 n 的增大而减小。作为另一个重点，我们构建了一个 1024 位的复合材料，保证被 GNU GMP 库[Gt18]声明为素数，用于任何不超过并包括 15 轮测试(GMP 推荐的最小值)的任何东西。这是 GNU GMP 初始化其 PRNG 到静态状态的结果，因此在其 Miller-Rabin 测试中只依赖于 n，被测试的数字。我们还展示了如何通过从固定的素数列表中随机抽样进行碱基选择，如 Apple 的 corecrypto 库、Cryptlib、LibTomCrypt、JavaScript Big Number (JSBN)和 WolfSSL，可以被颠覆:我们构建了加密大小的复合 n，无论执行了多少轮测试，这些库都保证将其声明为素数。

我们继续检查我们的发现对应用程序的影响，重点是 DH 参数测试。好消息是 OpenSSL 没有受到影响，因为它坚持在 DH 中使用安全素数;也就是说，它要求 DH 参数(p, q, g)，其中 q = (p − 1)/2 和 p, q 都要进行质数测试。在这种情况下，我们目前的技术无法产生恶意参数。另一方面，当允许更自由的参数选择时，就像 Bouncy 中的情况一样

Castle 和 Botan，我们能够构建恶意的 DH 参数集，这些参数集通过了库的测试，但对于它们来说，g 生成的子群中的离散对数问题很容易。

最后，我们在第 6 节中讨论了提高对抗性环境中素性测试鲁棒性的途径。

1.3 披露和缓解

我们报告了我们的发现，并根据我们对 OpenSSL、GMP、Apple、JSBN、Cryptlib、LibTomMath、LibTomCrypt、WolfSSL、Bouncy Castle 和 Botan 的分析结果提出了合适的缓解措施。我们对这些讨论的结果进行了简短的回顾。

-当我们联系到 OpenSSL 开发人员时，他们正在修改他们的质数测试代码，使其成为 fips 投诉[OP18a]。这些变化包括在 OpenSSL 版本 1.0.2p 和 1.1.0i(2018 年 8 月 14 日发布)中。[5] 然而，这些变化并没有考虑到我们论文所关注的对抗性场景，在该场景下 OpenSSL 的默认设置仍然很弱。

- Apple 将他们的 corecrypto 库从 Miller-Rabin 测试中使用固定基改为使用伪随机基。该漏洞被分配为 CVE-2018-4398。

LibTomMath 和 LibTomCrypt 的开发人员正在调整其库中的质数测试函数。他们计划移除固定基础的 Miller-Rabin 测试，并根据我们的建议用 Baillie-PSW 测试替换该函数[Lib18]。

-为了响应我们的发现，WolfSSL 在即将发布的版本[Wol18a]中对他们的质数测试进行了一些修改。这包括现在使用伪随机基执行 Miller-Rabin，不覆盖用户选择的迭代，以及增加 DH 和 DSA 检查函数中对质数参数执行的轮数。

- Bouncy Castle 也根据我们的发现做出了改变，在即将发布的 1.8.3 版本中，删除了 DH 验证功能并替换为白名单方法。他们也在考虑根据我们的建议在未来的版本中执行 Baillie-PSW。

-博坦版本 2.7.0 [Llo18b]增加了 DH 验证中 Miller-Rabin 的轮次，并根据我们的建议，包括增加 Lucas 测试来执行 Baillie-PSW。

- GNU GMP、Mini-GMP 和 Cryptlib 都保持不变，但 Cryptlib 的作者指出了一个代码注释，指出了它们的素性测试的局限性。

-我们没有收到来自 JSBN 的回应。

2 质数测试的背景

质数测试是一种用来判断给定数字是否为质数的算法。这些质数测试有两种不同的形式;确定性和概率性。确定性素数测试算法能结论性地证明一个数是素数，但往往速度较慢，在实践中应用并不广泛。一个著名的例子是 AKS 测试[AKS04]。除了在某些数学软件中出现的情况外，我们在本文中不会进一步讨论此类测试。

概率素数测试利用了所有素数必须满足的算术条件，并针对感兴趣的数字 n 测试这些条件。如果条件不成立，我们就知道 n 必须是合数。然而，如果条件成立，我们可能只能推断 n 可能是质数，因为一些合数也可能通过测试。通过重复测试，在 n 通过了某些 t 次测试的条件下，它是合数的概率可以使其对于密码学应用来说足够小。一个典型的目标概率是 2^{-80},cf. [MVOV96, 4.49]。这里有一个关键的考虑因素是 n 是否是反向生成的，因为在两种情况下，概率上可以推断出的界限可能是完全不同的;下文将对此进行更多介绍。

我们现在讨论三个广泛使用的检验:费马检验、米勒-拉宾检验和卢卡斯检验。

[5] 见 https://www.openssl.org/news/changelog.html。

[6] 参见 https://nvd.nist.gov/vuln/detail/CVE-2018-4398 和 https://support.apple.com/en-gb/HT201222

2.1 费马测验

费马素数检验是基于下面的定理。

定理 1(费马小定理)。如果 p 是质数，a 不能被 p 整除，那么

$$A^{p-1} \equiv 1 \pmod{p}。$$

要测试 n 是否为素数，只需选择一个碱基 a 并计算 $a^{n-1} \pmod{n}$。如果 $a^{n-1} 6 \equiv 1 \pmod{n}$，那么我们可以确定 n 是合数。如果在测试了多种基 a_i 之后，我们发现它们都满足 $a_i^{n-1} \equiv 1 \pmod{n}$，那么我们可以得出结论，n 很可能是素数。

众所周知，对于所有不能被 n 整除的整数 a，存在满足 $^{n-1}a \equiv 1 \pmod{n}$的合数。这些数完全挫败了费马检验，被称为卡迈克尔数。这些在续集中将会有关联。下面的结果是构建卡迈克尔数的基础。

定理 2(科塞尔特准则)。一个正的合数 n 当且仅当 n 是无平方数时是一个 carmic - michael 数，$p-1 \mid n-1$ 对于 n 的所有质因数 p。

2.2 米勒-拉宾检验

米勒-拉宾[Mil75,Rab80]素数检验是基于一个素数模的单位根不存在非平凡根的事实。设 n > 1 是要检验的奇整数，其中 d 为奇数时，记为 $n = 2^{ed} + 1$。如果 n 是质数，那么对于任意 $1 \leqslant a < n$ 的整数 a，我们有:

$$A^D \equiv -1 \quad \text{Mod } n \text{ 或 } a^{2^d} \equiv -1 \text{ Mod } n \text{ 对于 } 0 \leqslant I < e。$$

Miller-Rabin 测试则包括检查上述条件，如果两个条件中有一个成立，则声明一个数(可能)为素数，如果两个条件都不成立，则声明该数为合数。如果一个条件成立，那么我们说 n 是以 a 为基数的伪质数，或者 a 是 n 的合数的非见证数(因为 n 可能是合数，但 a 没有证明这个事实)。

对于合数 n，让 S(n)表示非见证数 $a \in [1, n-1]$。S(n)的上界由[Mon80,Rab80]的结果给出:

定理 3 (moner - rabin 界)。设 n 6= 9 为奇合数。然后

$$S(n) \leq \frac{\varphi(n)}{4}$$

其中 ϕ 表示欧拉 totient 函数。

这个界限对于确定反向生成的 n 通过 Miller-Rabin 检验的概率至关重要。因为对于大 n，我们有 $\phi(n) \approx n$，这表明对于概率大于(1/4)的 t 个随机基，没有复合 n 可以通过 Miller-Rabin 检验。因此，要达到 2^{-80} 的目标概率，需要 $t \geqslant 40$。测试通常使用(a)一组固定的基础(例如 JSBN)或(b)随机选择的基础(例如 OpenSSL)来实现。当然，(1/4)边界只在随机选择的基的情况下成立。

2.3 卢卡斯检验

Lucas 素数检验[BW80]使用 Lucas 序列，定义如下:

定义 1 (Lucas 序列[Arn97])。设 P 和 Q 为整数，$D = P^2 - 4Q$。那么 Lucas 序列(U_k)和(V_k) $(k \geqslant 0)$递归定义为:

$$U_{k+2} = PU_{k+1} - QU_k \quad where,$$
$$V_{k+2} = PV_{k+1} - QV_k$$
$$U_0 = 0, U_1 = 1,$$
$$V_0 = 2, V_1 = P.$$

吗?吗?Lucas 或然素数检验则依赖于以下定理(其中 x 表示 p

Legendre 符号，如果 x 是对 p 取模的平方，值为 1，否则值为-1):

定理 4 ([CP06])。设 P、Q、D 和 Lucas 序列(U_k)、(V_k)定义如上。如果 p 是素数，且 gcd(p, 2QD) = 1，则

$$U_{p-\left(\frac{x}{p}\right)} \equiv 0 \pmod{p}. \tag{1}$$

Lucas 或然素数测试反复测试不同对(P, Q)的性质(1)，这就引出了关于这样一对的 Lucas 伪素数的概念。

定义 2 (Lucas 伪素数)。设 n 是一个合数，使得 gcd(n, 2QD) = 1。如果 $U_{n-(x)} \equiv 0 \pmod{n}$，那么 n 对于参数(P, Q)被称为 Lucas 伪素数。

现在我们可以通过下面的定理引入关于参数(P, Q)的强 Lucas 或然素数和强 Lucas 伪素数的概念。

吗？

定理 5 ([Arn97])。设 p 是一个不除 2QD 的质数。设 p−

吗?p =2kqwith q

奇数。则满足下列条件之一:

$$p \mid U_q \quad or \quad \exists i \ such \ that \ 0 \leq i < k \ and \ p \mid V_{2^i q}.$$

强 Lucas 可能素数测试反复测试不同对(P, Q)的性质(2)，这导致了关于参数(P, Q)的强 Lucas 伪素数的定义如下。

定义 3?(强 Lucas 伪素数)。设 n 是一个合数，使得 gcd(n, 2QD) = 1。设 n− Dn=2kq, q 为奇数。假设:

n |Uq 或∃ i 使 0≤i < k 和 n |V2q。那么 n 对于参数(P, Q)

称为强 Lucas 伪素数。

强 Lucas 伪素数也是 Lucas 伪素数(对于相同的(P, Q)对)，但反过来不一定成立。因此，强版本的测试被视为更严格的选择。

备注 1。Lucas pseudprime 和 strong Lucas pseudprime 测试也分别被称为 Lucas-Selfridge 测试和 strong Lucas-Selfridge 测试，特别是在使用 Selfridge 的参数 P = 1, Q =− 1 时。

类似于 Miller-Rabin 素数测试的伪素数莫内-拉宾定理，Arnault [Arn97]表明对于一个整数 D 和 na 的复合物，gcd(D, n) = 1 和 n 6= 9, 0≤P, Q < n, gcd(Q, n) = 1, $P^2 - 4Q \equiv D \pmod{n}$使得 n 是关于(P, Q)的强 Lucas 伪素数最多为 4n/15。有一个例外结果对某些形式的孪生质数(我们这里省略细节),但 Arnualt 继续证明,即使这些特殊形式的双胞胎' n 最多 n / 2 双(P, Q), n 是一个强大的卢卡斯 pseudoprime (P, Q)。由此,我们可以推断 t 的应用强卢卡斯测试可能将宣布一个复合 n '概率最多(4/15)。

2.4 Baillie-PSW

Baillie-PSW [Pom84]检验是将以 2 为基数的单一 Miller- Rabin 检验与 Lucas 或强 Lucas 伪素数检验相结合而形成的概率素数检验。这个测试的想法是，两个成分是“正交的”，因此数字 n 将通过两个部分的可能性非常小。事实上，目前还没有已知的复合数 n 通过 Baillie-PSW 检验。吉尔[Gil13]

确认没有小于 2^{64} 的 Baillie-PSW 伪质数。PRIMO [Mar16]是一个基于椭圆曲线的素数证明程序，它使用 Baillie-PSW 检验来检查所有中间的可能素数。如果这些值中的任何一个确实是合成的，那么最终的证明必然会失败。由于在使用过程中从未发生过这种情况，PRIMO 的作者 Martin 估计[Wei18]，不存在小于大约 10000 位的 Baillie-PSW 伪质数。这一经验证据表明，在 Diffie-Hellman 和 RSA 中使用的密码大小的数字不太可能是 Baillie-PSW 伪素数。然而，Pomerance 在[Pom84]中给出了一个启发式论证，即实际上存在无穷多个 Baillie-PSW 伪素数。单个实例的构造是数论中一个重要的开放问题。

3 构造伪素数

在本节中，我们将回顾 Miller-Rabin 和 Lucas 测试中已知的伪素数构造方法。我们还提供了这些方法的变体。我们将在下一节中使用这一节的结果，在那里我们研究对抗环境中用于素性测试的密码库的鲁棒性。

3.1 米勒-拉宾伪素数

给定 n 的因式分解[CP06]，可以计算出任意合数 n 的非目击者的确切数量 S(n)。生成具有大量非目击者的合数 n 就没那么简单了。在实证工作中，Pomerance 等人[PSW80]表明，许多通过米勒-拉宾素性检验的合数具有 $n = (k + 1)(rk + 1)$的形式，其中 r很小，$k + 1$ 和 $rk + 1$ 都是素数。最近，H ¨ oglund [H ¨ o16]和 nice [Nic16]使用 GNU GMP 中实现的 Miller-Rabin 质数测试来测试这种形式的随机生成的数字对于不同的 r 值和不同大小的 k。他们的结果支持了[PSW80]所做的声明。

我们现在考虑生产复合材料的现有方法，它们有许多非见证，对于两种形式的米勒-拉宾试验：首先是随机选择的碱基，其次是使用固定的碱基集。

3.1.1 随机碱基。对于随机基，我们感兴趣的是构造具有大量非目击者的复合 n，即对于它 S(n)很大。这样的数字将通过米勒-拉宾检验，概率为 S(n)/n 每次试验;当然，根据莫尼尔-拉宾定理，这个概率的边界是 $\phi(n)/4n \approx 1/4$，但我们感兴趣的是我们能得到多接近这个边界的值。我们依赖于以下几点:

定理 6 ([Mon80])。考虑一个具有 m 个不同素数因子 p1 的奇复合整数 n，…,点。假设 $n = 2e \cdot d + 1$，其中 d 为奇数。又假设 $n = Q m_{pti}$ 其中每个 $i=1$ i pi 可以表示为 $2e \cdot di + 1$，每个 di 为奇数。[i] 然后

$$S(n) = \prod_{i=1}^{m} \gcd(d, d_i) \cdot \left(\frac{2^{\min(e_i) \cdot m} - 1}{2^m - 1} + 1 \right). \tag{3}$$

注意这个定理中的界是如何不依赖于指数 t_i 的，这表明无平方数将具有相对较大的 S(n)。还要注意对项 $\gcd(d, d_i)$的依赖，这表明要为 S(n)实现较大的值，就必须确保每个质因数 p_i _的奇数部分与 n 的奇数部分具有较大的 gcd。作为这个定理的一个简单推论，我们得到:

推论 1 ([Mon80])。设 x是一个奇整数，使 $2x + 1$ 和 $4x + 1$ 都是素数。那么 $n = (2x + 1)(4x + 1)$ 有 $\phi(n) = 8^2 x$，达到莫尼尔-拉宾界，即满足

$$S(n) = \varphi(n)/4.$$

这个推论的证明很容易遵循的观察，我们可以取 m = 2 和 d= d_1 = d_2 = x 在前面的定理。Narayanan [Nar14]还表明，如果 n 是形式为 ppp_1 的卡迈克尔数，$_2$ 其中 $_3$ 每个 pi 都是一个独特 $_i$ 的素数，p≡3 (mod 4)，那么 $_i$ S(n)达到莫内-拉宾界。他还给出了进一步的结果，表明 n 的这两种形式是唯一达到 Monier-Rabin 界限的形式，所有其他的 n 都满足 S(n)≤ϕ (n)/6。

3.1.2 固定基底。Miller-Rabin 素性测试的一些实现是从固定列表(通常是质数)中选择基，而不是随机选择。例如，直到 2010 年，PyCrypto 2.1.0 (2009) [Lit09]素数测试 isPrime()执行了 7 轮 Miller-Rabin，使用前 7 个素数作为基数，而 LibTomMath 则从硬编码的素数列表中选择前 t 个条目作为基数。

Arnault [Arn95]提出了一种生成合数 n = pp_1 • $_2$ • • p 的方法，对于任意固定的素数基 $_h$ 集 a = {a_{21}a，…}。$_t$ 我们在附录 A 中给出了阿尔诺方法的概述和例子。

由于固定基础的米勒-拉宾测试在实现中相对不常见，因此看起来阿尔诺的方法可能不会很有用。然而，我们将看到，当实现从大量固定的可能性列表中随机选择基时，这种方法特别有用。例如，一个实现可能从 1000 以下的质数列表中随机选择质数基;由于阿尔诺的方法可扩展性好(我们只需要同时解决与 CRT 更多的同余)，我们可以使用这种方法来产生一个复合 n，这样所有低于 1000 的质数都是非 n 的目击者。我们将在 4.3、4.5、4.7、4.9、4.10 和 4.11 节中看到这种方法对不同库的应用。

3.1.3 混合技术。上述方法产生的复合材料实际上总是卡迈克尔数。从 3.1.1 节我们知道，如果 n 是一个有 3 个不同质数因子都同余于 3 (mod 4)的卡迈克尔数，那么 n 有最大的非证人数 ϕ (n)/4。我们可以在 Arnault 的方法中设置 h = 3，并稍微调整它以确保，除了用指定的非目击者集合 a 产生 n 外，它还产生一个满足莫尼尔-拉宾界的 n，这样随机基米勒-拉宾检验也将以最大概率通过。这个微调非常简单:我们确保 2∈A;这迫使 p_1 ≡3 或 5 (mod 8);然后我们选择 p_1 ≡3 (mod 8)，使 p_1 ≡3 (mod 4)。阿尔诺的方法使 p_i = k_i(p_1 − 1)+ 1，其中 k_i are 对 A 的所有元素都是质数。由于 2∈A,$_i$ k 必须都是奇数;很容易看出，这也迫使 p_i ≡3(mod 4)。

我们将在 4.6 节中给出这种技术的一个应用。

3.1.4 复合固定基的扩展。Arnault [Arn95]的方法(如所示)仅适用于素基，不适用于复合基。虽然不太常见，但有些实现在他们的 Miller-Rabin 测试中同时使用了素数基和复合基。通过设置 n≡3 (mod 4)，我们知道当对 d 奇数写成 n =$_{2e}$ • d + 1 时 e = 1。在这种情况下，通过米勒-拉宾检验的条件简单地变成了 $a^{(n-1)/2}$≡±1 (mod n)，因此，如果 n≡3 (mod 4)对某组碱基{a_{21}, a，⋯a_t}，那么 n 对于任何以乘积 b =a_{e11} · $_{ae22}$ · · · · · a_{ett}(mod$_n$)(对于任何一组指数 ei∈Z)形式出现的碱基 b 来说，也是伪素数。因此我们 $_1$ 可以构造一个对于任意一组碱基{b，⋯， b}(其中任何数 $_t$ 都可以是合数)通过使用第 3.1.3 节中描述的混合方法，但该方法中的集合 A 是 b 中出现的素数因子的完全集合。注意，在这种方法 $_i$ 中，n 的形式为 n = pppwhere each p≡3 (mod 4)，$_1$ 因此 $_2$ 我们 $_3$ 有 n≡3 (mod$_i$ 4)根据需要。此外，由于 n 的形式，以这种方式生成的复合数也将满足莫尼尔-拉宾界。

我们将在 4.3 节中给出这种技术的一个应用，我们研究了 Mini-GMP [Gt18]，它使用欧拉多项式来生成 Miller-Rabin 基。

3.2 Lucas 伪质数

和 Miller-Rabin 伪素数一样，Lucas 伪素数也与一些测试参数的选择有关。在整个工作中，我们遵循 Selfridge 的参数选择方法 A [BW80]，总结如下:

定义 4(塞尔弗里奇方法 A[BW80])。设 D 为序列的第一个元素

$$5, -7, 9, -11, 13, \ldots \text{ for which } \left(\frac{D}{n}\right) = -1. \text{ Then set } P = 1 \text{ and } Q = (1-D)/4.$$

研究这种特殊的参数设置方法有两个原因。首先是它是在执行 Baillie-PSW 素性检验[PSW80,BW80]中的 Lucas 部分时使用的参数选择。第二个是，这是我们研究的 Java [Cor18]和 Crypto++ [Dai18]库在实现 Lucas 测试时使用的方法。

带有这种参数选择的 Lucas 和 strong Lucas-probable prime tests 在文献中通常被称为 Lucas 和 strong Lucas- selfridge probable prime tests。关于这个参数选择的伪质数有很好的文献记载。OEIS 序列 A217120 [Bai13a]给出了其中的一个小列表，参考 Jacobsen [Jac15]汇编的 $10^{14} \approx 2^{47}$ 以下所有 Lucas 伪质数的表格。对于强 Lucas 伪素数，还有一个等价序列 A217255 [Bai13b]。任何关于某些参数集(P, Q)的强 Lucas 伪素数检验的伪素数，也是 Lucas 伪素数检验的伪素数。

Arnault [Arn95]还提出了一种可扩展的方法，将一组参数选择{(P1, Q1, D1)，(P2, Q2, D2)，…，(Pt, Qt, Dt)}，并返回形式为 n =p1p2・・・ph 的复合 n，对所有 i1≤ii≤t 的参数(P, Q, D)i 都是强 Lucas 伪素数。该方法类似于构造固定碱基的 Miller-Rabin 伪素数的方法，但在细节上有所不同。特别是，这两种构造方法差别很大，似乎很难推导出一种生成 n 的方法，同时对 Miller-Rabin 和 Lucas 测试都是伪素数。

3.2.1 Arnault [Arn95]对 Selfridge 's Method A 的特殊化对于 Selfridge 's Method A，我们知道，如果我们取一个这样 5 ? 的 n = - 1，那么对 n 进行单次测试

n

将执行参数 set (P, Q, D) =(1，－1,5)。接下来，我们将展示如何对 Arnault 的构造[Arn95]进行特殊化，以便它将产生复合材料 n，保证该复合材料 n 通过对该参数集的强 Lucas 测试被声明为素数。

按照 Arnault 的构造，我们考虑 n 的形式为 n =21ppp 其中 3 p= k(p1 +1)i－1 对于 i∈{2,3}，具有 kand kodd2 整数 3 。

我们首先注意 i 到，对于 Legendre 符号，p 必须满足某些条件(见[Arn95，引理 6.1 和 6.2]):

$$\left(\frac{D}{p_i}\right) = \left(\frac{Q}{p_i}\right) = -1 \qquad \text{对于所有使 } 1 \leqslant I \leqslant 3 \text{ 的 I。}$$

用我们的单参数集(P，Q, D) =(1，－1,5)，这就变成:

$$\left(\frac{-1}{p_i}\right) = \left(\frac{5}{p_i}\right) = -1 \qquad \text{对于使 } 1 \leqslant I \leqslant 3 \text{ 的所有 I。} \tag{4}$$

吗?吗?现在 －1 pi =－1 的流场占有率≡3 (mod 4)。由于 pi =ki(p1+1)－1 对于 i∈{2,3}，且 ki 为奇数，

那么很容易表明，如果 p?≡1 3(mod 4)，那么可以推导出 I =2, 3as 时 p≡i 3 (mod 4)

好。我们也有 (5 π =－1 一组 i 争斗争斗≡2 或 3(mod 5)。因此满足条件(4)

当 p1 ≡3 或 7 (mod 20)(按 CRT 计算)，且 ii ≥2 时 p≡2 或 3 (mod 5)。

在这一点上，我们必须选择 k2,3 k，并添加条件，以确保[Arn95，引理 6.1]中的系数确实是整数。这些条件很简单:

$$p_1 \equiv k_3^{-1} \pmod{k_2} \qquad \text{和} \qquad p_1 \equiv k_2^{-1} \pmod{k_3}.$$

我们选择固定 $p_1 \equiv 7 \pmod{20}$，并选择$(k_2, k_3) = (31,43)$。这就产生了质 1 数 p 必须满足的最终同余:$p_1 \equiv 6647 \pmod{26660}$。我们现在搜索一个满足这个同余的素 1 数 p，并且使得 p_2 and p_3 satisfaction $p_i = k_i(p_1 + 1) - 1$ for $i = 2,3$ 也是素数 $p_2 \equiv p_3 \equiv 2$ 或 3 $\pmod{5}$。

最小的解如下所示:

$$P_1 = 486527,\ P_2 = 15082367,\ P_3 = 20920703$$

这产生了一个 68 位的 n = 153515674455111174527，它确实通过了使用塞尔弗里奇方法 a 进行参数选择的强 Lucas 测试。当然，我们可以取任何满足上述条件的(p_{21}, p_3, p)(满足这些条件并不太繁重)，从这个意义上说，该方法可以很好地扩展到密码学上有趣的大小的数字 n。例如，附录 B 展示了使用上述过程生成的 2050 位的示例。

这种生成技术也是通用的，因为我们可以简单地根据特定测试使用的参数选择方法在我们的集合中包括额外的参数。这允许我们生成复合材料，这些复合材料被各种强 Lucas 测试声明为素数，代价是解决与 CRT 多几个同时同余的小成本。

4 个密码库和数学包

许多提供通用密码协议实现的密码库也提供了处理任意精度整数算术的工具包，包括素数测试。例如，这些函数将被用于测试 Diffie-Hellman 参数的素数。

本节提供了一个广泛且具有代表性的密码库(OpenSSL, GNU GMP 和 small -GMP, Java, JavaScript Big Number (JSBN)， Libgcrypt, Cryptlib, Apple corecrypto 和 CommonCrypto, LibTomMath, LibTomCrypt, WolfSSL, Bouncy Castle, Botan, crypto++和 GoLang)中素数测试的调查。对于每个库，我们首先描述它是如何实现素数测试的。然后，我们为每个特定的库定制一个可能被声明为素数的组合，并量化我们的组合通过库的素数测试的概率(从而使素数测试失败)。我们的发现总结在表 1 中。在整个过程中，我们将 Miller-Rabin 测试的轮数作为 t。

4.1 OpenSSL

OpenSSL 是使用最广泛的开源加密库和 TLS 实现。自始至终，我们考虑 OpenSSL 1.1.1-pre6 [OP18b]，尽管所研究的组件在各个版本中基本上是稳定的，并保持与早期版本(2000 年 9 月的 0.9.6 版本)相似。

分析。OpenSSL 中的质数测试位于 crypto 库中，该库还包含各种加密算法的实现。由加密库提供的服务被 SSL、TLS 和 S/MIME 的 OpenSSL 实现所使用，也被用于实现 SSH、OpenPGP 和其他加密标准。

在 OpenSSL BIGNUM 库中执行素数测试的函数是 BN 是素数 ex, BN 是在 BN prime.c 中找到的素数 fasttest ex。在 BN 中完成的质数测试算法的大部分是质数 fasttest ex，其中 t = Miller-Rabin 检查轮被执行，每轮都随机选择一个基数。checks 变量作为素数验证函数的参数提供。函数 BN is prime ex 简单地调用 BN is prime- fasttest ex 而不做任何试除法。我们生成的复合函数 n 的因子比 OpenSSL 执行的试除法中的因子大得多。这意味着，就我们的目的而言，调用任何一个函数的结果都是等效的。因此，我们将只关注 BN 是素数的最快测试 ex。

米勒-拉宾弹的数量。两个质数测试函数都允许用户确定米勒-拉宾执行的轮次。文档表明，如果用户将 checks 的值设置为变量 BN prime checks，则选择米勒-拉宾迭代次数 t，使得米勒-拉宾测试声明随机合数 n 为质数的概率小于 2^{-80}。然后，执行的轮数是基于被测试的数字 n 的比特大小 b。这两个值之间的关系如表 2 所示。这里的条目是基于应用密码学手册[MVOV96]的平均案例误差估计，该手册反过来引用了[DLP93]。

表 2。Miller-Rabin 的 t 轮由 OpenSSL 在测试 b 位整数时选择，检查= BN 质数检查。

b ≥	t	b ≥	t
1300	2	400	8
850	3	350	9
650	4	300	12
550	5	250	15
450	6	200	18
400	7	150	27

基础的选择。OpenSSL 以伪随机的方式选择它使用的米勒-拉宾基，通过使用 OpenSSL 的函数 BN rand range()并将可选标志设置为 PRIVATE。然后调用 bnrand 来生成一个 $1 \leq a < n$ 范围内的伪随机基数 a，使用一个密码学强的伪随机数生成器，从操作系统收集熵输入，cf. [Str16]了解 OpenSSL 随机数生成的细节。

Pseudoprimes。如第 1 节所述，[DLP93]的平均病例估计仅设计用于质数生成期间的测试数字。事实上，OpenSSL 在这种情况下正确地应用了质数测试，如上所述。然而，我们在文档中没有发现任何关于对抗性设置的警告。相反，它似乎是留给用户决定需要多少轮测试，如果他们设置检查= BN 素数检查，那么表 2 将规定应用多少轮。在这种情况下，我们可以通过使用 3.1.1 节中描述的方法产生合数来破坏 OpenSSL 的保证。也就是说，我们可以很容易地用 x 奇数和 2x + 1,4x + 1 质数构造出 n = (2x + 1)(4x + 1)形式的数字，并确保 n 将通过随机基米勒-拉宾测试，每次测试的概率大约为 1/4。例如，对于有 b = 2048 位的 n, OpenSSL 将应用 t = 2 个测试，我们有 1/16 的机会让我们的复合 n 欺骗 OpenSSL。

4.2 gnu GMP

GNU 多重精度算术库[Gt18]，简称 GNU GMP 或简称 GMP，是一个流行的开源任意精度整数库，广泛部署在数学软件包中。我们全程考虑最新版本 GMP 6.1.2。

分析。GMP 提供了自己的数据类型来处理称为 mpz t 的大整数。GMP 的素数测试在 mpz probab 素数 p(mpz t n, int reps)中实现。在输入 n 上，这个函数执行一些试划分，然后是基数 $210 = 2 \cdot 3 \cdot 5 \cdot 7$ 的固定基费马测试，最后是 t = rep 轮的米勒-拉宾;后者在 MPZ millerrabin 函数中实现。reps 的值由调用者选择。文档保证了一个合数将被识别为概率小于 $(1/4)^{reps}$ 的质数，并声明“rep 的合理值在 15 到 50 之间”。

基础的选择。GMP 使用伪随机数生成器(PRNG)来选择用于每个米勒-拉宾测试的碱基。PRNG 的状态在 mpz millerrabin 函数中通过调用 gmp randinit default(rstate)进行初始化，该函数使用了 Mersenne Twister 算法。然后这个初始的种子状态被用作 mpz urandomm(a, rstate, n)中的随机源，以生成一个以 a 为基数的 2 到 n − 2(含)的均匀随机整数。

虽然 GMP 提供种子 PRNG 并显式地将它们传递给需要访问伪随机数的函数，但此选项不适用于质数测试，即每次对 mpz millerrabin 的调用将与相同的 PRNG 状态一起工作。因此，由于初始种子状态是恒定的，由 mpz urandomm 为固定 n 选择的值的结果序列也是恒定的。注意，尽管，不同的 a 可以为不同的 n 选择，因为基 a 在取决于 n 的范围内均匀采样。这实际上意味着，当测试 n 时所选择的基被定义为 n 的函数。因此，对于固定的 n 和 t 值，mpz 概率素数 p(mpz t n, int reps)的结果是确定的

Pseudoprimes。对于整数 n, t, let (a_{21}, a，…， a_t)表示 GMP 使用的确定碱基列表，其中 t = rep。通过设置 n = (2x + 1)(4x + 1)，其中 x 为奇数，2x + 1,4x + 1 都为素数，我们将获得一个随机碱基 MR 测试将以大约 1/4 的概率通过的数字。由于(a_{21}, a，…， a_t)是伪随机的，我们可以预期以这种方式构建的 n 将以概率$(1/4)^t$通过 GMP 中的 MR 测试。因此，例如，对于最小推荐值 t = 15，通过尝试足够多的随机 x 值来构造一个合适的 n 可能是可行的，它总是被声明为素数。

然而，回想一下，我们需要 2x + 1 和 4x + 1 同时是质数，而且我们还必须通过以 210 为底的费马检验。这使得用这种直接方法构造 n 的成本高得让人无法接受，因为随机 x 给出素数对(2x + 1,4x + 1)的概率约为$(2/\ln x)^2$，而 n 的特殊形式意味着费马检验以大约 1/2 的概率通过(见附录 C)，而通过 t 轮 MR 检验的概率仅为(1/4)。把这些放在一起，每个 x 通过的概率约为 $2t-1\ 2\ 1/2(\ln x)$;要想 99%的几率成功找到一个 ln x = s 的好 x，我们需要大约 $5 \cdot 2^{2\ 2t-1}$次尝试，每次尝试至少涉及对 2x + 1 的质数测试。对于 1024 位的 n 和 t = 15 次试验(GMP 推荐的最低试验)，大约需要 247 次试验，每次试验至少涉及 512 位素数试验。

相反，部分受到 ROCA 攻击[NSS+17]和那里利用的素数形式[JPV00,JP06]的启发，我们考虑特殊形式 x = kM + 189 的 x，其中 M 是集合 P ={2,3，…， 373}和 k 是一个随机选择的大小的整数，使 n = (2x + 1)(4x + 1)具有理想的目标大小(例如 1024 位)。选择这种形式的 x 确保 2x + 1 = 2kM + 379 和 4x + 1 = 4kM + 757 不能被 P 中的第一'质数整除，从而提高 2x + 1 和 4x + 1 都是质数的几率(x 的形式本质上确保 2x + 1,4x + 1 通过 P 中的第一'质数的试除法;这里我们依赖的是 379 和 757 都是质数且大于 373 的事实)。189 的偏移量是特别选定的，这样所选形式的 n 在以 210 为底的 n 上的费马检验总是通过。这源于一个预定的数学分析，该分析被推迟到附录 C。

我们用于构造这种特殊形式的 x(和 n)的代码首先为 n 选择一个目标位大小，然后选择'尽可能大，以便 k 有足够的选择，以便有足够多的候选项，从而产生一个合适的 x。对于每个结果 x，我们的代码测试 2x + 1，然后 4x + 1 质数，并且(如果这些测试通过)应用 GMP 质数测试所需的 t 轮 MR 测试。

对于 1024 位的 n，我们设置` = 69，将 M 作为质数到 349 的乘积，并为 k 留下一个 51 位的值。M 的选择增加了 2x + 1 和 4x + 1 都是质数的概率，大约是 25 的倍数，而 x 的形式确保费马测试总是通过，给出了另一个 2 因子的改进。总共使用 33,885 core-hours (3.87 core-years)的

[7] 我们注意到，当 n 仅略小于 2 的幂时，即使 n 不同，也会产生相同的 amiay 序列。这是由于应用了通过与 n 进行比较的拒绝抽样来在高达 n 的范围内进行抽样。

在运行 2.4GHz 的 872 个核心上并行计算(由 CloudFlare 捐赠)，我们发现以下 1024 位示例通过了 GMP 的素数测试，t = 15 轮 MR 测试:

$$n = 2^{960} \cdot 0\ \text{x}000000000000000000000000000000081d564fbdd20b406$$

$$768$$

$$+2 \cdot 0\text{x}750af7bd334dcf547b131a1d8f8235fd603dba44e22e0775$$

$$+ \ ^{,576} \cdot 0\ \text{x}0ecf755051d33cb8895413f5d69f5a3df701889e3a69f92e$$

$$384$$

$$+2 \cdot 0\text{x}dd3f5f36662521877231ba4753a3e7185a89ddb0b2d73a35$$

$$+ \ ^{,192} \cdot 0\ \text{x}9e976a9bcfeae1a7c026d74bc7515a5010f3cd62c69fa9ad$$

$$_0 +2 \cdot 0\text{x}7b699f40e7a85192e1a4aa95537363fcb93d789aee32bbbf。$$

我们回想一下，对于 15 个 MR 轮，这个 n 总是能通过 GMP 的质数测试，因为 MR 碱基的生成确定地依赖于 n。

4.3 Mini-GMP

Mini-GMP 是 GMP 6.1.2 [Gt18]中 mpn 和 mpz 接口子集的一个小实现。这个库包括它自己的 mpz probab prime- p(n, reps)的微型实现。与 GMP 相比，最显著的变化是米勒-拉宾显式地执行了 t 碱基的确定性序列，通过计算欧拉多项式 $a(x^2) = x+ x+ 41$ 在 x = 0,1,2⋯， t − 1。也省略了 GMP 的费马检验。

Pseudoprimes。在 Mini-GMP 中使用一个确定的碱基序列使我们能够预测对 rep 的任何特定值 t 所选择的碱基。碱基并不都是素数(尽管著名的欧拉多项式确实产生许多素数)，因此我们不能直接使用 3.1.2 节中的阿尔诺方法。相反，我们使用 3.1.4 节中的复合、固定基方法的扩展。

使用这种方法，我们构建了一个 2960 位的复合材料 $n = {}_{21}ppp_3$，它通过了 t = 101 轮 Mini-GMP 的米勒-拉宾测试。在欧拉多项式产生的 101 个碱基中，有 86 个已经是素数，剩下的 15 个碱基全部分解为 163、167、179 和 199 这 4 个素数的各种组合。然后使用 3.1.4 节中描述的方法来使用 90 个唯一素数的组合列表来产生 n。这个 n 在附录 d 中给出。我们注意到 Mini-GMP 的文档与主要 GMP 库共享，这意味着用户需要进行 15 到 50 轮 MR 测试。

4.4 Java

Java 实现为任意精度的运算提供了自己的方法，包括素数检验，如 Java .math. biginteger 所示。我们考虑 OpenJDK10 [Cor18]，尽管在 JDK8 等旧版本中似乎没有对这部分代码进行重大更改。

分析。素数测试函数 isProbablePrime 被传递了一个确定的单一参数。这是一个由用户选择的值，在文档中被描述为:“调用者愿意容忍的不确定性的度量:如果调用返回 true，这个大整数是质数的概率超过$(1 - 1/2^{certainty})$。”然后，确定性参数用于确定将执行多少轮测试。这是通过调用附录 e 中所示的 primeToCertainty 函数来完成的。该函数首先将变量 n 设置为(certainty + 1)/2。当确定性为偶数时，这将产生一个非整数结果，但结果被转换为整数，隐式地限制结果。

[8] 由于 n 在确定 Miller-Rabin 要执行的轮数时所起的作用，结果是当 k 为奇数时，测试 isProbablePrime(k)和 isProbablePrime(k+1)没有区别。这对给用户的保证有影响——$1 - 1/2^{certainty}$ 的保证对于一半的确定性值来说不再准确。

该函数还考虑了被测试数字的位大小;如果小于 100，那么 Miller-Rabin 最多执行 50 轮;如果大于 100，则同时进行米勒-拉宾和带有塞尔 fridge 参数的 Lucas 可能素数检验，如 3.2 节所述。在后一种情况下，Miller-Rabin 的最大轮数是根据测试数的比特大小来确定的，类似于 OpenSSL。在这两种情况下，用户对确定性的选择将决定米勒-拉宾执行的实际轮数，只有当它小于内部指定的该位大小的轮数。

Pseudoprimes。对于密码学上有趣的大小数字，Java 执行 Miller- Rabin 和 Lucas 可能素数测试。使用 3.2 节中概述的方法，我们可以生产出由 Lucas 测试保证被声明为素数的复合材料。然而，由此产生的形式并不适用于任何已知的具有大量米勒-拉宾非证人的复合材料族。因此，我们无法利用我们目前的技术以高概率构造出任何通过 Java 素数测试的数字。

4.5 JavaScript 大数(JSBN)

由 Tom Wu [Wu17]编写的 Java Script Big Number (JSBN)库为 Java Script 应用程序提供了一个小型的加密工具包。在这里，我们研究了 2013 年最新发布的 JSBN 1.4。根据其主页，该库已在各种应用程序中使用，包括:Forge (SSL/TLS 的纯 JavaScript 实现)，谷歌的 V8 基准套件版本 6,JavaScript 加密工具包和 RSA-Sign JavaScript 库。

分析。该库提供了素数测试 bnIsProbablePrime(t)，其中参数 t 定义了用户希望执行的 Miller-Rabin 轮数。代码文档指出，该函数将“以确定性≥1− .5 的方式测试素数 t”。该函数伪随机地从硬编码的所有低于 1000 的素数列表(称为 lowprimes)中为每一轮米勒-拉宾(Miller-Rabin)选择一个基数 a。

Pseudoprimes。我们可以把这种实现看作是用固定的基数进行测试，其中所选的基数都是 2 到 1000 之间的质数。然后，我们可以使用阿尔诺的方法(3.1.2 节)来构造通过 JSBN 素数测试的合数 n，无论用户希望执行多少轮测试 t。例如，我们使用 SageMath 7.6 [S+17]来获得具有 3 个质数因子的 4279 位合数 n，详情参见附录 F。

4.6 Libgcrypt

Libgcrypt [Koc18]是一个通用密码库，最初基于 GnuPG 的代码。该库提供了各种加密函数，包括公钥算法、大整数函数和素数测试。我们分析了 2017 年 12 月发布的当前稳定版本 1.8.2

分析。Libgcrypt 的文档指出，用于检查质数的质数的函数是在 primegen.c 中找到的 gcry 质数检查。然后这个函数调用 _check prime，在其中执行实际的测试。这个函数执行三个测试步骤。第一步是 4999 之前所有质数的试除法。第二步是基数为 a = 2 的费马检验。最后一步包括 t 轮 Miller-Rabin，其中碱基是伪随机选择的。我们注意到，t 是用户定义的，但不能设置为小于 5。检查素数生成算法中产生的数字的默认设置为 5，但当用户调用 gcry 素数检查时，t 的选择被硬编码为 64。

Pseudoprimes。在 3.1 节之后，如果我们选择 n 作为形式为 n = pqr 的 Carmichael 数(其中 p, q, r > 4999)，在检查素数中执行的测试的第 1 步和第 2 步是微不足道的。通过使用 3.1.3 节中的混合技术，我们可以创建一个 Carmichael 数，该数也具有随机分布的非目击者的最大数量。然后，我们只需要用伪随机基克服 t 米勒-拉宾检验。这是以概率(1/4)发生的。如果用户调用 gcry 质数检查，那么我们可以欺骗这个测试的概率将只有 2^{-128}。然而，执行 64 轮 Miller-Rabin 是相当耗时的，用户可能会被诱惑绕过 gcry 质数检查，用更少轮调用质数检查。在这种假设的情况下，或者在 Libgcrypt 1.3.0 (2007) [Koc05]之前的版本中(其中 gcry prime check 会默认调用 t = 5 轮)，我们能实现的最好结果是以 1/1024 的概率通过测试(对于 t = 5)。

- -

4.7 Cryptlib

Cryptlib 3.4.3 [Gut18]是一个由 Peter Gutmann 开发的开源安全工具包库。它提供了多种服务，包括:公钥算法、各种密码函数和素数测试。

分析。Cryptlib 中的素性检验是在 kg prime.c 中找到的函数 primeProbable，由 t 轮 Miller-Rabin 组成，其中 t 的值必须在 1 到 100(包括)之间，由用户在调用时选择。然后，该函数从固定的素数列表开始，递增地为每个测试选择基数。这要么是前 54 个质数(2 到 251)的列表，要么是前 2048 个质数(2 到 17863)的列表，取决于预处理器指令 CONFIG save MEMORY。

- -

Pseudoprimes。由于 t≤100，我们最多只会使用 2 和 541 之间的质数(第 100 个质数)作为底数进行测试。因此，对于任何有效的输入 t，我们可以通过这个测试来生成保证被声明为质数的数字，只需使用阿尔诺的方法来生成一个复合 n，其中前 100 个质数作为非证人。实际上，使用 3.1.2 节中描述的方法，我们可以生成一个 2329 位的合数，它对所有素数基(包括 541)都是伪素数。详情见附录 G。

4.8 苹果的 corecrypto 和 CommonCrypto 库

苹果的 CommonCrypto [App18a]库提供 iOS 和 OS X 的加密服务。Common- Crypto 依赖于苹果的 corecrypto 库[App18b]来提供底层加密基元的实现。

分析苹果的素数测试可以在 corecrypto 文件 ccz is prime.c 中找到，其中包含函数 ccz is prime。这个函数将一个要测试的数字 n 和用户选择的测试轮数 t 作为输入。这个函数然后调用在 ccprime rabin miller.c 中找到的函数 ccprime rabin miller。这反过来又可选地检查被测试的数字是奇数，不是前 256 个质数之一，并且不能被前 119 个质数之一整除(通过 gcd 计算)。然后它执行 min{t, 256}轮米勒-拉宾测试，从硬编码的前 256 个素数列表中递增地选择基数。代码文档指出，当执行 t = 32 轮测试时，错误素数分类的概率估计为 2^{-64}。

CommonCrypto 提供 primailty 测试 CCBigNumIsPrime，它调用 ccz 是 corecrypto 提供的素数。然而，在这种情况下，用户对测试轮数 t 没有选择，因为它被硬编码为 16。

- -

由于基的选择是基于 t 的值确定的，我们可以实现关于该值 t 的 100%的失败率，只需使用 3.1.2 节的方法来产生一个复合 n，其中前 t 个素数作为非证人。这样的 n 实际上保证被 ccz 声明为素数，对于任何用户输入≤t。我们在附录 I 中给出组合 n 的例子，这些组合 n 将被 corecrypto 声明为素数，对于 t≤256 和 t≤40。

由于 CommonCrypto 强制 ccz 为素数来执行 16 轮测试，我们在这种情况下甚至可以更容易地实现 100%的成功率。事实上，附录 I 中的两个例子在这种情况下都保证被声明为素数，附录 H 中提供的各种比特大小的补充例子也是如此。

4.9 LibTomMath

LibTomMath v1.0.1 [Den18b]是一个带有数论工具包的开源多精度整数库。

分析。LibTomMath 包括几种以试除法、费马检验和米勒-拉宾检验形式进行素性检验的方法。后两种方法以单个碱基 a 和数字 n 作为参数进行测试，并返回 a 是目击者还是非目击者。主素数检验由函数 mp prime is prime 定义，该函数接受参数 n(要检验的数)和 1≤t≤256 的整数 t。然后它执行一些试除法(默认前 256 个素数)，然后 t 轮 Miller-Rabin。要使用的基数的选择与 Cryptlib 中的做法类似:它只是从一个硬编码的素数列表中增量选择(但这次使用的是一个截至 1619 的 256 个素数的列表)。

LibTomMath (bn.pdf)的文档讨论了所需的米勒-拉宾轮数:“一般来说，为了确保一个数字很可能是质数，你必须使用至少 6 个或 6 个独特的基数来执行米勒-拉宾运算。”与此相补充的是一个 mp prime rabin miller trials 函数，该函数给出了根据测试数字的比特大小(类似于 OpenSSL 和[DLP93])实现错误率小于 2^{-96} 所需的轮数，以及 mp prime rabin miller trials 上面的头文件 tommath.h 中的一个注释，该注释指出错误分类的概率不超过(1/4)。

Pseudoprimes。由于基是根据 t 的值确定选择的，我们可以简单地通过使用 3.1.2 节的方法来产生一个有前 256 个素数作为非证人的复合 n，从而实现 100%的失败率;对于任意的 t 值(包括由 mp 素数 rabin miller 试验选择的描述错误率小于 2^{-96} 的 t)，这样的 n 保证被 mp 素数是素数声明为素数。附录 I 提供了这样一个 n 的 7023 位示例。如果保证使用更小的 t 值，则可以获得更小的示例;特别地，我们可以很容易地获得一个 t≤40 的 1024 位的例子(也见附录 I)。

4.10 LibTomCrypt

LibTomCrypt v1.18.1 [Den18a]是一个额外的加密工具包，与 LibTomMath 共享许多资源。

分析。LibTomCrypt 中的素数检验称为 isprime(n,t,result)。它以一个 n 作为参数进行测试，并执行 t 轮 Miller-Rabin。LibTomCrypt 的文档建议，每一轮 Miller-Rabin 都会将 n 为伪素数的概率降低 4 倍，因此推断出整体误差最多为(1/4)。LibTomCrypt 在运行时支持从 3 个不同的大整数库中进行选择。

如果选择 LibTomMath，则 isprime 将调用 mp prime isprime，如 4.9 节所述，传递参数 n 和 t。如果选择 TomsFastMath [Den18c]，则 isprime 将调用 fp_isprime_ex，一个在数学库 TomsFastMath 中定义的函数，执行与 LibTomMath 的 mp prime isprime 等价的测试。如果 GMP 被选中，那么 isprime 将被调用

mpz_probab 素数 p.如 4.2 节所述。三种选择中任意一种所使用的 t 的值继承自对 isprime 的原始调用，但是如果 t = 0，该值将被覆盖为 t = 40。

Pseudoprimes。如果选择了 LibTomMath 或 TomsFastMath, 4.9 节(见附录 I)中描述的伪素数将始终通过素数测试被声明为素数。如果 GMP 被选中，我们可以应用 4.2 节中的分析来生成伪质数(见附录 C)。

4.11 WolfSSL

WolfSSL 3.13.0 [Wol18b](以前的 CyaSSL)是一个小型的 SSL/TLS 库，目标是在嵌入式系统中使用。WolfSSL 提供基于公共域 TomsFastMath 0.10 [Den18c]和 LibTomMath 0.38 [Den18b]函数的质数测试工具。

分析。WolfSSL 中的质数测试是函数 mp prime is prime，它以要测试的数字 n 和测试的轮数 t 作为参数。该函数直接取自 LibTomMath 的旧版本，即 0.38 [Den18b]。WolfSSL 将默认使用 LibTomMath，但可以在运行时选择性地编译为使用 TomsFastMath 0.10 [Den18c]。LibTomMath 0.38 中的素性测试与 4.9 节中 1.0.1 版本中分析的素性测试相同。当使用 TomsFastMath 时，mp 素数是素数调用 fp isprime，这剥夺了用户选择的 t，并简单地调用 fp isprime ex 与硬编码的 t = 8 的值。函数 fp isprime ex 然后执行试除法(默认前 256 个素数)，然后使用前 8 个素数作为基数进行 8 轮米勒-拉宾运算。因此，它的行为等同于 mp prime is prime 在 LibTomMath 中，但 t = 8。

Pseudoprimes。由于 WolfSSL 中的测试实际上与 LibTom- Math 中执行的测试相同(但在使用 TomsFastMath 时仅使用 8 轮 Miller-Rabin)，因此附录 I 中给出的复合示例也被声明为 prime, 100% 成功。

4.12 Bouncy Castle

Bouncy Castle 是一个用 Java 和 c#编写的加密库[otBCI18]。Bouncy Castle Java 中的素数测试基于 JDK 中的 BigInteger 类，如 4.4 节所述。Bouncy Castle c#实现了自己的素数测试。我们分析了 Bouncy Castle c# 1.8.2 版本。

分析。负责素数测试的相关函数位于 BigInteger 类中。这个类提供了 IsProbablePrime 方法，它接受确定性作为参数。然后，该方法使用 t 轮的米勒-拉宾检验，其中 t 被计算为 t =((确定性 − 1)/2)+ 1。在每一轮中，使用由 Bouncy Castle 库提供的安全随机数生成器(SecureRandom)选择基础。

确定性参数必须始终提供给 IsProbablePrime 方法的调用。因此，用户的选择完全决定了执行多少 Miller-Rabin 轮。例如，这个方法直接在 TlsDHUtilities 类中使用，该类为 TLS 提供了 Diffie-Hellman 操作。当验证传入的 DH 参数时，ValidateDHParameters 方法调用确定性= 2 的 isProbablePrime。这导致只进行了单一的 Miller-Rabin 测试。

Pseudoprimes。我们可以用 3.1 节中的任何一种方法制作复合材料 n;这样的 n 满足 Monier-Rabin 界，因此将以概率 t 通过 Bouncy Castle 的素数检验

(1/4)以 t 为确定性导出。虽然没有正式的文档，但素数测试代码上方的注释表明，这个测试函数的失败率应该是$(1/2)^{certainty}$，这样就实现了用户对确定性的选择。

4.13 Botan

Botan 是一个用 c++ 11 [Llo18a]编写的密码库。除了加密功能之外，它还提供 TLS 客户端和服务器实现。我们分析了 Botan 2.6.0。

分析。相关的素数测试实现可以在 numthry.cpp 中找到，其中包含 function is prime。这个函数首先评估被测试的数是否能被 65521 以内的小素数整除。然后它用随机选择的基执行 Miller-Rabin 素数测试。随机性的来源和 Miller-Rabin 轮数基于传递给 is 素数函数的参数。轮数根据参数 prob 计算，t 设为(prob + 2)/2。Botan 的文档非常清楚地区分了随机来源和可能的对抗性来源的测试数。要区分来源，函数是素数包含布尔标志是随机的。如果设置，则代码使用[DLP93]根据被测试数字的比特大小分配 t，目标失败率小于 2^{-80}。

Pseudoprimes。和蹦蹦城堡一样，我们可以用 3.1 节中的任何一种方法制作合成 n;这样的 n 满足莫尼尔-拉宾界限，并将通过博坦与 $_t$ 的素数检验

(1/4)的最高概率，其中 t 来自于用户通过 t 选择 prob = (prob + 2)/2。从这个意义上说，测试的保证与用户的期望相匹配。

4.14 加密++

Crypto++ 7.0 是一个开源的 c++密码库，最初由 Wei Dai [Dai18]编写。在 nbtheory.cpp 中，Crypto++有多种素数测试算法。这些算法包括试除法、费马、米勒-拉宾以及强 Lucas 和标准 Lucas 可能素数测试。Crypto++的素数测试函数 isprime 同时执行 Miller-Rabin 和 strong Lucas 测试。因此，要骗过它，我们需要找到 Baillie-PSW 的伪质数(尽管 Miller-Rabin 测试是一个随机基测试，不同于 Baillie-PSW 中执行的测试)。目前我们还不知道有任何这样的伪质数。

4.15 GoLang

2009 年在谷歌上创建的 Go 编程语言(GoLang) 1.10.3 [Goo18]是一个开源项目，包括任意精度的算术和加密功能。

分析相关的素数测试实现可以在 int 中找到。go，其中包含函数 probaylyprime (t)。参数 t 定义了用户希望执行的米勒-拉宾轮次。该函数首先用一系列小素数进行试除法，然后是 t 轮的米勒-拉宾(其中一个基数被强制为 2，其他所有基数都是伪随机选择的)，最后是卢卡斯概率素数测试。因此，该函数正在进行 Baillie-PSW 检验。在版本 1.8 之前，Go 的 ProbablyPrime(t)函数只应用了 Miller- Rabin 检验。GoLang 提供的文档明确指出，该函数声明随机选择的复合输入为素数的概率最多为$(1/4)^t$。它还指出，“很可能 prime (t)不适合判断敌手可能精心设计来欺骗测试的质数”。

从攻击的角度来看，有趣的是，在这个质数测试中使用的伪随机数生成器是被测试的数字 n 的种子。因此，攻击者可以可靠地预测伪随机生成的米勒-拉宾基。

由于正在进行 Baillie-PSW 测试，我们知道没有复合材料被 GoLang 错误地声明为质数。然而，对于 2017 年发布的 1.8 之前的版本，我们能够利用米勒-拉宾基选择的不安全性质来产生合数，这些合数保证相对于参数 t 被声明为素数。由于这是 GNU GMP 为米勒-拉宾选择基所用的相同方法，我们可以使用第 4.2 节中描述的方法来产生这样的复合材料。我们在附录 J 中给出了一个复合数 n 的例子，对于 t≤13,n 总是被声明为素数。

4.16 数学软件包

我们还检查了在流行的数学软件包和计算机代数系统中发现的素数检验，即:Magma、Maple、Maxima、SageMath、SymPy 和 Wolfram Mathematica。我们将这些包含在我们的分析中，因为开发人员在手动检查标准或代码中的值时可能会依赖它们。一些库使用确定性测试来证明质数，尽管大多数库在测试大于 64 位的候选项时仍然依赖于概率方法。Maple、Maxima 和 SymPy 依赖于 GMP，因此继承了 4.2 节中讨论的质数测试的相同问题;然而，他们在最新版本中也都执行 Lucas 测试，因此这种“交叉污染”不会导致可利用的弱点。完整的细节在附录 L 中提供。

5 申请 Diffie-Hellman

验证 Diffie-Hellman (DH)参数的正确性是验证密钥交换完整性的关键步骤。正如引言中提到的，由于 DH 参数集(p, q, g)，其中 $g \in_p z$ 生成一组 q 阶，是公开的，它们可以来自第三方来源，如服务器或标准。一个熟练的 DH 参数验证函数应该检查 p, q 都是素数，并且对于某个整数 k $p = kq + 1$，它还会测试给定的生成器 g 生成 q 阶的子群，并且任何接收到的 DH 值都位于正确的子群中。一个常见的选择是设置 $k = 2$，因此 p 是一个安全素数。对于不是安全素数的 p，组阶 q 可以比 p 小得多，从而提供性能提升。然后安全等级是基于 q 的比特大小，它必须仍然足够大，以挫败解决离散对数问题(DLP)的波利格-赫尔曼算法，这对于素数 q 是 √

运行时间 O(q)，一个常见的参数选择是 160 位的 q 和 1024 位的 p 或 256 位的 q 和 2048 位的 p。

√

更准确地说，pohligi - hellman 算法的运行时间为 O(t)，其中 t 是 q 的最大素数因子。因此，攻击者可以通过欺骗素数测试来提供足够平滑的合成 q，使得 $p = kq+1$ 仍然是素数。例如，如果 q 是(2x+1)(4x+1)的形式，这将导致对复杂性为 2^{40} resp 的 DLP 的攻击。2^{64} 对于上面提到的大小。

但是，我们强调，在这项工作中，恶意合成的构造没有对坚持 $k = 2$ 的协议(如 Telegram)构成风险，即检查 $q = (p - 1)/2$ 和 p 的可合成性。例如，第 3.1.1 节的构造将设置 $q = (2x + 1)(4x + 1)$并产生总是能被 3 整除的 p;此外，q 不够平滑，不足以让波赫利格-赫尔曼对密码学上合适大小的参数构成威胁。这是一个有趣的开放问题，找到一个大的，足够平滑的合成 q 以高概率通过素数测试，使 $p = 2q + 1$ 是素数或也通过素数测试。

我们现在讨论各种库中的 DH 验证函数。对于每个库，我们应用第 4 节的分析来检查这些库的攻击健壮性。我们注意到，第 4 节中讨论的其他库没有实现用于验证 DH 参数的更高层次的函数。当然，这并不妨碍应用程序使用这些库来实现自己的验证功能。这样的应用程序将继承底层库的弱点和优点(当 $k =2$ 被允许时)。我们为下面的 GMP 库提供了这个场景的一个例子。我们以 SSL/TLS 的重要用例的讨论结束。

OpenSSL“dh check.c”文件中包含“dh check params”和“dh check”函数。前者是一个轻量级的检查，通过测试 p 是否为奇数和 1 < g < p − 1，只是确认 p 和 g“可能足够”是有效的。后一种函数则更为彻底，它调用 BN is prime ex 来检验 p 和 q = (p − 1)/2 的质数。这些素数检验是用 checks = BN 素数检查来调用的，因此 Miller-Rabin 的轮数由表 2 确定。这意味着，例如，当 n 有超过 1300 位时，他们将声明为质数，概率为 1/16，复合特殊形式 n = (2x + 1)(4x + 1)(x 奇数和 2x + 1,4x + 1 质数)。由于不需要隐私数据，这个测试函数最有可能的用例是检查由其他人(可能来自不可信的服务器或未知的来源)生成的 Diffie-Hellman 参数，因此显然误用了 OpenSSL 自己的质数测试函数。

然而，由于 OpenSSL 将参数集(p, q, g)限制为安全素数 p，因此高效的攻击是不可行的。利用我们目前的技术，我们无法生成一个集合，它将以高概率同时通过 p 和 q 的质数测试，并允许高效求解 DLP。

Bouncy Castle ValidateDHParameters 中的 DH 参数验证从 DH 参数集中提取 p、g，然后仅用 1 轮 Miller-Rabin 检查 p 的素数性。因此，我们可以生产出被接受为概率为 1/4 的 DH 模的复合材料。更严重的是，q 并没有给到校验函数，因此即使有素数 p, g 的值也可以被选择，使其具有小阶，使得波赫利-赫尔曼(polhligi - hellman)如所期望的那样简单。即使 g 有大的素数阶数，选择参数的灵活性将允许 Lim-Lee 进行小的子群攻击，如[VAS+17]中所探讨的。

Botan The Botan function is prime 用于类 DL 组(位于 DL Group .cpp 中)，也用于验证 DH 参数。这个类包含验证组函数，可以用布尔参数 strong 调用。如果 strong 设置为 true，则使用 prob=128 调用 is prime 函数。这将导致 t = 65 的米勒-拉宾计算。否则，执行 prob=10 和 6 个米勒-拉宾计算。对 p 和 q 都进行此检验;代码还检查 q|(p − 1)，但不坚持 p 是安全素数。

使用 3.1 节中描述的方法，我们可以找到一个 160 位的 q，它以 1/4096 的概率通过 6 轮 MR 测试，使得 q 具有 2 或 3 个素数因子。然后，我们可以通过使用 k 中的灵活性，将 1024 位素数 p 构造为 p = kq + 1，以及一个生成大小为 q 的子群的 g。由于这个 p 确实是素数，并且 q|(p − 1)，如果 strong 设置为 false，则所有对参数集(p, q, g)的波坦测试将以概率 1/4096 通过。我们随后可以使用 pohligi - hellman 算法来求解 g 生成的子群中的 DLP，用大约 2^{28} 个努力就能破 DH。有关这样的参数集的例子，请参见附录 K。

GNU GMP 256 位整数 q = (2x + 1)(4x + 1) with

X = 0x400286bac15132db85b1c936709f369b

通过 15 轮 GMP 素数检验 mpz 是 probab 素数 p;挑选 k = 21792 + 1254 产生 2048 位素数 p = kq + 1。如果潜在的质数测试是基于 GNU GMP 的最低推荐轮数 Miller-Rabin 的代码，那么由此产生的参数集(p, q, g)甚至可以确定地通过完全熟练的 DH 验证。

SSL/TLS 我们最后对 SSL/TLS 中 DH 参数测试的情况进行了评论。这里，服务器选择参数，但只发送(p, g)给客户端。没有要求 p 是安全素数。这使得客户端很难验证 DH 参数(他们需要因式 p − 1，然后尝试不同的约数来确定 q 的顺序)或对收到的 DH 值进行组成员测试。因此，大多数客户端只执行简单的心智检查，例如检查 g ∈/ {0，±1}。这使得 SSL/TLS 容易受到各种恶意的 DH 参数攻击 cf. [Won16,VAS+17]，鉴于这些，展示复合质数 p 来欺骗质数测试对于目前形式的 SSL/TLS 标准来说是过度的。然而，我们的工作表明，即使客户端试图通过因子化来验证 DH 参数

$p-1$，找到 g 的阶，然后测试它的质数，他们仍然可能与恶意的 DH 参数发生冲突。而且如果修改 SSL/TLS 协议，让服务器提供完整的 DH 参数，仍然需要仔细检查。最后我们注意到，TLS 1.3 中只允许少量固定的、安全的 prime DH 参数集。这些参数最近在 RFC 7919 [Gil16]中标准化，为协议的未来版本缓解了这些问题。

6 结论和建议

我们的工作探索了对抗性环境中的素性测试及其对 Diffie-Hellman 参数测试的影响。我们的主要发现是，领先的库不是为这种设置而设计的，因此往往容易被接受为恶意选择的素数合成输入，见表 1。

非对抗性(或随机)和对抗性素数测试之间需要仔细区分，这一点在密码学研究界当然得到了充分的理解。然而，这种区分并不一定反映和实现在密码学库及其文档。因此，我们通常可以将素数分类精度失败的潜在原因归类为未考虑对抗性设置。更明确地说，我们可以根据米勒-拉宾的基是如何选择的来对大多数失败进行分类，即固定基、可预测基、基数不足。迷你-GMP、JSBN、Cryptlib、LibTomMath、LibTomCrypt 和 WolfSSL 都因为从固定列表中选择碱基而失败，而 GNU GMP 和 GoLang pre 1.8 都受到可预测碱基的影响。OpenSSL、Libgcrypt、Botan 和 Bouncy Castle c#都可以选择运行任意多轮的 Miller-Rabin，但要么默认运行，要么调用测试(在库的其他地方)轮数太少。

基于我们的分析，我们提出以下建议:

-在没有已知伪质数的情况下，我们建议库在可能的情况下切换到使用 Baillie- PSW 质数检验。对性能的负面影响是适度的，对安全性的正面影响是显著的。在计算机代数系统 PARI/GP [The18b] (Sage 的素性测试函数基于此)的文档中可以找到这种权衡的现有基准。PARI/GP 实现了用户定义 t 的 Miller-Rabin 测试和 Baillie-PSW 测试，并表明[The18a]他们的 Baillie-PSW 测试大约与 $t = 3$ 的 Miller-Rabin 测试一样快。

-希望继续只使用 Miller-Rabin 的库(例如，维护一个小的代码库)应该使用 pseudorandom bases, cf. Cryptlib, LibTomCrypt, JavaScript Big Number, WolfSSL。特别要注意的是，这些基库不应该只依赖于 n, cf. GNU GMP。

-我们还建议在选择迭代次数时默认为最坏情况的界限，并且在用户明确指示时只假设平均情况的行为。具体来说，我们建议使用 64 次迭代来确保合数被错误地识别为质数，其概率最多为 2^{-128}。默认到最坏情况界限对性能的影响在非对抗性设置中应该是最小的，因为只要识别出复合的 Miller-Rabin 证人，测试就可以被放弃，这些是非常常见的(如[DLP93]的界限所示)。另一方面，正是在对抗性设置中，需要最坏情况边界。采用这一建议可能需要对素数测试代码的接口进行更改。

-新协议的设计者应该避免 SSL/TLS 中的陷阱，即 DH 参数验证对客户端不切实际。TLS 1.3 通过修复并要求使用一小部分参数集来做到这一点。

密码学文献中的定义通常以“让 p 是一个素数……，然而我们的工作强调，许多实现不一定为这个假设提供强有力的保证。因此，这是一个有趣的开放问题，文献中有关领域参数的其他看似无害的假设可以以类似的方式被破坏。

致谢

Albrecht 得到了 EPSRC 的资助 EP/P009417/1。Massimo 得到了 EPSRC 和英国政府的支持，是伦敦大学皇家霍洛威学院网络安全博士培训中心(EP/K035584/1)的一部分。帕特森获得 EPSRC 资助 EP/M013472/1、EP/K035584/1 和 EP/P009301/1。Somorovsky 获得了“地平线 2020”计划的支持，项目编号为 700542 (FutureTrust)。

我们感谢 Christian Elsholtz 对数学文献的初步指导，以及 Ian myers 对我们分析 OpenSSL 的帮助。我们感谢来自 Cloudflare 的 Brendan McMillion 和 Nick Sullivan 慷慨提供的计算资源，使我们能够找到第 4.2 节中的示例。

参考文献

AKS04 Manindra Agrawal, Neeraj Kayal 和 Nitin Saxena。PRIMES 在 P. Annals of mathematics，页 781-793,2004。

AM93 奥立弗 • L • 阿特金与恰朗 • 科伊斯 • 莫兰。椭圆曲线与素数证明。计算数学，61(203):29-68,1993。

App18a 。苹果(aapl . o: 行情)。Common crypto - Security 苹果开发者。https://opensource.apple.com/ /source/CommonCrypto/， 2018 年 8 月。

App18b。Apple Inc. corecrypto -安全苹果开发者。https://developer.apple.com/security/， 2018 年 8 月。

Arn95。弗兰,coi 阿尔诺。构造几个碱基的强伪质数卡迈克尔数。符号计算学报，20(2):151-161,1995。

Arn97。弗兰,coi 阿尔诺。卢卡斯伪素数的拉宾-莫尼耶定理。《美国数学学会的计算数学》，66(218):869-881,1997。

Bai13a。罗伯特柏丽。OEIS A217120: Lucas pseudoprimes。https://oeis.org/A217120, 2013 年 3 月。

Bai13b。罗伯特柏丽。OEIS A217255:强卢卡斯伪质数。https://oeis.org/A217255, 2013 年 3 月。

BCP97。Wieb Bosma, John Cannon, Catherine Playoust。岩浆代数系统。J.符号计算机。， 24, 1997。《计算代数与数论》(伦敦，1993)。

Ble05。丹尼尔 Bleichenbacher。用伪素数破解密码协议。In Serge Vaudenay，编辑，PKC 2005, LNCS 第 3386 卷，第 9-15 页。《施普林格》，海德堡，2005 年 1 月。

BW80。罗伯特 • 贝利和塞缪尔 • 瓦格斯塔夫。卢卡斯 pseudoprimes。计算数学，35(152):1391-1417,1980。

CMG+16。Stephen Checkoway、Jacob Maskiewicz、Christina Garman、Joshua Fried、Shaanan Cohney、Matthew Green、Nadia Heninger、Ralf-Philipp Weinmann、Eric Rescorla、Hovav Shacham。系统分析 juniper dual EC 事件。在 Edgar R. Weippl、Stefan Katzenbeisser、Christopher Kruegel、Andrew C. Myers 和 Shai Halevi，编辑，ACM CCS 16, 468-479 页。ACM 出版社，2016 年 10 月。

CNE+14。Stephen Checkoway、Ruben Niederhagen、Adam Everspaugh、Matthew Green、Tanja Lange、Thomas Ristenpart、Daniel J. Bernstein、Jake Maskiewicz、Hovav Shacham、Matthew Fredrikson。谈 TLS 实现中双重 EC 的实际可利用性。在第 23 届 USENIX 安全研讨会(USENIX Security 14)，第 319-335 页，圣地亚哥，CA, 2014。USENIX 协会。

Cor18。甲骨文公司。OpenJDK 10 Open Java Development Kit, 2018。openjdk.java.net。

CP06。理查德 • 克兰德尔和卡尔 • 波梅朗斯。《质数:计算视角》，第 182 卷。施普林格科学与商业媒体，2006 年。pp.136 - 140。

Dai18。魏戴。加密++。https://www.cryptopp.com/， 2018 年 4 月。

Den18a。汤姆 • 圣丹尼斯。LibTomCrypt。http://www.libtom.net/LibTomCrypt/， 2018 年 4 月。

Den18b。汤姆 • 圣丹尼斯。LibTomMath。http://www.libtom.net/LibTomMath/， 2018 年 4 月。

Den18c。汤姆 • 圣丹尼斯。TomsFastMath。http://www.libtom.net/TomsFastMath/， 2018 年 4 月。

DLP93。Ivan Damg° ard, Peter Landrock 和 Carl Pomerance。强可能质数检验的平均案例误差估计。计算数学，61(203):177-194,1993。

DR08。T. Dierks 和 E. Rescorla。传输层安全(TLS)协议 1.2 版本。RFC 5246(建议标准)，2008 年 8 月。由 rfc 5746、5878、6176、7465、7507、7568、7627、7685、7905、7919 更新。

FGHT17。Joshua Fried, Pierrick Gaudry, Nadia Heninger, Emmanuel Thom´e。1 千比特隐式 SNFS 离散对数计算。在 Jean-S´ebastien Coron 和 Jesper busus Nielsen，编辑，EUROCRYPT 2017，第一部分，LNCS 卷 10210,202-231 页。施普林格，海德堡，2017 年 4 / 5 月。

fSidI17 《德国信息技术报》(Bundesamt f ¨ur Sicherheit in Informationstechnik)BSI tr - 02101 -1 密码机制：建议和密钥长度。《技术指南》，联邦信息安全办公室，2017 年 1 月。

Gil13 杰夫·吉尔克莱斯特。带有概率素数测试的伪素数枚举。http://gilchrist。ca/jeff/factorings / pseudoprime .html，2013 年 8 月。

Gil16 d·吉尔默。协商传输层安全(TLS)的有限域 Diffie-Hellman 临时参数。RFC 7919(建议标准)，2016 年 8 月。

Goo18 谷歌。Go 编程语言。https://golang.org，2018 年 7 月。
。

Torbjorn Granlund 和 GMP 开发团队。GNU MP: GNU 多重精度算术库。https://gmplib.org，2018 年 4 月。

Gut18 彼得·古特曼。CryptLib。http://www.cryptlib.com/，2018 年 4 月。
。

H ¨o16 Andreas H ¨oglund。MPZ SPSP 低于 GMP 5.0.1。http://www.hoegge.dk/gmp/gmp501.htm，2016。最后访问 2016-10-31。
。

Jac15 达纳·雅各布森。伪质数统计、表格和数据。http://ntheory.org/pseudoprimes。html，2015 年 8 月。
。

Jae93 格哈德 Jaeschke。对几个基地的强伪质数。计算数学，61(204):915 - 926,1993。

Marc Joye 和 Pascal Paillier。便携设备上的质数快速生成:一个更新。在 Louis Goubin 和 Mitsuru Matsui，编辑，CHES 2006，第 4249 卷的 LNCS 160-173 页。施普林格，海德堡，2006 年 10

JPV00 Marc Joye、Pascal Paillier、Serge Vaudenay。质数的高效生成。In C etin Kaya Ko C and Christof Paar，主编，CHES 2000，《LNCS》1965 卷，340-354 页。《施普林格》，海德堡，2000 年 8 月。

Koc05 维尔纳·科赫。Github - libgcrypt 更改为默认质数测试。https:

//github.com/gpg/libgcrypt/commit/78a84338cb36748f17cc444b17ab7033ce384c34#diff-96a06fc4d0080caec00d423ca08a6c86，2005 年 4 月。

Koc18。维尔纳·科赫。Libgcrypt。https://gnupg.org/software/libgcrypt/index.html，2018 年 4 月。

Lib18。LibTomMath。Pull request -增加了 Fips 186.4 合规，一个额外的强大的 Lucas-Selfridge(用于 BPSW)。https://github.com/libtom/libtommath/pull/113，2018 年 8 月。

Lit09。Dwayne C. Litzenberger。PyCrypto 魅惑。https://pypi.python.org/pypi/pycrypto/2.1.0，2009 年 12 月。

LK08。M.莱平斯基和 S.肯特。用于 IETF 标准的额外 Diffie-Hellman 组。RFC 5114(信息)，2008 年 1 月。

LLC18。Telegram FZ 有限责任公司，Telegram Messenger。https://telegram.org，2018。

Llo18a。杰克劳埃德。Botan。https://github.com/randombit/botan，2018 年 8 月。

Llo18b。杰克劳埃德。Botan pull request - 添加来自 FIPS 186-4 的 Lucas 测试。https://github.com/ randombit/botan/pull/1636，2018 年 8 月。

Mac17。Macsyma 组。Maxima 5.41.0，文档，2017。可在 http://maxima 下载。sourceforge.net/docs/manual/maxima_10.html。

Mac18。Macsyma 组。Maxima 5.41.0，计算机代数系统，2018。可在 http://maxima 下载。sourceforge.net/index.html。

Mar16。马塞尔·马丁。primo - priality proof，2016 年。https://www.ellipsa.eu。

McC97。Jud McCranie。OEIS A014233:小于或等于 n 个素数的基上的 Miller-Rabin 素数检验不能揭示复合性的最小奇数。https://oeis.org/A014233，1997 年 2 月。

Mil75。加里·米勒。《黎曼的假设与质心检验》。第七届 ACM 计算理论年会论文集，第 234-239 页。ACM，1975 年。

Mon80。路易 Monier。两种有效的概率素性测试算法的评估和比较。理论计算机科学，12(1):97-108,1980。

MVOV96。阿尔弗雷德·J·梅内塞斯、保罗·C·范·奥尔肖特和斯科特·A·范斯通。《应用密码学手册》。CRC 出版社，1996 年。

Nar14。Shyam Narayanan。提高了米勒-拉宾素性检验的速度和精度。MIT primes-usa，2014 年。https://math.mit.edu/research/highschool/primes/materials/2014/ Narayanan.pdf。

Nic16。托马斯。GNU GMP mpz probab prime p - - - pseudoprimes

http://www.trnicely.net/misc/mpzspsp.html，2016。最后访问 2016-10-31。

NSS+17。Mat´us Nemec, Marek S´ys, Petr Svenda, Dusan Klinec 和 Vashek Matyas。The return of coppersmith’s attack:对广泛使用的 RSA 模的实用分解。在 Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu，编辑，ACM CCS 17, 1631-1648 页。ACM 出版社，2017 年 10 月/ 11 月。

OP18a。GitHub OpenSSL 项目。Pull request -增加 RSA 素数代的 MR 测试数量#6075。https://github.com/openssl/openssl/pull/6075，2018 年 8 月。

OP18b。OpenSSL 项目。OpenSSL：SSL/TLS 的开源工具包。www.openssl.org，2018 年 5 月。

otBCI18。充气城堡军团公司。The Bouncy Castle Crypto Package For C Sharp，2018。https://github.com/bcgit/bc-csharp。

Pom84。卡尔 Pomerance。Baillie-PSW 素性检验是否有反例。Dopo Le Parole aangeboden aan A. K. Lenstra 博士。，1984 年。

PSW80。卡尔·波梅兰斯、约翰·塞尔弗里奇和塞缪尔·瓦格斯塔夫。$25 \cdot 10^9$ 的伪质数。计算数学，35(151):1003 - 1026,1980。

Pub13。联邦信息处理标准出版物。FIPS PUB 186-4 数字签名标准(DSS)。标准，国家标准与技术研究所，盖瑟斯堡，MD，2013 年 7 月。

Rab80。迈克尔·拉宾。测试素性的概率算法。数论学报，12(1):128-138,1980。

RI18。Wolfram 研究公司 Mathematica，2018 年 11.3 版。《香槟》，IL，2018 年。

Rie16。格哈德 Rieger。Socat 安全咨询 7 - Openwall oss-security 邮件列表，2016。http://www.openwall.com/lists/oss-security/2016/02/01/4。

肌+力。威廉·斯坦等人。Sage 数学软件 8.2 版。Sage 开发团队，2017 年。可在 http://www.sagemath.org 下载。

Sta05。英国的标准。ISO/IEC 18032:2005 信息技术。安全技术。素数生成。标准，国际标准化组织，2005 年 1 月。

Str16。法尔 Strenzke。OpenSSL 随机数生成器的分析。在 Marc Fischlin 和 Jean-S´ebastien Coron，编辑，EUROCRYPT 2016，第一部分，LNCS 的 9665 卷，第 644-669 页。施普林格，海德堡，2016 年 5 月。

Sym17a。SymPy。SymPy GitHub 仓库。可在 https://github.com/sympy/sympy/commit/9e35a94eceaff73b350794dcc70b4a412dc2f6e6#diff-e20bc128d13486b598a04fce77584900 下载，

2017.

Sym17b。symy 开发团队。SymPy:符号数学 Python 库，2017 年。

The18a。巴黎集团，波尔多大学。《PARI/GP 常见问题》，2018。可从 http://pari.math.u-bordeaux.fr/faq.html#primetest 获取。

The18b。巴黎集团，波尔多大学。2018 年 PARI/GP 版本 2.9.0。可从 http://pari.math.u-bordeaux.fr/下载。

VAS+17。卢克·瓦伦塔、大卫·阿德里安、安东尼奥·桑索、沙南·柯尼、约书亚·弗里德、玛赛拉·黑斯廷斯、j·亚历克斯·哈德曼、纳迪亚·海宁格。衡量针对 Diffie-Hellman 的小亚群攻击。在 NDSS 2017。《互联网社会》，2017 年 2 / 3 月。

Wat17。Waterloo Maple (Maplesoft)。枫叶 2017 版，2017。https://www.maplesoft 有售。com/products/Maple/。

Wei18。埃里克·w·韦斯坦。Baillie-PSW 素性测试来自于 MathWorld- Wolfram 网站资源。http://mathworld.wolfram.com/Baillie-PSWPrimalityTest.html，2018 年 4 月。

Wol18a。WolfSSL Inc .)Pull request -素数测试。https://github.com/wolfSSL/wolfssl/ pull/1665，2018 年 8 月。

Wol18b。WolfSSL Inc .)WolfSSL。https://www.wolfssl.com/wolfSSL/Home.html，2018 年 4 月。

Won16。大卫·王。How to Backdoor Diffie-Hellman。密码学 ePrint Archive，报告 2016/644,2016。https://eprint.iacr.org/2016/644。

Wu17。汤姆吴。JSBN: JavaScript 中的 RSA 和 ECC。http://www-cs-students.stanford.edu/~tjw/jsbn/，2017 年 4 月。

A 阿尔诺方法概述

阿尔诺的方法生成 n，形式为 n = pp$_{2\ 1}$⋯p 其中 $_{h}$ 有两个不同 $_{i}$ 的奇素数，使得 n 对一组 t 素数基底{a, a，⋯$_{12}$}。$_{t}$ 由[Arn95，引理 3.2]可知

如果 gcd(a, n) = 1，并且 $\left(\frac{a}{p_i}\right)$ = - 1 对于所有 1≤i≤h，那么 a 将是米勒-拉宾非证人

相对于 n(这一组条件是 a 相对于 n 是米勒-拉宾非证人的充分条件，但不是必要条件)。

现在，根据高斯二次互易定律，我们知道，对于任何素数 p， $\left(\frac{a}{p}\right)$ 可以

从? p一
个

a，我们可以计算出潜在质数 p 的可能非剩余的集合 amod 4a。也就是说，我们可以计算出满足 a 条件的集合 ssatisfaction

$$\left(\frac{a}{p}\right) = -1 \iff p \bmod 4a \in S_a.$$

阿尔诺的方法选择 1p，然后确定形式为 $p_i = k_i(p_1 - 1) + 1$ 的其他 p_i from 方程，其中 k_i are 值也被选择为方法的一部分($k_1 = 1$)。这样??做是为了确保结果 n = pp2 1⋯pish 一个卡迈克尔数。但是，

条件 $\frac{a}{p_i}$ = - 1 对于所有 1≤i≤h 意味着，对于每个 a∈a 和每个 1≤i≤h，我们有

$k_i(p_1 - 1) + 1 \in {}_a s$ 重写此，我们得到：

$$P_1 \bmod 4a \in \bigcap_{i=1}^{h} k_i^{-1}(S_a + k_i - 1), \tag{5}$$

其中 ki-1(Sa+ki-1)表示集合{ki-1(s + ki-1)mod 4a|s∈Sa}。这就给出了一组关于每个 a∈a 的 pmodul1 o 4a 值的条件;通常，对于 1 每个 a 的值，pmod 4a 的几个候选者仍然存在。通过为每个 a∈a 选择这些候选者中的一个 zfor 并使用 aCRT，这些条件可以组合成 pmodulo m = lcm(4, a，⋯11， a).必须选择 k 值，t 使(5)右侧 i 的集合是非空的;通常，它们被设置为比 a∈a 的最大值更大的小素数，这样对于每个 a, k 都存在 mod 4a。i -1

然后，阿尔诺的方法发挥了对每个 i = 2 的 $p_1 \bmod k_i$ for 的其他限制，⋯， h.这些限制源于 n 是卡迈克尔数的要求。我们省略了完整的细节，但是，例如，当 h=3 时，额外的限制可以写成：

P1 = k3-1 mod k2 和 P1 = k2- 1 modk3

使 ki co-prime 相互作用和对 a∈a 保证了 CRT 的另一个应用可以纳入这些条件。最终的结果是一个单一的条件的形式：

P1=z mod lcm(4,a1，⋯， at,k2，⋯kh)

其中 z 是一个固定值，由 z 值的选择 a 和额外的限制决定。

最后，该方法反复生成满足上述约束的候选 1p，并使用方程 $p_i = k_i(p_1 - 1) + 1$ 来确定另一个 pi。1 对于给定的 pif1，该方法是成功的，所有得到的 p，⋯， pareh 素数。

显然，该方法是复杂的，并不能保证对给定集合 a 的每次尝试都成功。然而，它可以用 k 的不同选择 i 进行迭代，直到(5)右侧的集合是非空的;此外，可以使用回溯方法来选择 z 值 a，以加快构造 p1 的整个过程。全素数解(p，1⋯， p)h 在所有可能的候选解(p1，⋯， ph)满足 p1=z mod lcm(4,a1，⋯，at,k2，⋯， kh)和 PI=ki(p1-1)+ 1 对于 I = 2，⋯， h 可以使用关于大小为 L = lcm(4, a1，⋯)的质数分布的标准启发式来估计。， at,k2，⋯、kh);大致为 1/(logh(L)•Phi=2log(ki))。

注意，集合 A 越大，在决定 1pl 的条件下模量 L 就越大。因此，如果 A 包含许多碱基，那么更大的 pi will，因此更大的 n 将会产生。此外，全素数解的密度会降低。举个例子，在分析 Maple V.2 中的素数检验时，Arnault [Arn95]认为 h = 3 so n = pp1p2and3 A = {2,3,5,7,11} (so t = 5);他在 k= 132 和 k= 4133 的情况下工作，最终得到了这样的条件：

P1 = 827443 mod 49249 20。

对于 p_1 = 286472803，这产生了一个通过 Maple 的固定基 Miller-Rabin 素性测试的 29 位十进制数字合成。

我们给出了一个描述 n 的形式为 n = $pp_1p_2for_3$ 的方法的简短示例，其中前 10 个素数是米勒-拉宾非证人。也就是说，我们的目标 A =$\{2,3,5,7,11,13,17,19,_{?}23_{?},29\}$。

我们首先生成素数 p 对 4a 取模的剩余集合 $_a$，使得 $\left(\frac{a}{p}\right) = -1$ for

每个碱基 a∈a。表 3 给出了所选 $_a$ 集合 a 的集合 s。

表 3。值 a 和素数 p 的模 4a 残基的子集 $_a$ 使得 $\left(\frac{a}{p}\right) = -1.$

一个	某人
2	{3、5}
3.	{5、7}
5	{3, 7, 13, 17}
7	{5, 11, 13, 15, 17, 23}
11	{3, 13, 15, 17, 21, 23, 27, 29, 31, 41}
13	{5、7、11、15、19、21、31、33、37、41、45、47}
17	{3、5、7、11、23、27、29、31、37、39、41、45、57、61、63、65}
19	{7, 11, 13, 21, 23, 29, 33, 35, 37, 39, 41, 43, 47, 53, 55, 63, 65, 69}
23	{3, 5, 17, 21, 27, 31, 33, 35, 37, 39, 45, 47, 53, 55, 57, 59, 61, 65, 71, 75, 87, 89}
29	{3、11、15、17、19、21、27、31、37、39、41、43、47、55、69、73、75、77、79、85、89、95、97、99、101、105、113}

我们现在设 k_2 = 41, k_3 = 101;这些是所有 a∈a 的互质。我们找到满足要求的 s 的子集 $_a$:

$$p_1 \pmod{4a} \in \bigcap_{i=1}^{h} k_i^{-1}(S_a + k_i - 1).$$

这就给了我们一组对每个 a∈a 取模 4a 的余项，$_1$ 这些余项 p 必须满足。我们为表 4 中的前 10 个质数给出了一个这样的例子。

一个	Th i=1 ki−1(Sa + ki− 1)	模
2	{3、5}	8
3.	{7}	12
5	{3, 7, 13, 17}	20.
7	{15}	28
11	{21 日 23}	44
13	47}{21 日	52
17	{5、29、31、39、63、65}	68
19	{33, 37, 39, 47, 69}	76
23	{31, 47, 57, 87, 89}	92
29	{19、37、41、55、77、95、99、113}	116

表 4。值 a 和集合 $\text{Th}_{i=1}\ k_{i-1}$ $(S_a + k_i - 1)$当 k_2 = 41 和 k_3 = 101 时。

然后我们需要选择一个剩余的 z_a per 集合。这个选择是任意的，但我们注意到，并不是所有的选择组合都会导致一个解决方案。我们在表 4 中用粗体给出了一组不错的选择的例子。

然后，根据我们对 k 值的选择 $_i$ ，我们有两个额外的条件需要添加。这些可以写成:

$$P1 = k3 - 1 \mod k2 \text{ 和 } P1 = k2 - 1 \mod k3$$

在我们的例子中，我们选择 $k_1 = 41$ 和 $k_2 = 101$，这给我们:

$$p_1 \equiv 28 \pmod{41} \quad \text{和} \quad P1 \equiv 32 \quad 101 \text{ 年}$$

然后，我们可以使用中国剩余定理同时求解表 4 中加粗项所隐含的 10 个条件和上面的两个条件。在这种情况下，我们就有了解:

$$P_1 \equiv 362530308344 83 \mod 107163998661720。$$

总理

$$p_1 = 1424453871614154824048263654181759622666891330061 63$$

满足这个条件，并产生质数

$$p_2 = 584026087361803477859788098214521445293425445325 2643$$

$$p_3 = 143869841033029637228874629072357721889356024336223 63$$

使乘积 n = $pp_{21}pi$ 是 3 一个 512 位的数，它是一个米勒-拉宾伪素数，其基底为 2、3、5、7、11、13、17、19、23 和 29。

B 一个大的强 Lucas 伪素数

使用我们在 3.2.1 节中描述的 SAGE 实现方法，我们构造了一个形式为 n =$p_1p_2p_3$ 的 n，其中 pi = ki(p1+1)- 1 与(k2,k3) =(31,43)和

$$p_1 = 2^{576} \cdot 0\ x000000000000000000000bc508ae6dacc43b138c0e9f22d$$

$$+ 2^{384} \cdot 0\ xfb99b146bedd0ac93f84e8cfe2780a881fdbad85918a6b75$$

$$+ 2^{192} \cdot 0\ xbd3af841123bad7438fe08c5433ec8b5fa7b0a1b149876bf$$

$$+ {}^{0}\ x5af73cd9a608317066029e0cff4171ce336ff0b666344757 \cdot 0$$

然后 n = $pp_{21}pi_3$ 是塞尔弗里奇 a 法参数选择的 2050 位强 Lucas 伪素数。

C 构建 GMP 伪素

回想一下，我们的工作对象是形式为 x = kM + 189 的候选素 x，然后考虑 n = (2x + 1)(4x + 1); 我们选择 x，使 2x + 1 和 4x + 1 都是素数，我们选择 M 作为集合 P ={2,3，…373}。我们在这里论证这种构造。

首先，注意 2x + 1 = 2kM + 379，而 4x + 1 = 4kM + 757，其中 379 和 757 都是质数。考虑对 M 中的每一个‘质数因子 p 取 2x+1 模，我们看到 2x+1 = 379 mod p 6= 0 mod p，因为 p < 379; 类似地，我们得到 4x+1 = 757 mod p =0 mod p。因此，没有这样的 p 可以除 2x +1 或 4x+1，所以这些数字不能被乘积 M 中的任何质数整除(即第一个质数)。因此，在随机选择 k 和 x = kM + 189 的情况下，可以得出 2x + 1 和 4x + 1 比随机选择 x时更有可能是质数。对效果的分析涉及应用容斥原理来确定有多少数字被这个过程“筛掉”。我们在这里省略了完整的分析，但请注意，对于密码学上有趣的大小和` = 69 的数字，我们在构建我们的 1024 位 n 示例中使用，其效果是将每个数字的质数概率从 1/ ln x增加到大约 5/ ln x。由于我们有两个数字 2x + 1,4x + 1，它们的质数在 x的选择上基本上独立，相比于尝试随机 x值的直接方法，这产生了 25 倍的性能提升。

接下来，我们考虑以 a = 210 为底的 n 的费马检验，假设因子 2x + 1 和 4x + 1 是质数。这个检验计算 am^{n-1} od n 的值，并将其与 1 进行比较。现在

$n - 1 = (2x + 1)(4x + 1) = 8x^2 + 6x = 2x(4x + 3)$, so we obtain:

$$a^{n-1} = (a^{4x+3})^{2x} = 1 \mod 2x + 1$$

和

$$a^{n-1} = a^{8x^2+6x} = (a^{2x+1})^{4x} \cdot a^{2x} = 1 \cdot a^{2x} = a^{2x} \bmod 4x+1.$$

在这里，我们反复使用了费马小定理(即对于素数 p，$a^{p-1} = 1 \bmod p$，而 $a \neq 0 \bmod p$)。

由此可见，当且仅当 a 是对 4x +1 模的二次剩余时，$a^{n-1} = 1 \bmod n$。由此可见，对于大约一半可能的基数 a，n 通过了以 a 为底的费马检验(因为大约有一半的 a 模 n 值是模 4x + 1 的二次剩余)。

现在我们用 a = 210 = 2 • 3 • 5 • 7 这一事实来写:

$$\left(\frac{210}{4x+1}\right) = \left(\frac{2}{4x+1}\right)\left(\frac{3}{4x+1}\right)\left(\frac{5}{4x+1}\right)\left(\frac{7}{4x+1}\right).$$

由于 M 是偶数，我们可以写成 4x+1 = 8k(M/2) + 757 = 5 mod 8，因此$(\frac{4x+12}{}) = -1$。同样$(\frac{4x+1}{3}) = (\frac{4kM+757}{}) = (\frac{757}{}) = (\frac{1}{}) = 1$，其中我们使用高斯二次律和 3 3 3

3 |。同样，我们得到$(\frac{4x+1}{5}) = -1$，$(\frac{4x+1}{7}) = 1$。结合一切，我们最终得到

$$\left(\frac{210}{4x+1}\right) = (-1) \cdot 1 \cdot (-1) \cdot 1 = 1.$$

我们得出结论，以 a 为底 210 的给定形式的 n 的费马检验总是通过的。

D 是 Mini-GMP 的伪素

使用我们在 3.1.4 节中描述的复合固定基技术的 SAGE 实现，我们构建了形式为 $n = p_1 p_2 p_3$ 的 n，其中 pi = ki(p1 – 1)+1 with(k2, k3) = (10937,11257) and

$$p_1 = 2^{960} \cdot 0\ x000000000000000000000000000000000000000000002e394$$

$$768 + 2 \cdot 0x1a2fe4aa9e66358347f63732494d08635ccc9ae0a3c17764576$$

$$+2 \cdot 0xa8e266f4d26758ab804a702c235f63b1e109a81fc007f94b$$

$$+^{384} \cdot 0\ xec5158f231a30b1cbf96a7fc444c09be62f5a809f049cc5d$$

$$192$$

$$+2 \cdot 0xe94b84275c38885c9b61a6bdc44111501527722a8ac87ea2\ 0$$

$$+2 \cdot 0xa5d4498caa2d9d07b34001a508fa53063991206268c547d7。$$

这产生了一个 2960 位的复合 n，保证通过任何数字，直到并包括 t = 101 轮 Mini-GMP 的质数测试。

E Java 代码清单

我们包含了 java.math.BigInteger 类中的 primeToCertainty 函数的源代码。清单 1.1。OpenJDK10

java.math.BigInteger 函数 primeToCertainty

```
布尔值 primeToCertainty (int c ertainty, Random Random) {
    Int 轮数= 0;
    intn =(数学。min (c e r t a in t y, in t e g r.MAX 值 – 1)+1)/2;
    //确定性和轮数之间的关系在标准草案 ANSI X9 中给出 8 0，“素数//数字生成、素数检验
    和素数证书”。int s i z e in B i t s = this。bitLength ();
    if (s i z e in B i t s < 100) {
        轮数= 50;
        Rounds = n < Rounds ?N:回合;
```

```
if (sizeinBits < 256) {
```

```
        轮数= 27;
    } else if(sizeInBits < 512) {rounds =
        15;
    } else if(sizeInBits < 768) {rounds =
        8;
    } else if(sizeInBits < 1024) {rounds =
        4;
    } else {
        回合数= 2;                          随机)&& passesLucasLehmer();
}
```

F JSBN 的伪素数例子

使用我们在 3.1.2 节中描述的 SAGE 实现的方法，A 包含前 1000 个素数，我们构造了一个形式为 n = ppp$_1$ 的 4279 位 n，$_2$ 其中 $_3$p= k(p− 1$_i$)+$_i$1 与 $_1$（k, k) =(1013$_2$, 20$_3$53)和

$$p_1 = 2^{1344} \cdot 0\ \text{x00000000000000000000000000083dda18eb04a7597ca3}$$
$$+ 2^{1152} \cdot 0\ \text{xc6bc877df8a08eec6725fa0832cba270c42adc358bc0cf50}$$
$$+ 2^{960} \cdot 0\ \text{xc82cb10f2733c3fb8875231fc1498a7b14cb675fac1bf3c5}$$
$$+ 2^{768} \cdot 0\ \text{x127a76fc11e5d20e27940c95ceba671fe1c4232250b74cbd}$$
$$+ 2^{576} \cdot 0\ \text{xf8448c90321513324c0681afb4ba003353b1afb0f1e8b91c}$$
$$+ 2^{384} \cdot 0\ \text{x60af672a5a6f4d06dd0070a4bc74e425f3eae90379e57754}$$
$$+ 2^{192} \cdot 0\ \text{x82d26e80e247464a4bb817dfcf7572f89f8b9cacd059b584}$$
$$+ {}^{0}\ \text{x0e4389c8af84f6a6ea15a3ea5d62cb994b082731ba4cde73} \cdot 0$$

这产生了一个 n，通过 JSBN 对任何确定性参数 t 的质数测试，保证该 n 被声明为质数。

G Cryptlib 的伪素数示例

使用我们在 3.1.2 节中描述的 SAGE 实现的方法，A 包含前 100 个素数，我们构造了一个形式为 n = ppp$_1$ 的 2315 位 n，$_2$ 其中 $_3$p= k(p− 1$_i$)+$_i$1 与 $_1$（k, k) =(641,$_2$677$_3$)和

$$p_1 = 2^{576} \cdot 0\ \text{x24a027808260908b96d740bef8355ded63f6edb7f70de9a9}$$
$$+ 2^{384} \cdot 0\ \text{xb99c408f131cef3855b4b0aea6b17a4469ed5a7ec8b2be62}$$
$$+ 2^{192} \cdot 0\ \text{x66c3a9eae83a6769e175cb2598256da977b9e191b9b847a7}$$
$$+ {}^{0}\ \text{xe2cf4750d9bc2d64ccd3406f5db662c22c3fc65e3c56eff3} \cdot 0$$

对于 Cryptlib 的素数测试执行的任何有效轮数 t,n 都被声明为素数。

H Apple CommonCrypto 的伪素数例子

使用我们在 3.1.2 节中描述的 SAGE 实现的方法 A 包含前 16 个素数，我们构造了形式为 n = ppp$_1$ 的 3 个复合 n，$_2$ 其中 $_3$p= k(p− 1$_i$)+$_i$1 with$_1$ （k, k) =(113,$_2$173$_3$)其中：

p_i = 0 x32972d4e607a45f57d7144df60a7abf8b473a1680b

30.

产生一个 512 位的 n，

$$p_1 = 2^{192} \cdot 0\ x0000000000151452e0a832f27b9eead0000000000000000$$

$$+ \ ^{0} \cdot 0\ x0000000000000000000000000aff3796792e7ceb8d55206a3$$

产生 1024 位的 n，并且

$$p_1 = 2^{576} \cdot 0\ x0000000000000000000000032972d4e607a45f57d66c00000$$

$$+ \ ^{0} \cdot 0\ x0000000000000000000000001f2a7b5c6a50fc1e38aae911b$$

产生一个 2048 位的 n。

这些组合总是被 CommonCrypto 声明为素数。

我举例说明 Apple corecrypto 和 CommonCrypto, LibTomMath, LibTomCrypt 和 WolfSSL 的伪素数

使用我们在 3.1.2 节中描述的 SAGE 实现的包含前 256 个素数的方法，我们构造了一个形式为 n = ppp$_1$ 的 7023 位 n，$_2$ 其中 $_3$p= k(p − 1$_i$)+$_i$1 与 $_1$ (k, k) =(2633$_2$, 58$_3$81)和

$$P1 = \ ^{22304} \cdot 0x000000000000000000000000000000000000000000000001$$
$$e46d6a8\ 2112$$
$$+2 \cdot 0x4d42d684ddb3415e871b661303b1c60f0388dfb9e525f8bc$$
$$+ \ ^{1920} \cdot 0\ x51c9de3c9f45627608de2f75dee580d9d4d97cab6fa86dad$$
$$+ \ ^{1728} \cdot 0\ x9e6bbfd721f297472480a9bed9508aa884bda9dc56833752$$
$$^{1536}\ +2 \cdot 0xa452f960facc9b65f6962cb26131b42650c29c8735083c7e\ ^{1152}$$
$$+2 \cdot 0x6c3a220d77d1cbe7f9628885a7b79465287d4b02ad546007\ 960$$
$$+2 \cdot 0x1d43306a8813836de5ccd162fbeca4f117552dba01975451\ 768$$
$$+2 \cdot 0x2f7684e32b0377e76f87b96906f8fa276381db612f76c2c7\ 576$$
$$+2 \cdot 0xdd97ab4380042c991a4719884377c70065a3614237a41289\ 384$$
$$+2 \cdot 0x24a1017fbb529443b0ad43c5424753db5b518cee5a1fcd87\ 192$$
$$+2 \cdot 0xea038ffcad33380db1d89cd4e0b15b480cf0c62e8999924d\ 0$$
$$+2 \cdot 0x0284af806081ea106f35f85a664456166b864650ef034cf3。$$

对于 Apple corecrypto 和 Common- Crypto, LibTomMath, LibTomCrypt 和 WolfSSL 库的任何有效轮数 t, n 都被声明为素数。

同样使用 3.1.2 节中描述的方法，但现在使用 A 包含前 40 个素数，我们可以构建一个形式为 n = ppp$_1$ 的 1024 位 n，$_2$ 其中 $_3$p= k(p − 1$_i$)+$_i$1 与 $_1$ (k, k) =($_2$233,241$_3$)和

$$p_1 = 2^{192} \cdot 0\ x000000000000e17516504450e648b6aedb0c0784e17dda33$$

$$+ \ ^{0}\ x63e1956a843076a9e5b6d15a819cf0907a96154d47662d0b \cdot 0$$

这个 n 保证被 mp prime 声明为素数，t≤40，因此也保证被 mp prime 声明为素数，就像 LibTomCrypt 1.18.1 和 WolfSSL 3.13.0 中 t 的相同值一样。同样适用于 Apple corecrypto 和 CommonCrypto。

J 1.8 之前 GoLang 的伪素数示例

使用 4.2 节中描述的方法，我们构造了一个 1024 位的复合数 n，在 1.8 之前的版本中，通过 GoLang 的素数测试将其声明为素数，对于 t≤13，该方法 100%成功。我们把

$$p = 2^{960} + 0\ x000000000000000000000000000000000ff7d428a8a9f9ffc$$

$$+2^{768} + 0x2ea178501115ec855f1154c054f5f67e15967a139a92fe15^{576}$$
$$+2^{384} + 0xddf2c49b044820ea8c58551b74f81b45b116da4e1f11b926$$
$$+2^{192} + 0x93e0cdc58006bc2052eb9b2fc32c71dd041d1907225e2814$$

$$+2^{0} + 0xebe18736f626fea57c965b67b296a6461455226b39aba263$$

$$+2^{0} + 0x3faeb483847a715c6a01d8d0e401a4aaf8f3d22121fd142f。$$

K 一个为 Botan 设置的恶意 DH 参数的例子

使用我们在 3.1.3 节中对该方法的 SAGE 实现，我们构建了一个 160 位 q，形式为 q =q1q2q3，其中 qi=ki(q1− 1)+ 1 与(k2, k3)=(61, 101)和 q1=537242417098003。

这个 q 被 Botan 的验证群函数声明为质数，概率为 1/4096。通过在 p = kq + 1 中设置 k = 2^{864} + 134，我们得到一个素数 p，因此通过设置生成器 g 为:

$$g = 2^{960} + 0\ x000000000000000000000000000000075ead4f9fa60a06e$$
$$+2^{768} + 0\ x0787a1e0708f5e2055b2899691f7dd73303d5643e57b1636$$

$$+2^{576} + 0x66ce328086bd6a0df756175c35549ba7a5ffe517036c0ef1^{384}$$
$$+2^{192} + 0x44a9542f698255efb66cda28b0b8a5ebebf2c0892f8147d3$$

$$+2^{0} + 0\ x72083822a36098addcd30a1767ccefaae65d1dcd6b45de92$$

$$+2^{0} + 0x09047326d40b622af6a76218664ba3df13eb0fead02d772a$$

我们得到一个参数集(p, q, g)使 g 产生 q 阶的子群，这个集合被 Botan 的验证群函数接受的概率是 1/4096。g 生成的子群中的 DLP 可以在 49 位、55 位和 56 位因子 q、q 和 q 的 q_1 上分别使用 pohligi - hellman 算法求解，$_{32}$其代价由最大素数因子支配，约为 2^{28} 次运算。

L 数学软件包的细节

L.1 岩浆

Magma V2.23-9 [BCP97]是一个数学软件包，设计用于代数、数论、代数几何和代数组合数学的计算。

分析。Magma 提供了一个素数测试函数，根据调用时给出的参数，可以调用素数证明算法，也可以调用他们所谓的概率素数测试。素数测试的主要函数调用是 IsPrime(n:- Proof)。更严格的素数证明方法是基于 ECPP(椭圆曲线素数证明)方法[AM93]的一个实现，默认使用，除非测试的数字大于 34 × 10^{13} 或参数证明= False。在这种情况下，取而代之的是调用概率素数测试 IsProbablePrime。默认情况下，它由随机基数的 20 轮米勒-拉宾测试组成。通过将可选参数 Bases 设置为某个值 B，使用的基数是 B 而不是 20。

Pseudoprimes。第 3 节生成的伪素数仅试图克服概率素数测试，并不是为了克服 ECPP 等素数证明方法而设计的。

然而，如果参数被设置为调用带有默认参数的概率素数测试，那么 3.1 节中方法生成的复合数被错误声明为素数的概率为 2^{-40}。这个概率被正确地暗示为最坏情况下的文档给出的这个函数。

L.2 枫

Maple 2017 [Wat17]是由 Maplesoft 开发的计算机代数系统，为数学、数据分析、可视化和编程提供了一个通用的软件工具。

分析。Maple 中的质数测试在待测试的候选 n 上称为 isprime(n)。文档指出，“如果 n 在一个强伪素数测试和一个 Lucas 测试中被证明是复合的，则返回 false。否则返回 true”。该函数在调用 gmp isprime(n)之前，先对一系列小素数进行一些试验除法。如果 gmp isprime(n)的结果是 1(即数字“可能是素数”)，被测试的候选 n 大于 $5 \times 10^9 \approx 2^{33}$，那么 isprime 将继续对 n 进行 Lucas 测试。在所有其他情况下，Lucas 测试被省略。

虽然我们不能直接检查 gmp isprime(n)的代码(因为 Maple 是专有软件)，但我们可以通过对我们自己的输入 n 调用它来反向工程该函数，并评估它的执行情况。Maple 的文档说明它执行 Miller-Rabin 测试并使用 GMP 来实现此功能，但由于在 GMP isprime(n)中没有其他代码指示 Miller-Rabin 测试，我们推断 Maple 正在调用 GMP 的函数 mpz probab prime p- (n, reps)。由于 gmp isprime(n)只接受一个参数，我们推断 Maple 将 reps 的硬编码值传递给 gmp。我们能够验证 reps 的值实际上是 5。我们通过使用 4.2 节中描述的方法来生成由 mpz probab 质数 p (n, reps)声明为质数的各种比特长度的合数，对于 reps = 1,2,3,4,5。对于只能在大多数 reps = 4 通过的复合数，Maple 的 gmp isprime 正确地将这些识别为复合数。但对于通过了 reps = 5 的复合材料，该函数错误地将它们声明为 prime。

Pseudoprimes。在测试数字 $n \leqslant 5 \times 10^9$ 时，isprime 充当了米勒-拉宾测试的确定性版本。我们通过对所有 $n \leqslant 5 \times 10$ 的数调用 mpz probab 素数 p(n,5)，并将结果与低于 5×10^9 的素数列表进行比较来验证这一点。GMP 为每个 n 选择的不同碱基集是这样的，没有低于这个阈值的复合材料被 mpz probab 素数 p 声明为素数，代表为> 3。然而，任何改变(有缺陷)的方式，GMP 目前选择其测试的基础可能实际上使 Maple 的 isprime 功能不太准确(不再确定)的 $n \leqslant 5 \times 10!^9$

为了欺骗 Maple 对大于 5×109 的数字的素数测试，我们需要一个通过 Lucas 测试和 5 轮 Miller-Rabin 测试的复合 n。目前我们还不知道有这样的 n。

L.3 最大值

Maxima 5.41.0 [Mac18]是由 Macsyma 集团开发的免费开源计算机代数系统。Maxima 是一个通用系统，包括用于各种数学函数以及操纵符号和数值表达式的工具。

分析。Maxima 提供的素数检验就是 primep(n)函数。当测试 n 小于 341550071728321($\approx 2^{49}$)时，使用米勒-拉宾检验的确定性版本。这是通过使用一组已被验证没有复合材料被错误地声明为素数的基来调用重复轮的米勒-拉宾测试来实现的。这些是在[Jae93,McC97]中定义的，因此通常可以用来创建小于 264 的数字的确定性测试。

当测试大于 341550071728321 的 n 时，primep(n)执行 25 个随机基础 Miller- Rabin 测试，然后进行一个 Lucas 测试。Maxima 用于基选择的源然后由 Maxima 随机数生成器提供，这是 Mersenne twister MT 19937 [Mac17]的一个实现。

Maxima 的文档正确地说明了“非素数 n 通过 Miller-Rabin 测试的概率小于 1/4。使用质数测试的默认值 25,n 是复合的概率要比 10^{-15} 小得多。”

Pseudoprimes。当测试数 n < 341550071728321(≈ 2^{49})时，primep(n)函数是确定的，因此不可能出现伪素数。如果 n > 341550071728321，那么 Miller-Rabin 测试和 Lucas 测试的结合意味着测试中没有已知的伪素数。

L.4 SageMath

SageMath 8.2(或简称 Sage)是一个免费的基于 python 的开源数学软件系统，最初由 William Stein [S+17]创建，但现在由许多志愿者开发。Sage 提供了代数、组合数学、数值数学、数论和微积分等领域的数学函数工具包。

分析虽然在 Sage 中有许多方法可以用来测试数字的素数，但旗舰函数是在/src/ Sage /rings/integer.pyx 中找到的 is_prime(n, proof)。如果调用时将 proof 的值设置为 True(启动 Sage 时的默认值)，该函数将执行 use a provable priality test。如果设置为 False，它将使用强伪素数测试，取而代之的调用是伪素数(n)。

当 proof = True 时调用的“可证明的素数测试”是 PARI [The18b] isprime 函数。然后使用 Baillie-PSW 检验、Selfridge“p − 1”和 Adleman- Pomerance-Rumely-Cohen-Lenstra (APRCL)的组合。文档中指出，当测试一个“有 1000 位以上”的质数时，这种测试可能会“非常慢”。

当 proof = False 时调用的“强伪素数测试”准确率较低，但速度快得多，因此在测试大的候选者时很可能是一个选择。然后对候选项进行 PARI’s is pseudprime (n)测试，该测试由 Baillie-PSW 测试组成。

-

Pseudoprimes。由于正在进行 bailli - psw 测试，我们知道没有合数被 SageMath 错误地声明为质数的证明布尔值。

L.5 SymPy

symy [Sym17b]是一个免费开源并被广泛使用的符号计算 Python 库，它提供了类似计算机代数系统的功能。

分析。SymPy 提供了素数检验 isprime(n)，它和 Maxima 一样，在测试候选 n < 2^{64} 时，使用选择基来执行米勒-拉宾的确定性版本。我们将考虑 symy 最新的几个版本(symy 1.0 和 symy 1.1)，因为最近对函数 isprime 进行了重大更改。

symy 1.0 在 2017 年 7 月发布 1.1 之前，symy 1.0 以同样的方式进行了 isprime 中发现的质数性检验。经过一些初步的试验划分后，如果没有发现因子，该函数将调用确定性版本的米勒-拉宾测试，使用[Jae93,McC97]中描述的基。对于大于≈ 2^{53} 的数字，该测试将调用额外的 Miller- Rabin 轮数。在 2009 年的 0.6.6 之前的所有版本中，这将简单地在基础{2,3,5,7,11,13,17,19}上执行 8 轮米勒-拉宾。在 0.6.7 版本[Sym17a]中，这增加到 46 轮米勒-拉宾，使用前 46 个质数作为基数。之后的测试基本没有变化，直到 2017 年的 1.1 版本。

从 2017 年 7 月起，symy 1.1 对 isprime 函数进行了修订，删除了 46 个基上的最终米勒-拉宾测试，并将其替换为 2.4 节所述的贝利- psw 测试。

Pseudoprimes。symy 1.0 和所有之前的版本都容易受到 3.1.2 节中方法生成的合数 n 的影响。当最终的米勒-拉宾检验是基于前 8 个质数时，这些数字构造起来是微不足道的，但即使在 0.6.7 中进行了更改之后，1.1 之前的所有版本也会错误地宣布以这种方式生成的合数为质数。例如，使用 3.1.2 节的方法，我们能够构造出一个形式为 n = pp$_1$p$_2$1024 位的 n，它 $_3$ 对 symy 在 1.1 之前的所有版本中所选择的所有碱基都是伪素数。在这里

$$p_i = k_i(p_1 - 1) + 1 \text{ with } (k_2, k_3) = (241, 257) \text{ and}$$

$$p_1 = 2^{192} \cdot 0\ \text{x000000000000f8ae31e07964373e4997647e75fa186dd5e7}$$

$$\pm\ ^{0}\ \text{xe42ada869da0b3a333813f8102b1fb5f20623d6543e78a3b} \cdot 0$$

由于 SymPy 1.1 引入了 Baillie-PSW 测试，我们无法再生成由 SymPy 声明为 prime 的复合材料。

L.6 Wolfram Mathematica

Wolfram Mathematica 是由 Wolfram Research 开发的一个涵盖科学、工程、数学和计算领域的计算软件包。当前版本为 Mathematica 11.3 [RI18]，具有与 Wolfram Alpha 的内置集成功能。

分析。Mathematica 提供了内置素数测试 PrimeQ，据说可以使用基数 2 和 3 进行两次米勒-拉宾测试，并结合“卢卡斯伪素数”测试。由于源代码不是开源的，我们无法验证“Lucas 伪素数”测试中使用的参数。我们注意到，文档中引用了 Baillie 和 Wagstaff [BW80]，而 Selfridge 的参数正是来源于此。该函数的文档也表明，这个过程只知道对 $n < 10^{16}$ 是正确的，并且“可以想象，对于更大的 n，它可以声称一个合数是质数”。

Pseudoprimes。由于正在进行 Baillie-PSW 测试，我们知道没有复合材料被错误地声明为质数。