16 | 为什么说随机数都是骗人的?

2020-12-30 范学雷

实用密码学 进入课程 >

讲述：范学雷
时长 15:07 大小 13.86M

你好，我是范学雷。

今天，我们来讲一个之前总提到的知识点：随机数。比如，我们在讲初始化向量时，提到可以使用随机数来避免重复。事实上，随机数还有更广泛的用途。可以这么说，**只要使用密码学的技术，我们一定会用到随机数。**

说起来也很巧，当我准备随机数这篇文章思路的时候，朋友圈里有一条感慨，说道：

> “人能做出两种随机数。一种是欺骗自然的，算的很快，状态域超大，统计上随机；一种是欺骗人的，通常无法预测下一个是多少，通常算的慢”。

其实，欺骗自然的随机数，也是用来骗人的。这么说来，难道所有的随机数都是骗人的？为什么说随机数是骗人的？骗人的随机数还有现实的意义吗？这是这一次我们要讨论的问题。

今天的内容有点多，不过我给你多分了几个小模块，详细地分析了随机数的每个问题，这样会让你更清楚地、一次性地解决随机数的相关知识点。好，我们开始！

真的有随机数吗？

第一个问题就是，随机数真的存在吗？

数这个东西是人类的发明，不是自然界产物。所以，当我们说随机数的时候，一定是人造的东西。随机是自然世界的主旋律，你不可能找到相同的两片叶子，也不可能两次踏进同一条河流。

但是，在科学的世界里，随机从来就不存在，或者说随机都被当做噪音简化过滤掉了。因为科学研究的是确定性、可重复的科学。不确定的、不可重复的随机数，不属于科学的范畴，也走不进科学的领域。

> 冯·诺伊曼：若人们不相信数学简单，只因他们未意识到生命之复杂。

想一想计算机吧。计算机之所以存在，就是因为它的确定性。同一台计算机，相同的数据输入，一定要产生相同的结果。两台不同的计算机，相同的数据输入，也一定要产生相同的结果。

这是计算机存在的基础。怎么会有随机的数据和计算结果呢？所以说，在计算机的世界里，随机数的确是不存在的。

> 冯·诺伊曼：任何考虑使用数学产生随机数的方法都是不合理的。

这是不是在开玩笑？那我们前面多次提到过的随机数算是怎么回事？其实，我们能够创造出来的，都不是真的随机数。这些所谓的随机数，要么是骗机器的，要么是骗人的，或者两者都骗。严格一点，我们把它们叫做伪随机数。

既然没有真正的随机数，当然也就没有必要区分真的随机数和假的随机数。所以，我们通常把伪随机数简单地叫做随机数。只要我们明白，没有真正的随机数，一个更顺溜的名字节省了很多口水，却不会引起太多的麻烦。以后没有特别声明，我们讨论的随机数，都是伪随机数。

那么，既然没有真随机数，假随机数的边界在哪里呢？什么样子的伪随机数能够骗得过机器、骗得过人呢？要想回答这个问题，我们先要了解什么样的数才算是随机数。

随机数怎么来的？

先来问你一个问题，“123456”是一个随机数吗？那“123456”比“654321”更随机吗？我们很容易有这样的问题。遗憾的是，这样的问题在密码学里是没有实用意义的。一个数的随机性说的永远是还不存在的、未来的那个数的不可预测性。

即便我们知道了随机数产生的机制，以及所有的已经产生的随机数，我们也无法预测下一个随机数是什么。这就是随机数的第一个特点：不可预测。

那怎么检查未来的那个数是随机的呢？未来的那个数还不存在，当然也没有直接的方法。间接的办法，就是产生很多随机数，然后检测这些随机数能不能通过所有的随机性统计检验。在数学上，有很多种随机性统计检验的理论和方法，我们就不去讨论这么折磨人的数学问题了。

记住一点，只要有一个随机性统计检验方法没有通过，这个随机数产生的机制就是有问题的。换句话说，对于合格的随机数，我们没有办法验证下一个随机数不随机。这就是随机数的第二个特点：无法证伪。

到这里，不知道你有没有找到一对矛盾的地方。我们上面不是说过吗，“同一台计算机，相同的数据输入，一定要产生相同的结果”，那怎么才能够产生不可预测的随机数呢？

产生随机数的诀窍，就是把“相同的数据输入”拆成两部分。一部分是私密的数据，一部分是公开的数据。如果我们能保护住私密的数据，不让人知道，也不让机器拷贝，那么下一个数据是什么，对于别人或者别的机器来说，可能就是无法预测的。

当然，对于秘密的持有者，下一个数据是什么是确定的，不是不可预测的。那么，随机性统计检验呢，当然也得假装这个私密数据不存在。

这个私密数据的质量和计算输出的算法，就决定了随机数的质量。

幸运的是，现在几乎所有主流的计算机，都内嵌了随机数发生器；主流的操作系统或安全类库，都提供了随机数的接口。一般来说，我们不需要自行设计随机数算法，保守私密数据的秘密。

但是这并不意味着我们就可以高枕无忧了。随机数的应用，我们还需要注意两点：

第一点是，随机数的产生可能会阻塞；

第二点是，随机数的强度要匹配。

阻塞的随机数有什么麻烦？

随机数也是有质量要求的，为了保证随机数的质量，随机数发生器的设计需要收集随机信息，比如计算机的噪音，周围的温度，CPU 的状态，硬盘的状态，用户的行为等等。

这些信息收集，是需要时间的，所以有的时候，产生下一个随机数的时候，就会阻塞。阻塞的时间还不确定，有的可能是纳秒级别的，有的可能是秒甚至分钟级别的。而对于一个高吞吐量的系统，微秒级别的阻塞可能都是不能忍受的。

随机数阻塞会带来什么麻烦呢？一般来说，在一个应用程序里，密码学算法要保护的都是关键的信息或者流程，比如说身份认证或者数据加密。如果随机数有阻塞，这个应用程序就会停顿。应用程序的停顿，会使得占有的资源不能及时地释放，降低系统的效率，增加用户等待时间。

更要命的是，随机数阻塞的时间不确定，有时候时间长到无法忍受，有时候短到毫无影响。除了影响系统的效率和吞吐量之外，还会影响用户体验的一致性，影响程序运行的一致性，也使得出现的问题难以排查。所以，除非万不得已，**我们尽量不要使用阻塞的随机数发生器**。

非阻塞的随机数还能随机吗？

不使用阻塞的随机数发生器，也就意味着有非阻塞的随机数发生器。那么，难道非阻塞的随机数发生器就不需要收集随机信息了吗？不是的，非阻塞的随机数发生器也需要收集随机信息。**区别在于随机信息怎么使用，以及使用的频率。**

阻塞的随机数发生器的每一个随机数，都要损耗随机信息；而非阻塞的随机数发生器，可能仅仅在开始的时候就损耗随机信息（比如说一台计算机开机的时候），然后，随机数的发生就不再损耗随机信息了。只要不再损耗随机信息，就不会有收集随机信息带来的阻塞了。

不再损耗随机信息，还能保证随机数的随机性吗？

什么是确定性的随机数发生器？

有一类随机数的算法，叫做确定性的随机数发生器（Deterministic Random Bit Generator， DRBG），使用的就是单向散列函数。

怎么解释确定性？确定性说的就是相同的输入，有相同的运算结果。这也就意味着，对于确定性的随机数发生器来说，它的下一个随机数是确定的，是可以计算出来的，当然也是可以预测的。

计算结果可以预测，这还算什么随机数？

我们上面说过，产生随机数的诀窍，就是持有一部分私密的数据。随机数发生器持有私密的数据，所以计算结果对它来说，是确定的。如果我们不知道私密的数据，从随机数发生器外面看起来，下一个随机数还是能够做到貌似不可预测的。

怎么做呢？

怎么使用哈希函数实现随机？

你还记得我们前面讨论过的单向散列函数的性质和消息验证码的工作原理吗？**使用单向散列函数实现非阻塞的随机数发生器的关键，就是把这部分私密的数据，当做单向散列函数输入的一部分，然后再加入不会重复的数据**，比如序列数。

为什么这个办法是可行的呢？使用单向散列函数和有限的随机信息，随机性能够满足“不可预测”和“无法证伪”这两个检验指标吗？这需要我们回头再看看单向散列函数的特点。

第一个特点，单向散列函数正向计算容易，逆向运算困难。我们使用了私密数据作为单向散列函数输入的一部分，由于逆向运算困难，只要这部分私密数据保护得好，攻击者是没有办法通过运算结果计算出私密数据的。否则，也就意味着破解了单向散列函数。

第二个特点，单向散列函数运算结果均匀分布，构造碰撞困难。我们使用了不会重复的数据作为单向散列函数输入的另外一部分，由于运算结果均匀分布（也就是严格的雪崩效应），如果不知道私密数据，运算结果就是无法预测的。

同时，严格的雪崩效应，也就意味着每一次计算，输出数据的每一位都有 50% 的概率会发生变化。如果不知道私密数据，运算结果的随机性也是无法证伪的。

所以，你看，即使只损耗有限的随机信息，使用好单向散列函数，也能做出来合格的随机数发生器。

既然使用单向散列函数，也逃不掉单向散列函数的种种陷阱。所以，一般来说，我也不建议你自己设计、实现随机数发生器。现在大部分的密码学类库里，都应该支持了非阻塞的随机数发生器。**我们要优先使用这些已经经过验证的、成熟的设计和实现。**

使用这些类库时，有没有需要注意的事情呢？这是我们要重点关注的问题。其中，最重要的，就是安全强度的匹配。

随机数也有强度吗？

对于阻塞的随机数发生器来说。随机数每一位损耗的都是计算机收集的随机信息。**随机数的位数也就是它的安全强度。**所以，通常的，我们不担心它的安全强度。

但是对于非阻塞的随机数发生器，它只损耗有限的随机信息，通过单向散列函数来计算随机数。这也就意味着两个问题。

第一个问题是，单向散列函数本身是有安全强度的。相应的随机数发生器的安全强度，不会超过单向散列函数的安全强度。比如说，SHA-256 的安全强度是 128 位，那么使用它的随机数发生器的安全强度一般也不会高于 128 位。

第二个问题是，单向散列函数的输出长度是固定的，而随机数的长度可能并不匹配散列值的长度。SHA-256 的散列值长度是 256 位，如果要产生一个 512 位的随机数，就需要至

少两个散列值的拼接。

这会带来什么问题呢？通常地，一个 256 位的随机数就应该有 256 位的安全强度，一个 512 位的随机数就应该有 512 位的安全强度。但是，由于单向散列函数自身的限制，基于 SHA-256 的随机数发生器，它的安全强度不会超过 128 位，无论这个随机数是 256 位还是 512 位。这就带来了很大的困扰。

需要多长的随机数，我们就使用随机数发生器生成多长的随机数，而不需要考虑随机数的强度够不够，这是人们的使用习惯。 由于这样的习惯，要想漂亮地解决上面的问题就变得很棘手。

甚至，很多密码学类库都没有准备好解决这样的问题的接口，其中包括 Java 语言。Java 语言默认使用很高的安全强度，这样应用程序就不用担心当前会有实质性的安全问题了。但是从长远看，这样的解决方法还不够灵活，没有给应用程序提供更高安全强度的选项。

有哪些常见的随机数算法?

目前，有两类常见的随机数算法，它们是基于单向散列函数的 Hash-DRBG 和基于 HMAC 的 HMAC-DRBG。一般来说，Hash-DRBG 算法初始化的时候需要使用有限的随机信息。而对于 HMAC-DRBG 算法来说，这个有限的随机信息被保密的对称密钥取代了。

无论是随机信息还是已知的对称密钥，都是需要保密的信息。所以，使用保密的对称密钥取代随机信息，并没有安全上的妥协。

不过，使用保密的对称密钥取代随机信息还有一个好处。一般来说，对于应用程序来说，随机信息是透明的，只有机器知道，应用程序并不知道随机信息是什么。

所以，这样的随机数是应用程序无法复制的。而使用对称密钥，只要知道了对称密钥，应用程序就有复制随机数的办法。这一点，对于很多应用场景，都有着重要的意义。可以说，HMAC-DRBG 给了应用程序更多的灵活性。

无论 Hash-DRBG 还是 HMAC-DRBG，它们的安全强度都是由所使用的单向散列函数的安全强度决定的。**使用的时候，要注意安全强度的匹配问题，确保随机数能够提供足够的安全强度。**

上一讲结尾的时候，我们说过随机数是对称密码依赖的技术。除了初始化向量以外，随机数也是生成对称密钥的基础。该怎么使用随机数生成对称密码？我们下一次讨论这个话题。

Take Away（今日收获）

今天，我们讨论了和随机数概念相关的话题，比如，真的随机数存不存在？伪随机数有什么特点？我们还讨论了阻塞的随机数发生器和非阻塞的随机数发生器，以及怎么使用单向散列函数来实现随机函数。

我们还讨论了随机数的强度，随机数的位数也就是它的安全强度。还有两类常见的随机数算法，也就是 Hash-DRBG 还是 HMAC-DRBG。我们使用的时候，要注意安全强度的匹配问题，确保随机数能够提供足够的安全强度。

通过今天的讨论，我们要：

知道随机数发生器可能会阻塞，我们应该尽量使用非阻塞的随机数发生器；

知道随机数的算法也有安全强度的限制，我们要小心安全强度错配的问题。

思考题

今天的思考题，是一个动手题。

在你正在开发的项目中，或者你关注的开放源代码项目中，试着搜索一下随机数的使用。重点关注如下的问题：

随机数发生器使用了什么算法？

随机数的安全强度足够吗？

随机数发生器会不会阻塞？

如果你发现了问题，有没有改进的建议？

欢迎在留言区留言，记录、讨论你的发现和建议。

好的，今天就这样，我们下次再聊。

提建议

上一篇 15 | AEAD有哪些安全陷阱?

下一篇 17 | 加密密钥是怎么来的?

精选留言 (1)

写留言

John

2020-12-30

有了量子计算机就可以生成真正的随机数了。

展开

作者回复: 嗯，量子计算机对密码学既是打击，也是促进。

1