True Randomness Can't be Left to Chance: Why entropy is important for information security

Introduction and Motivation

Cryptography is fundamentally important for securing data, whether that data is in transit over the internet or at rest on a storage device. In the past, cryptography relied on algorithms and codes that were kept secret. This, however, proved impractical when parties who did not know each other in advance wanted to communicate securely. Modern encryption and authentication algorithms are public and have been extensively analyzed. Today, the security of cryptography depends primarily on having strong keys and keeping them secret. A key is strong only to the degree that it is hard to guess or – to put it another way – that it is random. Information security standards [1] require at least 112-bits of security strength for cryptographic keys, meaning that the effort for breaking them would be equivalent to trying at least 2^{112} possible keys as the best available attack. Although other factors such as cryptographic algorithm soundness and secure implementation play a big role in the security cryptography provides we focus our attention on the randomness of keys – an issue often misunderstood or neglected by the crypto community at large.

Deterministic random bit generators (DRBGs), also known as pseudo-random bit or number generators, are used to generate keys, but the sequence of numbers they generate can be traced predictably to the seed (initial value) that was supplied to the generator in the beginning. In other words, knowing the seed one can reconstruct the sequence of numbers a particular DRBG produces. Thus, DRBGs must be seeded with sufficient entropy from a reliable entropy source. Entropy sources that provide true randomness are usually based on non-deterministic physical processes or unpredictable events. Ring oscillators are an example of the first, and human driven mouse movements and keyboard stroke timings are examples of the second.

The importance of obtaining and using highly unpredictable keys is not just an academic question. There are many practical examples showing that failure to obtain sufficient entropy destroys any security provided by long keys and sound algorithms. Even the best algorithms cannot compensate for weak keys generated using insufficient entropy. Such systems are vulnerable to attackers – with potentially disastrous results. A study [2] revealed that thousands of network devices had generated easily guessable keys because of insufficient entropy from the entropy source used. Some keys on different systems were even identical, making it very easy for attackers to unlock the supposedly secret data protected by these keys.

Background

What is entropy? Entropy in the information theoretic sense is a measure of randomness or uncertainty in a signal. The typical units of measure are bits for entropy and bits per sample for entropy rate.

Cryptography is usually considered part of the realm of mathematics but today many of the critical issues in this discipline span the domains of computer science and engineering. Codes, ciphers and algorithms are the stock in trade for cryptographers, cryptologists, software developers and hardware engineers. They are implemented in the deterministic media of digital circuits and software. Computers are designed to be deterministic dynamic systems that execute well-defined instructions in order to produce predictable outcomes. This makes it hard to obtain good entropy from real computer systems because the sources of unpredictable behavior are minimized by design. Therefore, in order to obtain good entropy one needs to find sources of true randomness, those which run contrary to the nature of the typical computer system and thus may be difficult to use.

Though easy to understand intuitively, the notion of true randomness is hard to define and quantify. One might approach it by studying infinite sequences of bits or samples and measure their properties, some statistical such as bias, others non-statistical such as lack of computable correlations. Three of the most frequently used characteristics of true randomness are i) unpredictability, which is a measure of the strong non-computability of the bits in the sequence; ii) uniform distribution of the bits in the sequence; iii) lack of patterns in the sequence. It is worth pointing out that iii) implies both i) and ii). However, i) does not imply ii) and similarly ii) does not guarantee i).

Random numbers are used in other scientific and engineering fields, but the goals and needs are different. In simulation and modeling, one or more pseudorandom number generators (PRNGs) are used to generate values according to a given probability distribution, for example, modeling an arrival process with a Poisson distribution. The PRNG is seeded with a different initial value for each run of the simulation. The initial seed may be saved in order to repeat or recreate individual runs of interest. For many simulation uses, the only requirement on the seed is that it does not repeat or, equivalently, that it is unique across runs of the simulation. The current time or the tick count of a system clock can serve as a suitable source of seed values. Such applications do not pose any challenge to modern computing in terms of obtaining unpredictable outcomes from deterministic rules.

In cryptographic applications the requirements for seeding a DRBG are much more precise and stringent. The seed must possess sufficient entropy as illustrated by Heninger et al [2]. NIST SP 800-90A requires the seed to have at least the number of bits of entropy as the rated security strength of the DRBG mechanism. For example, a block cipher counter mode DRBG (CTR_DRBG) using AES-256 requires at least 256 bits of entropy input in order to achieve its maximum security strength of 256 bits [3].

We define an entropy source as comprising a noise source, sampling and quantization if needed, and minimal conditioning, such as unbiasing. The noise source is the physical process or observable activity that has true randomness. Sampling and quantization may be needed if the source is continuous. In most cases the output from a particular source contains bias and correlations – symptoms of non-randomness - due to imperfections in measurement or design. While eliminating correlation is very difficult in practice, bias can be mitigated. Von Neumann unbiasing is a common and useful simple

conditioner. Often the entropy source output is then passed through a pseudorandom function (PRF) conditioner to distribute entropy uniformly across the bits of output samples. Hash functions and block ciphers are common PRFs used.

There are a several types of entropy and noise sources on computing systems but they can be divided in two broad categories: hardware and software. Hardware entropy sources rely on processing variation in individual gates or circuit elements or on putting circuit elements in an undefined or unstable state. Examples of this type are ring oscillators, noisy diodes and techniques for using flash memory. Hardware entropy sources of this type, often called hardware random number generators, can be incorporated directly into devices and systems and provide entropy on-demand. Dedicated high-quality entropy sources such as those that use quantum effects are usually physically separate from the systems that use them. The NIST Randomness Beacon [4] is an example of this type, though it is not recommended for use in generating private keys since it is a public source.

Software entropy sources or software random number generators can run stand-alone or be incorporated into software applications directly. The name "software" is misleading, as the randomness comes from exploiting physical device phenomena such as the drift between the operating system (OS) timer and processor clock. Additional randomness may come from variation in events and processes in the system. TrueRand and the CPU Jitter Random Number Generator [5] are examples of this kind of entropy source[1].

Modern OSes have system-level entropy sources that provide entropy as a service to calling programs or system components. This relieves each application from the burden of needing a built-in entropy source or having a dependency on an external source that is not guaranteed to be on every system. OS-level sources typically use an entropy pool that is fed by multiple sources, such as system, network and user activity. /dev/random on Linux [6] and CryptoAPI on Windows are examples. Even though OS-level sources provide a convenient source of entropy to applications, they must be used with caution. The quality of a particular OS-level source is a function of a specific test configuration, which may include environment elements such as human-computer interactivity through peripheral devices. Therefore it is important to understand the dependencies of a given OS-level source on its environment and retest it when it is ported to a different environment. In fact, the recent discovery of weak keys found on network devices [2] resulted from improper use of an otherwise reasonable OS-level entropy source ported to a new environment where the assumed operating conditions (noise sources) did not hold.

Testing

Estimating the entropy rate or measuring the true randomness of an entropy source is difficult. For example, to show the unpredictability of a sequence of bits one has to establish that no Turing machine can compute more than finitely many scattered bits of the sequence. Since computability implies

[1] The identification of any commercial product or trade name does not imply endorsement or recommendation by the National institute of Standards and Technology.

predictability, the unpredictability depends not only on the impossibility to compute but also on the strength of that impossibility. In general, no bit sequence is absent of all possible patterns and this is the reason why it is hard to characterize true randomness in practice. Moreover, as mentioned above, assessing and eliminating correlation in the output from a source is practically impossible. Therefore, all known practical techniques for measuring true randomness are limited and imperfect. Still, objective, repeatable estimation methods are preferred, such as statistical estimators run on sample datasets. In addition to requiring good tests, such methods require careful data collection. For some noise source mechanisms, operating conditions are significant. Those based on physical processes may be temperature dependent and subject to degradation over time. Those based on network or user activity are heavily dependent on the presence and level of such activity.

To evaluate an entropy source and estimate its entropy rate current popular approaches use a battery or suite of tests, each test using a different method. For example, a suite may combine tests looking for bit patterns with tests trying to fit sample statistics to a particular distribution model for the sequence. The lowest estimate out of all the tests is typically used as the overall entropy estimate for the source [7, 8, 9, 10].

This approach is essentially ad hoc and has limitations. Widely varying estimates among the individual tests do not inspire confidence in the final estimate and provide little or no qualitative understanding.

A different class of methods computes entropy estimates from the spectral characteristics of digital signals. Spectral methods are widely used in digital signal processing and provide not only good quantitative results but also insight into the nature of the signal. The relationship between power spectral density and entropy for stochastic signals is well known in the literature [11]. However, spectral methods have not been used much in estimating the entropy for cryptographic purposes. Any bit or byte sequence produced by a true random number generator may be viewed as a digital signal. Then spectral techniques may be applied to calculate the power spectral density of that signal and from it an estimate for the entropy of the source. This insight is particularly useful for designers of hardware entropy sources as it allows them to identify non-random components in the output sequence and remove them by fine-tuning the design.

Because entropy sources can drift over time and are subject to malfunction due to aging or errors, health tests need to be performed in the field, in addition to the initial testing and entropy rate estimation performed. The health tests for entropy sources are of a different nature than those for cryptographic algorithms. Simple known-answer tests cannot be used. Instead, a set of samples must be collected and simple statistic tests run on the fly to determine whether the entropy source has degraded. The health tests are thus computationally demanding but too important to skip [8].

Naturally, the quality of a given random source is determined in large part by its ability to produce high rates of entropy consistently to meet the demand of all of its consumers without degradation of service resulting in locking or inability to provide cryptographic services. Generators with high entropy rate tend to tolerate health tests without noticeable performance degradation.

References:

1. NIST Special Publication (SP) 800-131A, “Transitions: Recommendation for Transitioning the Use of Cryptographic Algorithms and Key Lengths”, January 2011, (http://csrc.nist.gov/publications/nistpubs/800-131A/sp800-131A.pdf)
2. N. Heninger, Z. Durumeric, E. Wustrow and J. A. Halderman, “Mining Your Ps and Qs: Detection of Widespread Weak Keys in Network Devices”, Proceedings of the 21st USENIX Security Symposium, August 2012.
3. NIST Special Publication (SP) 800-90A, “Recommendation for Random Number Generation Using Deterministic Random Bit Generators”, January 2012, http://csrc.nist.gov/publications/nistpubs/800-90A/SP800-90A.pdf.
4. NIST Randomness Beacon, http://www.nist.gov/itl/csd/ct/nist_beacon.cfm.
5. CPU Jitter Random Number Generator, http://www.chronox.de
6. “The Linux Pseudorandom Number Generator Revisited”, P. Lacharme, A. Röck, V. Strubel and M. Videau, Cryptology ePrint Archive, Report 2012/251, 2012 (http://eprint.iacr.org)
7. NIST Special Publication (SP) 800-22 Rev. 1a, “A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications”, Rev. 1, April 2010 (http://csrc.nist.gov/publications/nistpubs/800-22-rev1a/SP800-22rev1a.pdf).
8. NIST Special Publication (SP) 800-90B, “Recommendation for the Entropy Sources Used for Random Bit Generation”, DRAFT, August 2012, NIST (http://csrc.nist.gov/publications/drafts/800-90/draft-sp800-90b.pdf)
9. DIEHARD Battery of Tests of Randomness: http://www.stat.fsu.edu/pub/diehard.
10. D. Knuth, “The art of computer programming”, vol. 2, 3rd ed., 1997, Adison-Wesley, ISBN 0-201-89684-2.
11. S. Haykin, “Adaptive filter theory”, 2nd ed., 1991, Prentice-Hall, ISBN 0-13-013236-5.