Design of a True Random Number Generator Using Audio Input

Roger Morrison
Department of Electrical and Computer Engineering,
Oregon State University, Corvallis, Oregon 97331 - USA.
E-mail: morrison@engr.orst.edu

Abstract— **Even the strongest cryptography is breakable if it is easy to guess the key. Therefore, it is critical that keys generated be highly random and difficult to guess. A common source for these random bits is sampling an analog signal generated by a physical noise source. This method has the potential to provide high quality random bits, but usually requires specialized analog circuitry be added to the computer. This paper demonstrates that the common microphone and audio input found on most home personal computers also provides cryptography quality random numbers.**

I. Introduction

Several cryptographic algorithms have shown they are secure by resisting attacks for years and surviving the test of time. Others are even provably secure by relying on known hard problems such as discrete logarithm or factoring. But the Achilles heal to all these algorithms is that the key must not be guessable and, in theory, should be as hard to discover as a truly random number.

When the key is guessable then all of the security is lost. If an attacker can discover any type of predictability in the way a cryptography key is generated, it will lower the difficulty if finding the correct key. The difficulty comes in generating a random, unpredictable, key using a computer that is designed to be as predictable and regular as possible. John Von Neumann saw this problem in 1951 when he wrote "Anyone who considers arithmetical methods of producing random digits is, of course, in a state of sin." [1]

II. Software/User Input Methods

There are several software techniques used to create random numbers, but each has weaknesses. Pseudo-random number generators create random bit strings that are strong enough to be used in cryptography. However, these require a random initial seed value, which brings back the issue of finding a random input source. Another software technique is to use an input source such as a mouse or keyboard to get random data from the user. This has worked for years in personal security systems such as PGP. When the software needs a random seed value it instructs the user to type random keys or move the mouse. By recording the input values and the times that they occur, fairly unpredictable seed values are created. This method does have drawbacks. One problem is that a user may not be available. A computer server may be located in a locked basement and does not interface directly with a mouse or keyboard. A computer server may also be running several processes, each of which is requesting random inputs. It could potentially require more input than the user is capable of, and most certainly more input than the user would like to perform. This leads to another problem of lax user input. The user may tire of typing random keys and instead hits only the space bar or holds the space bar down and lets the computer repeat key inputs. This more predictable user input has the potential to lower the randomness of the seed. [2]

Another option is to create random seed from hard to predict values found on the computer. These may include the date and time, process ID files and disk contents. These values are then combined in a convoluted way to further increase the appearance of randomness. The problem with this is two-fold. First, the number generation is based on semi-predictable data. If an attacker were able to gain access to the computer that is generating the seed it could be possible to find out or at least narrow down each of the values used to create the seed. Secondly, the appearance of randomness is based on a method of processing data. If the process is publicly available then the attacker only needs to guess the input data and enter it into the same process the software is using to generate the random number. If the process is secret, then in can be reverse-engineered or leaked to the public. The process can then be used just as if it were public. [3]

III. Hardware Methods

To avoid the disadvantages of random numbers generated by software and user input they could be generated using a physical process that provides a source of random noise. This noise is then sampled and recorded as random bits. For this system to work several characteristics must be met. The noise must be from a random and unpredictable source. In general, these sources are natural analog processes that are based on fluctuations on an atomic scale such as thermal, nuclear decay and electromagnetic noise. These events are based on such a large number variables it is impractical to predict the outcome. Many times the Johnson noise (thermal noise), shot noise and flicker noise that is found in resistors are used. [1] These sources are sufficiently random and can be designed compactly and cheaply, but have very small magnitudes, and must be amplified before they can be sampled.

This poses two problems. The first is that no amplifier is perfect, so distortion and bandwidth limitations cause

discrepancies between the initial pure noise and the amplified output noise. This is counteracted by using an amplifier with low distortion in a specific bandwidth and filtering out all other frequencies. The amplified noise is sent through a low pass filter and then sampled at a frequency slow enough that the high-frequency roll off of the amplifier is not a factor. The second problem is the difficulty in shielding such a weak signal from the strong digital signals in its surroundings. Care must be taken that the noise generators environment is as shielded as possible. This shield will help keep predictable digital signals from entering the system all also thwart attackers who may try to introduce a specific noise into the system to increase predictability. An additional step of protection is to use a differential noise from two similar nearby sources such as adjacent resistors. This will cancel any environmental noises that affect the sources identically. Once the signal is amplified, it must be sampled and converted to a usable binary format. Several of these methods use two clocks signals, one to sample the noise, and one to sample the data that the noise source has created. Bucci and Bagini suggest using the first clock to create a pulse that then triggers the sampling switch[4]. This data is stored in a binary counter. The first clock is also fed into a counter that creates a slower clock that samples the binary counter data. By using the binary counter any bias from the amplifier and comparator can be eliminated.

The Intel random number generator uses a slightly different approach. The frequency of the first clock is actually a function of the random data. The Johnson thermal noise of a resistor is fed into an amplifier and then to a voltage controlled oscillator. This clock now contains the noise data. When this clock transitions it triggers a latch that samples a high frequency clock. This method also eliminates bias from the noise amplifier and voltage controlled oscillator.

IV. Randomness

Using two clocks to eliminate noise sampling bias is not without drawbacks. The entropy of the noise signal does not increase and the bias is actually converted into correlation. [5] This can be more easily understood by looking at an extreme example. In the case of the first generator, assume that the signal coming out of the comparator is all 1s. This bias has taken all of the entropy out of the signal. The output of the generator may not be biased because it is coming out of the binary counter. Nevertheless, it will be strongly correlated because the binary counter and the second sampling clock will perform in a predictable manner. The binary counter will increment every clock cycle and the second clock will sample it at a regular synchronous frequency. This will produce all 0s, all 1s or alternating 1s and 0s. The same idea holds true for the second random number generator, the entropy of the output can be no greater than the entropy of the sampled noise. In fact, entropy is further decreased by any bias in the second high frequency clock. There are methods, however, to concentrate the entropy of a signal by trading the entropy of the output bits with the output bit-rate.

One method used by Intel [1] is the Von Neumann corrector. This method takes pairs of bits as inputs and converts then into output bits using the table below:

Input Bits	Output Bit
0,0	nothing
0,1	1
1.0	0
1,1	nothing

Table 1: Von Neumann corrector

Output bits only occur when there is a transition in the input bit stream. Because transitions in each direction occur an equal number of times in a bit stream the bias is eliminated. In this case the bias is not converted into correlation, but instead results in a lower bit rate. The greater the bias the more often 0,0 or 1,1 states occur which results in no output and thus a lower bit rate. In fact, the Von Neumann corrector reduced the average bit-rate of an unbiased signal by a factor of 6. In addition, the output now has a variable bit rate, instead of the constant bit-rate of the input.

This corrector also illustrates that correlation can be converted to bias. If the bits being used as an input to the Von Neumann corrector are correlated the output will also be correlated and/or biased. Again, taking the extreme example by assuming the input is alternating 0s and 1s. The input is unbiased, but the output will be all 0s or all 1s depending on when the corrector begins to read the input data.

V. Analog to Digital Converter Chaos

An obstacle to extracting the entropy from weak analog signals is filtering out digital signal interference. This must be done to stop the deterministic signals from introducing predictability into the output. A different method is to sample the signals at a high resolution such that minuet, unpredictable fluctuations are the source of entropy. It is analogous to measuring the temperature of room temperature air with a thermometer that is accurate to 1/1000th of a degree, and trying to predict the least significant digit. The most significant digits would be easily guessed, especially if the air was being controlled with a thermal source, but the digits of higher precision are a function of so many elements that the digits becomes chaotic and unpredictable. The least significant bits (LSBs) of high-resolution A/Ds also tend to produce independent bits[4]. This is true even when an outside source attempts to control or interfere with the sampled signal. The key is that the A/D is able to extract characteristics of a signal that are too small to be accurately controlled or predicted. The analog noise source no longer requires protection from deterministic signals because the predictable part of the interference is not being sampled.

VI. Audio Input Sampling

Two A/Ds common in the personal computer are the audio input and the mouse. The mouse, as we found before, is not a satisfactory source because it requires user input. The microphone, on the other hand, is bombarded constantly with analog signals from the surroundings. Movements create sound waves that prorogate through the air and reflect off objects, which makes the amplitude at microphone more unpredictable.

The analog sound is converted to voltage by the microphone and then into data bits by and audio A/D converter. At high sampling rates, audio signal data samples have high correlation properties. These are expected because the sampled sound is a continuous and sometimes slow changing analog signal. The graphic below shows how nearby samples can have similar values. This is a side effect of the A/D having limited precision and having to group amplitudes of similar strength into the same digital value.

To overcome this, the samples are added and the sum is sampled at a large period. This way the entropy of every sample is contributing to the output bits. The tradeoff is that as the sampling period is increased the bitrate at the output decreases.

Also notice that the entropy of the LSBs is greatest and diminishes as the bits become less precise all the way up to the most significant bits (MSBs). This would mean only the LSB of the sampled signals are useful to extract the maximum amount of entropy, and all of other bits are simply wasted. However, there is a method to use entropy of the entire sample. The XOR function has the property of adding the entropy of uncorrelated bits. The samples from the A/D are correlated from sample to sample, but not bitwise from MSB to LSB. If an analog signal is sampled at random over the full range of the A/Ds input then the bits should have no correlation between them. In addition, if the signal is not evenly distributed over the input range then the MSB bits, which will be biased toward zero, will not detract from the entropy. To better understand this concept think of an example using decimal digits. If a random number is added to zeros, it retains its randomness.

An added benefit to XORing the bits of the sampled signal is that bias is reduced. If the LSB is the only source for the output bits then it is subject to any bias of the A/D toward 1 or 0. By XORing the entire string of bits the bias is shifted to the number of 1s in the sample is even or odd. Tests performed on sample audio inputs showed heavy bias. For large samples of data bias was 48.2 % 1s. By XORing the entire sample the bias was virtually eliminated. It should be pointed out that the XORing function does not increase entropy by reducing the bias. The XORing of the bits only adds the entropy of the uncorrelated bits. If the bits were correlated then the bias would propagate. For example, if the two of the bits tended to be the same, then XORing the bits results in a zero bias.

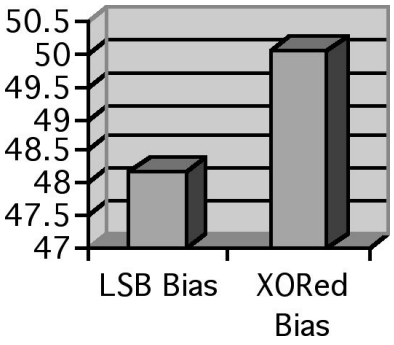

Figure 1: Bias

The final output data is determined by combining the large sampling period and XORing techniques. Each time the A/D samples the audio the bits are XORed across MSB to LSB and then added to a counter. Because the counter is only one bit wide, the addition is also a simple XOR function. This counter is sampled at a period of N, where N is a fixed number of A/D samples. This data is put into a buffer as a random bit string.

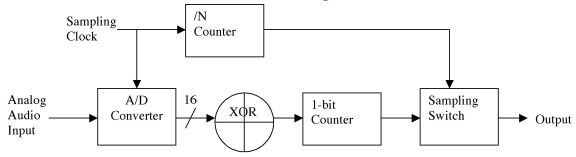

Figure 1: Sampling

The noise level at the microphone, the A/D sampling rate, and the value of N were all varied to determine there impact on the final bit stream. The noise level variations consisted of:

- quiet environment that effected no more than 2 of the LSB at any time
- noisy radio playing in the background but not causing the A/D to clip (exceed the range of the A/D)
- loud environment that did cause significant A/D clipping
- 10kHz interference tone projected directly into the microphone.
- 11.025kHz interference tone generated synchronously using the sound inputs sampling clock
- normal office environment with over half of the A/D bits are effected more than 10% of the time (this occurs at default microphone settings)

The sampling rate of the A/D was tested 11.025 and 44.100 KHz and the N value was set at 1, 10 and 100 samples.

Although randomness can never be proven, several test suits can be used to test correlation and bias. After at least 1 megabit of data was collected for each setting the samples were passed through the following tests:

- Block means Spectral analyses
- Random walk test
- Block mean correlations, 1-129
- Overall mean
- Column means
- Spectral analyses; hi, medium and low smoothing
- Run length variances
- FIPS 140-1 test suite

Failures occurred whenever N was set to 1 and when N was set to 10 and the environment was very quiet or so loud that clipping occurred at the A/D. This is reasonable because the sampling rates were set high and the input signals were intentionally robbed of entropy. The microphone and A/D was not able to gather enough entropy from the signal during this short time period nearby samples became correlated. For all other settings, the output bits passed all of the statistical tests for randomness. Even when the test was intentionally sabotaged by forcing a single fixed frequency into the microphone, no significant correlations or biases were found when N was greater than 10.

As an added precaution a preliminary test should be performed on the microphone. The samples should fluctuate enough to effect more than half of the bits a significant part of the sampling time. Care is also needed to ensure the input signal is never clipped by the A/D. Either of these situations have the potential to cause a loss is entropy. For added security, N should be 10 or above. This provides a bitrate of 4.41 Kbit/sec when the sampling rate is 44.1 KHz, which is enough for most cryptographic applications.

VII. Conclusion

Cryptographys desire for random numbers, and the inadequacies of software and user input methods, has created a need for inexpensive and widely available method of generating random bits with hardware. The common microphone and audio input found on person computers provides a means for sampling an analog audio signal with random properties. Sampling techniques further concentrate the entropy of these samples resulting in an output bit stream that consistently passes tests for randomness. Using this method the home computer user can generate cryptographically secure random bits without any specialized hardware.

References

[1] P. Kocher B. Jun, "The intel random number generator," Tech. Rep., Cryptography Research, Inc. White Paper Prepared for Intel Corporation, 1999.

[2] RSA, "Hardware-based random number generation," Tech. Rep., An RSA Data Security White Paper, 1999.

[3] David Wagner Ian Goldberg, "Randomness in the netscape browser," *Dr. Dobb's Journal*, January 1996.

[4] Marco Bucci Vittorio Bagini, "A design of reliable true random number generator for cryptographic applications," *Cryptographic Hardware and Embedded Systems*, vol. 1717, pp. 204–218, August 1999.

[5] Robert J. Rance David P. Maher, "Random number generators founded on signal and information therory," *Cryptographic Hardware and Embedded Systems*, vol. 1717, pp. 219–230, August 1999.