Random Number Generation Using AES

Toby Prescott

Use of Random Numbers in Security Protocols

Random number generation has applications in many areas, but this article will focus on the use of these numbers in security protocols. Specifically, the security protocols used to protect the opening and starting of an automobile will be examined. While equally important, much of the focus has been on the choice and implementation of the cryptographic engine that protects the exchange of information between the vehicle and key fob used to establish the authenticity of the owner. Many of these security protocols use a random number as a way to ensure "freshness" of the communication. These random numbers are vastly important for achieving a high level of security, and in many cases are incorrectly assumed to be easily obtained. There are numerous examples of security protocols being compromised not by a direct attack on the encryption itself but instead by focusing on removing the unpredictability from the random number generation process. [Mifare.pdf] This article attempts to bring this critical part of the security system into full focus and present an effective method for generating random numbers.

Random Number Generation Background

Random numbers have been generated in many ways in order to introduce a needed element of unpredictability. This can range from being as simple as flipping a coin to more complicated procedures such as measuring the decay of radioactive elements. The desired result is that it should be impossible to predict with any accuracy the next outcome based on past or present knowledge. Much of the application of random numbers to security systems is based on Claude E Shannon's work with information theory. A general definition of a random number is that when drawn from a given set of numbers, the probability of any one number should be equal to that of all others, in order for it to be truly random.

The process of generating a random number will typically fall into two distinct categories: True Random Number Generators (TRNG) and Pseudo-Random Number Generators (PRNG). Typically, the TRNG is tied to some physical noise source, ranging from a noisy diode to quantum noise. PRNG results from the need to produce a random number in an application where a TRNG is not practical to implement. The PRNG attempts to achieve a result sufficiently indistinguishable from TRNG for the application using the random number. Examples of this would be measuring the movement of a computer mouse over time or sampling a high-speed counter each time a button is pressed by the user. While it is technically possible to manipulate or control these in such a way as to produce numbers that are not random, the feasibility of implementing the attack ensures that these methods of PRNG are sufficient for low-risk applications. This article focuses on a distinct subset of PRNG known as Deterministic Random Bit Generators (DRBG). These are unique in that they are based on the use of algorithms that are very predictable by nature. While it is possible to generate good-quality random numbers using these methods, there are some specific conditions that must be carefully considered. The recommendations in this article are based on the National Institute of Standards and Technology (NIST) publication SP800-90.

Quality of the Random Number

For a generated output to be a high-quality random number, it must satisfy several criteria. One important requirement holds that it must not be possible to predict future numbers based on past performance (prediction resistance). Also, for a given range of possible numbers the results of the DRBG should be uniformly distributed. To make it easy to compare different approaches, the quality of the random number is typically specified in terms of Entropy (H). The entropy of a system is a function of the probabilities of the individual results that are possible. As a result, all evaluation of random numbers is heavily based in probability and statistical mathematics. For security applications, the NIST recommends the use of a minimum level of Entropy (H_{min}) dependent on the application requiring the random number. [SP800-90revised_March2007.pdf] This H_{min} should meet or exceed the requirements specified for the cryptography level used in the application. H_{min} can be calculated by the following formula $H_{min} = -Log_2(p_{max})$ where p_{max} is the maximum probability of a single event. This maximum probability must be characterized or specified for a given system. The NIST article uses an example based on 4-bit sampling of a diode with a maximum probability of 0.19462 for two individual events. The H_{min} is then calculated to be 2.38487 for this approach. Please note that the Entropy will always be equal or less than the number of bits used to generate the output. The total entropy of a system can be increased by concatenation. To achieve entropy of 128 bits using the above example, the diode would need to be sampled 54 times and the resulting values concatenated.

One very easy test that can only be used to determine that a DRBG is NOT truly random is the visual test [Analysis2005.pdf]. While this method cannot say that the numbers generated are high quality, it does give a satisfying indication that you may be on the correct path. The human eye is very good at picking out repeated patterns from a graphical representation. If such a pattern exists, there is a good probability that a serious weakness exists in the process. [http://www.boallen.com/random-numbers.html] Figures 1 to 3 show the output of 256, 5000, and 10000 random numbers using the modified DRBG routine that Atmel proposes as an option for automotive applications.

For a thorough evaluation of the random number generation method, the NIST recommends that a true statistical analysis be performed. They have documented several methods in SP800-22, and compiled an exhaustive list of possible

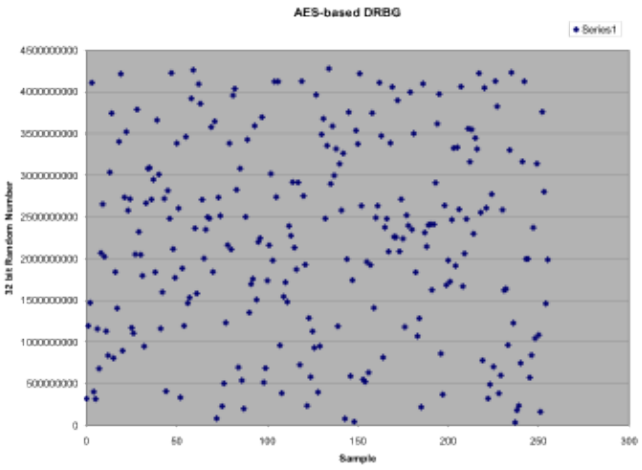

Figure 1. 250 Iterations of the DRBG

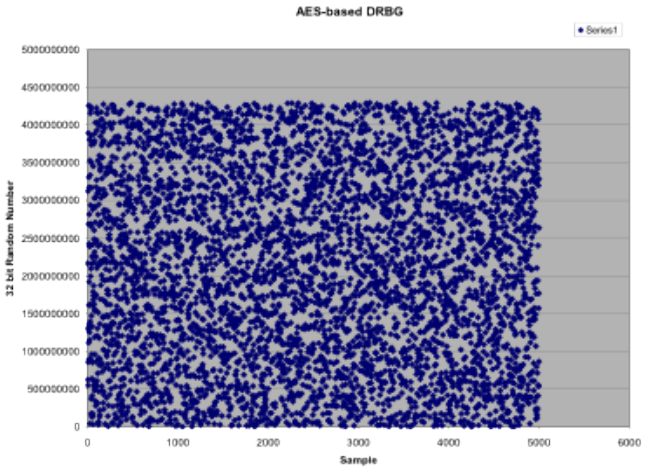

Figure 2. 5000 Iterations of the DRBG

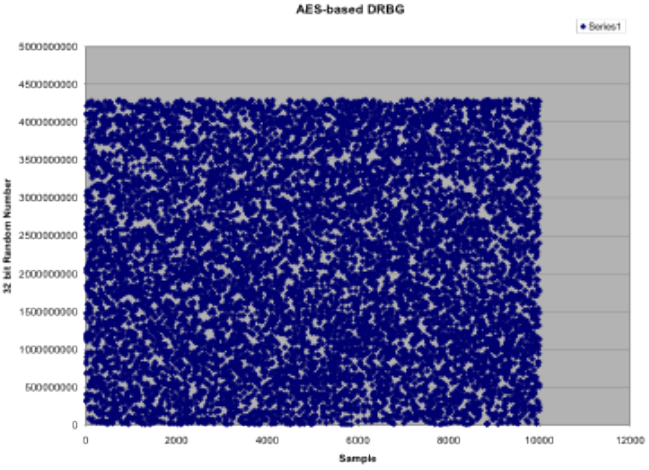

Figure 3. 10000 Iterations of the DRBG

tests for many possible weaknesses. [SP800-22rev1a.pdf] In addition, correct interpretation of the test results is necessary and can be quite challenging to achieve. NIST has also released a software tool package that can be used to run many of these tests on the random number generation method chosen for a given application.

[http://csrc.nist.gov/groups/ST/toolkit/rng/documentation_software.html]

AES Encryption Introduced

The recommendations from the NIST are very general in scope and this article will focus only on a small subset from this document. Included in the many deterministic algorithms that can be used are a class termed Block Ciphers. The one Atmel selected is the Advanced Encryption Standard (AES). This encryption algorithm was the result of an international competition sponsored by the US Government to create a modern security engine that leverages the concept of peer review by remaining completely open source. In 2000 the AES standard was accepted as the encryption of choice by the US Government, and remains secure with no known successful attacks.

AES is considered deterministic because the output is very predictable given knowledge of inputs such as the plain text and secret key. Also, like most symmetric block ciphers, AES is capable of both encoding and decoding using the same secret key. This property is a very useful function when transferring information, but when applied to generating random numbers it causes some limitations that must be addressed.

So, how is AES used to generate random numbers if the fundamental structure imposes considerable limitations? One very important factor applies to most encryption algorithms in general. For an algorithm to have a high level of security, the output resulting from one single bit change on the input should be a large number of bit changes that are evenly distributed across the entire possible range. This should be accomplished in a manner that is completely unpredictable with only knowledge of the input and the output. In essence, a high level of entropy is present as long as the secret key remains unknown. The properties lend to a very good starting point for generating random numbers.

Proposed Method for Generating Random Numbers Using AES

The following will focus on two main areas in a vehicle with security protocols that rely on quality random numbers. Passive Entry/Passive Start and immobilizer systems both generate random numbers in the vehicle that are critical in determining the authenticity of a key fob before allowing access to the vehicle or starting the engine. The process used to generate these numbers is normally not well defined in the security protocol, even though it can have serious implications on the overall security of the vehicle. Atmel selected AES because this encryption is gaining popularity in the automotive security area and Atmel is leveraging the reuse of already present encryption blocks. While it is not practical to implement the entirety of the NIST recommendations, the remaining sections attempt to create a proposed solution that is a modified subset of their main blocks. The modifications specifically address the challenge of storing and modifying large amounts of data in non-volatile memory in an automotive application. It should also be noted that only the generate function has been implemented, not the complete support functions, which include instantiate, reseed, test, and uninstantiate. Figure 4 shows the basic operation of the modified DRNG generate function.

EEPROM Variables

The basic generate function proposed by Atmel uses three EEPROM variables. The size of these as well as the way Atmel suggests using them is modified from the NIST recommendation because of the data retention requirements in automotive applications.

It is common practice to use the following two methods for storing variables in EEPROM. The first is to triplicate copies for error detection and correction where the same value is written in three different locations (should not be sequential addresses) and all three are compared using majority voting when the value is read out. The second is cell wear leveling where the value is really an array of locations with a pointer to the current cell(s) used. Each time the value is written and read out, the pointer is moved so that the next write cycle uses a different set of EEPROM cells. It is easy to see that the two methods combined can require a large amount of physical memory to store a small number of variables.

With this in mind, the size of the Counter_V variable is set to 32 bits. This variable is updated each time the generate function is called. The next variable is the DRBG_Key, which is a full 128 bits in length. This variable is only updated during

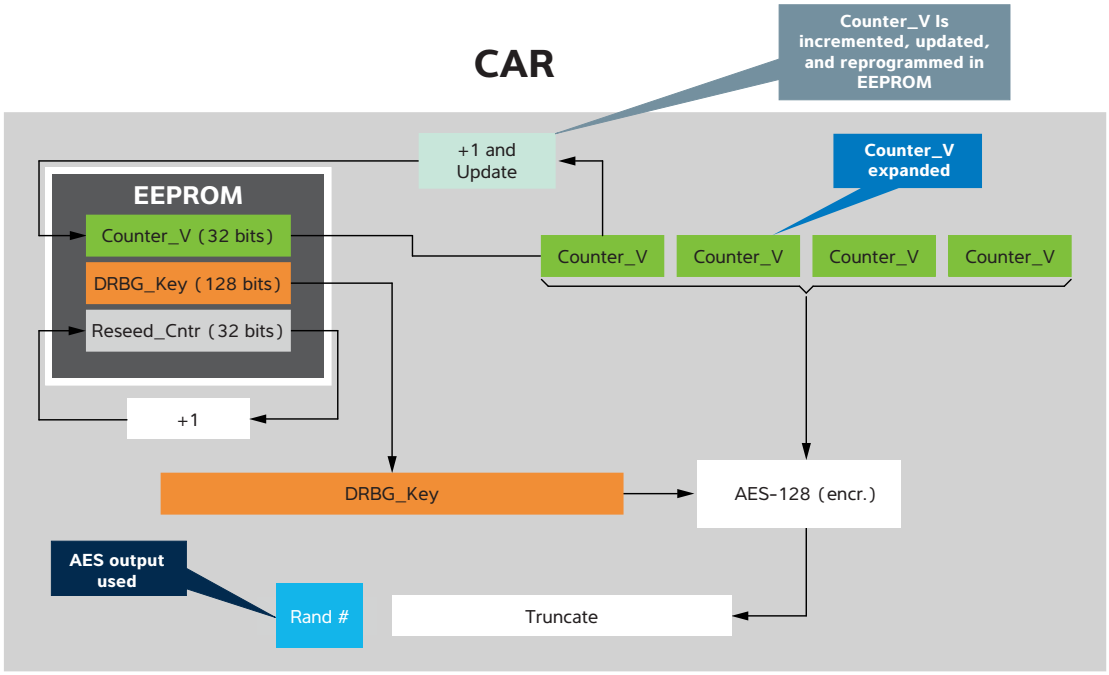

Figure 4. Generate Function

the initialization and reseed portions of our approach, so it should not require cell wear leveling depending on the reseed interval used. The third variable is the Reseed_Cntr, which is also 32 bits in length. This is incremented by one each time.

Inputs to the AES Block

The encryption block requires two different inputs, both sized to 128 bits. One of the inputs is the secret key (DRBG_Key), which is loaded in without modification. This value is fixed between reseed intervals. The other input is the plain text to be encrypted. This is created by expanding the Counter_V from 32 bits to a full 128 bits. There are many ways to pad inputs but in this example the value is simply concatenated four times. Once these inputs are loaded the AES encryption process is carried out.

Use of the AES Output

The result of the AES encryption is a 128-bit number. This is used as the random number output from the generate function. Many security protocols in car access and security require a shorter value due to system response requirements. For example, the random number may only need to be 32 bits in order to increase communication speeds. The proposed method uses truncation of the right-most bits until the desired length is reached.

Update Function and Counters

Once the random number is generated, the counters must be updated in preparation for the next execution of the function. In this case the NIST recommended update function is also modified so that it does not update the 128-bit DRBG_Key variable. If feasible, this is left to the reseed function to implement in the application. The update function modified by Atmel is shown in figure 5. The process for updating the Reseed_Cntr is straightforward by simply incrementing each time the generate function is executed. To ensure that the DRBG process is not susceptible to backtracking attacks, the Counter_V is not simply incremented. Instead the counter is initially incremented, then padded with the Unique ID of the vehicle and zeros to a length of 128 bits. The UID could be the VIN number or a hardware module ID. This is used as the input to a second AES encryption. The output is truncated to 32 bits and stored as the new Counter_V variable.

Considerations for Ensuring a Quality Random Number

As mentioned in section 2, the use of a deterministic algorithm such as AES to generate random numbers faces some challenges that must be overcome to produce results that contain sufficient entropy to be used in security applications. This section looks at some of these aspects and provides suggestions for further enhancements that can be made to the system.

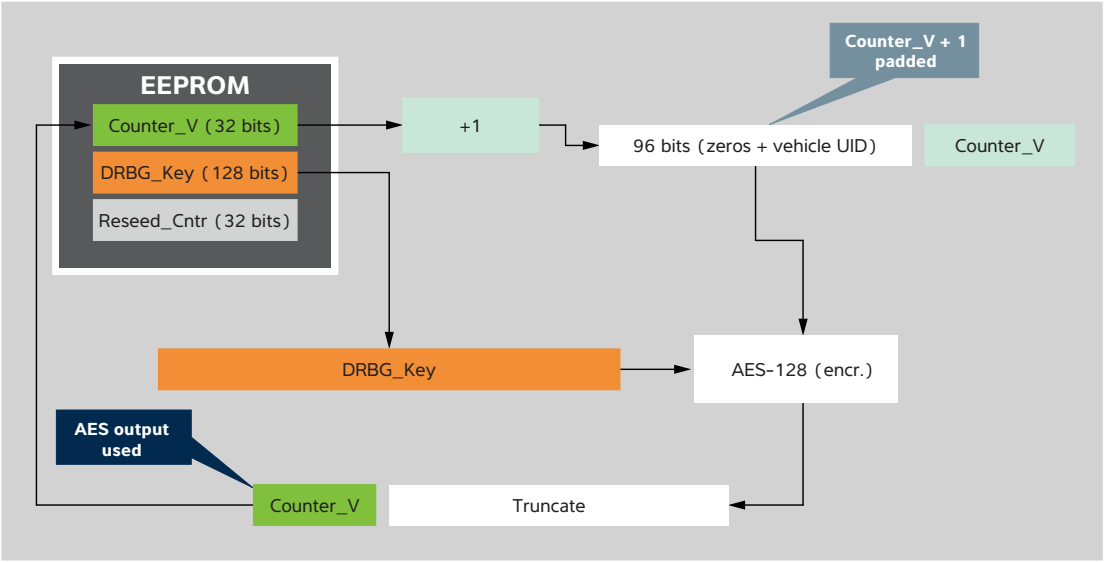

Figure 5. Update Function

Initial Values of the EEPROM

The EEPROM variables that are used to exercise the AES encryption engine and generate the random number output have already mentioned. One topic, however, that has not been covered yet is how these values are set initially. This falls outside the scope of the article but it is crucial to convey the importance of the process used to set the initial state of the DRBG_Key and Counter_V variables. It would be preferable to use a RNG with very high entropy during the manufacturing and test process to generate these initial values. This can be done with specialized equipment, for example, and transferred along with final configuration data.

Maintaining Secrecy

All of the variables stored in EEPROM should be treated as a part of the security system and handled appropriately. Because of the deterministic nature of the AES algorithm, it is possible to calculate the next sequence of numbers if the internal state of the variables is known. If it is possible for the attacker to access these variables in any way, they could devise an attack on the security protocol that uses these random numbers without having to "break" the encryption algorithm that is used. Protecting these variables should begin with the initialization process, which should ensure that a "common" or "master" secret key is not used for all devices.

Use of Reseed Counter

The Reseed_Cntr variable has been included but the use of this variable has not yet been discussed. As mentioned previously, the counter value is incremented each time the generate function is executed. The purpose of this variable is to send a limit on how many times the generate and update functions can be used without reseeding the system with new variables from an external entropy source, preferably from a TRNG. This provides prediction resistance because it limits the number of deterministic outputs that are possible. In the case where all internal state variables are compromised, it would be possible for the attacker to calculate forward the random numbers. The use of the reseed counter limits the usefulness of this in breaking the system. A lower limit provides stronger resistance but the reseeding process requires access to a second entropy source.

Other reference citations

[http://www.random.org/randomness/]
[http://www.randomnumber.info/content/Random.htm]