Otoniel Cisneros
Dr. Martin
CS 4960s
Research paper
10/27/10

True Random and Pseudo-Random Numbers

Randomness is something we, as humans, do not desire most of the time because we like to manipulate the outcome of events to our advantage. Yet, sometimes randomness is one of the best strategies we can use once we understand how to use it, when to use it, and why we use it. True random numbers and pseudo-random numbers are a way to implement randomness to our advantage, and we now have the knowledge of how, when, and why it is a really good resource. Randomness, in computer science, is mostly used in cryptography, the field that is in charge of encrypting messages in a random way in which only the encoder knows how to transform the message into a code and only the decoder knows how to understand the message. These encryptions are specially found within networks and Internet, which have many users sharing data through a common medium. Since there is plenty of information floating around, very confidential and important data needs to be hidden by a code. This is where pseudo-random numbers and true random numbers are useful as a way to hide this information through a different code so the information cannot be understood by a bogus user. By encrypting this data, no one knows what those bytes express but the decoder who has the means to reveal it. This paper intends to expose some general problems with network security, the role that it plays with encryption, and what encryption does to a message. It will also explain the nature of pseudo-random numbers and true random numbers, how they are generated, the manner they are used to help protect data, their implementation, and the most convenient time to use true random numbers or pseudo-random numbers depending on the situation.

Cryptography is an old technique first used by Julius Caesar during the time of the Roman

Empire. Cryptography takes a message and then transforms it into an unreadable format which only the person that knows the cryptographic format can read. It started with a simple encryption of letters, or a scrambling of characters. However, with the passage of years, this technique has been manipulated to create more complex codes.

By using cryptography to hide information, programmers started to create virtual keys made up of numerical numbers that would reveal confidential data. However, once a computer is surfing through the Internet, there is more insecurity. "The Internet is global in scope, but this global Internet-working is an open insecure medium" (Rhee, XIII). Since the Internet is a global means of communication, the amount of on-line users surpass the millions. Unfortunately, some of them are not plain users that would like to check their banking account or send a simple message to their friends. Some users are lurking behind their computer for a way to steal something of monetary value that will in turn affect someone else. Since there is a very wide variety of users throughout the network, there are a lot of users that would like to break codes and search for an useful way to steal important data. "Some cryptographic algorithms are very trivial to understand, replicate, and therefore, crack. Some others are highly complicated and therefore very difficult to crack" (Kahate, 39). In other words, cryptography used in computer science can definitely help to reduce the effect of malicious users who want to use others' private data like credit card numbers by making those codes hard to break. For example, let's pretend there is an e-mail with 200 characters. When the e-mail is passed through the encryption process, all characters change to numbers, other characters, or symbols to hide the data from plain sight. The encryption process is then composed of difficult mathematical functions that will output integers as final result and use these integers to add to the ASCII code in every letter from the message, thus transforming the message. It passes through the network and then is decrypted back so the other user can read it. In this manner, the private message from one sender is protected from outsiders. In this

manner, Internet security will increase and therefore will reduce the risk of viewing private data by random users.

There are many types of ways to encrypt data and many functions to encode it. However, when using pseudo-random numbers there is a higher security factor and a tougher difficulty when trying to break the code. Pseudo-random numbers are numbers generated by an algorithm or some mathematical function which simulate randomness. Although the simulation is very close to random, these numbers are not random since they obey some mathematical pattern, and they have the benefit that they can produce the same numbers again and again for debugging processes. According to Gentle, a pseudo-random number generator must have the ability to set or retrieve the seed, select seed that yields to different streams, and possibly select the method for a limited amount of numbers given (Gentle, 285). This author refers to the seed as the first number that will start the simulation. In order words, the number that will be manipulated by a mathematical function to start generating random numbers. As an example, the seed in C/C++ programming language is srand() function founded in stdlib.h. Using the srand() in a function, uses mathematical procedures into such function, which formats the program to give X amount of numbers, and output the number(s). This whole process results in a simple pseudo-random number generator. The following is an example of using srand() in C++:

```
#include <iostream>
#include <stdlib.h>
using namespace std;


int main(){

        int seed;
        int times;
        cout << "enter the seed: " << endl;
```

```
        cin >> seed;
        cout << "enter amount of numbers" << endl;
        cin >> times;
        srand(seed);

        for(int i=1; i < times + 1; i++){

                cout << rand()%6 + 1 << '\t';                    //it will simulate dice
        } //end for loop

        return 0;

} //end main
```

This code simulates the outcome of a dice by inserting a seed number and then indicating the amount of times the dice will be rolled. The output will consist on the number given by the simulation of the rolled dice. Yet, this is not the only way to create "random" numbers. When someone wants to build a pseudo-random number generator without the use of an embedded seed in some language, there could be very simple things to do. The usage of the residue of a division is important and can simulate a lower degree of randomness (Bowman, 228). By using a big residue space, a division from a function generating numbers can have results that can simulate random numbers. As an example, here is the following code:

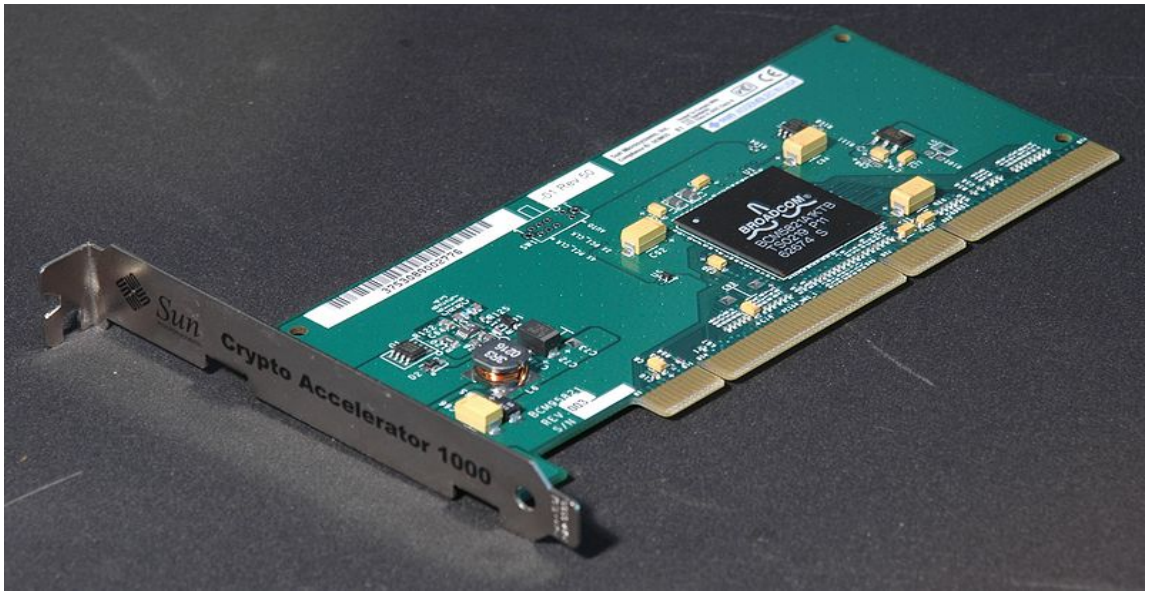
```
#include <iostream>
using namespace std;

int nSeed;
int loopnum;
int mid;
int x;

int main (){

        cout << "Please enter a number" << endl;
        cin >> nSeed;
        cout << "Please enter the number of random numbers desired"  << endl;
        cin >> loopnum;
        cout << "Enter coding number" << endl;
```

```
        cin >> mid;

        for (x=0; x< loopnum; x++){
        nSeed = (333453 * nSeed + 234453 - mid);
        cout << nSeed % 32767;
        cout << "\t" ;
        }


        return 0;
}
```

In this example, an array of numbers is outputted by inserting a number, then selecting how many numbers the user wants, and some other number to randomize the function a little bit further. Once this calculation is done, then it takes the residue of 32767 to increase the chance of not repeating a number. By doing this, there is no special library used to create random numbers and there is still a simulation of random numbers. However, once again, pseudo-random numbers follow a mathematical pattern that will simulate randomness. On the other hand, true random numbers do not follow mathematical patterns and in the same manner they cannot be followed.

True random numbers are numbers that do not have a logical pattern in any function. They do not have any correlation with the number before it nor the one after it. Unlike pseudo-random numbers, these numbers are very difficult to achieve and they are even tough to imagine creating them. Since all programming languages obey mathematical laws to program, true randomness cannot be created by following a function or using a random seed to generate numbers. Therefore, more sophisticated and difficult approaches are needed to obtain a True Random Number Generator (TRNG). TRNG's need a program to obtain data and use it (except non-electric generators such as drawing ticket for lottery), yet they do not receive the random data from the program but from an outside source (for the most part). An example of a TRNG is the work done by Yue Hu, Xiaofeng Liao, Kwok-wo Wong and Qing Zhou which generates a 256-bit random number by computer mouse movement. This program eliminates the

effect of similar movements by using discretized 2D chaotic map permutations, spatiotemporal chaos, and "MASK" algorithm. By using this program, the user can have true randomness by just moving the mouse and getting plenty of data out of his movement. These programmers ensured randomness by eliminating the effect of having similar movements. Another example of a TRNG is a hardware random number generator. However, this is not only a program, this hardware device generates random numbers based on microscopic phenomena such as heat or the photoelectric effect or other quantum phenomena (Kaliski, Koç, Paar, 465). This hardware creates true random numbers based on temperature, electricity, light, or other physical sources that will ensure randomness for a higher protection. Another different way to create true random numbers is used by the webpage random.org created by Mads Haahr, a doctor in computer science, that takes noise from the atmosphere to create numbers successfully (Fulton). TRNGs are very impressive and very hard to create. TRNGs along with Pseudo-random number generators have their unique features and they are both used for a defined purpose. The following image depicts a TRNG:



Physical true random number generator from *Wikipedia.org*

Since both of these number generators are used in encryption, both of these are used in security. Yet, the difference consists in how securely encrypted data must be and how important such data is. For the most part, pseudo-random numbers are used for creation of keys that need some sort of security. For example, the creation of keys for software programs, creation of email accounts, and other type of keys in which security will not make a major change around the globe. Since these keys are used to protect software of minimal value, companies will use pseudo-random numbers to create keys but consisting of plenty of numbers, so users will not be able to generate their own key to obtain free software (Fulton 130). Pseudo-random numbers are quite useful since time to create them or money to invest in creating them is minimal. In addition, they consist in mathematical functions that can be created and implemented using a simple computer without using complex code. Truly random numbers are used to make sure there is no way to track down the numbers. For example, using true randomness to select a lottery winner will be a great advantage since pseudo-random numbers are created by a mathematical function (Fulton, pg. 200). By using true randomness, the lottery winner will be selected just as if someone would pull a paper or a numbered ball out of the basket. As mentioned before, true random numbers cannot be followed and therefore there is no way to determine by any means what will be the next number. Pseudo-random numbers have been known for quite some time while true randomness in computers is a topic that is fairly new and contains future work to do.

True randomness has shown to leave no relation between numbers and therefore it is very useful and reliable. One of the future research to generate true randomness is the creation of generators by using Internet noise (Kugiumtzis & Boudourides). Since the Internet contains so many different connections, and the data sent is usually not the same, this could create plenty of different numbers that have no relation with the numbers before or after it. In this manner, there could be a new way to

create a new type of true random number generator and possibly decreasing the cost of creating it since some of these generators consist in expensive hardware to detect motion or noise. Although this sounds like a great idea, there is research going on to examine and test the type of data flowing through the Internet to observe any abnormalities and check whether it will be good the create the number generator.

Overall, randomness has been successfully used in history and has initialized encryption and ways to hide data information from others. Pseudo-random numbers and true random numbers have been the way that computers are most protected with our important data. Both have a different use and they both are convenient for the purpose needed.

CITATIONS

1. Atul, Kahate "Cryptography and network security". 1st edition. McGraw-Hill Education. 2009.
2. Gentle, James E., "Random number generation and Monte Carlo methods". 2nd edition. Springer publishers. 2003.
3. Rhee, Man Young "Internet security: Cryptographic principles, algorithms and protocols". 1st edition. Willey publishers. 2004.
4. Bowman, Kenneth P. "An introduction to programming with IDL: Interactive Data language". 1st edition. Academic Press. 2005.
5. Yue Hu, Xiaofeng Liao, Kwok-wo Wong and Qing Zhou**.** "A true random number generator based on mouse movement and chaotic cryptography". Chaos, Solitons & Fractals Vol. 40. Issue 5. June 15 2009. Pg. 2286-2293.
6. Burton S. Kaliski, Çetin K. Koç, Christof Paar. "Cryptographic hardware and embedded systems". 4th edition. Springer. 2002.
7. Fulton, Hal E. "The Ruby Way". 1st edition. Sams publishers. 2001.
8. Kugiumtzis, Dimitris & Boudourides, Moses A. "Chaotic Analysis of Ping Data". Department of Electrical and Computer Engineering , Democritus University of Xanthi, Greece.