A Personalized Random Number Generator with external physical and atmospheric inputs

Ramya Rajan – 636547507

Sudharshan Krishnamurthy – 925615183

Saket Thombre - 899913802

1. What is the project about? What do you plan to do?

We plan to implement a secure random number generator with an attempt to increase the overall entropy using external physical inputs. Almost every cryptographic application requires random numbers. The random number generators in use today produce predictable results when generating large numbers of random numbers. **Our aim is to use physical inputs like mouse movements and atmospheric inputs like sound captured from the microphone to increase the entropy of the random numbers which would be generated.**

Note: The sound captured from the microphone is not stored anywhere to ensure privacy of the user.

2. How is it done today and the limits of the current practice? How, what you are proposing is different than what has already been done?

Ramya Rajan – Review of two papers

Review of paper 1: Understanding random number generators and their limitations in Linux

Linux has an in-built crypto PRNG in the form of dev/random and dev/urandom. The kernel maintains an entropy pool which is used to store random data by various system events like keypress timings, interrupt timings etc. Though it is not cryptographically strong, it is quick enough to produce random data. On each generation of random data, the kernel's estimate of true randomness contained in the pool decreases. The dev/urandom device does not have a limit of maximum random number generation and will return as many bytes as requested. With more and more bytes requested, the entropy pool will not have time to recharge and will output numbers that are merely cryptographically strong.

Limitations: Once the entropy pool is exhausted, it will pause until sufficient randomness is recharged. These types of pauses are unacceptable and risky as it can lead to a DoS attack against the application or the entire system. Also, with more applications being deployed on the cloud, it can be a problem for virtual machines as there are no hardware devices attached which can seed the entropy pool.

Review of Paper 2: Testing Cryptographically Secure Pseudo Random Number Generators with Artificial Neural Networks

Random Number Generators including Cryptographically Secure PRNGs are tested with Artificial Neural Networks to get a better understanding of its entropy. A handful of random number generators from different platforms are taken into consideration and a dataset is created. An analysis of the system is done during the training phase and regarding the expected behaviour of random numbers, an optimizer is defined.

Long Short-Term Memory method is used for the training and the neural network model is used against PRNGs and CSPRNGs like dev/urandom, SecureRandom, rand() etc. **This test proved to be useful as it pointed out weaknesses in PRNGs and even in some CSPRNGs too.** It was further concluded that the RNN can be used as a testbench to test various other RNGs and find the level of entropy that it possesses.

Sudharshan Krishnamurthy – Review of two papers

Review of paper 1: A Design for a Cryptographically Secure Pseudo Random Number Generator

This paper discusses designing a cryptographically secure pseudo-random number generator. The elementary and main techniques to it are diffusion and confusion but with proper guesses, the process of diffusion becomes into confusion. In other words, all encryptions are instances of substitutions. It is well known that these techniques are not capable of hiding patterns.

The next step towards increasing randomness was to use Permuted Congruential Generator. This is built around a Linear Congruential Generator where the data is passed through a series of permutations, some bit manipulations and then used as a PRNG output.

Limitations: In the above-mentioned technique too, the permutations were exhausted, and the running time became slow while generating high-quality random numbers. This meant that significant subset of seeds will produce poor quality randomness and each seed must be tested before it can be used for cryptographic use.

Review of paper 2: Security Analysis of Java SecureRandom Library

We all know that standard random number generation libraries are used by developers, but it does not have a high level of entropy. As a result, if an attacker comes to know about the generation time of a random number, he/she may generate the same sequence.

On top of generic random number libraries, SecureRandom library gives developers some generation upon predefined algorithms. This library is tested against various tests consisting of 1000 samples being tested 1000 times and then obtaining the average rate.

Limitations: It was concluded that though the library produces secure sequences, the reliability percentage tends to decrease when the length of the sequence increases. Also, the SecureRandom library shows weak security requirements with pattern-based tests

Saket Thombre – Review of two papers

Review of paper 1: Pseudo-Random Number Generator based on Logistic Chaotic System

Chaos and cryptography have a very special relationship where many characteristics of a chaotic systems such as sensitivity of initial value/system parameters, ergodicity, deterministic dynamics, and structural complexity are confusion and diffusion of keys. A Pseudo-Random Number Generator (PRNG) generation can be classified into initial state where the seed parameters need to be selected manually and normal state where generation of seed parameters ais related to the antecedent pseudo-random sequences.

Limitations: Security of PRNG for generating random number sequences uses floating point numbers generated by the current system state which is then rearranged and extracted. The next random sequences are generated on the previous 15 number sequences which increases the key length and makes the system parameters of the chaotic system uncontrollable. Thus, it has no statistical properties, and increased key space makes it difficult for exhaustive attacks.

Review of paper 2: Analysis of the Linux Random Number Generator

This paper studies the pseudo-random generator used in Linux operating systems (LRNG) since it is the most popular open-source pseudo-random number generator. This system is currently embedded in all Linux Environments including desktops, servers, PDAs, smart phones, media centres and routers.

Limitations: This study does not include any Linux Kernel options like multi-cpu hardware configurations or unique hardware configurations such as Qtronix keyboard or a mouse device which has a different entropy collection mechanism other than the one mentioned in this paper. Also, the LRNG is an open-source project and hence any adversary can read the whole source code and even trace changes in the configurations which might give the adversary an upper hand.

Common Project Proposal

We plan to develop a small working application in Java to demonstrate random number generation by taking physical input data from mouse movements and also from microphones. The random numbers generated can be used for further processing in the form of seed variables for cryptographic key, password, and binary data generations.

3. What is your approach to solve the problem?

The approach for this problem is to take the X and Y coordinates of the mouse movements, translating microphone data and then using these inputs to generate random numbers and passwords with higher entropy.

This personalized approach would make it harder to form patterns out of the random data generated. This tool can be used by developers to create a randomized seed and use it for further processing in cryptographic and hashing algorithms.

4. What is the novelty of the work? What is the deliverable? By the end of the project what you would have accomplished?

The novel feature in this project would be to get more physical input data by capturing sounds from the device's microphone. This data captured can be used along with mouse movements. This will make the RNG more personalized. Each mouse movement would be associated by X and Y coordinates of the respective screen or display. Additionally, we plan to take the sound captured from the microphone to add another layer of randomness to produce random numbers with higher entropy. The microphone's capturing frequency can also be altered as per the environment it is situated in.

All these would be coupled into a user-friendly application that displays the random numbers generated as per the user's requirements.