**THE REAL-TIME PAD**

By-Mohan Vamsi Kasukurthi (Computer Science)

-Tharani Konreddy (Computer Science)

**Introduction:**

When rolling a pair of dice, can you predict what digits are going to appear on each dice? Or when picking a single card from well shuffled deck of cards can we predict what could be that card without any magic tricks? So, the digit on a die thus rolled is random with a probability of 1/6 and the card thus picked is a complete random one with a probability of 1/52. Therefore, we can summarize random numbers as, **“Numbers produced from a process which results in unpredictable series of numbers and cannot be statistically reproduced”**. A series of random numbers once produced must never repeat. For computational uses, we require many random number series. If it were just one or two series, we can ask few of our friends to pick some cards from a deck or can ask them to roll a pair of dice several times or any one of the methods mentioned in the section “History”. But, we nowadays need hundreds of thousands of random numbers which is a bit difficult to obtain and real time methodologies like cards, dice etc. can’t be used. So, scientists came up with different methods to obtain random numbers. Some of them are based on a mathematical formula/algorithm which are called “pseudorandom number generators”. Many other types of random number generators have been developed and some of them are discussed in the section, “Existing and popular randomness generators”. In this chapter, we came up with a new algorithm to produce a true random series which is based on social media and the methodology produces a randomness such that the series satisfies most of the properties of ideal random numbers.

**Properties of true randomness:**

We can measure the randomness of a sequence based on how unpredictable the given sequence is. “**Unpredictability”** can be measured by checking if that succession of numbers meets all the characteristics of true random numbers. Whereas, unpredictability itself is one of the properties of true randomness. Some of the other properties are discussed here, “**distribution”** is one of the most basic properties. Random number sequences need to be uniformly distributed. Numbers in series must also be “**independent”**, i.e.., current value in the series must have no relation with any of the previous values in the series. “**Range”** is one of the simple yet most important properties of random number sequence. Most of the people don’t care much about the range, but for example if there is a series of N numbers, having range 0 to N, and N is small enough to a random number generator to generate a decent amount of different numbers which can be found by brute force, then the series of numbers are not unpredictable anymore. “**Coverage”** is another property, it refers to values generated by the generator must result all the numbers present in the output range of that sequence. For example, if there is a series of numbers ranging from 0 to A, then the generator must produce all the values between 0 and A when it allowed to run for optimal amount of time. Any true random number generators must not have a specific patterns or frequencies in their result set, this is because if a series is generated truly randomly then sometimes it is quite natural some values to be repeated. By the way, “**Serial Correlation”** is also other well-known property of random numbers, there must be no correlation between the consecutive numbers. It is worthwhile to mention that, not all the random number generators have all the properties mentioned above. Most of them lag many basic properties of true randomness. In this chapter, the algorithm thus mentioned satisfies most of the properties of true randomness. Greater details about this can be found in rest of the document.

**History:**

The usage of random numbers started very early in time. Even during the time of world war II, people used random numbers to encrypt their messages using ciphers like “One Time Pad”, which are completely based on random numbers and were trusted to be completely secure ciphers even now but only when we have enough random numbers to produce. Ancient random number generators achieve true randomness based on major/minute physical activities like dice, playing cards, coins, spinning wheels, keyboard stroke timing, lava lamps, fish tank bubbles, thermal noise, photoelectric effect, etc., But it is very hard to harness an infinite source of randomness using these physical activities. Which then lead to the development of modern random generators.

**Why in the world do we need Random Numbers...???**

Have you ever been to a casino? Or ever played a dice game? Or ever bought a lottery ticket? What if you already know which number is going to win a million dollars? Sounds awesome, isn’t it? What happens if you are playing a dice game with a friend and you already know what your turn is going to return? If you can guess the stream, how many people in the world do you think can do the same? (you are exceptional if you are a wizard or something beyond human). This can’t happen, right? Exactly, this is why we need randomness in the world.

Randomness is useful in a wide variety of streams like arts, sciences, statistics, cryptography, gaming etc. Most of the casinos’ popular, money making games such as slot machines, games of chance, gambling and many other modern electronic casino games work on randomness generators. Randomness is used in physical studies like electronic noise studies, engineering, operations research and in statistical analysis methods like bootstrapping. It is also used in arts, literature, military and music and even in our day to day lives.

Among all the applications of random numbers, cryptography needs very strict, true random numbers. To better understand the use of these random numbers in cryptography, let us consider an example. Alice wants to send a secret message to Bob. she got her own reasons to keep the message secret. But the attacker, Eve want to know what’s going on between Alice and Bob. So, Alice try to encrypt the secret message with a key which is known to just herself and to Bob so that the message looks completely senseless. But, how can Alice and Bob who are several thousands of miles away from each other know exactly same key? If Bob can know the key or guess the key from thousand miles away, Eve also can know the exact same key because he is sitting in the room next to you. So, cryptographic keys should be unpredictable but should be known to only the sender and the receiver. This is one of the areas in cryptography, where we need true randomness. If the key is not random, then it is said that the encryption technique is compromised and can be easily attacked by Eve. Hence, there are ample number of random number generators developed for cryptography. Some of them are, Middle-square method, linear congruential generators, Mersenne twister, pseudorandom generators, linear feedback shift register, Blum-Blum- Shub, etc., But, why are the cryptographic professionals still looking for true random generators. Because, as explained in the section “Properties of ideal random number set”, harnessing true randomness is almost impossible with all the properties being satisfied.

**Existing and popular Randomness generators.**

Any type of random generators can be classified as one of the two types, 1) true random generators and 2) pseudorandom number generators. True random number generators are sometimes called as hardware random numbers which produce random numbers based on natural events occurring in and around the world, such as fish movements in fish tanks\aquariums, baby movements in mother womb, planets and galaxies movements, by flipping coins and dice, and using all kinds of atmospheric noises. Even though they are considered to be truly random, physical devices used to capture the random phenomenon are expensive and can break after some point of time. Pseudorandom number generators are generators that use mathematical formulas or standard algorithms to produce sequences of random numbers. These are assumed to be random by expecting that the initial seed is not known to anybody. Any of the existing random number generators are considered as not secure for using in cryptographic uses, this is because if the initial values are known then anybody can easily generate the next values in the succession. Some of the pseudorandom number generators include, Linear congruential generators, which have serial correlation between successive values of the sequence which leads to violation of  the randomness property. Pseudorandom generators(PRNG) also have a period from where the sequence generated will be repeated, hence PRG is also predictable. Blum-Blum-Shub Random number generator is secure to use in cryptographic uses but it is very slow to implement. It can be speedup at cost of some security. All polynomial random number generators are insecure to use for cryptographic applications, but they are still in use only because they can be generated very fast. It tells us that till date cryptography is in search of one such random number generator that is quick to implement and at the same time which is unpredictable. This leads to our idea of real time random number generators based on an infinite real time source.

**Standards for RTP:**

The standards for RTP are defined so that, the random stream thus generated can be given several levels of security and can only be accessed by sender and the receiver.

**Social media account standards:**

* Must use at least 2 accounts.
* Number of accounts must always be even.
* Accounts should be private.
* No friends are allowed in the accounts.
* Accounts should not follow or be followed by other accounts that belongs to ‘n’ (set of accounts we use for the RTP).
* Posting from any account is not accepted i.e., using it for personal use.
* Verification methods for the accounts must only be using a phone number instead of email.
* Number of accounts should be kept secret.
* Accounts can only be known by sender and the receiver.

**User name Standards:**

* The usernames for the accounts should be at least 10 characters.
* Must use alphanumeric characters with Capitals and symbols.
* Usernames in the accounts that belong to ‘n’ must not be similar either syntactically or semantically.
* Usernames must make no sense with any of your personal details.
* Must be unpredictable.
* Should be kept very secret.

**Password Standards:**

According to [4] we have set up few password standards for RTP to have a secure password,

* For RTP, we choose the password length should be at least 15 characters in length.
* Must contain at least 3 lower case characters.
* Must have at least 3 upper case characters.
* Must contain at least 3 special symbols.
* Must contain at least 3 numbers.
* Must not be a keyboard sequence or something very common such as “abcdefghIJK 1234” etc.
* Must not match your username.
* Must have at least one space.
* Do not share the password.

**Friendship standards:**

* Can follow from the account but cannot be followed by anyone.
* Must follow at least 500 people from each account.
* The people you follow must be from different places around the world so that we can have data 24/7.

**Methodology:**

Nowadays social media has been playing a very important role in our day to day lives. It is an invisible network that is spread all over the world. According to a survey, seven out of ten Americans use some kind of social media i.e., 69% of the public [3]. If this is just in the united states, what about across the globe? And is there any possible way that all kinds of social media collapse and probably wouldn’t be back? It is completely hypothetical and could be happened only when the world bans social media. This is our inspiration to use the social media as a source of randomness generator. So, it obviously is an “**infinite source of randomness**”. Further in the section we illustrate our methodology in detail and a set of standards we defined to make the randomness generator very secure and useful for cryptography. We named our generator “The Real-Time Pad” (RTP). The name is chosen in the honour of one of the most secure ciphers “One Time Pad” (OTP). It was considered the secure and safest ciphers back in the days but it has two noteworthy drawbacks to be used in modern times. They are 1) Lack of infinite randomness source 2) Key distribution. Not only in OTP, but many other ciphers have these two drawbacks in common. Instead of much more research in these areas, the mankind has just moved on to some other ciphers which are based on lack of computational capacity of an attacker. For example, most of the modern ciphers are in such a way that, even though an attacker has the ciphertext and knew the methodology, they might not have enough computational capacity to brute force the ciphers. So, we have also introduced this kind of feature in our RTP. The methodology, standards, different types of attacks possible on the RTP are described further in detail. In this section, we describe our methodology in detail.

The steps involved in the development of RTP are:

1. Choosing the type of social media (S).
2. Choosing number of accounts (n).
3. Account set-up.
4. Working of RTP algorithm.
5. **Choosing the type of social media:**

Among all the social media across the world, twitter is ranked number 10 as of April 2017, based on the active users. It has around 300 million active users per month. While the top 9 are Facebook, YouTube etc. [2] which are not very famous for providing data for computational uses. Most of the data mining researches, big data analyses use twitter as a source of real time data over the other social media. Therefore, as of now, twitter provides very good support for their data to be downloaded for computational purposes. Twitter has different types of methods available for their data download such as rest APIs, streaming APIs etc. as mentioned in [1].

**2. Choosing number of accounts:**

Once we choose the best social media with a good source of data, the next thing to do is to choose how many accounts to get the data from. The more accounts we can have, the more secure the random stream and the most unpredictable the random series is. The value of ‘n’ should always be even. Here, in our implementation, we took ‘n’ as 2 for ease of use. The reason for an even number of accounts is that it would be simpler to deskew pairs of accounts together. Deskewing is to make the obtained series much more unpredictable. More on deskewing is discussed in the “Deskewing two accounts” section. ‘n’ can be any even number but all the accounts must satisfy the set of standards we described for RTP. There is another reason to choose multiple accounts i.e., for security purposes. If we just use a single account, an attacker might first compromise the user account and then can access the similar random stream. So, it is always very secure to have multiple user accounts.

**3. Account set-up:**

Once we create the required number of accounts according to the RTP standards, we should get the keys for the accounts to access them programmatically. In this section, we discuss the way to obtain the developer keys for one of the twitter accounts and the keys for all the accounts must be obtained in the very similar manner.

1. Login to your twitter account and go to <https://dev.twitter.com/> .
2. Now click, “My apps” on the page.
3. Press “Create New App”.
4. In the resultant page, enter name, description, website as “XXXX [Current Time]”, “XXXX”, “<http://example.com>”. (if name is already taken, you can choose anything you feel unique).
5. Check the “Developer Agreement” check box.
6. Now, click “Create your twitter application”.
7. Go to “Keys and tokens” tabs.
8. Click “Create Access Tokens”.

Once all the above steps are done, there are 4 things we should keep handy, 1) Consumer Key 2) Consumer Secret 3) Access Token 4) Access Token Secret.

After getting these keys, account setup is done and we are ready for RTP algorithm.

**4. The working of RTP algorithm:**

The algorithm is described for two user accounts. When the number of accounts is more than 2, the same algorithm can be applied for the first account from the first and last account from the last and second account from first and last but one account from last and so on. For ex, if we have 6 multiple accounts, we can apply the RTP algorithm for account pairs like [1,6], [2,5], [3,4]. And finally concatenating the results. The RTP is implemented in Java. Any preferred language with twitter support can be chosen. First, the project was built using python using the “Tweepy” module for twitter. Later, for our convenience in the GUI development, we implemented the whole project again in Java. The java class that was very helpful for the application development is “twitter4j”. The class provides methods to access the user timelines and those that can also be used to post on user’s timeline programmatically. The RTP algorithm pseudocode is as follows:

//Generates the random bits.

String Generate Key()

{

define consumerkey1, consumersecret1, accesstoken1, accesstokensecret1; //first account keys.

define consumerkey2, consumersecret2, accesstoken2, accesstokensecret2; //second account keys.

Set key1=null;

set key2=null;

ConfigurationBuilder cb1(first account keys);//configure the account1.

ConfigurationBuilder cb2(second account keys);configure the account 2.

twitter1= TwitterFactory(cb1); //get instance of account 1.

twitter2= TwitterFactory(cb2);//get instance of account 2.

List<twitter1Statuses>= twitter1.getHomeTimeLine(); //get account 1 home timeline tweets.

List<twitter2Statuses>= twitter2.getHomeTimeLine(); //get account 2 home timeline tweets.

//Cleaning the twitter 1 statuses.

for every status in twitter1Statuses do:

Split each status into separate words;

For every word do:

If word starts with ‘.’, skip the word.

If word contains ‘http’, skip the word.

If word equals ‘the’, skip the word.

If word equals ‘a’, skip the word.

If word contains ‘retweeted’, skip the word.

Try:

If word is not empty:

If the first character ‘x’ of the word is in [a-z] or [A-Z]:

If key1 is not null:

key1=key1+’x’;

Else

key1=x;

Catch:

Exception e;

//Cleaning the twitter 2 statuses.

for every status in twitter2Statuses do:

Split each status into separate words;

For every word do:

If word starts with ‘.’, skip the word.

If word contains ‘http’, skip the word.

If word equals ‘the’, skip the word.

If word equals ‘a’, skip the word.

If word contains ‘retweeted’, skip the word.

Try:

If word is not empty:

If the first character ‘x’ of the word is in [a-z] or [A-Z]:

If key2 is not null:

key2=key2+’x’;

Else

key2=x;

Catch:

Exception e;

Return getBitString(key1,key2);

}

//takes two strings and returns the deskewed value of both strings mixed together.

string getBitString(key1,key2)

{

result=null;

If key1 length< key2 length

len= key1 length;

Else len= key2 length;

Repeat for len times:

result=result+ characterDeskew( key1(len) character, key2(len) character);

return result;

}

//take two characters and converts them to binary and then deskews them.

characterDeskew(character 1, character 2) {

result=null;

char1binary=Convert character1 to binary;

char2binary=Convert character2 to binary;

chars1[]= split char1 binary to digits;

chars2[]= split char2 binary to digits;

For every digit ‘m’ in chars1[] and chars2[]:

result=result+ deskew(chars1[m],chars2[m]);

return result;

}

//Takes two digits and deskews them.

String deskew( digit1, digit2){

If digit1=0 and digit2=0

Return ‘’;

Else if digit1=0 and digit2=1

Return 0;

Else if digit1=1 and digit2=1

Return ‘’;

Else

Return 1;

}

For better understandability, simply the algorithm is:

1. Take first account’s first tweet’s first pure alphabetic word’s first character.
2. Take second account’s first tweet’s first pure alphabetic word’s first character.
3. Convert both to binary.
4. Deskew them both together.
5. Repeat for every word of every tweet for latest 20 tweets in each account.

**DE skewing:**

De skewing is a technique that is used to get equal number of ones and zeros from a pair of bit sequences. The process eliminates the bit patterns like ‘00’ and ‘11’ and makes the RTP perfectly random. Given two sequences of bits ‘01101101’ and ‘11100011’, first we need to pair them up as [01], [11], [11], [00], [10], [10], [01], [11]. In these sequences, we have a total of 6 zeros and 10 ones. DE skewing eliminates ‘00’ and ‘11’ and converts [10] to ‘1’ and [01] to ‘0’. Therefore, the resultant sequence of the paired-up bits is ‘0110’.

**GUI Development:**

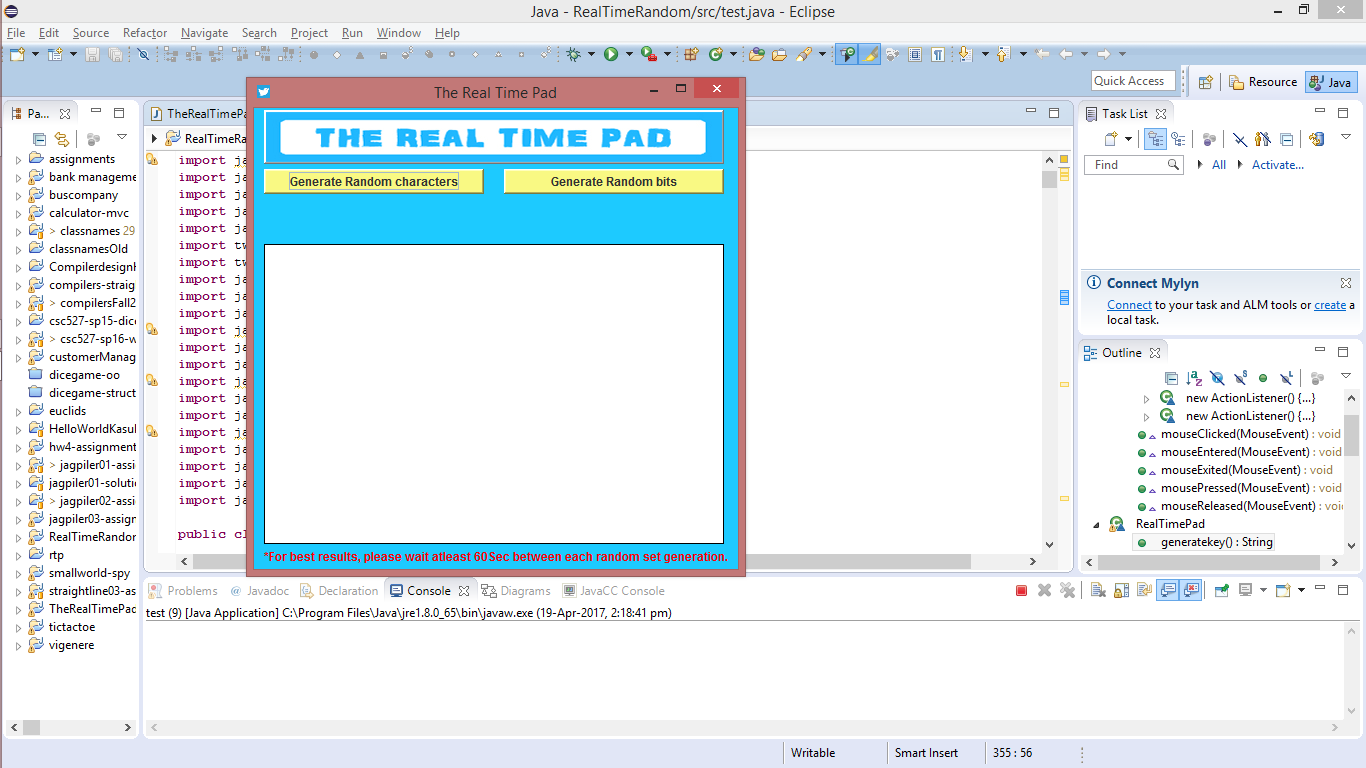
Once we obtain the random bits, we provide a user friendly, graphical user interface for our RTP. The GUI is developed using “Java Swings”. We made the UI as friendly and attractive as possible by using different components such as “Mouse Listeners”, “Images”, “Text Labels”, “TextArea”, “Buttons”, “Tooltips”, “User friendly messages” etc. The images and icons required for the UI are developed by us using Adobe Photoshop 7.0.

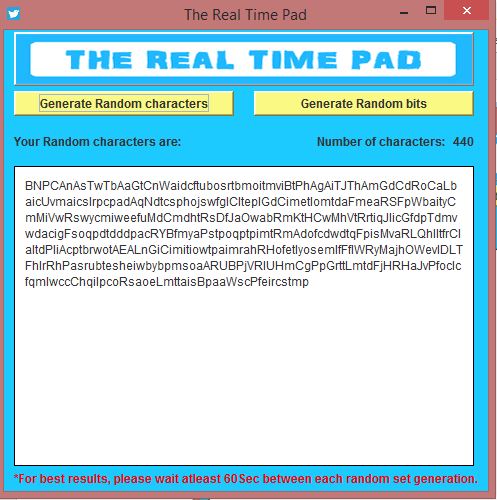
Below are a few screen shots of the RTP GUI:

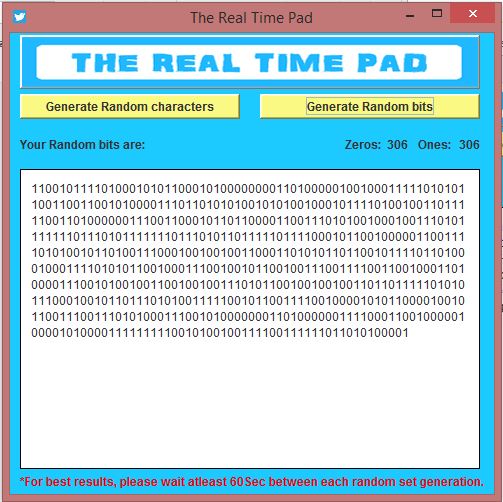
**ScreenShots and images:**

****

****

****

****

****

**References:**

1. <https://dev.twitter.com/docs>.
2. <https://www.statista.com/statistics/272014/global-social-networks-ranked-by-number-of-users/>.
3. <http://www.pewinternet.org/fact-sheet/social-media/>.
4. <http://its.ucsc.edu/policies/password.html>.
5. <https://en.wikipedia.org/wiki/Applications_of_randomness>
6. <http://arminstraub.com/teaching/cryptography-spring17>
7. <https://www.eg.bucknell.edu/~xmeng/Course/CS6337/Note/master/node37.html>
8. <http://anomaly.org/ThinAir/charactr.html>