Intelligent Data Analysis 10 (2006) 539–554
IOS Press

Analyzing a class of pseudo-random bit generator through inductive machine learning paradigm

Shri Kant[a] and Shehroz S. Khan[b]
[a]*Scientific Analysis Group, Defence R&D Organization, Metcalfe House Complex, Delhi-54, India*
E-mail: shrikant@scientist.com
[b]*America Online India Pvt. Ltd, RMZ Titanium, 3rd Floor, 135, Airport Road, Bangalore-560017, Karnataka, India*
Tel.: +91 9845935310; E-mail: shehrozkhan@rediffmail.com

Received 11 October 2005
Revised 22 December 2005
Accepted 19 February 2006

Abstract. Random number generation is an integral part of strong cipher systems. If a pseudo-random sequence can be predicted with better than chance probability then the generator is considered to be cryptographically weak. This paper deals with next bit prediction of pseudo-random binary sequences generated by Linear Feedback Shift Register (LFSR) and LFSR-based Pseudo-Random Bit Generators (PRBG), using inductive Machine Learning (ML) paradigm, namely C4.5 the most common and widely used inductive data mining algorithm. This machine learning technique has been introduced to convert the theoretical prediction problem into a classification problem, which we coined as *Classificatory Prediction* problem. We further extended the use of this technique to predict next bit without having any knowledge of subsequent bits of the PRBG and can be termed as true *Next Bit Predictor*. The technique used is independent of the parameters and domain knowledge of the pseudo-random bit generators. The present study is a comprehensive extension of the work done by Hernandez et al. [15]. We performed meticulous experiments (over wide range of LFSRs) and came out with a more explanatory analysis. Our classificatory prediction results paved the way for the evolution of the next bit prediction model.

Keywords: Machine learning, classificatory prediction problem, next-bit prediction, stream cipher, linear feedback shift register, pseudo random bit generator

1. Introduction

Random number generation is a critical issue in cryptography. Generating true random numbers, especially on computers that are typically designed to be deterministic is a real challenge in the cryptographic world. The best a computer can produce are pseudo-random numbers, which are generated from some random initial values. The pseudo-random numbers look like random numbers and have good statistical properties [5]. The period of the pseudo-random number generator should be very high and should not repeat for a large length. Normally in cryptographic applications, random number is a number that cannot be predicted by an eavesdropper before it is generated. It must be computationally infeasible to predict the next bit of cryptographically secure pseudo-random sequences, given complete knowledge

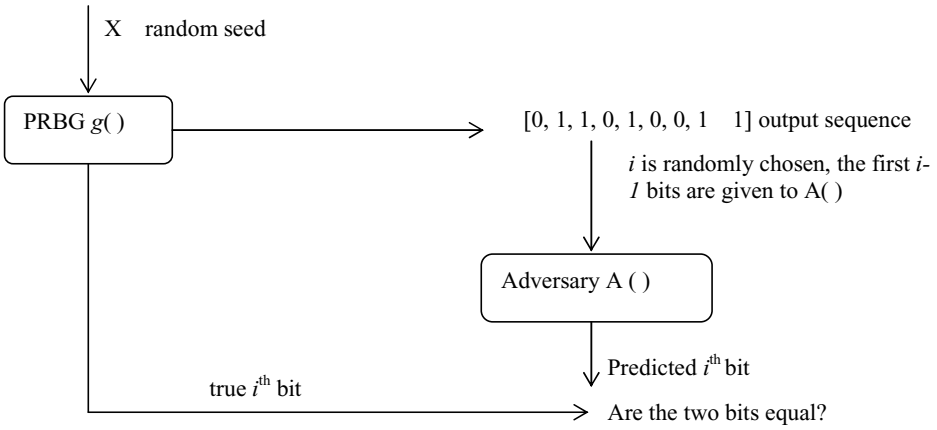

Fig. 1. Theoretical Next-Bit Prediction Model.

of the algorithm or hardware generating the sequence and all of the previous bits in the stream [1]. Typically, if the pseudo-random numbers are to be in the range [OK $n-1$], an adversary cannot predict that number with probability slightly better than $1/n$ (i.e. chance probability). A more formal definition is given in [23]. Let $g:\{0,1\}^n \rightarrow \{0,1\}^{l(n)}$ be an efficient (computable in polynomial time) function, $l(n)$ being a polynomial with $l(n)$>n. Let X and I be random variables uniformly distributed respectively on $\{0,1\}^n$ and on $\{1,\ldots,l\,(n)\}$. Then g is a next bit unpredictable PRBG, if for all adversaries A running in polynomial time the success probability (prediction probability) of A for g is

$$P\left[A\left(I, g\,(X)_{\{1,\ldots,I-1\}}\right) = g\,(X)_I\right] \;\langle\; \frac{1}{p\,(n)} \forall p,$$

where p is a polynomial.

Figure 1 explains the working of a next bit predictor. First a seed and a number i-1 of bits are randomly chosen; the adversary must then predict the i^{th} bit with the complete knowledge of first i-1 bits as input, in polynomial time.

Machine learning techniques are very widely used by data mining community for knowledge discovery, pattern enumeration and acquiring the predictive ability/knowledge in real-life applications. This paper deals with prediction of next bit of a pseudo-random bit generator of a class of Pseudo-random Number Bit Generator namely Linear Feedback Shift Register (LFSR) and LFSR-based PRBGs, using Inductive Machine Learning (ML) paradigm [21].

Hernandez et al. [15] reported a General Next Bit Predictor (GNBP) for predicting next bit for LFSR by converting the next bit predictor theoretical model into a classification problem using C4.5 as inductive algorithm. They have used their GNBP for one particular primitive polynomial $\left(x^{15}+x^1+1\right)$ and depicted the prediction rule for that polynomial to predict the subsequent next bits. The claims made in their research work are not comprehensive; hence the interpretation of their results can not be generalized. S.S. Khan [27] extended this work on various other LFSRs and came out with an analysis to completely break LFSRs by constructing the generating primitive polynomial using decision trees. The present work is a more comprehensive, exhaustive and analytical study of both the above mentioned works and introduces the concept and algorithm for the 'next bit prediction', to assist an analyst in a scenario when he is left with fewer amount of bits and has to predict the future bits, without possessing the domain knowledge and parameters involved in the PRBG.

The rationale behind this work is to learn and extract knowledge from the bit sequences generated by LFSR based PRBG's and to be able to predict the future output sequences and hence to capture it's all parameters for solution. The analytical study has been carried out without any domain knowledge and apriori knowledge of parameter of the LFSR, so that this model can be generalized for analysis of any stream cipher crypto primitive. The aim is to find regularities and hidden patterns in the output of key stream generator i.e. PRBGs and to develop a generalized predictive classification model.

The paper is organized as follows. Section 2 presents a brief introduction to LFSR as a stream cipher. Section 3 gives an introduction to machine learning, Minimum Description Length (MDL) and C4.5 inductive algorithm. We present the technique of next bit prediction as a classification problem, which we termed as C*lassificatory Prediction* in Section 4. Classificatory Prediction for Linear Feedback Shift Register and Geffe generator [1] is detailed in Section 5. Section 6 introduces the concept and presents the algorithm for next bit prediction. We conclude our presentation in Section 7.

2. Linear feedback shift register (LFSR) as stream cipher

Before introducing stream cipher and LFSR we introduce here the basic definition of cryptography, because stream cipher is one of the prevalent methods for cryptography. Cryptography is a science of transforming the plain messages into the disguised form in such a way so that an unauthorized receiver should not be able to deduce any information out of it [1,3]. Cryptography is categorized as symmetric key and asymmetric key on the basis of whether the sender and receiver uses the same key or different key. Since the present work is more related to symmetric key cryptography we will describe it briefly as follows.

In symmetric key cryptography plaintext bits $p_1, p_2, \ldots, p_n$ are transformed into ciphertext bits $c_1, c_2, \ldots, c_n$ by using certain invertible transformation together with a *key*. These transformation mechanisms are broadly categorized as block cipher cryptography and stream cipher cryptography. Block cipher specifies a memory-less device which transforms a message block $[p_1, p_2, \ldots, p_n]$ under control of a key into cipher text block $[c_1, c_2, \ldots, c_n]$, where the message text alphabet and cipher text alphabet usually are identical. A stream cipher is a device that transforms each message bit p_i into a cipher bit c_i by means of a function which depends on both the secret key K and the internal state of the stream cipher at time i. In general the stream ciphers algorithm converts each p_i into c_i as follows

$$c_i = m_i \otimes k_i, (\otimes \text{ is Exlusive-OR})$$

where k_i's are generated through a key stream generator. In general, a symmetric key cipher is considered secure if there is no method less expensive (in time, memory requirements, etc) than brute force. A **brute force attack** is a method of defeating a cryptographic scheme by trying a large number of possibilities; for example, exhaustively working through all possible keys in order to decrypt a message. In most schemes, the theoretical possibility of a brute force attack is recognised, but it is set up in such a way that it would be computationally infeasible to carry out. In a brute force attack, the expected number of trials before the correct key is found is equal to half the size of the key space. For example, if there are 2^{64} possible keys, a brute force attack would, on average, be expected to find a key after 2^{63} trials. The security of stream cipher depends entirely on the non-linear structure of the key stream generator. If the generator's output is endless stream of 0's or 1's then the c_i's and m_i's are same or merely transpose will give us m_i's. If it produces long repeating bit pattern the algorithm will be a simple XOR with negligible security. If the generator spits out an endless stream of random bits (true random), then it behave like a

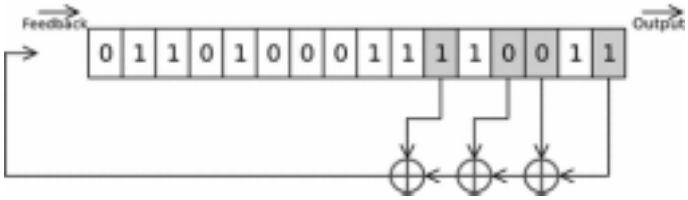

Fig. 2. Linear Feedback Shift Register.

Vernam cipher [22]. In such ciphers the device emits a random sequence where each bit is equally likely to be 0 or 1 independently of the preceding bits. Such system in general is impractical due to both key generation time and key distribution.

The operational disadvantages of such system led to the development of synchronous key stream generator generating the same key stream at the both end (enciphering and deciphering) under the influence of the same shared secret key. The basic problem of key stream generator design is to generate a running key k_i with very large period, large linear complexity and uniform distribution properties. The large period ensure the repetition of not getting the same cipher text for same plain text; the large linear complexity implies large period and uniform distribution properties frustrate the prediction of next bit with knowledge of previous bits. One of the most useful devices in the generation of running keys is the Linear Feedback Shift Register (LFSR).

A linear feedback shift register [1] is a shift register whose input is the exclusive-or (XOR) of some of its outputs (Fig. 2). The outputs that influence the input are called *taps*. A maximal LFSR produces an n-sequence ($n = 2^l - 1$, where l is the number of stages i.e. the number of bit positions in the register, before the shift register returns to its original state and the n-bit output sequence repeats), unless it contains all zeros. The tap sequence of an LFSR can be represented as a polynomial mod 2 – called the feedback polynomial. For example, if the taps are at 11^{th}, 13^{th}, 14^{th} and 16^{th} bits (Fig. 2), the polynomial is $x^{16} + x^{14} + x^{13} + x^{11} + 1$. If this polynomial is primitive, then the LFSR is maximal. A primitive polynomial of degree m is an irreducible polynomial that divides $x^{2^{m-1}} + 1$, but not $x^t + 1$ for any t that divides $2^m - 1$ [29].

LFSRs have long been used as a pseudo-random number generator for use in stream ciphers (especially in military cryptography), due to the ease of construction from simple electromechanical or electronic circuits, long periods, and very uniformly distributed outputs. J.L. Massey [16], showed that iterative algorithm introduced by Berlekamp [7] for decoding BCH codes provides general solution to the problem of synthesizing the shortest linear feedback shift register capable of generating a prescribed finite sequences of digits. Since the outputs of LFSRs are completely linear, they lead to fairly easy cryptanalysis. Given an output sequence of a l stage LFSR, a minimal size LFSR can be easily constructed using Massey algorithm [16].

3. Machine learning and introduction to C4.5

Machine Learning objectives to develop algorithms that can learn from the observations (data), adapt its behavior and continuously improve upon that as human intelligence does. Inductive machine learning is the study of learning by examples so that accurate predictions can be made for future examples.

The Minimum Description Length (MDL) Principle [24] is a relatively recent method for inductive inference that provides a generic solution to the model selection problem. MDL is based on the following

insight: any regularity in the data can be used to compress the data, i.e. to describe it using fewer symbols than the number of symbols needed to describe the data literally. The more regularities there are, the more the data can be compressed. Equating 'learning' with 'finding regularity', we can therefore say that the more we are able to compress the data, the more we have learned about the data. The MDL is widely used for model selection in various machine learning problems. In practice, MDL works well on inference of decision trees. Among efforts that have been put into development of tree based classification techniques in recent years, Quinlan and Rivest [17] proposed a method of inferring decision trees using MDL.

Inductive paradigm in machine learning have many inductive algorithms like CART [8], ID3 [19], C4.5 [18] and SLIQ [26]. We have chosen C4.5 algorithm as inductive technique for classification and later for next bit prediction of Linear Feedback Shift Register (LFSR) and LFSR-based PRBGs. C4.5 is a successor of ID3 with some improvement and added capabilities as described in [18]. The main difference of C4.5 with respect to ID3 are its handling of data with missing values, capability of using continuous attribute values, minimizing error during pruning and forming rules sets (if then else rules) from the constructed decision trees.

C4.5 induces classification rules from a training set to form a decision tree. The decision tree is defined as a tree in which each node is an attribute, each arc from this node is a possible value for that attribute, and each leaf is the expected value for the category of the pattern obtained following the entire path from the root of the tree to that leaf. The general idea to construct a decision tree is to decide at each node which of the non-used attributes is most informative for the classification of all the patterns represented by the path from the root to that node. Applying this idea, recursively, for every node generates the decision tree. C4.5 uses the concept of gain ratio [20] to make a tree of classificatory decisions with respect to a previously chosen target classification. The information gain can be described as the effective decrease in entropy (usually measured in terms of 'bits') resulting from making a choice as to which attribute to use and at what level. The gain ratio is an information-based measure that takes into account different numbers (and different probabilities) of test outcomes. Let C denote the number of classes and $p(D,j)$ the proportion of cases in D that belong to the j^{th} class. The residual uncertainty about the class to which a case in D belongs can be expressed as

$$\textit{Info}(D) = -\sum_{j=1}^{C} p(D,j) \times \log_2 (p(D,j))$$

and the corresponding information gained by a test T with k outcomes as

$$\textit{Gain}(D,T) = \textit{Info}(D) - \sum_{i=1}^{k} \frac{|D_i|}{|D|} \times \textit{Info}(D_i)$$

The information gained by a test is strongly affected by the number of outcomes and is maximal when there is one case in each subset D_i. On the other hand, the potential information obtained by partitioning a set of cases is based on knowing the subset D_i into which a case falls; this *split information*

$$\textit{Split}(D,T) = -\sum_{i=1}^{k} \frac{|D_i|}{|D|} \times \log_2 \left(\frac{|D_i|}{|D|} \right)$$

tends to increase with the number of outcomes of a test. The gain ratio criterion assesses the desirability of a test as the ratio of its information gain to its split information. The gain ratio of every possible test is determined and, among those with at least average gain, the split with maximum gain ratio is selected.

In some situations, every possible test splits D into subsets that have the same class distribution. All tests then have zero gain, and C4.5 uses this as an additional stopping criterion.

The recursive partitioning strategy above results in trees that are consistent with the training data, if this is possible. In practical applications data are often noisy – attribute values are incorrectly recorded and cases are misclassified. Noise leads to overly complex trees that attempt to account for these anomalies. Most systems *prune* the initial tree, identifying sub-trees that contribute little to predictive accuracy and replacing each by a leaf.

We have chosen inductive paradigm approach of machine learning, because we generate a concept for Class Label (CL) that is the "next bit either 0 or 1" and then generalize with set of training patterns. Finally we arrive at a concept description that can predict the value of the class lebel for all the previously observed patterns.

3.1. C4.5 package

We downloaded the C4.5 package Release 8 for Unix/Linux version [10]. This package consists of four programs viz. *C4.5, C4.5rules, Consult* and *Consultr*. A brief description of each program is given below

1. *C4.5 Program* – $C4.5$ is a program for inducing classification rules in the form of decision trees from a set of given examples. All trees generated in the process are saved. After each tree is generated, it is pruned in an attempt to simplify it. The "best" pruned tree is saved in machine-readable form in a file. All trees produced, both pre-simplification and post-simplification, are evaluated on the training data. If required, they can also be evaluated on unseen data.
2. *C4.5rules – C4.5rules* reads the decision tree or trees produced by $C4.5$ and generate a set of classification rules from each tree and from all trees together. For each tree that it finds, the program generates a set of pruned rules, and then sifts this set in an attempt to find the most useful subset of them. If more than one tree was found, all subsets are then merged and the resulting composite set of rules is then sifted. The final set of rules is saved in a machine-readable format in a rules file. Each of the rule sets produced is then evaluated on the original training data and (optionally) on the test data.
3. *Consult – Consult* reads a decision tree produced by $C4.5$ and uses this to classify items whose description is provided by the user. When all the relevant attributes have been determined, *consult* will give one or more classes that the item might belong to. The likelihood of a class may be indicated by a probability, followed sometimes by a probability interval
4. *Consultr: – Consultr* reads a rule set produced by *C4.5rules* and uses this to classify items whose description is provided by the user. When all relevant attributes have been determined, *consultr* will give a class that the item might belong to. The likelihood of the class may be indicated by a probability.

4. Classificatory Prediction

The next bit predictor as presented in [3], suggests that, an algorithm when, given all previous bits generated from a pseudo-random generator (PRBG), can efficiently predict the next bit with higher than chance probability. Suppose we have a pseudo-random sequence of bits $p_1, p_2, \ldots, p_n$ generated by a PRBG, then a next bit predictor should compute the p_{n+1}^{th} bit given the previous ones, with probability greater than $1/2$ (chance probability) without knowing the particular set of parameters used by the PRBG.

We concentrate our efforts to find patterns and regularities in the pseudo-random sequences using the machine learning technique. Here, the advantage lies in the fact that no a priori domain knowledge is required to predict the next bit. We propose the technique of next bit prediction in two ways.

1. As a traditional classification problem to predict the future bits of the PRBG which we coined as *Classificatory Prediction* problem. The idea is to adjudge the prediction accuracy of the known bits to be predicted.
2. Once we gain confidence in prediction (through classification) and analyzing the PRBG then we go for *Next Bit Prediction*, in which limited bits are needed to predict the unseen bits, which subsequently paves the way for the future bits to be predicted. We first explain the methodology adopted for *Classificatory Prediction* problem.

Methodology adopted: Classificatory Prediction Model:
Consider n bits generated from a PRBG are $p_1, p_2, \ldots, p_n$. Choose a suitable block size, b, as a training pattern (P_i) associated with a Class Label (*CL* $\rightarrow$ 0 or 1) such that

$$P_1 = p_1, p_2, p_3, \ldots, p_b \quad CL \rightarrow p_{b+1}$$

$$P_2 = p_2, p_3, p_4, \ldots, p_{b+1} \quad CL \rightarrow p_{b+2}$$

$$\vdots$$

$$P_{n-b} = p_{n-b}, p_{n-b+1}, \ldots, p_{n-1} \quad CL \rightarrow p_n$$

The $n - b$ patterns from P_1 to P_{n-b} serves as the pattern space for the classification model we have adopted. Out of these $n - b$ patterns, an appropriate number of patterns say (α) are used for learning and the remaining patterns are used to predict the ($n - b - \alpha$) bits of the pseudo-random sequence.

Here, we are not considering all previous bits at once, as presented in Fig. 1. This is a slight deviation from the theoretical next bit predictor model. The learning process is dependent on a number of prefixed block sizes of generated sequence so as to accommodate maximum possible regularities, patterns and combinations.

To use C4.5 package as a next bit predictor for classification problem, we supply α patterns as training data set, and $n - b - \alpha$ patterns as test data for the prediction of $n - b - \alpha$ bits. Intuitively, the size of the training data set should be sufficient to capture maximum regularities and extract generalized conclusions that yield high degree of prediction. The algorithmic steps followed are presented below:

4.1. *Algorithm: Classification Prediction* ($p_{i,n}$)

Input: p_i – the pseudo-random number sequences generated by a generator.
n – the length of the pseudo-random sequence taken for study
Output: Classificatory prediction results on test data
begin

1. Select a suitable block size (b).
2. Generate $n - b$ blocks (P_1,P_2, …,P_{n-b}) and associate every pattern with their class label (CL), as described above.
3. Select 'α' number of patterns for training.
4. Select remaining $n - b - \alpha$ number of patterns as test data.

5. Run the $C4.5$ program to generate decision trees.
6. Run the *C4.5rules* program to generate the classification rules from the decision trees generated by $C4.5$ program.
7. Choose the simplified classification rule set for analysis.

end

If prediction results are not satisfactory, then by tuning the parameters: the block size, b, number of patterns for learning, α and the length of the pseudo random sequence, n, we may arrive at satisfactory results.

In the succeeding sub-sections we show the use of our proposed methodology for classificatory prediction of two different pseudo-random number generators:

1. Linear Feedback Shift Register (LFSR) and
2. Geffe Generator

5. Classificatory prediction of Linear Feedback Shift Register

The essential difference between Massey algorithm and the proposed next bit predictor model is that the former algorithm uses the domain knowledge of the LFSR whereas the proposed model is free from this limitation.

Recently, Hernandez et al. [15] reported a General Next Bit Predictor for predicting two families of pseudo-random number generator namely truncated linear congruential generator (LCG) and linear feedback shift register by converting the next bit predictor theoretical model into a classification problem using C4.5 as inductive algorithm. They thoroughly examined the case of LCG. This type of solution is particularly very significant from cryptanalysis point of view because it does not use any domain knowledge of PRBG. Where as Massey algorithm need the complete domain knowledge. In the case of LFSR they have studied one particular primitive polynomial $\left(x^{15}+x^{1}+1\right)$. The claims made in their research work are not comprehensive; hence the interpretation of their results cannot be generalized. They did not comment about the optimal block size requirement to predict correctly. We analyze the problem in a more comprehensive manner by considering various primitive polynomials ranging from degree 10 to 41 to arrive at general conclusion. We looked into the problem of classificatory prediction on three counts

1. *Issue of block size:* To check the minimum block size required for correct rule formation
2. *Sequence length requirement*: To check how many bits are needed to learn from the pattern space such that the prediction is maximally correct
3. *Determination of primitive polynomial:* To arrive at classification rules for constructing the primitive polynomial used in LFSR

5.1. Issue of block size

The primitive polynomials (listed in Table 1 – column 2) are taken for analysis. As discussed in Section 4, we have generated $n-b$ patterns for each of these polynomials respectively. Here n is the period of the LFSR of degree d i.e. $n=2^{d}-1$ and b is the block size arbitrarily chosen, initially. We carried out the experiment to check the minimum size of block required to learn correctly using C4.5 algorithm. Initially, we took 99% of the pattern space for training and the remaining 1% for testing

Table 1
Implication of Block Size over Classificatory Prediction Error

Degree of polynomial (d)	Primitive polynomial chosen	Block size $b < d$	Classificatory prediction Error (%age)	Block size $b \geqslant d$	Classificatory prediction Error (%age)
10	$x^{10} + x^3 + 1$	9	36.4	10	0
11	$x^{11} + x^2 + 1$	10	52.4	11	0
12	$x^{12} + x^6 + x^4 + x + 1$	11	56.1	12	0
13	$x^{13} + x^4 + x^3 + x + 1$	12	50.0	13	0
14	$x^{14} + x^5 + x^3 + x + 1$	13	56.1	14	0
15	$x^{15} + x + 1$	14	50.0	15	0
16	$x^{16} + x^5 + x^3 + x^2 + 1$	15	51.5	16	0
17(a)	$x^{17} + x^3 + 1$	16	50.5	17	0
17(b)	$x^{17} + x^5 + 1$	16	54.4	17	0
17(c)	$x^{17} + x^6 + 1$	16	56.6	17	0
18	$x^{18} + x^5 + x^2 + x + 1$	17	33.9	18	0
19	$x^{19} + x^5 + x^2 + x + 1$	18	55.0	19	0
20	$x^{20} + x^3 + 1$	19	55.0	20	0
21	$x^{21} + x^2 + 1$	20	52.4	21	0
22	$x^{22} + x + 1$	21	49.0	22	0
23	$x^{23} + x^5 + 1$	22	51.8	23	0
24	$x^{24} + x^4 + x^3 + +1$	23	46.6	24	0
25	$x^{25} + x^3 + 1$	24	52.8	25	0
26	$x^{26} + x^6 + x^2 + x + 1$	25	52.2	26	0
27	$x^{27} + x^5 + x^2 + x^1 + 1$	26	50.3	27	0
28	$x^{28} + x^3 + 1$	27	50.8	28	0
29	$x^{29} + x^2 + 1$	28	50.1	29	0
30	$x^{30} + x^6 + x^4 + x + 1$	29	48.1	30	0
31	$x^{31} + x^6 + 1$	30	49.3	31	0
36	$x^{36} + x^{11} + 1$	35	49.3	36	0
39	$x^{39} + x^4 + 1$	38	50.8	39	0
41	$x^{41} + x^3 + 1$	40	54.1	40	0

the classificatory prediction accuracy. It has been found experimentally that for a LFSR of degree d, the minimum block size required is d. Hernandez et al. claimed in [15], the larger the block size, the better the prediction. They presented a value of block length equal to $10 * \log(n)$ to distinguish an unpredictable source from a predictable one. We agree to the first claim, but they did not justify this numerical value. Experimentally we found a lower bound for the block size for maximum prediction. The results are summarized in Table 1.

It can be seen from Table 1 (column 3)that if the block size for a primitive polynomial is less than its degree then the Classificatory Prediction Error (CPE) is nearly equivalent to chance probability. Hence from this point onwards it is established that *for a primitive polynomial of degree d, the minimum block size will be d for accurate classificatory prediction*. If we increase the block size further, there will be no significant change in the formation of decision trees and rules as generated by C4.5 algorithm.

5.2. Sequence length requirement

The next important point to consider is how many bits are required to learn from the pseudo-random output of LFSR to get correct prediction. To check these results we performed a comprehensive analysis for every primitive polynomial from degree 10 to 30 and then randomly up to 41 degree. We fix the block size as equal to d. We generated 100 different pseudorandom sequences corresponding to every primitive polynomial by varying its initial settings. We take only the upper bounds results into account.

Table 2
Minimum bits required and Bit Prediction Ratio (BPR)

Degree of primitive polynomial(*d*)	Correct Classificatory Prediction requirement		
	Training patterns, *x*	Minimum bits needed	*BPR*
10	70	80	8.00
11	61	72	6.54
12	571	583	47.58
13	899	912	69.15
14	1473	1487	106.21
15	81	96	6.40
16	2558	2574	160.87
17(a)	159	176	10.35
17(b)	319	336	19.76
17(c)	319	336	19.76
18	2010	2028	112.66
19	2648	2667	140.36
20	29	49	2.45
21	374	395	18.80
22	2749	2771	125.95
23	874	897	39.00
24	20989	21013	875.54
25	7996	8021	320.84
26	48987	49013	1885.11
27	90975	91002	3370.44
28	2499	2527	90.25
29	2749	2778	95.79
30	64980	65010	2167
31	2749	2780	89.67
36	11995	12031	334.19
39	21741	21780	558.46
41	3998	4039	98.51

Hernandez et al. [15] claimed that experimentally they found that training patterns needed to predict accurately is close to 1% of period of generator. Their analysis is based on a particular LFSR of degree 15; hence limitations exist in their claim.

We define an index *Bit Prediction Ratio* (*BPR*), to estimate the ratio between minimum bits required $(x + b - 1)$, where x is the number of training patterns, for correct classificatory prediction to the degree of the corresponding LFSR. Mathematically,

$$BPR = \frac{x + b - 1}{d}$$

Experimentally, we found that *BPR* for each of the primitive polynomial varies from 2.45 to 3370.44. This variation in the values of *BPR* is due to the number of taps in LFSR's. It has been observed that, *the higher the number of taps the more the number of bits required for correct classificatory prediction*. We can also infer that *x+b-1 bits are required to predict the next n-x-b+1* bits *correctly*. Table 2 summarizes this result.

5.3. Determination of primitive polynomial

C4.5 makes classification decision on the basis of the decision trees and classification rules generated from the learning data. We observe that the primitive polynomial can be determined from the pruned trees but not the way Hernandez et al. [15] showed in his work. We observed experimentally that all the

classification rules lead to primitive polynomial construction in a different fashion. Suppose we consider the primitive polynomial of degree 10, as presented in Table 3 (column 2), the simplified classification rules generated after pruning are

If bit at position 1 is 0 and bit at position 8 is 1 then class label is 1
If bit at position 1 is 1 and bit at position 8 is 0 then class label is 1
If bit at position 1 is 1 and bit at position 8 is 1 then class label is 0
If bit at position 1 is 0 and bit at position 8 is 0 then class label is 0

C4.5rules program identifies bit 1 and 8 as significant attributes or bits. Here, we can observe that *class label* $=$ *bit1* **XOR** *bit8*

Now consider bit 1 and bit 8 as the significant bits as pointed out by the classification rules. This rule is generated for a 10-stage LFSR; therefore subtract these significant bits from 10. Alternatively we can also say that subtract the significant bits from the chosen block length d, which is 10 in this case. Add 1 to them to get the desired primitive polynomial. The steps can be summarized as

$(1, 8)$ – Significant bits from classification rules
$(9, 2)$ – Subtract from 10
$(10, 3)$ – Add 1

Hence, $x^{10} + x^3 + 1$ is the required polynomial.

For any n-stage LFSR the above rule can be generalized to obtain the required primitive polynomial as:

$(a_1, a_2, a_3, a_4, \ldots)$ – Significant bits from classification rules
$(n - a_1, n - a_2, n - a_3, n - a_4, \ldots)$ – Subtract from chosen block length of LFSR
$(n - a_1 + 1, n - a_2 + 1, n - a_3 + 1, n - a_4 + 1, \ldots)$ – Add 1

Hence, $x^{n-a_1+1} + x^{n-a_2+1} + x^{n-a_3+1} + x^{n-a_4+1} + \ldots + 1$, is the required polynomial used in LFSR.

Therefore we conclude that *if significant attributes/bits are known then the exact primitive polynomial can be constructed*. The rule sets that we discovered for different polynomial are depicted in the form of significant attributes in Table 3.

If we summarize the experimental results of Tables 1, 2 and 3 we can estimate the minimum number of bits and the block size for correct classificatory prediction. Finally, the corresponding primitive polynomial can also be constructed. And also from the graph (Fig. 3) we can draw some basic conclusions

– The graph shows the plot between the degree of primitive polynomials and their respective *BPR* values (minimum bits required for correct classification). We tried to show two types of plots. First plot (dark lines) is the *BPR* values on primitive polynomials with 2 tap points. The other plot (dotted lines) shows the *BPR* values for those primitive polynomials that have more than 2 tap points.
– As the degree of primitive polynomial increases there is increase in the number of "minimum bits required" and hence increase in BPR for correct classification. But this increase is significant in the case, where number of taps is more than 2. So a larger primitive polynomial with more number of taps needs more bits to be totally broken upon (inference from Fig. 3).
– For all experimentation with primitive polynomial of degree greater than 16, we have generated the first 1, 00,000 bits of the respective generators, as it is difficult to store the whole period of larger primitive polynomials. Also, the time it will consume to run C4.5 on different size of patterns to approximate minimum bits requirement would be very high.

Table 3
Classification Rules generated for Primitive Polynomials of various degrees

Deg. of primitive polynomial (*d*)	Rules generated	Deg. of primitive polynomial (*d*)	Rules generated
10	1, 8	22	1,22
11	1, 10	23	1,19
12	1, 7, 9, 12	24	1,21,22,24
13	1, 10, 11, 13	25	1,23
14	1, 10, 12, 14	26	1,21,25,26
15	1, 15	27	1,23,26,27
16	1, 12, 14, 15	28	1,26
17(a)	1, 15	29	1,28
17(b)	1, 13	30	1,25,27,30
17(c)	1, 12	31	1,26
18	1,14,17,18	36	1,26
19	1,15,18,19	39	1,36
20	1,18	41	1,39
21	1,20		

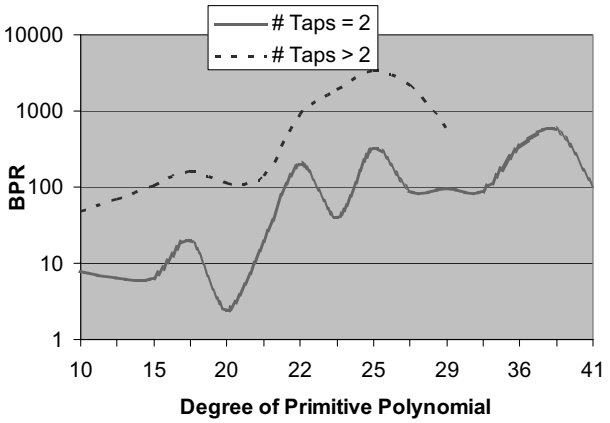

Fig. 3. Plot showing higher number of taps increases the minimum bit requirement for correct classification.

– In Table 2, the values of "minimum bits required" and hence "BPR" should not be treated as discrete. The values we have presented here are the average of 100 different experiments on each primitive polynomial with random initial settings. An experiment with different random setting conducted different number of times may yield slightly different values. But we observed that the overall band of *BPR* values will not be affected (i.e. insignificant standard deviation), since the output of a primitive polynomial, even with random initial contents is cyclic in nature and therefore exhibit similar statistical properties.

5.4. Classificatory prediction of Geffe Generator

Geffe Generator uses three LFSR's combined in nonlinear manner. Two of the LFSR's are inputs into a multiplexer, and the third LFSR controls the output of the multiplexer. If g_1, g_2 and g_3 are the outputs of the three LFSR's, the output of the Geffe Generator can be described by

$$s = (g_1 \wedge g_2) \oplus ((\neg g_1) \wedge g_3)$$

If the LFSRs have length n_1, n_2 and n_3 respectively, then the linear complexity of the Geffe Generator is $(n_1 + 1)\, n_2 + n_1 n_3$. The period of the generator is the least common multiple (*LCM*) of the periods of the three generators. This generator falls prey to correlation attack [6].

As explained in Section 5, a similar exercise was conducted to check the classificatory predictive behavior of pseudo-random bits Generator by Geffe generator. We took various combinations of different LFSR's to carry out the experiment. We generated 100 different pseudo-random sequences from Geffe Generator using different sets of LFSR's and also with different initial settings to have more confidence on the results thus obtained.

We generate various sequences of length n, equal to *LCM* of period of three LFSR's. A suitable block size b is chosen and subsequently P_{n-b} patterns are generated to create the pattern space. To check the minimum block size for correct classificatory prediction, we took 99% of the patterns for training and remaining 1% for testing. It has been found experimentally that the block size, b, should be greater than the product of the length of each of the LFSR's for good classificatory prediction results. The classification rules generated by *C4.5rules* for various block sizes vary in size and interpretation; hence no general comment can be made about them. The rules generated are typically large and do not depend on few bits as in the case of LFSR. Hence to physically interpret each of them they should be tabulated extensively before taking any decision about the classificatory prediction. Presently, we are experimenting to fix the lower bound on the number of training patterns required for correct classificatory prediction of Geffe Generator.

6. Next bit prediction

Next bit prediction is one of the important activities of the cryptanalyst and information theorist. In this direction the first celebrated paper is by Shannon "Prediction and entropy of printed English" [2]. This model is purely based on entropy and redundancy of the language. Inspired by Shannon work several authors have given models for either predicting next bit or character for a particular generator or general predictor models. Methods for inferring Linear Congruential Generator and its variants have been studied in detail and can be seen in [4,9,11]. The current work of prediction by BlackBurn et al. [28] and Gathen and Shparlinski [12] are also specific to particular generator like non-linear generator and subset sum generator. General Next Bit Prediction models have also been quoted in literature by Cover [30], Ziv [13,14] and Jacquet et al. [25]. All of these general models are probability based and requires very large data set to fix the bound of learning. All of the above prediction models have been put on mathematical foundation but have very little practical application. In contrast the present model which we are proposing is based on inductive machine learning paradigm for which the theory is already established. We have to only customize the existing classification algorithms as a prediction algorithm.

We presented the theoretical prediction model in section 1. In that model at any time, previous i-1 bits are needed to predict the i^{th} bit. We used this concept and presented slightly modified way to predict bits as a classification problem (Section 4). Once appropriate numbers of patterns are given to the C4.5 inductive algorithm, and it has generated decision trees and rules out of that, it can be used for *next bit prediction*. In classificatory prediction we have to have full pseudo-random sequence in hand, so that we can train few patterns and check the prediction accuracy of the remaining bits. In practical scenario, the analyst may not have full pseudo-random sequence with him but he would like to know the subsequent bits of the PRBG with available bits in hand. Keeping this motivation in mind we used the C4.5 classification algorithm for next bit prediction. The algorithm is presented as under.

Algorithm: *Nextbit_Prediction*(p_i, *n*)

Input: $\{p_i\}$ – the pseudo-random number sequence generated by the generator
n – the length of the pseudo-random sequence in hand
K – *The* number of bits to be predicted
Output: $\{B_i\}$ - predicted bits
begin

1. Select a suitable block size (b)
2. Generate $n - b$ blocks and associate every pattern with their class label, i.e. next bit (described in Section 2.2)

$$P_1 = p_1, p_2, p_3, \ldots, p_b \quad CL \rightarrow p_{b+1}$$
$$P_2 = p_2, p_3, p_4, \ldots, p_{b+1} \quad CL \rightarrow p_{b+2}$$
$$\vdots$$
$$P_{n-b} = p_{n-b}, p_{n-b+1}, , \ldots, p_{n-1} \quad CL \rightarrow p_n$$

3. Start the C4.5 algorithm to learn from the pattern set and tabulate the results
4. Set counter, i=1;
5. Repeat step 6 till K bits are predicted i.e. while (i $\leqslant K$)
6. To predict $n + i^{th}$ bit, B_i
   (a) Choose $B_i = p_{n+i-b}, p_{n+i-b+1}, \ldots, p_{n+i-1}$ as a test pattern
   (b) Run the C4.5 program to predict the class.
   (c) Store B_i
   (d) Increment counter, $i++$

end

Repeat this exercise for different block sizes and observe the K – bits in a known environment for validation of the true next bit predictor model.

The output of this algorithm is B_i number of predicted bits of the PRBG. We used the above algorithm to predict subsequent bits of different LFSR and Geffe Generator. As shown in Table 2, we fix the block size and the minimum bits needed for prediction as the training set parameters. By fixing these parameters, we could predict the remaining bits of each of the LFSR's with 100% prediction accuracy. In the case of Geffe Generator we predict few subsequent bits with better than chance probability.

7. Conclusions

In this paper we have improved upon the alternative approach of prediction of next bit of a pseudo-random generator using machine learning technique as proposed by Hernandez et al. [15]. We also proposed a model for *next bit prediction* for PRBG. The utility of the presented approach of predicting next bit for LFSR's and Geffe Generators by *Classificatory Prediction* and true *Next Bit Prediction* (with few bits in hand) has its impact on finding the flaw with other crypto primitives. There exist techniques to predict and cryptanalyze the above considered PRBG's, but they are domain specific. The advantage of our proposed approach is that it is domain independent and does not rely on the type of generator and the parameters used by PRBG. In a more practical application scenario, most of the time a cryptanalyst has little knowledge about the PRBG and he is inquisitive about the future bit sequences. In this case, he

can take a lead from the higher prediction accuracy of the proposed model for true next bit prediction. By predicting better than chance probability the analyst would be left with reduced set of unknown bits that can be subjected to a brute force attack in real time.

Acknowledgements

The authors are grateful to Dr. P K Saxena, Director SAG for keen interest and support to carry out this work. We are also thankful to Dr. S.S. Bedi, Scientist 'G' whose comments and suggestion gave us enough insight for analyzing the generators. The authors wish to thank the referees for their helpful and constructive comments that led to a much better presentation of the classificatory prediction problem.

References

[1] B. Schneier, *Applied Cryptography*, (second edition), Protocols, Algorithms and Source Code in C, John Wiley & sons, Inc, 1996.
[2] C.E. Shannon, Prediction and entropy of printed english, *Bell System Technical Journal* **30** (1951), 50–64.
[3] D.R. Stinson, *Cryptography*, Theory and Practice, CRC Press, 1995, 367–368.
[4] D.E. Knuth, Deciphering a linear congruential encryption, *IEEE Trans Inf Theory* **31** (1985), 49–52.
[5] D.E. Knuth, *Semi numerical Algorithms*, (third edition), Volume 2 of The Art of Computer Programming, Addison-Wesley, Reading, MA, USA, 1997.
[6] E.L. Key, An analysis of the structure and complexity of nonlinear binary sequence generators, *IEEE Transactions on Information Theory*, **IT-22**(6) (1976), 732–736.
[7] E.R. Berlekamp, Nonbinary BCH decoding, presented at the1967 International Sypmp. on Information Theory, San Remo, Italy. Algebraic Coding Theory. New York: McGraw-Hill, chs. 7 and 10, 1968.
[8] F.J. Breiman, R. Olshen and C. Stone, Classification and Regression Trees, 1984.
[9] H. Krawczyk, How to predict congruential generators, *J.Algorithms* **13** (1992), 527–545.
[10] http://www2.cs.uregina.ca/˜hamilton/courses/831/notes/ml/dtrees/c4.5/c4.5r8.tar.gz.
[11] J.B. Plumstead, *Inferring a Sequence Generated by a Linear Congruence*, 23rd Annual Sympos. On foundations of computer science, 1982, 153–159.
[12] J.V.Z. Gathen and I.E. Shparlinski, *Predicting Subset Sum Pseudorandom Generators*, SAC, LNCS 3375, 2004, 241–251.
[13] J. Ziv, A universal prediction lemma and applications to universal data compression and prediction, *IEEE Trans Inform Theory* **47** (2001), 528–532.
[14] J. Ziv, An efficient universal prediction algorithm for unknown sources with limited training data, *IEEE Trans Inform Theory* **48**(6) (2002), 1690–1693.
[15] J.C. Hernandez, J.M. Sierra, C. Mex-Perera, D. Borrajo, A. Ribagorda and P. Isasi, *Using the General Next Bit Predictor Like an Evaluation Criteria*, Proceedings of NESSIE workshop, Leuven, Belgium, 2000.
[16] J.L. Massey, Shift Register synthesis and BCH Decoding, *IEEE Transactions on Information Theory* **IT-15**(1) (1969), 122–127.
[17] J.R. Quinlan and R. Rivest, Inferring decision trees using the minimum description length principle, *Information and Computation* **80** (1989), 227–248.
[18] J.R. Quinlan, *C4.5: Programs for Machine Learning*, Morgan Kaufmann, San Francisco, CA, 1993.
[19] J.R. Quinlan, *Induction of Decision Trees*, Machine Learning Journal 1986.
[20] J.R. Quinlan, Improved use of continuous attributes, *Journal of Artificial Intelligence Research* **4** (1996), 77–90.
[21] J. Carbonell, ed., *Machine learning, Paradigms and Methods*, PP 1-9, The MIT Press, 1992.
[22] K.C. Zeng, C.Y. Yang, D.Y. Wei and T.R.M. Rao, Pseudorandom Bit Generator in stream – cipher cryptology, *IEEE Computer*, 8–16, Feb. 1991.
[23] M. Blum and S. Micali, How to generate cryptographically strong sequences of pseudo-random bits, *SIAM J Computing* **13**(4) (1984), 850–863.
[24] P. Grunwald, I.J. Myung and M.A. Pitt, eds, *Advances in Minimum Description Length: Theory and Applications*, MIT Press, 2005.
[25] P. Jacquet, W. Szpankowski and I. Apostal, A universal predictor based on pattern matching, *IEEE Trans. Inform. Theory* **48** (2002), 1462–1472.
[26] R. Agrawal, A. Arning, T. Bollinger, M. Mehta, J. Shafer and R. Srikant, *The Quest Data Mining System*, Proc. Of 2nd International Conference on Knowledge Discovery in Databases and Data mining, Portland, Oregon, August, 1996.

[27] S.S. Khan, Classificatory Prediction and Primitive Polynomial Construction of Linear Feedback Shift Registers using Decision Tree Approach, KBCS-2004, Fifth International Conference On Knowledge Based Computer Systems, 2004.
[28] S.R. Blackburn, D. Gomez-Perez, J. Gutierrez and I.E. Shaparlinski, Predicting nonlinear pseudorandom number generators, Math. Comp. (To appear).
[29] S.W. Golomb, *Shift Register Sequences*, San Francisco, Holden-Day, 1967, Reprinted by Aegean Park Press, 1982.
[30] T.M. Cover, *Behavior of Sequential Predictors of Binary Sequence*, in Proc. 4^{th} Prague Conf. Information theory, Statistical decision functions, Random processes, 1965, 263–272.