A Novel Authentication Procedure for Secured Web Login using Coloured Petri Net

D. Lalitha [1*], S. Vaithyasubramanian [2*], K. Vengatakrishnan [3], A. Christy [4], M. I. Mary Metilda [5]

[1, 2, 3, 5]*Department of Mathematics,* [4] *Department of CSE*
Sathyabama Institute of Science and Technology, Chennai, India.

* Corresponding Authors: [1] lalkrish2007@gmail.com, [2] discretevs@gmail.com
[3] vengat0809@gmail.com, [4] ac.christy@gmail.com, [5] metilda81@gmail.com

***Abstract* - To secure the information in the World Wide Web, websites ask users to create their own login identification and password, where in general a password is a simple string of characters. To make the information more secure, an access code of an array of alphanumeric and special characters can be generated by using a Petri net model. A string language can be generated by labelling transitions of a Petri net. Similarly two dimensional array languages can also be generated by Petri nets. Coloured Petri net has also been defined to generate array language. This kind of access would be a three factor authentication. In this paper we propose and develop the application of array generating Petri net to enhance information security.**

Keywords - Access Code, Array Languages, Coloured Tokens, Inhibitor Arcs, Information Security, Petri net, Three Factor Authentication

I. INTRODUCTION

Petri net [3] is one of many Mathematical models available to model distributed systems. Some of the components of such systems may exhibit concurrency and parallelism. Tokens in Petri net generally represent the resources required for activities to take place. All the tokens in a basic Petri net are black dots. In a complex system the resources may have different attributes. It is not possible to represent the various attributes of the resources by just black tokens. Hence the basic Petri nets are not suitable to model such systems. Several extensions of the basic Petri net are available in literature. In (CPN) Coloured Petri Nets[1, 2,4] tokens carry different values.

String languages and two dimensional array languages have been discussed in detail in Formal Languages. Array language can be generated by a Petri net [5-9]. Tokens are rectangular or hexagonal arrays over a given alphabet. Catenation rules are defined as label of the transitions. When transitions fire the array grows in size and will reach the output place. When enabled transitions are fired, arrays move around. The set of arrays that reach the final places is defined as the array language generated by the net. Coloured tokens have also been used to generate arrays [5]. CPN is used to facilitate more control over firing and also to have more variety on the data to be used.

The Petri net model introduced in this paper has tokens with three attributes. The first attribute gives the identification of the token. It is used to differentiate the various tokens, which may reside in the same place. The second attribute will signify the position of the token or the place in which it resides. The third attribute gives the value it takes at a given point of time. The transition gets enabled or disabled, depending on the conditions placed. The attribute of the tokens released will also depend on the conditions attached with the transition and also on the attribute of the tokens consumed.

Password is in general a string of characters used for authenticity to gain access to any resource. To gain access the password entered by the user has to match with the original password created. Otherwise the resource becomes inaccessible. Passwords are generally short so that it can be easily memorized. The easier the password is to remember, the easier it is for the hackers to crack. Several papers have been published for tackling this issue [10-12]. The user has to balance between the necessity of a highly secure password and a password which can be easily recalled. Creating a string password could be like handling a two edged sword. The number of combinations to create a string password consisting of three lower case letters and two digitsis36^5. For a cracker using a standard personal computer, the maximum time required to guess the password of length 5 is 1 second. The length of the password is also a factor involved in authenticating a string password. If six characters are to be used in a string password the number of combinations is 36^6. As an alternative to alphanumeric password, array passwords were defined [9, 10]. If six characters are to be used in an array password, it can be created in 4 different ways. The array size could be $1\times 6 \; or \; 6\times 1 \; or \; 2\times 3 \; or \; 3\times 2$. Hence there are 4×36^6 number of different combinations with 6 characters. Hence obviously it would take a longer time to guess an array password which is made up of the same number of characters as the string password.

This paper is organized as follows. The second section defines the basic Petri net model which generates rectangular arrays over an alphabet. Examples are given to show how the introduction of inhibitor arcs in the net increases the generative power of the model. The third section defines and also gives examples of the Coloured Petri net model which generates rectangular arrays. In the fourth section, Petri net has been used to create an access code. This access code is an array made up of any of the 95 printable characters. Flow chart is drawn to show how these Petri nets are called for the creation of the access code. Whenever the user logs in to use the resource, the access code has to be authenticated. Hence another flow chart is drawn to give the steps for authenticating the user. Three factors are involved in this authentication process. Number of rows and number of columns are two factors. The user also has to decide, when creating the access code, the order in which the characters are keyed in. It could be row-wise or column-wise. To access the resource he has to use the same order for entering the characters, otherwise the access will be denied. Before even entering the exact characters of the access code, three different factors are verified. The last section is the conclusion, which connects all the sections of the chapter.

II. BASIC NOTATIONS

All the preliminary notations required are explained in this section. Differences between an ordinary Petri net and an array generating Petri net are explained in detail. The Petri net model called Array Generating Petri net (AGPn) is defined and explained with an example.

Notations used in Array Languages: Two arrays can be joined column-wise if they have the same number of rows and can be joined row-wise if they have the same number of columns. If A and B are two arrays having same number of columns then $A \otimes B$ denotes the array which is got by joining the two arrays row-wise. If A and B are two arrays having same number of rows then $A \oplus B$ denotes the array which is got by joining the two arrays column-wise. In formal language theory, a_m denotes a column of m a's and a^n denotes a row of n a's. Generally for an m x n array A, $A \otimes b^n$ will denote joining a row of b's to the array A after its last row and $b^n \otimes A$ will denote joining a row of b's to the array A, before its first row. Similarly $A \oplus b_m$ will denote joining a column of b's to the array A, after its last column and $b_m \oplus A$ will denote joining a column of b's to the array A, before its first column. If A is an $m \times n$ array and B is a $k \times n$ array, then $A \otimes b^n \otimes B$ denotes the joining of A, followed by a row of b's and then followed by the array B. The resulting array would be of size $(m+k+1) \times n$. If A is an $m \times n$ array and B is an $m \times k$ array, then $A \oplus b_m \oplus B$ denotes the joining of A, followed by a column of b's and then followed by the array B. The size of the resulting array would be $m \times (n+k+1)$.

Differences between Ordinary Petri Net and Array Generating Petri Net:

In a basic Petri net tokens are just black dots. In a basic Petri net firing a transition moves the tokens from input places to output places. But in array generating Petri net model tokens have attributes. Array over a given alphabet is used as one of the attributes of the token used. A function is defined for every transition of the net. This function defines the changes that take place in the attributes of the token.

CPnAL: Coloured Petri net Array Language

Coloured Petri net is used whenever a variety of tokens is required for the model. In this section an Array Generating Coloured Petri net (AGCPn) is defined. In this model each token has three attributes, called the token attributes (TA). The language generated by AGCPn is denoted by CPnAL. The model is explained with an example. The declaration of the colour set is done in the net. In general integer values are denoted by 'int'. Two dimensional arrays are denoted by 'ARRAY2' and strings are denoted by 'str'.

Definition of AGCPn

An AGCPn is defined as a nine component tuple (Σ, P, V, T, I, O, TA, φ, F) where Σ a set of characters, P is the set of places of the net, V is the Colour Set, T is the set of transitions such that P∩T is empty, I is the input function from T to bags of places, O is the output function from T to set of places, TA is the initial Token Attributes which is a tuple of the form $\langle id, p, v \rangle$, where *id* is the identification of the token, *p* the position of the token and *v* is the value of the token. φ is a Transition function, defined as follows: for all t ϵ T, φ_t is a partial function from a set of attributes to a set of attributes. F a subset of P, is called the final set of places.

Definition of CPnAL

The language CPnAL, generated by AGCPn is the set of all arrays that come into the final set of places. The enabled transition t fires and the tokens are moved from the input place to its output place. The attributes associated with the token, changes according to the function φ_t. The resulting token will also depend on the conditions satisfied by the attributes of the token. The example 1 defines a net which generates all square arrays made up of a's whose side is of the form 2^k.

Example 1

AGCPn$_2$ = (Σ, P, T, I, O, V,TA, φ, F) where Σ={a, b}, P={P$_1$, P$_2$, P$_3$, P$_4$}, T ={t$_1$, t$_2$}, Input output functions can be seen in the net given in Figure 1. V- The tokens used are arrays and integers. The declarations are given in Figure 1. A is a two dimensional array whose initial value is assigned. I and J are integers whose initial value is also fixed.

The Initial Token Attribute is the set: $\{\langle A, P_1, A\rangle, \langle I, P_3, 1\rangle, \langle J, P_4, 1\rangle\}$

Where the function φ is defined in the net and F = {P$_1$}.

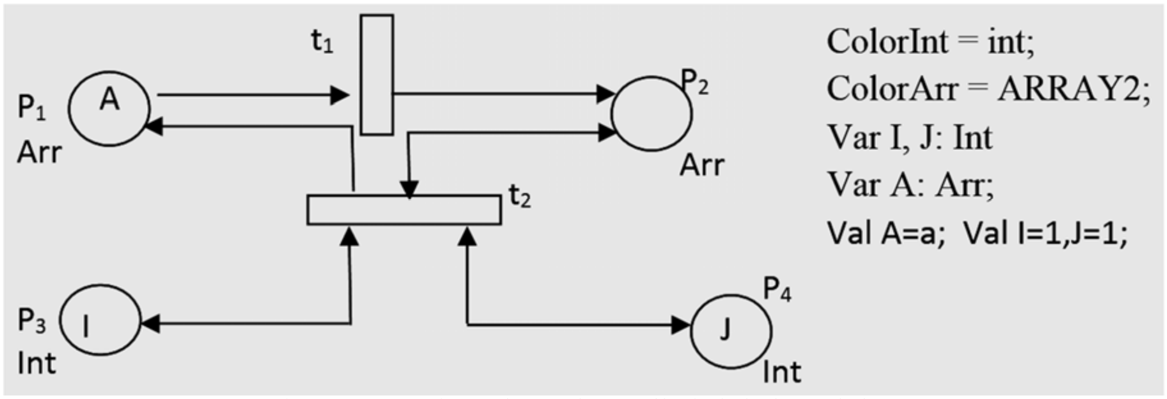


Figure1 Net generating a column and a row of b’s in the background of a’s

$$\varphi_{t_1}[\langle A, p_1, A\rangle] = [\langle A, p_2, A \oplus b_m\rangle]. if \quad I > 0, then$$

$$\varphi_{t_2}[\langle A, p_2, A\rangle, \langle I, p_3, I\rangle, \langle J, p_4, J\rangle] = [\langle A, p_2, A \oplus (a \quad a)_m\rangle, \langle I, p_3, I-1\rangle, \langle J, p_4, J\rangle].$$

$$if \quad I = 0, then$$

$$\varphi_{t_2}[\langle A, p_2, A\rangle, \langle I, p_3, I\rangle, \langle J, p_4, J\rangle] = [\langle A_{J+1}, p_1, A \otimes a^n\rangle, \langle I, p_3, J+1\rangle, \langle J, p_4, J+1\rangle]$$

The function φ_{t_1} *and* φ_{t_2} describe the actions of the transtions. Initially t$_1$ is enabled to fire. Whenever t$_1$ fires a column of b’s is joined to the array in P$_1$ and is put in P$_2$. This happens whenever an array reachesP$_1$. Now t$_2$ is enabled to fire. Since t$_2$ has three input places, the result of fiirng t$_2$ will depend on the I, J values. If I > 0, then two columns of a’s is joined to A and it is put in P$_2$, in P$_3$, the value of I is replaced by I - 1 and in P$_4$,the value J is unaltered. If I = 0, then a row of a’s is joined and is put in P$_1$, the values of both I and J are replaced by J+ 1. The arrays that reach P$_1$ are included in the language. The $k+1^{th}$ array is got from k^{th} array by adding a column of b’s, followed by an array of size $k \times 2k$ of a’s and finally a row of a’s is added. The size of the $k + 1^{th}$ array is $(k+1)\times(k^2+2k+1) = (k+1)\times(k+1)^2$. The language CPn$_2$AL consists of arrays of size $n \times n^2$. A derivation of the second array from the first array is given below:

$$\underset{P_1}{a} \xrightarrow{t_1} \underset{P_2}{a \quad b} \xrightarrow{t_2} \underset{P_2}{a \quad b \quad a \quad a} \xrightarrow{t_2} \underset{P_1}{\begin{matrix} a & b & a & a \\ a & a & a & a \end{matrix}}$$

III. CREATING AND AUTHENTICATING ACCESS CODE

AGCPn defined in the previous section, is used to create an access code for network security. When it comes to information security string password made up of alphanumeric and special characters are used for login authentication. Constantly efforts are made to create secure passwords so that it is not easy to guess. In spite of all the security provided, information gets hacked as these string passwords are easy to crack and there are tools for doing it. In this section a code which is in the form of a matrix is generated by AGCPn. In the process of generating the array password the user has to decide three different factors. As the first factor user needs to choose the array size, the row-size (m) and column-size (n). As the second factor user has to decide on how the characters are going to be entered into the array. The user can enter the characters row-wise(R) or column-wise(C). Deciding on every character of the array is the third factor involved. Hence when creating the access code, the user chooses the value of all the parameters (m, n, R/C). Every time the user is logging into the website, all these parameters will have to match, otherwise the access will be denied.

Two different Petri net models are used for generation of the access code. In the first model the array elements are entered one by one as a single character without using any of the "Tab" key or "Enter" key. In the second model a blank array is created of the required dimensions. The array elements are entered one by one as a single character, but in between every two character the "Tab" key is used to move the control to the next position in the array. Flow charts are drawn to give the procedure step by step in calling the Petri net. First time when the net is called, to generate the access code, the required parameters are given. Using the parameters and the characters entered, the access code is generated and sent back to flow chart. The access code for the user is saved for future reference. The second flow chart shows the steps involved in letting or refusing the user access into the website.

In the flow chart given in the Figure 2, the values for m and n are taken from the user to generate the array. The user has two options in entering the elements of the array, row-wise or column-wise. This order is also taken as the specified order (R/C) from the user. The flow chart gives the value of the parameters m, n and calls RPn (m, n) or CPn(m, n) depending on the choice of the order. The relevant Petri net generates the access code and sends it back to be stored. The user can use any of the 95 printable characters to generate his array password. The control characters cannot be used. If any other character is used, then the array does not get generated.

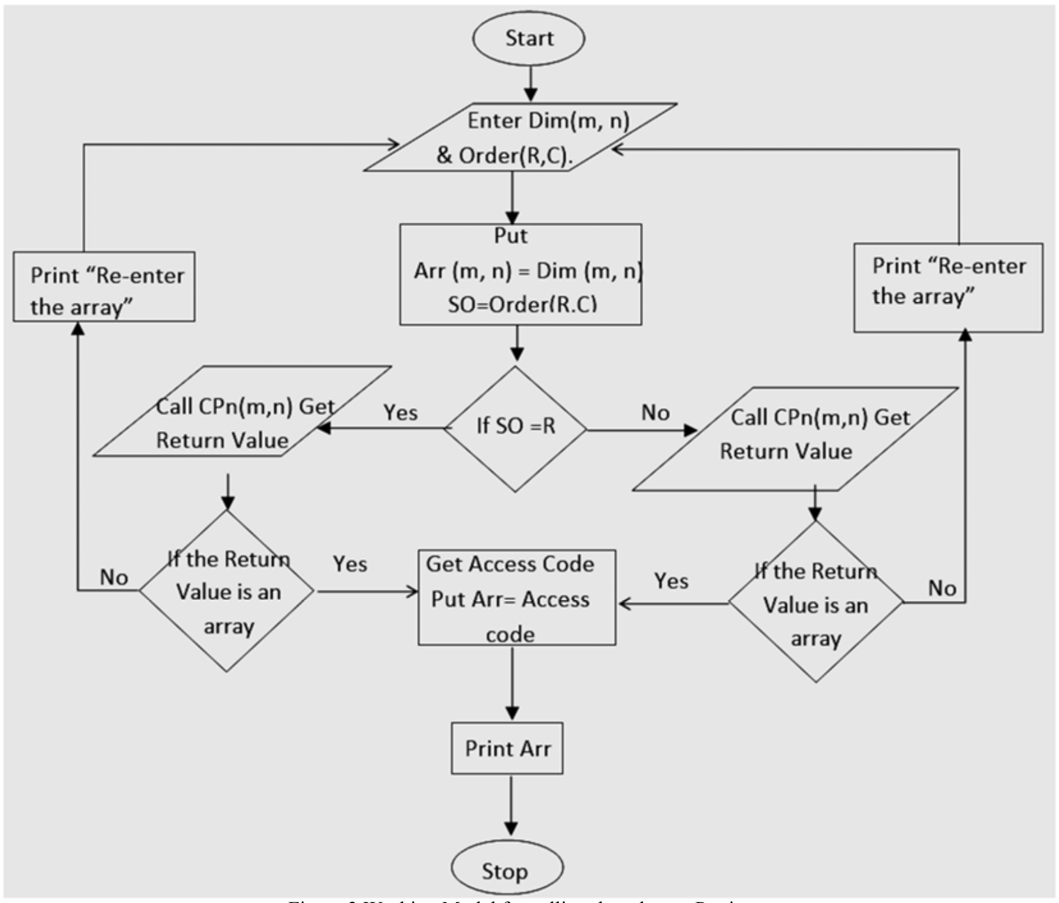

Start
Enter Dim(m, n) & Order(R,C).
Put Arr (m, n) = Dim (m, n) SO=Order(R.C)
Print "Re-enter the array"
Print "Re-enter the array"
If SO =R
Yes
No
Call CPn(m,n) Get Return Value
Call CPn(m,n) Get Return Value
If the Return Value is an array
If the Return Value is an array
No
Yes
Yes
No
Get Access Code Put Arr= Access code
Print Arr
Stop

Figure 2 Working Model for calling the relevant Petri net.

AGCPn to generate the access code: If R is chosen by the user, then the flow chart calls RPn (m, n, R) by giving the values of m and n as its parameters. The Petri net in example 2 is used to generate the array password.

Example 2

$RPn(m,n) = (\Sigma, P, V, T, I, O, TA, \varphi, F)$, Where Σ is the set of all 95 printable characters, P = {P_1, P_2, . . . ,P_8}, T={ t_1, t_2,t_3},Input and Output functions are shown in the Figure 3. The colour set involved in the net are two dimensional arrays, strings and integer. The arrays and strings are made of the elements of $\sum$. λ denotes the empty string. Initial Attributes of the tokens:

$$TA = \{\langle S,P_1,\lambda\rangle,\langle C,P_2,C\rangle,\langle i,P_3,1\rangle,\langle R,P_4,m\rangle,\langle C,P_5,n\rangle,\langle j,P_6,0\rangle\}$$

The partial function φ states how the attributes of each token changes, when any enabled transition is fired. F = {P$_8$}.The final place stores the access code which is an m x n array.

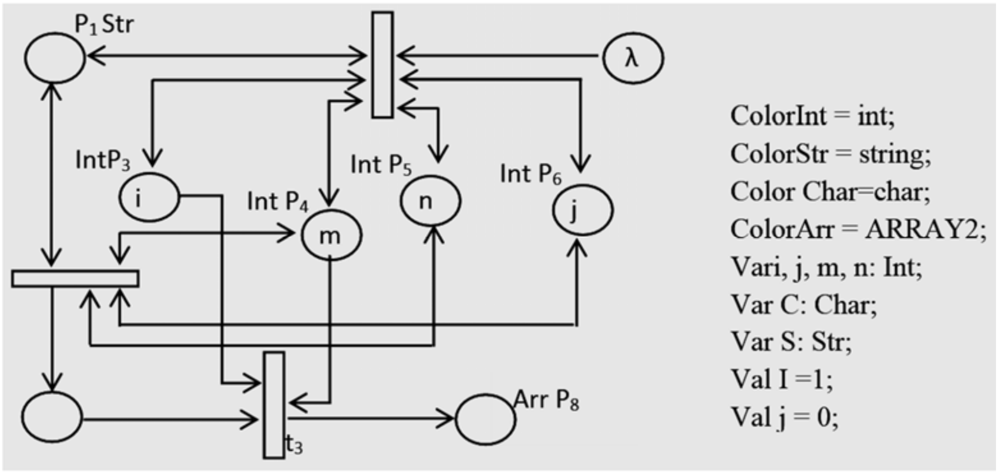


Figure 3 Net Generating Access Code.

$$\varphi_{t_1}[\langle S, P_1, S\rangle, \langle C, P_2, C\rangle, \langle in\,1, P_3, i\rangle, \langle r, P_4, m\rangle, \langle c, P_5, n\rangle\langle in\,2, P_6, j\rangle]$$
$$= [\langle S, P_1, S \oplus Ch\rangle, \langle in\,1, P_3, i\rangle, \langle r, P_4, m\rangle, \langle c, P_5, n\rangle\langle in\,2, P_5, j+1\rangle]$$
$$if \quad 33 \le Ascii\ (C) \le 126 \ \& \ i \le m \ \& \ j < n$$
$$\varphi_{t_2}[\langle S, P_1, Str\rangle, \langle in\,1, P_3, i\rangle, \langle r, P_4, m\rangle, \langle c, P_5, n\rangle\langle in\,2, P_6, j\rangle]$$
$$= [\langle S, P_1, \lambda\rangle, \langle in\,1, P_3, i+1\rangle, \langle r, P_4, m\rangle, \langle R[i], P_7, Str\rangle, \langle c, P_5, n\rangle\langle in\,2, P_6, 0\rangle]$$
$$if \quad i \le m \ \& \ j = n$$
$$\varphi_{t_3}[\langle R[1], P_7, Str\rangle, \langle R[2], P_7, Str\rangle, \ldots, \langle R[m], P_7, Str\rangle, \langle r, P_4, m\rangle, \langle in\,1, P_3, i\rangle]$$
$$= [\langle Access\_code, P_8, R[1] \otimes R[2] \otimes \ldots \otimes R[m]\rangle \ if \ i > m\ ]$$

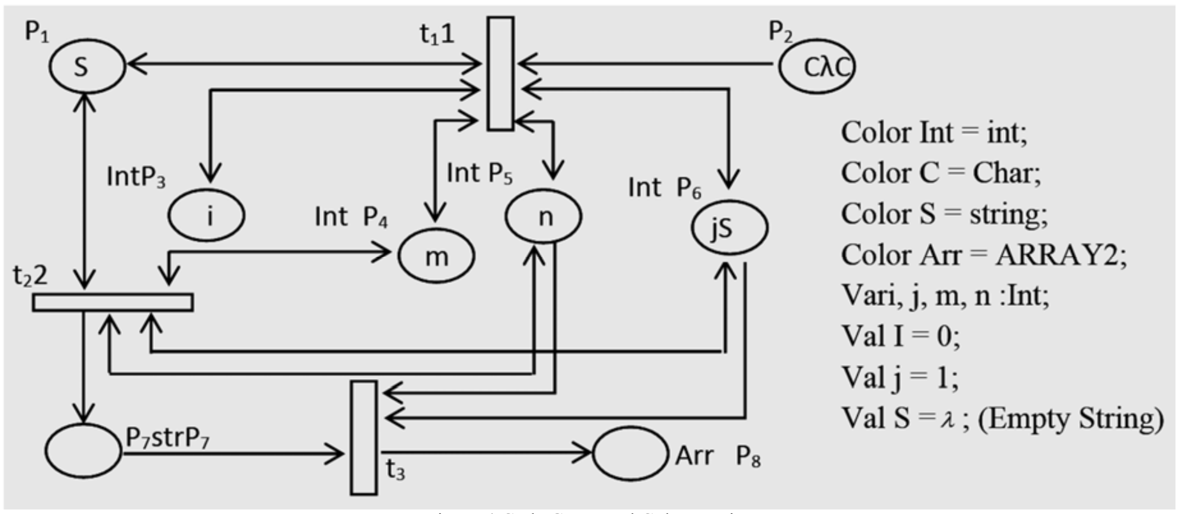
Initially there is an empty string in P$_1$, i = 1, j = 0. Assume the user is entering an access code made up of two rows and three columns. When the first character is entered, the condition for firing t$_1$ is satisfied and so the character is joined with λ column-wise and put in P$_1$. The value of i is retained but value of j is incremented by1.The same condition will hold till all the three characters of the first row are entered and it joins to form a string. At this stage j =3, so the condition for firing t$_1$ is not satisfied but the condition for firing t$_2$ is satisfied. When firing t$_2$ the string from P$_1$ is put into P$_7$and it is stored as R [1].The value of i is incremented by1and the value of j is initialized to 0. The empty string is put back in P$_1$. The whole process is repeated again and the second row gets generated using the next three characters put in P$_2$. Again j becomes three, with i = 2, and so t$_2$ fires moving the string in to P$_7$ and it is stored as R[2]. Now i = 3 which is greater than m and so t$_3$ fires. Both R[1] and R[2] are joined row-wise and is moved to the place P$_8$. The array which comes into P$_8$ is called Access code. If instead of the 95 printable characters any other control character is used, then the array does not reach the output place. If C is chosen by the user, then the flow chart calls CPn(m, n) by giving the values of m and n as its parameters. In this case the Petri net in example 3 is used to generate the array password.

Example 3

$CPn(m,n,R) = (\Sigma, P, V, T, I, O, TA, \varphi, F)$, where Σ is the set of all alphanumeric and special characters, P = {P$_1$, P$_2$, . . . , P$_8$}, T={t$_1$, t$_2$,t$_3$}, Input and Output functions are shown in the Figure 4. The colour set would be two dimensional arrays, strings and integers. λ denotes the empty string. Initial Attributes of the tokens:

$$TA = \{\langle S, P_1, \lambda\rangle, \langle C, P_2, C\rangle, \langle i, P_3, 0\rangle, \\ \langle R, P_4, m\rangle, \langle C, P_5, n\rangle, \langle j, P_6, 1\rangle\}$$

The partial function φ states how the attributes of each token changes, when any enabled transition is fired. F = {P$_8$}.The final place stores the access code which is an m x n array.



```
Color Int = int;
Color C = Char;
Color S = string;
Color Arr = ARRAY2;
Vari, j, m, n :Int;
Val I = 0;
Val j = 1;
Val S = λ ; (Empty String)
```

Figure 4 Code Generated Column-wise

In this example every character of the first column is filled. As each character is taken the value i increase till it reaches m and the value of j is not incremented. Once i reaches the value of m, i is initialized to 0 and j gets incremented. Finally all the columns reaching P_7 are joined column-wise.

If the ASCII values of the characters used are in the range 33 to 126, then the access code gets generated. Otherwise no array is sent from the net. In this case, the user has to start all over again to enter the various characters. When all the mn characters are entered properly, then the generated access code is sent back and stored as the login credential of the user. When the user tries to access the website once again, the user is asked to enter the size of the code along with the order in which the characters are entered in to the matrix. If any of them does not match then the access is denied. If these details, matches with what the user has saved already, then the array entered is checked with the access code generated. If the array matches then the access is authorized. The flow chart given in Figure 5 shows the steps involved in checking the credentials. If the access code is rightly entered, then access is given or else it is denied.

In examples 2 and 3, the entries of the array were taken one by one in the place P_2 of the net, without using a “Tab” key or an “Enter” key. Once the three parameters are cross checked, the array characters are entered just like a string password. The only difference between string password and array password is that there are three factors involved in authentication.

In example 4 the array characters are not taken as a single string. In between any two characters “Tab” key is used. A blank array of the required size is taken as the start array. Any of the 95 printable characters can be used to fill the array. Let us assume that user is filling the array row-wise. Then the first n printable characters are stored in the first row. In between any two of these n characters the “Tab” key is used. This will move the control horizontally. Once all the n positions are filled the control will move to the first element of the next row. This process is repeated till all the rows of the array are filled. The “End” key is used finally to indicate the end of array.

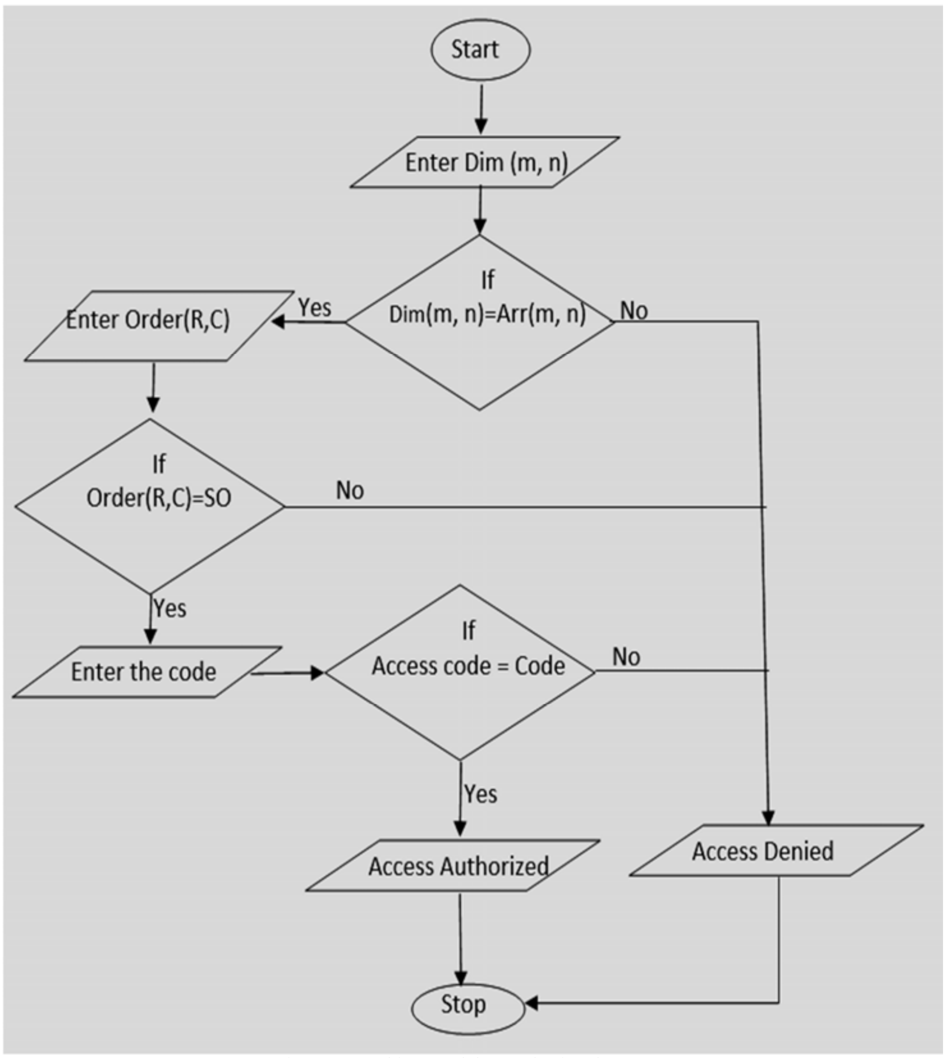


Figure 5 Working model to authorize the entree

In the next model only one net is used for generating the access code irrelevant of how the user is entering his characters. If the user is entering the characters row-wise, then the values of m and n are passed on as the row size and column size respectively. The access code generated by the net is stored as the array password. If the user is entering the characters column-wise, then the values of m and n are passed on as the column size and row size respectively. In this case the access code generated by the net would be of size n x m. The transpose of the received array is stored as the array password. If the full array is not entered by the user, then net sends an error message which will be a string. Hence if the return value is an error message, then the user has to re-create the array password. Flow chart given in Figure 6 shows the steps involved.

A blank matrix with m rows and n columns is denoted by B(mλ, nλ). Any entry which is blank is denoted by λ. All the mn entries of B(mλ, nλ) are blank:

$$B(2\lambda,3\lambda) = \begin{matrix} \lambda & \lambda & \lambda \\ \lambda & \lambda & \lambda \end{matrix}.$$

To start with a blank matrix of size m x n is in the place P_1. The array is built by the user. Initially i, j values are 1. The following keys are used: Any of the 95 printable characters. “TAB” as a separator between any two characters. “END” key to indicate the end of the array password.

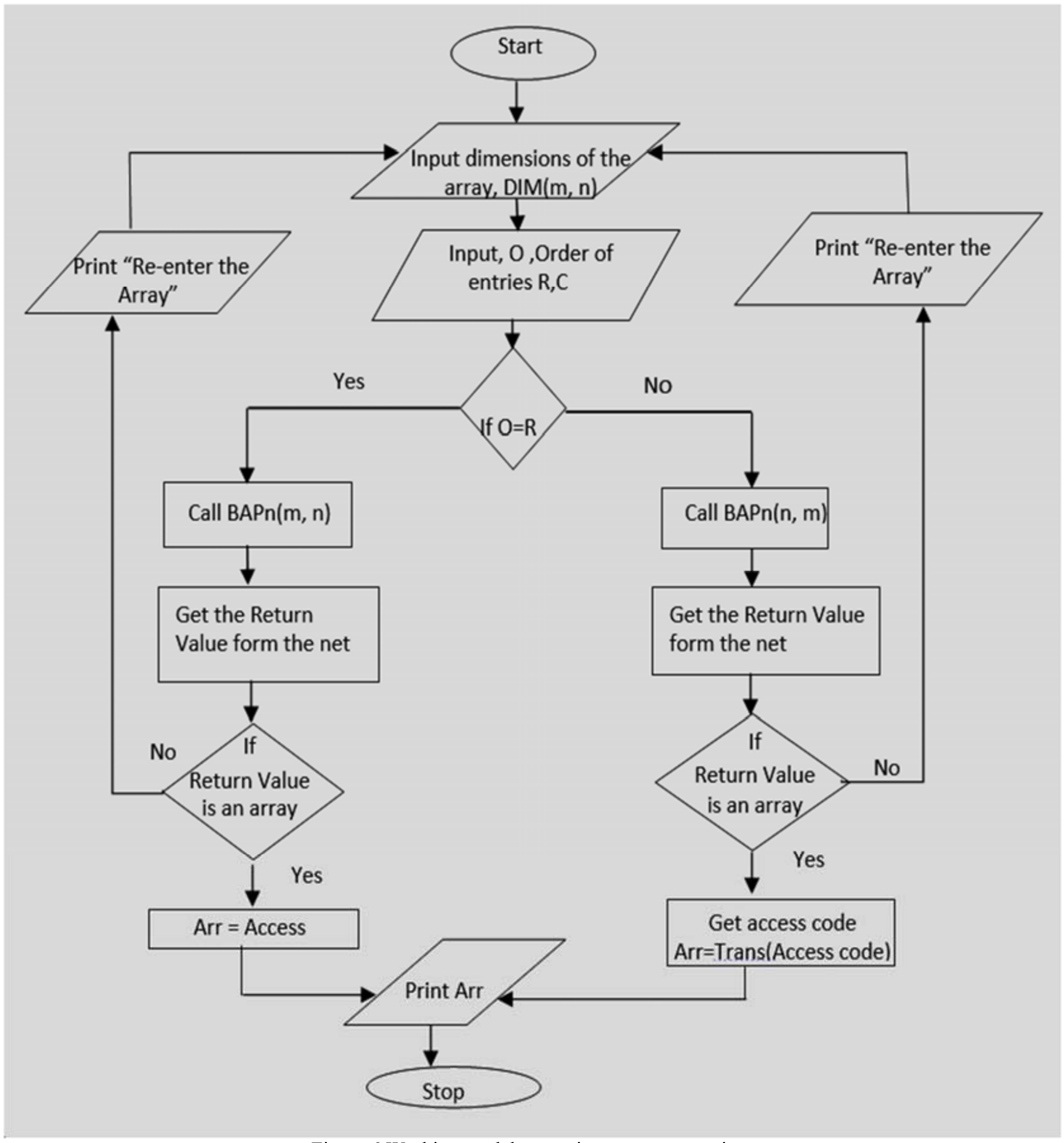

Start
Input dimensions of the array, DIM(m, n)
Input, O ,Order of entries R,C
Print "Re-enter the Array"
Print "Re-enter the Array"
If O=R
Yes
No
Call BAPn(m, n)
Call BAPn(n, m)
Get the Return Value form the net
Get the Return Value form the net
If Return Value is an array
No
Yes
If Return Value is an array
No
Yes
Arr = Access
Get access code Arr=Trans(Access code)
Print Arr
Stop

Figure 6 Working model to receive an array or string.

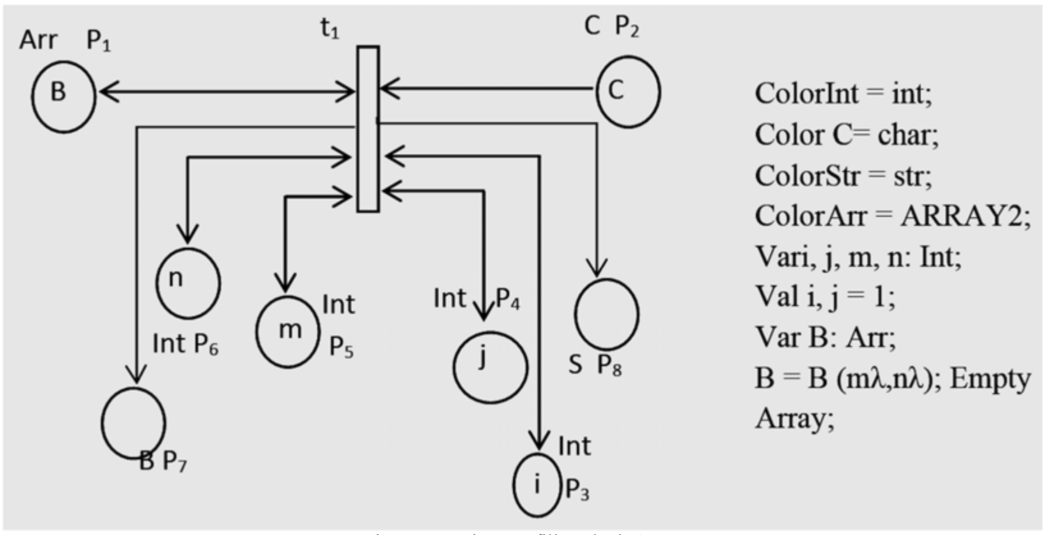
Every printable character is stored in *ij*th position of the blank array in the place P_1. When the "TAB" key is pressed the value of j is checked. If the value of *j* is less than *n* then it is incremented by 1. If the value of *j* equals n, then *i* is incremented by 1 and *j* is initialized back to 1. This process is repeated till the array is filled with all the printable characters. By pressing the "End" key the user signifies that all his characters have been entered.

Example 4

The parameters m and n are given by the flow chart which is invoking the Petri net. $BAPn(m,n)=(\Sigma, P, V, T, I, O, TA, \varphi, F)$, Where Σ is the set of all 95 printable characters and "Tab", P = {P_1, P_2, . . . , P_8}, T={ t_1}, Input and Output functions are shown in the Figure 7. The colour set would be two dimensional array, string, character and integers. The two dimensional array would be over ∑. The character can be chosen from ∑ and the "Tab" key. String is made up of ∑. Initial Attributes of the tokens:

$$TA=\{\langle B,P_1,B\rangle,\langle C,P_2,C\rangle,\langle i,P_3,1\rangle,\\ \langle j,P_4,1\rangle,\langle r,P_5,m\rangle,\langle c,P_6,n\rangle\}$$

The partial function φ states how the attributes of each token changes, when any enabled transition is fired. F = {P_7, P_8}.



```
ColorInt = int;
Color C= char;
ColorStr = str;
ColorArr = ARRAY2;
Vari, j, m, n: Int;
Val i, j = 1;
Var B: Arr;
B = B (mλ,nλ); Empty
Array;
```

Figure 7 Petri net to fill a Blank Array

$$
\begin{aligned}
&\varphi_{t_1}[\langle B, P_1, B\rangle, \langle C, P_2, C\rangle, \langle in1, P_3, i\rangle, \langle in2, P_4, j\rangle, \langle r, P_5, m\rangle\langle c, P_6, n\rangle]\\
&= [\langle B, P_1, b_{ij} = "C"\rangle, \langle in1, P_3, i\rangle, \langle in2, P_4, j\rangle, \langle r, P_5, m\rangle\langle c, P_6, n\rangle]\\
&\ \textit{if}\quad Ascii(C) \neq 9, 33 \leq Ascii(C) \leq 126 \ \& \ b_{ij} = \lambda\\
&\varphi_{t_1}[\langle B, P_1, B\rangle, \langle C, P_2, C\rangle, \langle in1, P_3, i\rangle, \langle in2, P_4, j\rangle, \langle r, P_5, m\rangle, \langle c, P_6, n\rangle]\\
&= [\langle B, P_1, B\rangle, \langle in1, P_3, i\rangle, \langle in2, P_4, j+1\rangle, \langle r, P_5, m\rangle\langle c, P_6, n\rangle]\\
&\ \textit{if}\quad Ascii(C) = 9 \ \& \ j < n\\
&\varphi_{t_1}[\langle B, P_1, B\rangle, \langle C, P_2, C\rangle, \langle in1, P_3, i\rangle, \langle in2, P_4, j\rangle, \langle r, P_5, m\rangle, \langle c, P_6, n\rangle]\\
&= [\langle B, P_1, B\rangle, \langle in1, P_3, i+1\rangle, \langle in2, P_4, 1\rangle, \langle r, P_5, m\rangle\langle c, P_6, n\rangle]\\
&\ \textit{if}\quad Ascii(C) = 9 \ \& \ j = n\\
&\varphi_{t_1}[\langle B, P_1, B\rangle, \langle C, P_2, C\rangle, \langle in1, P_3, i\rangle, \langle in2, P_4, j\rangle, \langle r, P_5, m\rangle, \langle c, P_6, n\rangle]\\
&= [\langle Access\_Code, P_7, B\rangle] \quad \textit{if}\quad Ascii(C) = 3 \ \& \ i = m, j = n\\
&\varphi_{t_1}[\langle B, P_1, B\rangle, \langle C, P_2, C\rangle, \langle in1, P_3, i\rangle, \langle in2, P_4, j\rangle, \langle r, P_5, m\rangle, \langle c, P_6, n\rangle]\\
&= [\langle EM, P_8, "Error - Message"\rangle]
\end{aligned}
$$

To start with a blank array of size m x n is in the place P_1. The array is built by the user. Initially i, j values will be 1. At a given time only one character is entered by the user. The first character is stored in b_{11}. The next character is "Tab" so the value of j is incremented, the head moves to b_{12} and the next character is stored in b_{12}. Once j reaches the value of n then i is incremented by 1and the value of j is initialized to 1.This goes on till the "Tab" key is pressed at i = m and j = n. Then the blank matrix is filled with all the required characters and the array is moved to P_7 as the Access Code. In any other case the Error message is put in P_8.If the user enters all the *mn* printable characters with the "Tab" in between every two printable characters, then the access code is formed as an array. In any other case the array does not reach P_7. Only an error message reaches the place P_8. The flow chart in Figure 6 either receives an array (access-code) or a string (error-message) from the net in the Figure 7. If an error message is received, then the user has to start all over again to form his access code. Till he has his *mn* printable characters in the positions of his array the net keeps giving error messages. The flow chart in Figure 5 is used once again to authenticate the access.

IV. CONCLUSION

Digital technology forces people to use the various websites on day to day basis. Password is the key factor to access any application in the computer or mobile phones. Alphanumeric, graphical passwords and biometric authentication are the different types of validation process available at present. Alphanumeric passwords are the most common mode of authentication. Since information security is always a concern for network users, instead of using a string password, an array password can be used. The Petri net model is used to generate an access code. Coloured Petri net models to generate array languages are defined. Attributes are used for tokens to impose more conditions for enabling transitions. Entries of the array password are taken from the user in two different ways. Either the "Tab" key can be used in between every two characters which will move the control accordingly or the characters can be entered without using the "tab" key. Petri net models have been designed for both cases. When an array password is used, the possibility of the information getting hacked is considerably less.

REFERENCES

[1] Jensen, Kurt, Rozenberg, Grzegorz (Eds.),High-level Petri Nets :Theory and Application, Springer Verlag, 1991.

[2] Jensen, Kurt, Coloured Petri Nets: A High Level Language for System Design and Analysis, in Advances in Petri Nets, G. Rozenberg, ed., vol. 483 of Lecture Notes in Computer Science, Springer-Verlag, New York, 1990, pp. 342- 416.

[3] J. L. Peterson, "Petri Net Theory and the Modeling of Systems," Prentice Hall, Englewood Cliffs, 1981.

[4] James L. Peterson, A Note on Colored Petri Nets, Information Processing Letters, Volume 11, Number 1, 1980, pages 40-43.

[5] Lalitha, D, Rectangular array languages generated by a Colored Petri net, Proceedings of IEEE International Conference on Electrical, Computer and Communication Technologies, ICECCT 2015.

[6] Lalitha, D. Rectangular array languages generated by a Petri net, Advances in Intelligent Systems and Computing ,volume 332,Computational Vision and Robotics, pp 17-27.

[7] D. Lalitha, K. Rangarajan, D.G. Thomas, Rectangular arrays and Petri nets, International Workshop on Combinatorial Image Analysis, Lecture Notes in Computer Science, volume 7655,pp 166-180.

[8] D. Lalitha, K. Rangarajan, Petri Net Generating Hexagonal Arrays, International Workshop on Combinatorial Image Analysis, Lecture Notes in Computer Science, volume 6636,pp235-247.

[9] D. Lalitha, K. Rangarajan, Column and row catenation Petri net system, Bio-Inspired Computing: Theories and Applications, IEEE Fifth International Conference on, 2010.

[10] Singhal M and Tapaswi S "Software Tokens Based Two Factor Authentication Scheme", International Journal of Information and Electronics Engineering, Vol. 2, No. 3, pp. 383, 2012.

[11] Silver D, Jana S, Boneh D, Chen E and Jackson C "Password managers: Attacks and defences", in 23rd USENIX Security Symposium- USENIX Security 14, pp. 449-464, 2014.

[12] Bafna A and Kumar S (2012). "Proactive Approach for Generating Random Passwords for Information Protection", Procedia Technology, Vol. 4, pp. 129-133.

.

.