DOI: 10.5604/01.3001.0013.2550

GENERATORS OF ONE-TIME TWO-FACTOR AUTHENTICATION PASSWORDS

Olga Ussatova[1], Saule Nyssanbayeva[2]
[1]Al-Farabi Kazakh National University, Almaty, Kazakhstan, [2]Institute of Information and Computational Technologies, Almaty, Kazakhstan

Abstract. *The paper presents algorithms for generating a one-time two-factor authentication passwords where application of trigonometric functions have been considered. To protect the opening of a one-time password, a secret string is read that consists of a sequence of randomly generated characters. The second factor is due to the fact that that the code has a certain validity period. The presented password generators allow the formation of secret words and trigonometric functions that the proposed two-factor authentication method consists of. The algorithm presented was implemented in Java Script. The algorithm includes blocks for checking randomly generated words and functions.*

Keywords: password generator, two-factor authentication, data protection

GENERATORY JEDNORAZOWYCH DWUCZYNNIKOWYCH HASEŁ AUTORYZACJI

Streszczenie. *W pracy przedstawiono algorytmy generowania jednorazowych, dwuczynnikowych haseł uwierzytelniających, w których uwzględniono zastosowanie funkcji trygonometrycznych. Aby chronić otwarcie jednorazowego hasła, odczytywany jest tajny ciąg składający się z sekwencji losowo generowanych znaków. Drugi składnik wynika z faktu, że kod ma określony okres ważności. Przedstawione generatory haseł umożliwiają tworzenie tajnych słów i funkcji trygonometrycznych, z których składa się proponowana metoda dwuczynnikowego uwierzytelniania. Przedstawiony algorytm został zaimplementowany w Java Script. Algorytm zawiera bloki do sprawdzania losowo generowanych słów i funkcji.*

Słowa kluczowe: generator hasła, uwierzytelnianie dwuskładnikowe, ochrona danych

Introduction

Ensuring the safety of confidential information must begin with identifying a system of threats, that is, negative processes that contribute to information leakage. In the modern world, the storage of electronic information, its value and significance has increased many times over. The need to ensure the safety of data storage, regular change and verification of passwords and control of the probability of information leakage have become an integral part of the information system. One of the most common methods of protecting information is password access to data. However, along with the undoubted advantages, this method of data protection has certain disadvantages: you can forget the password, it can be "hacked". A two-factor authentication data protection system based on one-time password generation is proposed.

This article describes the results obtained when developing a generator of trigonometric functions and secret words for generating a one-time two-factor authentication password based on an application using a smartphone.

1. Material and methods

The proposed system of information protection based on two-factor authentication using a combination of two factors: permanent and one-time passwords [7]. The user chooses a permanent password (the first factor) himself and uses it when registering an account. Before authorization must be registered in the application. After that, the application starts to enter user data (login and password), which must correspond to the registered data.

Then you need to enter the application on your smartphone and enter the initial data to generate a temporary password. A one-time or temporary password (the second factor) is generated on the server by the proposed algorithm [7] and is valid for a specific period of time for one authentication session. The time in the application is 20 seconds. The advantage of a one-time password is that the password is not reused. Thus, an attacker who intercepted data from a successful authentication session cannot use the copied password to gain access to the protected system. One-time password generation is possible online. The software sends a request to the authorization server to generate a temporary password. It is generated on the server and displayed to the user in additional software on the smartphone. The password has a short duration – 20 seconds. The one-time password is generated based on the result of the selected trigonometric function, which has a number of variables generated based on the result of the SHA256 hash function [4, 5]. The input string for the hash function is a combination of user credentials, the current Greenwich Mean Time, and an additional secret string. The result of the hash function is divided into individual numbers, which will be the indices for selecting the function and its initial data. The secret string is a required field that will be randomly selected from the array. The secret line is changed at each input, due to which it will be much more difficult to open the initial input line, which allows to further strengthen the protection. The data for the input string has the values: login, password, current date, time and secret string. The next step to generate a password will be the result of the SHA256 hash function, which underlies the choice of the trigonometric function.

2. Secret word generating

For the formation of a secret string, a generator has been developed that allows one to randomly form words. Word dictionaries were not used, as words are easier to crack [3, 6]. The generator is based on the use of the Latin alphabet of capital and uppercase characters in a total of 52. The length of the generated word is 5 characters.

In the final code, the algorithm described above is implemented in the Java Script language and has the following form:

```
letfuncVariblesList = []
letfuncComponents = []
let expressions = ['+', '-', '/', '*']

module.exports.getWord = function() { // word generator
let chars = 'ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz'
letwordLength = getRandomInt(5, 10)
let word = "
for (let i = 0; i <= wordLength; i++) {
letcharIndex = getRandomInt(0, chars.length - 1)
word += chars[charIndex]
  }
return word
}
```

For the analysis of the generator used the method of complete enumeration [1, 2]. According to this method, the length of the string is taken into account (the length of the string is 5 characters in the appendix) and, for example, the search speed of 100,000 words per second is used. The number of options is calculated by the formula:

$$S = A^n \qquad (1)$$

where A is the number of characters and n the length of the string.

An example of the analysis of the generator is presented in table 1.

Table 1. Analysis of the generator

Number of characters	Number of options	Persistence	Search Time
1	52	5 byte	Less than a second
5	380204032	26 byte	63 minutes

S= 52^5=380204032

Due to the fact that, according to the developed two-factor authentication algorithm, the generation of a one-time password occurs every 20 seconds, the probability of hacking the generated secret word is almost impossible. This confirms the efficiency of the proposed generator.

3. Trigonometric Function Generator

As stated above, the generation of a one-time password is based on the result of the selected trigonometric function, which has a number of variable parameters. The choice is made in accordance with the result of the obtained hash function of the SHA256 standards, where the first characters are used, which will be indices in a table of 256x256 dimension. By this index, the function will be selected and its parameters will be determined. According to the results of the calculation, digits after the comma are taken as a one-time temporary password, starting from the 5th position and 6 digits long.

The resulting number will be a temporary password that must be entered into the application. To implement this method, a generator of trigonometric functions has been developed, the use of which will greatly facilitate the formation of these functions. The algorithm of the generator of the trigonometric function is shown in Figure 1–3.

To generate a trigonometric function, the number of variables is taken as the basis. There are 7 of them in this generator: a, b, c, x, y, p1, p2. Initially, a list of variables is formed, resulting in a random number of variables Count from 1 to the number of variables minus 1. Then, the array is searched through the array with certain variables N- times based on a random number from 0 to the length of the array minus 1.

Read variable from the array, which is added in the new array and removed from the old.

After the cycle is completed, a list of variables for the function is formed.

Based on this list, the constituent parts (Math.sin (a), 1 / Math.tan (p2)) of the format – “['Math.sin ()', 'Math.cos ()', 'Math.tan ( ) ',' (1 / Math.tan ()) ',' () '] ”.The ComponentsCount function (the number of elements minus 1) starts the loop through the array with the generated variables. In the loop at each step, a random number componentIndex from 0 to componentsCount is formed. The element with the corresponding value of the componentIndex index is converted by replacing the symbol “with a variable from the list and added to the new array.

As a result, a list of component parts with variables is formed. Next, rows are formed based on a random number from 1 to 3. In the cycle, the components, separated by signs of mathematical expressions, merge randomly.

After receiving the strings, they are joined separated by mathematical expressions.

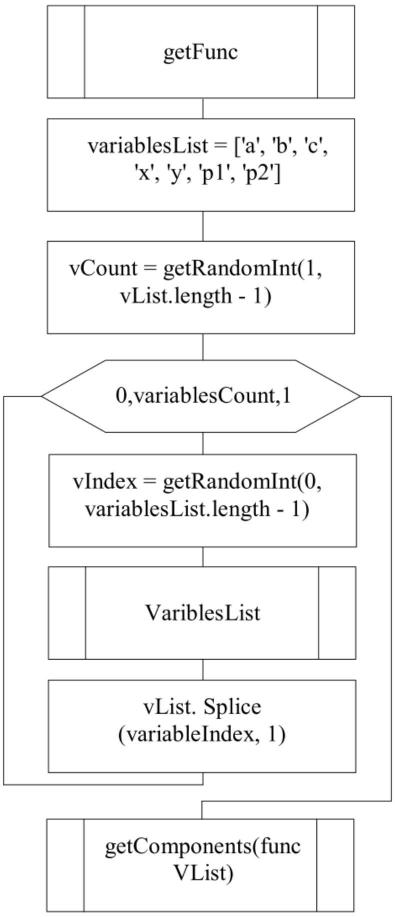
getFunc
variablesList = ['a', 'b', 'c', 'x', 'y', 'p1', 'p2']
vCount = getRandomInt(1, vList.length - 1)
0,variablesCount,1
vIndex = getRandomInt(0, variablesList.length - 1)
VariblesList
vList. Splice (variableIndex, 1)
getComponents(func VList)

Fig. 1. Function getFunc

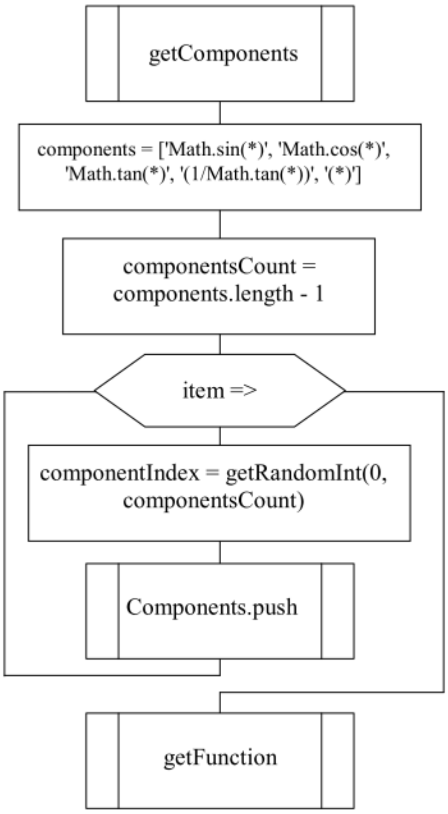
getComponents
components = ['Math.sin(*)', 'Math.cos(*)', 'Math.tan(*)', '(1/Math.tan(*))', '(*)']
componentsCount = components.length - 1
item =>
componentIndex = getRandomInt(0, componentsCount)
Components.push
getFunction

Fig. 2. Function getComponents

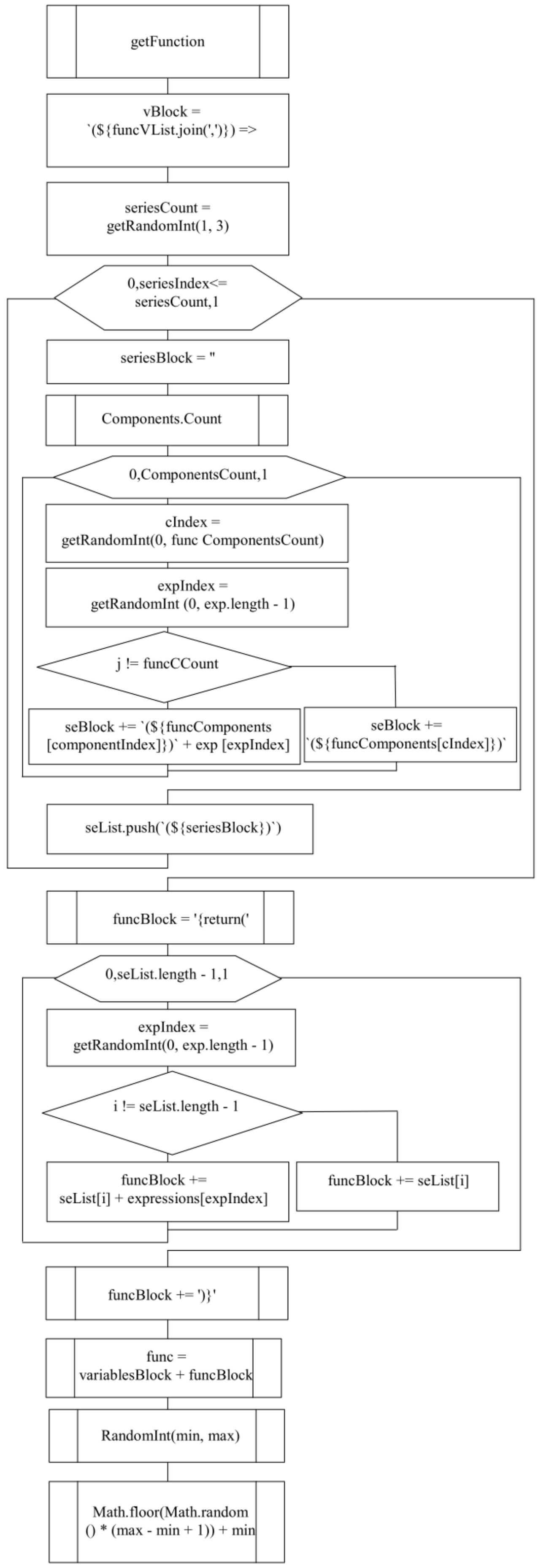
getFunction
vBlock = `(${funcVList.join(',')}) =>
seriesCount = getRandomInt(1, 3)
0,seriesIndex<= seriesCount,1
seriesBlock = ''
Components.Count
0,ComponentsCount,1
cIndex = getRandomInt(0, func ComponentsCount)
expIndex = getRandomInt (0, exp.length - 1)
j != funcCCount
seBlock += `(${funcComponents [componentIndex]})` + exp [expIndex]
seBlock += `(${funcComponents[cIndex]})`
seList.push(`(${seriesBlock})`)
funcBlock = '{return('
0,seList.length - 1,1
expIndex = getRandomInt(0, exp.length - 1)
i != seList.length - 1
funcBlock += seList[i] + expressions[expIndex]
funcBlock += seList[i]
funcBlock += ')}'
func = variablesBlock + funcBlock
RandomInt(min, max)
Math.floor(Math.random () * (max - min + 1)) + min

Fig. 3. Function getFunction

Software implementation of the generator has the following form:

```
module.exports.getFunc = function()
{ // function generator
funcVariblesList = []
  let variablesList = ['a', 'b', 'c', 'x', 'y', 'p1', 'p2']
letvariablesCount = getRandomInt(1, variablesList.length - 1)
for (let i = 0; i <= variablesCount; i++)
{
letvariableIndex = getRandomInt(0, variablesList.length - 1)
funcVariblesList.push(variablesList[variableIndex])
variablesList.splice(variableIndex, 1)
  }
returngetComponents(funcVariblesList)
}
functiongetComponents(variablesList)
{
funcComponents = []
let components = ['Math.sin(*)', 'Math.cos(*)', 'Math.tan(*)', '(1/Math.tan(*))', '(*)']
letcomponentsCount = components.length - 1
variablesList.forEach(item =>
{
letcomponentIndex = getRandomInt(0, componentsCount)
funcComponents.push(components[componentIndex].replace('*', item))
  })
returngetFunction()
}
functiongetFunction()
{
// forming a block of variables
letvariablesBlock = `(${funcVariblesList.join(',')}) => `
// forming a body function
letseriesCount = getRandomInt(1, 3)
letseriesList = []
for (let seriesIndex = 0; seriesIndex<= seriesCount; seriesIndex++) { // there will be 2 rows
letseriesBlock = ''
letfuncComponentsCount = funcComponents.length - 1
for (let j = 0; j <= funcComponentsCount; j++)
{
letcomponentIndex = getRandomInt(0, funcComponentsCount)
letexpressionIndex = getRandomInt(0, expressions.length - 1)
if (j != funcComponentsCount)
{
seriesBlock += `(${funcComponents[componentIndex]})` + expressions[expressionIndex]
      } else
{
seriesBlock += `(${funcComponents[componentIndex]})`
      }
    }
seriesList.push(`(${seriesBlock})`)
  }
letfuncBlock = '{return('
for (let i = 0; i<= seriesList.length - 1; i++)
{
letexpressionIndex = getRandomInt(0, expressions.length - 1)
if (i != seriesList.length - 1)
{
funcBlock += seriesList[i] + expressions[expressionIndex]
    } else {
funcBlock += seriesList[i]
    }
  }
funcBlock += ')}'
  // string of results
returnfunc = variablesBlock + funcBlock
}
```

```
FunctiongetRandomInt(min, max) { // function to get a random number for a given range
returnMath.floor(Math.random() * (max - min + 1)) + min;
}
```

As a result, we obtain a generated string function, which we use to calculate a one-time two-factor authentication password.

4. Conclusion

The use of generators to work in the formation of a one-time password, allows you to enhance the level of protection of the described system. Entropy is traditionally a measure of the strength of passwords - a measure of uncertainty, usually measured in bits. One bit entropy corresponds to the uncertainty of the choice of two passwords, two bits of 4 passwords, etc. The strength of a password should be considered only in the context of a specific password authentication system. This is due to the fact that different systems in varying degrees implement (or do not implement at all) the mechanisms for counteracting attacks aimed at breaking passwords, and also because some systems contain errors or use unreliable algorithms.

References

[1] Alata E., Nicomette V., Kaaniche M., Dacier M., Herrb M.: Lessons learned from the deployment of a high-interaction honeypot. Proc. Dependable Computing Conference (EDCC06), Coimbra, Portugal, October 18-20, 2006, 39–46.

[2] Ayankoya F., Ohwo B.: Brute-Force Attack Prevention in Cloud Computing Using One-Time Password and Cryptographic Hash Function. International Journal of Computer Science and Information Security (IJCSIS) 17(2)/2019, 7–19.

[3] Bahaa Q.M.: Preventing brute force attack through the analyzing log. Iraqi Journal of Science 55(3)/2013, 663–667.

[4] https://www.nist.gov (available 02.09.2018).

[5] https://www.seagate.com/files/www-content/solutions-content/security-and-encryption/id/docs/faq-fips-sed-lr-mb-605-2-1302-ru.pdf (available 12.10.2018).

[6] Lamar A.: Types of threats to database security (http://www.brighthub.com/computing/smbsecurity/articles/61402.aspx), 2012, (available 18.03.2019).

[7] Nyssanbayeva S., Ussatova O.: Two-factor authentication in the automated control system. International scientific conference Information Science and Applied Mathematics. Almaty, 2018, Vol. II, 239–242.

M.Sc. Olga Ussatova
e-mail: uoa_olga@mail.ru

Al-Farabi Kazakh National University, Institute of Information and Computational Technologies Ph.D. student in the specialty: «Information Security Systems». In 2003 graduated of the KazNTU – specialty 3704 – "Software computer technology and automated systems". In 2014 graduated of the Turan University – master's degree of "Computing system and Software". Research interests research theme - database protection using two-factor authentication.

ORCID ID: 0000-0002-5276-6118

Prof. Saule Nyssanbayeva
e-mail: sultasha1@mail.ru

Institute of Information and Computational Technologies, Information Security Laboratory. A specialist in the field of information security. The goals and objectives of the research are development and analysis of methods, algorithms and means of cryptographic protection of information based on modular arithmetic during its transmission and storage in info communication systems and networks.

ORCID ID: 0000-0002-5835-4958

otrzymano/received: 15.05.2019 *przyjęto do druku/accepted: 15.06.2019*