"Mircea cel Batran" Naval Academy Scientific Bulletin, Volume XVIII – 2015 – Issue 1
Published by "Mircea cel Batran" Naval Academy Press, Constanta, Romania // The journal is indexed in:
PROQUEST SciTech Journals, PROQUEST Engineering Journals, PROQUEST Illustrata: Technology, PROQUEST Technology Journals, PROQUEST Military Collection PROQUEST Advanced Technologies & Aerospace

A MULTI-FACTOR AUTHENTICATION SCHEME INCLUDING BIOMETRIC CHARACTERISTICS AS ONE FACTOR

Marius Iulian MIHAILESCU[1]
Ciprian RACUCIU[2]
Dan Laurentiu GRECU[3]
Loredana Stefania NITA[4]
[1]Ph.D, Informatics Department, University of ,,Titu Maiorescu", mihmariusiulian@gmail.com
[2]Ph.D, Informatics Department, University of ,,Titu Maiorescu", ciprian.racuciu@gmail.com
[3]Ph.D, Informatics Department, University of ,,Titu Maiorescu", danlaurentiugrecu@gmail.com
[4]MSc., Computer Science Department, University of Bucharest, stefanialoredanani@gmail.com

Abstract: *Multi-factor authentication schemes have been proved to be very useful in many authentication systems including biometric ones. In this paper we have proposed a multi-factor authentication scheme, in which one of the main components is represented by the generation of a token and a password (known as the kernel of the multi-factor scheme) and another component is represented by a module which will take one of the biometric characteristics (face image, handwriting, holographic signature). The token ID and passcodes generated values will be encrypted and decrypted with RSA. We will show how the scheme works using a simulator that we have developed for this goal.*

Introduction

Multi-factor authentication (MFA) represents a technique through which a user must follow more distinct authentication steps in order to gain access control on a computer.

There are many types of factors [5, 6]:

- **Knowledge factors** are the most used techniques of authentication, such as passwords, PIN, secret questions, where only the user knows the right answer.
- **Possesion factors**uses techniques like tokens. For example, it could be used connected tokens, where the data is sent directly between a computer and the token which is directly connected to that computer or disconnected tokens, where there is no connection between token and computer, the data being displayed on a built-in screen and the user should introduce it manually.
- **Inherence factors** are authentication techniques based on the biometric characteristics of the user, such as fingerprint, retina, and voice.

The MFA raises the security and reduces the risk that a forger cracks the security through addition of an obstruction to entry, making difficult for forger to login into the stolen account even if he knows the password. This type of authentication is used in many areas, such as companies, governments, and financial area, where the security needs to be at a high level according to the sensible data involved.

There are more tendencies in MFA. In [7] the authors present some authentication techniques using a graphical password, where the second authentication stage is password decoding employing user's device. The password is represented by an image, and the user should decode it finding the suitable clicks and their order on that image, which are given only through user's device as described in [7]. In [11] the authors propose an authentication mechanism based on two stages, where the first stage uses two factors in biometric authentication and the second one integrates some factors to authenticate depending on the biometric feature from the first stage. Another approach for MFA are authentication systems used for wireless payment [8].

We propose a software simulation application which uses multi factors used in authentication process. In the first step, the user is asked to introduce the username and password (knowledge factor), in the second step the system sends a passcode to the user's phone or email and the user introduce it (possession factor), and, in the last step, the user must authenticate with the biometric characteristic (face, signature or fingerprint) which he provided when the user account was created (inherence factor).

There are many methods for face recognition, according to [9]. For traditional face recognition system are used algorithms that extract the user's features from a picture, in 3D face recognition are used sensors which collect data about the face's shape, while the skin texture analysis are based on visual particularies of the skin.

Handwritten signature is one of the most used biometric characteristic which allows a person to authenticate in a system. It could be used in both online and offline applications. In the online applications, when the user signs, the data are dynamically retrieved from the signature, while the offline applications are based on a scanned picture of handwritten signature [10].

The goal of this article is to go further into the mechanism used in authentication process based on biometric characteristics. We will present our proposed algorithm that combine a mechanism based on a Token ID and a passcode, with a biometric characteristic in order to gain access to a system.

The First Version of Proposed Algorithm

Below, we will describe step-by-step how the algorithm is working.

The general steps are the followings:

1. The user enters the *user name* and *password*;
2. Based on the user name, the application will generate a token identification *(Token ID*– see Section 1.1*)*;
3. The user will take the *Token ID* and will enter on his device (mobile phone, tablet, token generator device etc.) and it will generate a *passcode*(see Section 1.2). Notice that the *passcode* it is generated based on the *Token ID.*
4. The application will validate the *Token ID* and *passcode.*
5. If everything is OK at *step 4* then the user is asked to choose his biometric characteristics (face image, holographic signature, and handwriting).

1.1. Generating the *Token ID*

The function for generating the *Token ID*it is based on two parameters, *token_id_length*and *generating_characters*. The second one, *characters,* is optional in our proposed function, because it comes already initialized in the declaration of the function. Anytime, the length of the token id can vary. This aspect will let us to improve the complexity of the token id and to be adapted to different systems in which the authentication based on a token and a passcode is necessary.

In order to generate the token ID, we will use the *RNGCryptoServiceProvider* class [3] from .NET Framework. The class will allow us to implement a cryptographic random

number used as generation. The generation process is based on the cryptographic service provider [1,2].
The steps of the algorithm for generating the passcode, is as follows:

1. We set a length for the *Token ID.*
2. If the length is greater than 0, then we create a data structure, called *HashSet*allowing only characters, in which we will store the characters from the optional variable, *generating characters.*
3. We will use a constant *byteSize*which is initialized with *0x100* which means 256 bytes and it will be used to test if the length of the allowed characters is less than 256 bytes.
4. With the help of *RNGCryptoServiceProvider* class we will build the token ID.
5. The token ID will be generated with the help of *StringBuilder* class [4].

The following function (C# Source Code) represents the algorithm and how it is implemented.

```
publicstringGeneratePassCode(inttoken_id_length,
stringgenerating_characters = "")
{
if (token_id_length< 0)
thrownewArgumentOutOfRangeException("length", "lungimea nu
poatesa fie mai mica de 0.");
if (string.IsNullOrEmpty(generating_characters))
thrownewArgumentException("generating_characters nu
poatesa fie gol.");
constintbyteSize = 0x100;
varallowed_characters = new
         HashSet<char>(generating_characters).ToArray();

if (byteSize<allowed_characters.Length)
thrownew
ArgumentException(String.Format("generating_characterspoat
esacontina nu maimult de {0} caractere.", byteSize));
using (varrng = newRNGCryptoServiceProvider())
{
var result = newStringBuilder();
varbuf = newbyte[128];
while (result.Length<lungime)
{
rng.GetBytes(buf);
for (var i = 0; i <buf.Length&&result.Length<
lungime; ++i)
{
varoutOfRangeStart = byteSize - (byteSize %
set_caractere_permis.Length);
if (outOfRangeStart<= buf[i])
continue;result.Append(allowed_characters[buf[i] %
allowed_characters.Length]);
}
}
returnresult.ToString();
 }
}
```

1.2. Generating the *passcode*

Generating *passcode* based on the token ID is done using a One-time password algorithm.
The general steps through which we have obtained the *passcode* are the followings:

1. We take the token ID as the starting value, known as seed.
2. We choose a hash function $f(tokenID)$ (we have used SHA256, SHA384, and SHA512) and we will apply many times (let's say for example, 500 times on the token ID). The obtained value $f^{500}(tokenID)$ is stored on the target computer.
3. After the user enters his user name and his passwords, based on the generated token ID, a passcode will be computed as being derived obtained by applying the function f for 499 times on the token ID, which looks like $f^{499}(tokenID)$. The target computer will authenticate the user with this passcode as being the right one, because $f(passcode)$ is$f^{500}(tokenID)$, obtaining a value that is store. The stored value is replaced with the *passcode*and if everything is ok, the user can login.
4. At the next authentication, the login process must be followed by the result applied as$f^{448}(tokenID)$. The result can be validated because if we apply the hashing function we will obtain $f^{449}(tokenID)$ which is represented by the $passcode$, the value that is stored somewhere on the computer after the previous login authentication. In this case, the value will replace p and the user is authenticated.

For each of the processes described above, generating the *Token ID*and *passcode,* are secured using a HMAC (keyed-hash message authentication code).
The general form of the HMAC algorithm is applied as it follows:

$$HMAC(key, tokenID) = H((key \oplus outer\_padding)\,|H((key \oplus inner\_padding)|tokenID))$$

and

$$HMAC(key, passcode) = H((key \oplus outer\_padding)\,|H((key \oplus inner\_padding)|passcode))$$

where:

- *HMAC*represent the construction for calculating the message authentication code (MAC) used for proving the integrity and authenticity of Token ID and passcode.
- *H*represent the hash function (in our case, we have implement some varieties for SHA256, SHA384, and SHA512). The implementations of these functions are presented in Section 1.3.
- *key*represent the secret key. In our proposal we have used some keys generated with OpenSSL 1.0.2a and imported them into the application, as we can see in Figure 1 and 2. The keys have been generated using DSA and RSA algorithms. In order to secure the token ID and passcode, you have to choose one of the two keys.
- *token ID* and *passcode* represent the two values that need to be authenticated.
- | represent the concatenation process.
- ⊕represent the *exlcusive OR* operation (XOR).
- *outer_padding*and *inner_padding*represent two hexadecimal constants.

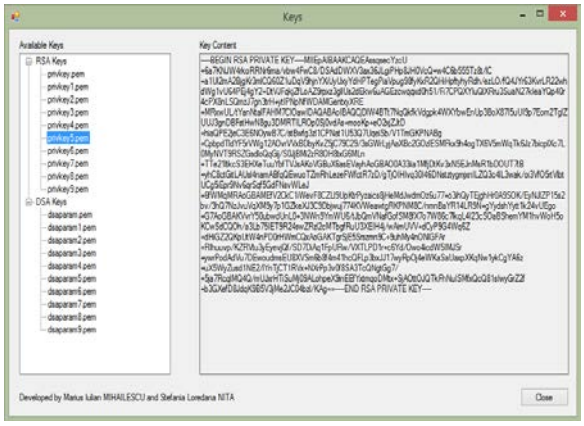

Figure 1 RSA key

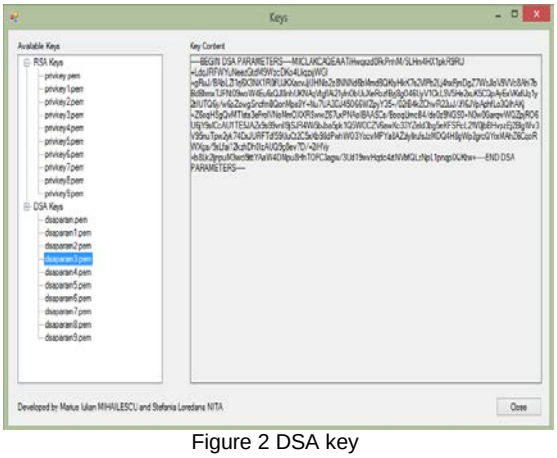

Figure 2 DSA key

1.3. Implementation of the hash functions SHA256, SHA384, and SHA512

In this section we will describe how the mentioned hash functions are implemented.
The hash functions are implemented using C# as programming language.
The following function, *ComputeHash,* will compute the hash for a specific value, in our cases the values will be token ID or passcode. The source code is very easy to understand and to implement it.

```
publicstringComputeHash(string value, string hash,
byte[] salt)
{
intminSaltLength = 4;
intmaxSaltLength = 16;

byte[] SaltBytes = null;

if (salt != null)
SaltBytes = salt;
else
    {
        Random r = newRandom();
intSaltLength = r.Next(minSaltLength,
maxSaltLength);
SaltBytes = newbyte[SaltLength];
RNGCryptoServiceProviderrng =
newRNGCryptoServiceProvider();
rng.GetNonZeroBytes(SaltBytes);
rng.Dispose();
}

byte[] plainData =
ASCIIEncoding.UTF8.GetBytes(value);

byte[] plainDataAndSalt = newbyte[plainData.Length
+ SaltBytes.Length];
for (int x = 0; x <plainData.Length; x++)
plainDataAndSalt[x] = plainData[x];

for (int n = 0; n <SaltBytes.Length; n++)
plainDataAndSalt[plainData.Length + n] =
SaltBytes[n];

byte[] hashValue = null;

switch (hash)
{
case"SHA256":
SHA256Managedsha = newSHA256Managed();
hashValue = sha.ComputeHash(plainDataAndSalt);
break;

case"SHA384":
SHA384Managed sha1 = newSHA384Managed();
hashValue = sha1.ComputeHash(plainDataAndSalt);
break;

case"SHA512":
SHA512Managed sha2 = newSHA512Managed();
hashValue = sha2.ComputeHash(plainDataAndSalt);
break;
}

byte[] result = newbyte[hashValue.Length +
SaltBytes.Length];

for (int x = 0; x <hashValue.Length; x++)
result[x] = hashValue[x];
for (int n = 0; n <SaltBytes.Length; n++)
result[hashValue.Length + n] = SaltBytes[n];

returnConvert.ToBase64String(result);
}
```

The next function, called *Confirm,* will return true or false if the hash value of the token ID or passcode is equal with the hash value stored on the target machine.

```
publicbool Confirm(stringplainText,
stringhashValue, string hash)
{
byte[] hashBytes =
Convert.FromBase64String(hashValue);
inthashSize = 0;
switch (hash)
    {
case"SHA256":
hashSize = 32;
break;
case"SHA384":
hashSize = 48;
break;
case"SHA512":
hashSize = 64;
break;
}

byte[] saltBytes = newbyte[hashBytes.Length -
hashSize];
for (int x = 0; x <saltBytes.Length; x++)
saltBytes[x] = hashBytes[hashSize + x];

stringnewHash = ComputeHash(plainText, hash,
saltBytes);

return (hashValue == newHash);
}
```

1.4. Biometric Characteristics Module

The biometric module authentication takes place when the Token ID and passcode has been validated. In this case, the user has to choose the biometric type (face, holographic signature, or handwriting) in order to authenticate. We assume that we have already the biometric characteristics enrolled into the system.
In Figure 3 we can observe that the user has the possibility to choose the biometric type.
In this moment we don’t have any security mechanism to assure the integrity of the biometric template used for storing the subject biometric data.

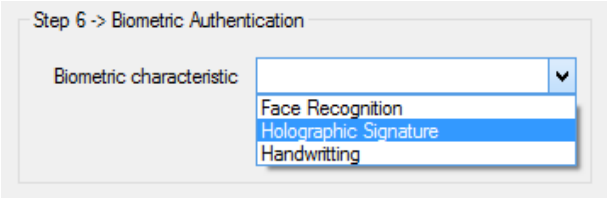

Figure 3 Biometric Types

Describing the Software Simulation Application (Multi-Factor)

The goal of simulation software application is to help us to understand how the multi-factor authentication schemes can be adapted to biometric systems.

The application is built in 5 easy steps, as follows:

1. Connecting to the database (Figure 4);
2. Enter the credentials (Figure 5);
3. Multi-factor Authentication Core (Figure 6);
   1.5. Generating Token ID (Figure 7);
   1.6. Generating passcode (Figure 8);
2. Passing the passcode to the authentication application (Figure 6);
3. Biometric Authentication (Figure 6).

Figure 4 Connecting to database

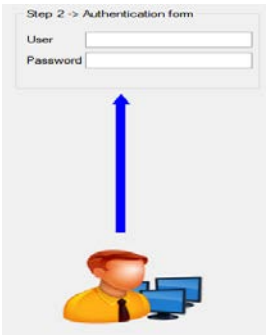

Figure 5 Enter the credentials

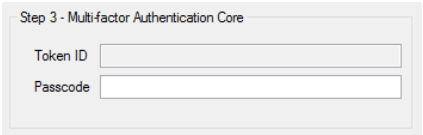

Figure 6 Multi-factor Authentication Core

Figure 7 Generating the Token ID

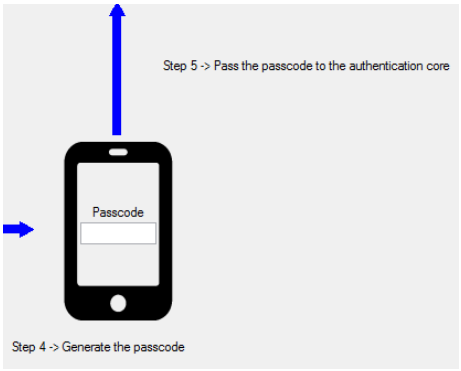

Figure 8 Generating the passcode

Conclusions

In this paper we have presented a framework that is used as simulation software in order to help us to understand how the multi-factor authentication schemes are working.

We have proposed some specific algorithms that can be used in order to generate token IDs and passcodes. Over these two processes we apply a security mechanism in order to assure the authenticity and integrity of the token ID and passcode.

The framework is not the final version of the proposed work. As we have stated above, this is one of the first version of the software and many other things will be done in future researching work.

Bibliography

[1]. Cryptographic Service Provider, http://en.wikipedia.org/wiki/Cryptographic_Service_Provider
[2]. Microsoft Cryptographic Service Providers, https://msdn.microsoft.com/en-us/library/windows/desktop/aa386983%28v=vs.85%29.aspx
[3]. RNGCryptoServiceProvider Class, https://msdn.microsoft.com/en-us/library/system.security.cryptography.rngcryptoserviceprovider%28v=vs.110%29.aspx
[4]. StringBuilder Class, https://msdn.microsoft.com/en-us/library/system.text.stringbuilder%28v=vs.110%29.aspx
[5]. Multi-factor authentication, http://en.wikipedia.org/wiki/Multi-factor_authentication#Implementation_considerations
[6]. Emiliano De Cristofaro, Honglu Du, JulienFreudiger, Greg Norcie, *A Comparative Usability Study ofTwo-Factor Authentication*, 8th NDSS Workshop on Usable Security (USEC 2014), http://arxiv.org/pdf/1309.5344.pdf.
[7]. AlirezaPirayeshSabzevar, Angelos Stavros, *Universal Multi-Factor Authentication Using Graphical Passwords*, Signal Image Technology and Internet Based Systems, 2008. SITIS '08. IEEE International Conference, Pages:625 – 632, ISBN: 978-0-7695-3493-0, http://cs.gmu.edu/~astavrou/research/StavrouUniversalMultiFactorAuthentication.pdf
[8]. SugataSanyal, AyuTiwari and SudipSanyal, *A Multifactor Secure Authentication System For Wireless Payment*, IADIS International Conference on Applied Computing Proceedings of the IADIS International Conference on Applied Computing, Salamanca, Spain, 18-20 February 2007, http://www.tifr.res.in/~sanyal/papers/Multifactor%20Secure%20Authentication.pdf.
[9]. *Facial recognition system*, http://en.wikipedia.org/wiki/Facial_recognition_system#Software.

[10]. AshishDhawan, Aditi R. Ganesan,*Handwritten Signature Verification*,ECE 533 – Project Report, https://homepages.cae.wisc.edu/~ece533/project/f05/ganesan_dhawanrpt.pdf
[11]. AbhilashaBhargav-Spantzel, Anna C. Squicciarini, Shimon Modi, Matthew Young, Elisa Bertino, Stephen J. Elliott, Privacy *Preserving Multi-Factor Authentication with Biometrics*,DIM '06 Proceedings of the second ACM workshop on Digital identity management, Pages 63 – 72, ACM New York, NY, USA 2006, ISBN: 1-59593-547-9, http://homes.cerias.purdue.edu/~bhargav/pdf/jcs07Bio.pdf