Journal of Information Security, 2016, 7, 206-214
Published Online April 2016 in SciRes. http://www.scirp.org/journal/jis
http://dx.doi.org/10.4236/jis.2016.73016

Scientific Research Publishing

Passwords Management via Split-Key

Kenneth Giuliani[1], V. Kumar Murty[1], Guangwu Xu[2]

[1]Department of Mathematics, University of Toronto, Toronto, Canada
[2]Department of EE & CS, University of Wisconsin-Milwaukee, Milwaukee, WI, USA
Email: ken.giuliani@utoronto.ca, murty@math.toronto.edu, gxu4uwm@uwm.edu

Received 2 March 2016; accepted 19 April 2016; published 22 April 2016

Abstract

This paper proposes a scheme for password management by storing password encryptions on a server. The method involves having the encryption key split into a share for the user and one for the server. The user's share shall be based solely on a selected passphrase. The server's share shall be generated from the user's share and the encryption key. The security and trust are achieved by performing both encryption and decryption on the client side. We also address the issue of countering dictionary attack by providing a further enhancement of the scheme.

Keywords

Password Encryption, Password Storage, Identity Management, Secret Sharing

1. Introduction

Being the first effective form of computer-based authentication, passwords are increasingly becoming a security problem in the modern age. There are an increasing number of websites emerging on the Internet, each demanding its own userid and password. A recent study reveals that Internet users, on average, have about 25 accounts that require password protection [1]. Users are suffering from "password fatigue" and this has led to some internet behavior on the part of users that makes them susceptible to attack. As indicated in [2], users write down passwords (making them vulnerable to onlookers) or select common or easily-guessed passwords. Users need a more effective method for managing the passwords which they require. Optimally, it may be beneficial to have a proper password management system. A straightforward solution would be to simply store the passwords in a certain location. Client password managers such as Roboform [3] keep the passwords on the client meaning that they can only be accessed when using the client machine. Browser-based password managers have been one of the most popular choices for user authentication and password management. Most popular browsers have provided a built-in feature for storing users' password (in encrypted form) and other login information into a database. However, as reported in [4], these password managers have vulnerabilities which can be exploited by

attackers to decrypt passwords easily. It is also noted that recently, security analysis of some popular web-based password managers have been reported in [5] [6], and security issues in auto fill polices, as well as vulnerabilities in one-time passwords, bookmarklets, and shared passwords have been identified.

It would be more optimal to store the passwords in a more accessible place, such as a web server. The problem with this approach is security and trust, as a user may not wish the server to have access to their plain passwords. This paper proposes a new scheme to assist with storing encrypted passwords on a server. The encryption key is split into a share derived from a user-selected passphrase and a share residing on a server. Only when both shares are combined can passwords be decrypted. Even though the data resides on the server, the processing is performed by the user which means that the server does not have access to the plaintext passwords, nor does it have the capability of deriving them. Hence, even if the user passphrase is compromised to an attacker, the attacker will still need to bypass authentication to the server to decrypt passwords.

The basic model in which this scheme will reside involves three parties: a user, a server, and a web service. The user will wish to store passwords and other login information on the server to be retrieved when it is needed. The user will then forward this data to the web service for processing. In practice, the web service will usually be a website which the user accesses, or some enterprise server. This will be of particular use as industry moves toward cloud computing. The basic data flow in this model is as follows (**Figure 1**):

1) User and server authenticate themselves to each other.
2) Server sends data needed to compute the password to the user.
3) User processes server data and sends to web service.

The authentication between user and server can be achieved by numerous different means. This may involve certificates, userid and password, two-factor authentication, or any other ways of network layer security protection. This step is out of scope for this paper. In the third step, the password data is forwarded to the web service. In practice this could be something such as a website or enterprise server. This step is dependent on mechanisms which already exist and thus is also out of scope for this paper. We will note here, though, that the security of this model will only be as good as the security of the third step. For example, if passwords are sent in the clear from the client to the third party, then they can be easily viewed by attackers regardless of any security mechanisms in place in the other two steps. The second step is the one in which the split-key algorithm will be effective. The passwords would need to be stored and retrieved from the server by the user after successful authentication. There are many different functional and security requirements for this step, including a fundamental security feature of performing both password encryption and decryption on the client side. From the detailed description of these functional and security requirements and their implementation we shall see that several security issues are addressed in this model [7]. Some ideas of this paper were patented in 2010 and the patent was approved in 2014. See [8].

This paper is organized into five sections, Section 2 describes functional and security requirements for our model, together with some relevant background of traditional secret sharing. Section 3 describes the scheme components in detail along with the security and functionality concerns which exist. Section 4 describes an enhancement of the scheme for countering dictionary attack. Finally we conclude the paper in Section 5.

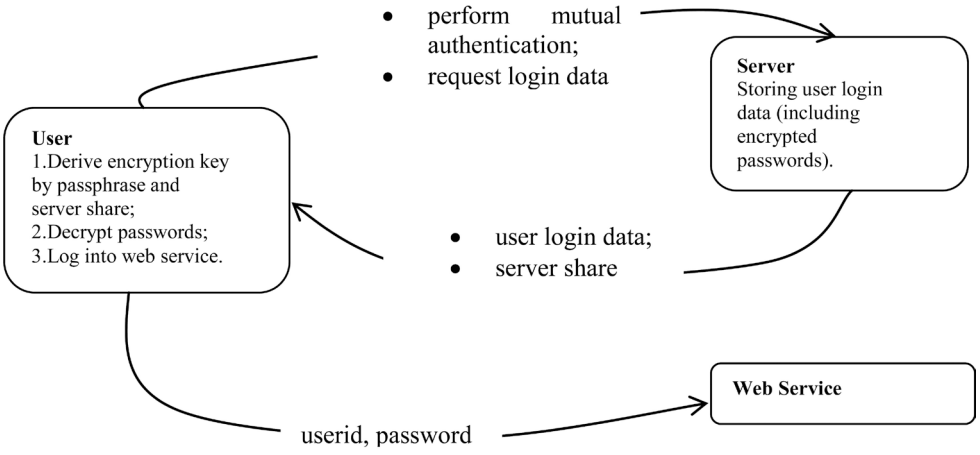

Figure 1. Model of split key.

2. Secret Sharing, Functional and Security Requirements

2.1. Traditional Secret Sharing

Traditional secret sharing schemes, such as those by Shamir [9] and Brickell [10] provide some motivation of our scheme. We demonstrate Shamir's scheme adapted to our particular case in this subsection. Suppose two parties Alice and Bob wish to split their secret K into two shares. We assume that the party Alice acts as the distributor by creating the two shares and sending one to the party Bob. We will represent K as an element of the finite field $\mathbb{F}_p$ where p is a prime sufficiently large. The following algorithm shows

Input	Secret $K \in \mathbb{F}_p$
Output	Public values x_1 for Alice and x_2 for Bob, private shares y_1 for Alice and y_2 for Bob

- Alice chooses nonzero elements $x_1, x_2 \in \mathbb{F}_p$ with $x_1 \neq x_2$
- Alice chooses a nonzero secret element $a \in \mathbb{F}_p$
- Alice computes

$$y_1 = ax_1 + K$$

$$y_2 = ax_2 + K$$

- Alice keeps y_1 and sends y_2 to Bob. The values x_1 and x_2 are made public

In our scenario, y_1 is kept by the user while y_2 is sent to the server. If the server sends y_2 back to the user, the user may simply solve for a by calculating

$$a = \frac{y_1 - y_2}{x_1 - x_2},$$

and get the secret key K.

The problem with this approach as with most traditional secret sharing schemes is that the shares are calculated instead of chosen. In our case, a different approach for creating shares is needed. More precisely, we require that the user's share is essentially chosen by the user in the sense that the user selects his/her own passphrase from which the share is generated. In the second part of this section, we shall list the function a land security requirements of our scheme which guided us in the design of the split key algorithm.

2.2. Functional and Security Requirements

According to our model, a user stores an encrypted password (which is associated with a web service) in the server. The encryption key is converted to shares for both user and server.

The specific requirements for the encryption key and passphrase are

(a-1)	Only the user may reconstitute the key
(a-2)	The encryption key will be split into shares between the user and the server
(a-2-i)	The user's share will be derived solely from a passphrase of the user's choosing
(a-2-ii)	The server's share will be derived from the user's share and the encryption key
(a-3)	The encryption key can only be reconstituted from both the user and server shares together
(a-4)	Compromise of the passphrase will not compromise future encryptions of the passwords

The functional and security requirements for dealing with the passwords are

(b-1)	Passwords will be allowed to have arbitrary length
(b-2)	Passwords should be bound to the web service with which they will be used
(b-3)	The passwords will reside encrypted on the server
(b-4)	Only the user will be able to encrypt/decrypt passwords
(b-5)	The encryption should allow multiple passwords to be associated to a web service
(b-6)	Passwords will be encrypted so that compromise of password will not affect future encryptions of new password

We will highlight how the split-key algorithm satisfies these requirements as it is described in the later sections.

3. The Split Key Algorithm

There are several different artifacts which are involved in the split-key scheme. These are listed below:

K	the encryption key
R	the user's selected ASCII passphrase of arbitrary length
P	the ASCII password to be encrypted
U	the URL or other ASCII identifier for the web service
C	the encrypted password
(d_1, d_2)	the user's share of the split encryption key (derived from R)
(x_1, x_2)	the server's share of the split encryption key (derived from K and (d_1, d_2))

The scheme will also have access to standard cryptographic algorithms such as a cryptographic hash function and a symmetric encryption algorithm. To simplify the explanation, we shall use SHA-512 and AES-256 in our algorithm description. We remark that other suitable functions can be used in their place.

3.1. System Setup

This subsection describes the steps required to set up the user and the server with the information needed to derive the requisite data. Prior to setup, the user will have to:

1) Generate a symmetric encryption key K;
2) Select a passphrase R.

For simplicity, we will take the key K to be a 256-bit AES key. In practice, this can be generated by the system which the user is using and be completely transparent to the user. The passphrase R should be selected with sufficient entropy so as not to be easily guessed.

3.2. Deriving the User's Encryption Key Share

In this subsection, we describe the process of creation of the user's share of the encryption key. We note that this depends only upon the user's passphrase.

Input: The user's passphrase R which is m bytes
Output: The user's encryption key share (d_1, d_2) where d_1 and d_2 are 256 bits each

- Represent R as $R = r_1 \| r_2$ where r_1 consists of the first $\left\lfloor \frac{m}{2} \right\rfloor$ bytes of R and r_2 consists of the rightmost $\left\lceil \frac{m}{2} \right\rceil$ bytes of R
- Let g_1 = SHA-512(r_1) and g_2 = SHA-512(r_2)
- Represent g_1 and g_2 as $g_1 = e_1 \| f_1$ and $g_2 = e_2 \| f_2$ where e_1, e_2, f_1, f_2 are each 256 bits
- Set $d_1 = e_1 \oplus e_2$ and $d_2 = f_1 \oplus f_2$
- If either d_1 or d_2 is $00\cdots0$ or $11\cdots1$, ask user to input new passphrase and start the process again
- Output (d_1, d_2)

Given the pseudorandomness properties of cryptographic hash functions, it is highly unlikely that either d_1 or d_2 will be $00\cdots0$ or $11\cdots1$.

We note that this step enables Requirement (a-2-i) to be satisfied. Since the user's passphrase is known by the user, the user's share does not need to be stored anywhere. The passphrase itself may be viewed as a long sentence which is easily remembered by the user but difficult for anyone else to guess correctly. For example, consider the sentence

The quick brown fox jumps over the lazy dog

SHA-512 is applied to this message so that any change to the message will result in an unpredictable change

in the output. Note, however, that the hash is used to generate both user share elements d_1 and d_2. Naively, one possibility of deriving this pair would be to split the passphrase into two parts (as is presently done) and use the first half to generate d_1 while the second half will generate d_2. The problem with this approach is that if the user changes their passphrase to something like

The quick brown fox jumps over the lazy cat

In which case, the private key element d_2 would change, but the element d_1 would remain the same. This could lead to a potential attack on the encryption key. To protect against this, notice that in our construction, the two pairs (e_1, e_2) and (f_1, f_2) use outputs from both halves of the hash output.

Another comment we wish to make is about the process of separating the passphrase in halves and hashing them independently. This is useful in preventing the user from choosing passphrases with undesirable patterns. For example, a passphrase of the form $R = r \| r$ will produce $d_1 = d_2 = 00\cdots0$, hence would be rejected. If we had simply set d_1 and d_2 to be the first and last 256 bits of SHA-512(*R*) respectively, then such an issue would not have been detected.

3.3. Deriving the Server's Key Share

Since the encryption is independently generated and the user's share is generated from the user-input passphrase, the server's share would need to be derived from these two pieces. The user performs the following task and sends the output to the server.

Input	The user's private key pair (d_1, d_2), the 256-bit encryption key *K*, pseudorandom number *S* of 256-bit
Output	The server's key share pair (x_1, x_2) where both x_1 and x_2 are 512-bit integers

- Calculate $x_1 = d_1K + S$
- Calculate $x_2 = d_2S + K$
- Output (x_1, x_2)

This derivation is simply the matrix multiplication

$$\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} d_1 & 1 \\ 1 & d_2 \end{pmatrix} \begin{pmatrix} K \\ S \end{pmatrix}$$

The pseudorandom number *S* is no longer needed after calculation of (x_1, x_2) and can be discarded.

Lemma 1. *Both* x_1 *and* x_2 *are* 512*-bit integers.*

Proof. Since d_1, K, and *S* are 256-bit integers, each of the $m \le 2^{256} - 1$. Then

$$\begin{aligned} d_1K + S &\le \left(2^{256} - 1\right)\left(2^{256} - 1\right) + 2^{256} - 1 \\ &= 2^{512} - 2 \times 2^{256} + 1 + 2^{256} - 1 \\ &= 2^{512} - 2^{256} < 2^{512} \end{aligned}$$

which shows that x_1 is a 512-bit integer. The analogous estimate holds for x_2. □

Note that this step enables Requirements (a-2-ii) and (a-3).

***Remarks*:**

1) When the user first registers with the server and selects its passphrase, the server key share can be generated by the user and sent to the server. This is done before any passwords are entered. If the user wishes to change their passphrase, a new encryption key *K* and the corresponding random number *S* should be generated.
2) In this model, the user encrypts and decrypts passwords to itself. Hence, there is only one entity involved with the encryption process. Thus, issues such as key management do not occur nor is there a need for public-key cryptography.
3) We should also remark that the pseudorandom number *S*, generated by the user, cannot be revealed to the server. Otherwise the server would be able to get the key *K* by factorizing the 512-bit integer $x_1 - S$. On average, this is a computationally feasible task.

3.4. Deriving the Encryption Key

In order to decrypt a password, the user will need access to the encryption key. The user knows its passphrase and hence, can derive its user key share. The server key share would need to be retrieved from the server.

Input: The user's key share (d_1, d_2), the server's key share (x_1, x_2)
Output: The 256-bit encryption key K

- Calculate $u = d_2 x_1 - x_2$
- Calculate $v = d_1 d_2 - 1$
- Output $K = \frac{u}{v}$

This operation is simply the matrix multiplication

$$\begin{pmatrix} K \\ S \end{pmatrix} = \frac{1}{d_1 d_2 - 1} \begin{pmatrix} d_2 & -1 \\ -1 & d_1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$$

Note the 2×2 matrix in this formula is the inverse of the matrix used from deriving the server key share. Since the value S is not needed, the second value needs not be calculated.

Note that this operation occurs on the user side since it is the only entity which knows the user share. This enables Requirement (a-1). Requirement (a-3) is also satisfied by this step.

***Remarks*:**

1) Over time, a user may wish to change their passphrase. If this were to happen, the encryption key K and salt S would need to be changed as well.

Suppose the same K and S values were to be used for a different user share $\left(\widehat{d_1}, \widehat{d_2}\right)$. Then, there would be new server share values

$$\widehat{x_1} = \widehat{d_1} K + S$$

$$\widehat{x_2} = \widehat{d_2} S + K$$

We see that $x_1 - \widehat{x_1} = \left(d_1 - \widehat{d_1}\right) K$. Hence, the secret key K is a factor of the 512-bit number $x_1 - \widehat{x_1}$. Factoring this number and checking the small number of factors would reveal K to the server.

Thus it is recommended to change K and S along with (d_1, d_2).

2) We also remark that the user should not just update the key K (or the salt S) alone. Indeed, suppose that S is changed to $\hat{S}$ but the other values remain the same. Then we would again get

$$\widehat{x_1} = d_1 K + \hat{S}$$

$$\widehat{x_2} = d_2 \hat{S} + K$$

Observe that

$$d_2 = \frac{x_2 - \widehat{x_2}}{x_1 - \widehat{x_1}}$$

This would give away part of the user's share to the server. Furthermore, since $K \equiv x_2 \pmod{d_2}$, this would narrow down the possibilities for K to a small set. A similar argument will show that a weakness is introduced if only K is changed.

This practice enables Requirement (a-4) to be satisfied.

3.5. Encrypting Passwords

When the user registers with a web service, it will obtain a URL, username, and set of passwords for that web

service. When the user wishes to store the password for the web service on the server, it must first encrypt them with the encryption key before transmitting them.

Input The 256-bit encryption key K; the URL U; the password P; a 256-bit nonce E; and the 32-bit counter i for indexing the password
Output The password encryption C

- Pad P with zero bits as necessary to make it $16r$ bytes
- Represent P as $P = P_1 \| P_2 \| \cdots \| P_r$ where each P_k has 128 bits
- For j from 1 to r do
  Concatenate the string $W_j = U \| E \| i \| j$ where j is of 32 bits
  Compute H_j = SHA-512(W_j)
  Write H_j as $H_j = e_j \| f_j \| g_j \| h_j$ where each substring is of 128 bits
  Let $N_j = e_j \oplus f_j \oplus g_j \oplus h_j$
  Let $C_j = P_j \oplus \mathrm{AES}_K(N_j)$
- Concatenate and output the string $C = C_1 \| C_2 \| \cdots \| C_r$

In addition to the password and the encryption key, the encryption mechanism makes use of other data such as the URL, a nonce, and counters. Let us describe how each of these inputs affects the encryption.

Binding URL's to Passwords. The URL U is presented within the hash value in order to bind each password to the URL on which it is used. Failure to do so would open up an implementation to a substitution attack. Consider the scenario where the URL is not bound in this way. Suppose a user has two passwords P_1 and P_2 which are to be used at the URL's U_1 and U_2 respectively. Suppose further that U_1 has high security associated with its passwords whereas U_2 has little or no security for passwords. Suppose a user wishes to decrypt both passwords by requesting them at the same time. An attacker may simply switch the URL's or the encrypted passwords in transit. In this case, the high security password would be sent to the low-security site. This is a potential breach of password security. It would certainly result in a breach if the second site has been maliciously constructed for this purpose. If the password encryption is bound to the URL, however, switching the sites would result in an incorrect decryption. Including the URL in the encryption enables Requirement (b-2) to be satisfied.

Nonces for Passwords. The nonce E used for password encryption is simply a string which is changed every time the user changes their password for the associated URL. Consider the following scenario. Suppose a user's password for a particular site is compromised in that the password$P$ and ciphertext C are revealed to another entity. In the real world, this may happen if the user loans their account to a friend for a short period of time. The other entity would then be able to determine $\mathrm{AES}_K(N_j)$ for each j as in the last step of the password encryption and decryption algorithms. Suppose now that the user changes their password but does not change E. Since the URL U does not change, nor do the counters i and j, the value for N_j would be the same, thus, so would $\mathrm{AES}_K(N_j)$. If the entity obtains C_i for all i, then they will be able to calculate the password P. This will happen regardless of what the user changes their password to. Hence, the value E corresponding to a password should be changed every time that particular password is changed. This enables Requirement (b-6) to be satisfied. Here we shall discuss an issue that is related to the Requirement (b-6). In the case that the client is compromised and a web service password is obtained by an attacker, the attacker is not able to get the encrypted version of the password and the server's share from the server side unless the communication credential between client and server has been leaked to the attacker. This is because the connection between client and server is supposed to be protected by other authentication services such as TLS/SSL.

Counters in Password Encryption and Decryption. A web service may have several different passwords associated to it. This is particularly the case during multi-stage login. In addition, some passwords may be longer than 16 characters, and hence, consist of multiple blocks. In practice, passwords are usually ASCII characters which have a specific encoding as bytes. If the counters were not used, the same value $\mathrm{AES}_K(N_j)$ would be xor-ed to each separate block for a web service. This could lead to potential correlation attacks to deduce the password. In general, the encryption is designed so that each x or value $\mathrm{AES}_K(N_j)$ is different for each block. Note that these counters enable Requirements (b-1) and (b-5) to be satisfied.

3.6. Decrypting Passwords

When the user wishes to access a web service, it will need the appropriate username and password. To get these values, it would first need to derive the encryption key from the passphrase and the server share. Once the encryption key is ready, the encrypted passwords can be sent from server to user for decryption and forwarding.

Input	The 256-bit encryption key K; the URL U; the encrypted password C; a 256-bit nonce E; and the 32-bit counter i for indexing the password
Output	The plaintext password P

- C has input size $16r$ bytes
- Represent C as $C = C_1 \| C_2 \| \cdots \| C_r$ where each C_k has 128 bits
- For j from 1 to r do

  Concatenate the string $R_j = U \| E \| i \| j$ where j is of 32 bits

  Compute H_j = SHA-512(R_j)

  Write H_j as $H_j = e_j \| f_j \| g_j \| h_j$ where each substring is of 128 bits

  Let $N_j = e_j \oplus f_j \oplus g_j \oplus h_j$

  Let $P_j = C_j \oplus \mathrm{AES}_K(N_j)$
- Remove any padding bytes from P_r
- Concatenate and output the string $P = P_1 \| P_2 \| \cdots \| P_r$

Decryption is very similar to encryption. The same x or value is generated. The only difference is that the encrypted value is xor-ed to obtain the clear text value. The way that decryption is structured, only the possessor of the encryption key, the user, can decrypt. This enables Requirements (b-3) and (b-4) to be satisfied. Removing any padding bytes from the last block P_r is easy since they are bytes consisting of a string of 0 and are not ASCII characters.

4. Further Enhancement

In this section, we shall present an enhancement of the split-key scheme for implementing a broader range of security features. More specifically the issue of dictionary attacks is considered in our modification.

An Extension to Counter Dictionary Attack

Using passphrases may appear to be stronger than using passwords. However, dictionary attacks can be applied to passphrases as well, as shown in [11]. In this subsection, we indicate an enhanced version of the split-key scheme by introducing an iterated hash operation to protect the scheme against a dictionary attack.

In order to prevent a dictionary attack to discover d_1 and d_2, we introduce the following variation on this scheme. As before, we generate r_1 and r_2 from R. We introduce a salt B which we write as $B_1 \| B_2$. The salt should be stored on the server side. This will reduce the chance for an attacker to obtain both the user's share (only used on the client side) and the salt.

Next we compute two sequences $\{e_i\}$ and $\{f_i\}$ as follows. Set

$$e_0 = \text{SHA-512}(r_1 \| B_1) \quad \text{and} \quad f_0 = \text{SHA-512}(r_2 \| B_2)$$

Then for $i \geq 1$, recursively define

$$e_i = \text{SHA-512}(r_1 \| B_1 \| e_{i-1}) \quad \text{and} \quad f_i = \text{SHA-512}(r_2 \| B_2 \| f_{i-1})$$

For a sufficiently large value of N, set

$$E = e_N \quad \text{and} \quad F = f_N .$$

In practice, we choose N of the order 1000 but it can be chosen much higher.

Now splitting each of E and F into equal segments $E = E_1 \| E_2$ and $F = F_1 \| F_2$, we set

$$d_i = E_i \oplus F_i$$

for $i = 1, 2$. We then generate x_1 and x_2 as before. The addition of the iterated hash step serves the purpose of introducing sufficient entropy into d_1 and d_2, thus making a dictionary attack less feasible.

5. Conclusion

We have described a scheme for password management by storing password encryptions on a server. The method involves having the encryption key split into a share for the user and one for the server. The user's share depends solely on a selected passphrase. The server's share is generated from the user's share and the encryption key. Only the user is able to recover the encryption key and to encrypt/decrypt the cipher text of a password. A further enhancement of the scheme is also proposed by providing a feature for countering dictionary attacks.

References

[1] Florêncio, D. and Herley, C. (2007) A Large-Scale Study of Web Password Habits. *Proceedings of the* 16*th International Conference on World Wide Web*, Banff, May 2007, 657-666. http://dx.doi.org/10.1145/1242572.1242661

[2] Hayday, G. (2002) Security Nightmare: How Do You Maintain 21 Different Passwords? Silicon.com.

[3] (2016) Roboform Reference Manual. Siber Systems Inc.

[4] Zhao, R. and Yue, C. (2013) All Your Browser-Saved Passwords Could Belong to Us: A Security Analysis and Acloud-Based New Design. *Proceedings of the* 3*rd ACM Conference on Data and Application Security and Privacy*, San Antonio, February, 2013, 333-340. http://dx.doi.org/10.1145/2435349.2435397

[5] Silver, D., Jana, S., Boneh, D., Chen, E. and Jackson, C. (2014) Password Managers: Attacks and Defenses. 23rd *USENIX Security Symposium* (*USENIX Security* 14), San Diago, August 2014, 449-464.

[6] Li, Z., He, W., Akhawe, D. and Song, D. (2014) The Emperor's New Password Manager: Security Analysis Ofweb-Based Password Managers. 23rd *USENIX Security Symposium* (*USENIX Security* 14), San Diago, August 2014, 465-480.

[7] Haque, T., Wright, M. and Scielzo, S. (2013) A Study of User Password Strategy for Multiple Accounts. *Proceedings of the* 3rd *ACM Conference on Data and Application Security and Privacy*, 173-176. http://dx.doi.org/10.1145/2435349.2435373

[8] Giuliani, K. and Murty, V.K. (2014) Split key Secure Access System. U.S. Patent No. 8,892,881.

[9] Shamir, A. (1979) How to Share a Secret. *Communications of the ACM*, **22**, 612-613. http://dx.doi.org/10.1145/359168.359176

[10] Brickell, E.F. (1989) Some Ideal Secret Sharing Schemes. *Journal of Combinatorial Mathematics and Combinatorial Computing*, **9**, 105-113.

[11] Bonneau, J. and Shutova, E. (2012) Linguistic Properties of Multi-Word Passphrases. *Proceedings of the* 16*th International Conference on Financial Cryptography and Data Security*, Kralendijk, March, 2012, 1-12. http://dx.doi.org/10.1007/978-3-642-34638-5_1