

UNLIMITED LENGTH RANDOM PASSWORDS FOR EXPONENTIALLY INCREASED SECURITY

Chemana Shaik

VISH Consulting Services Inc, 6242 N Hoyne Avenue, Chicago IL 60659, USA

ABSTRACT

Presented herein is a new method of exponentially strengthening user defined passwords against cracking. The enhanced security is achieved by injecting random strings of random length at random positions in the password string before encrypting and passing the ciphertext resulting after encryption over a network to its destination. Discussed also in detail is how the randomly injected strings are separated and the original password is extracted from the ciphertext. Also explained is how the method can be applied to any other confidential information such as credit and debit card information and cryptocurrency data.

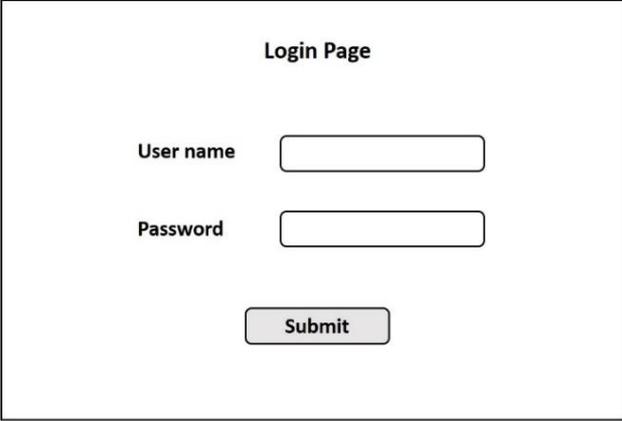
KEYWORDS

Padding, Random String Injection, Ciphertext, Delimiting String, Ciphertext Only Attack, Brute Force Attack, Security Factor

1. INTRODUCTION

Password is the basic and first means to control access to a web application that hosts any confidential data and resources. Passwords passed over a network in plain text are vulnerable to eavesdropping, thereby leading to hacking of websites and theft of confidential information[1]. In order to prevent this, passwords are encrypted or hashed before passing them over any network.

Fig. 1 below shows a login page wherein a web application user enters his username and password.



The diagram illustrates a traditional user login page. It features a central rectangular frame containing the following elements: a title 'Login Page' at the top center; a label 'User name' followed by a horizontal input field; a label 'Password' followed by a horizontal input field; and a 'Submit' button centered below the input fields.

Fig. 1 A traditional user login page

Passwords can be encrypted using a public key of any asymmetric encryption algorithm such as RSA, ECC. Today, most web applications mandate a minimum length of eight characters for

password when a user signs up, in order to guarantee minimum security against brute force attacks^[2]. Passwords too longer in size are difficult to remember and therefore can be forgotten. However, when an eight-character password is encrypted as is, it is highly vulnerable to ciphertext attacks and cracking would be very easy for attackers.

In ciphertext only attack, an attacker launches a brute force attack on the password trying all possible combinations of an eight-character string. In order to defeat this kind of attacks, password string is padded with random text to make it the size of the encrypting key to make the attack difficult, because the attacker needs to expand his brute force attack over to the padding string also which is multiple time larger than the original password.

Once the encrypted padded password reaches its destination, it is decrypted on server using the private key, the padding string is removed and the original password is extracted for verification. However, padding has its limitations as a password can be padded only up to the length of the encrypting key, and padding beyond this length is of no use as it can be ignored in brute force attacks^[3]. Fig.2 below shows how a password is padded before encrypting it.

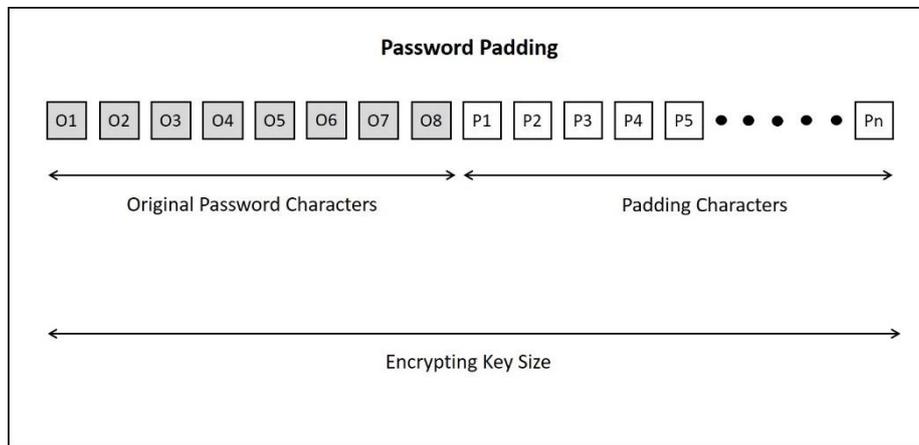


Fig. 2 Password padding

In the above figure O1, O2, ..., O8 are characters of the original user defined password. P1, P2, ..., Pn are random padding characters that are created by a typical public key encryption scheme that we use today. The total padded password string is encrypted into a ciphertext with the public key of the corresponding web application and submitted as soon as the user clicks the submit button. On the server side, the web application decrypts the ciphertext and derives the padded password string O1, O2, ..., O8, P1, P2, ..., Pn from which the padding string P1, P2, ..., Pn is removed to extract the original password.

2. RELATED WORK

In 1994, Bellare and Rogaway introduced Optimal Asymmetric Encryption Padding (OAEP) scheme that results into a probabilistic encryption, meaning encryption of a plain text message encrypted multiple times will result in different ciphertexts, which defeats chosen plain text attacks and chosen ciphertext attacks. In OAEP method, a random value R is hashed and the result is xor-ed with the zero padded input message. The resulting value S is hashed with another function and the result is xor-ed with R to obtain T. Subsequently, S and T are concatenated and the result is encrypted^[4].

In 2001, Dan Boneh proposed the SAEP (Simplified OAEP) padding scheme which allows to drop off one round of OAEP without compromising security. In SEAP method, a random value R is hashed and the result is xor-ed with input message padded with certain zero bits and the resulting value S is concatenated with the random value R and crypted into a ciphertext ^[5].

In 2002, Bellare et al presented several security fixes to the SSH authenticated encryption mechanism that defeat reaction attacks. Bellare suggested using random padding in CBC mode of encryption.

In 2012, Liu Chengxia discussed a new padding method in DES encryption which solves the ambiguity of padding zeros with the ending zeros of the actual message. Liu solves the problem by specifying the length of padding zeros in the last eight bits of the 64 bit block ^[6].

Later in 2012, Gilles Bartheet al proposed the ZAEP (Zero-Redundancy Asymmetric Padding) scheme that can defeat chosen ciphertext attacks ^[8]. ZAEP is surprisingly much simpler compared to the OAEP^[7].

In 2019, Prabavathiet al presented a Prime Padding Attribute based encryption to improve dataset security for publicly centralized cloud systems ^[8].

All the above works focus on padding and different variations of padding. Padding is always applied at the end of an original message. No evidences are found in literature teaching injecting random strings of random length at random locations in the original message.

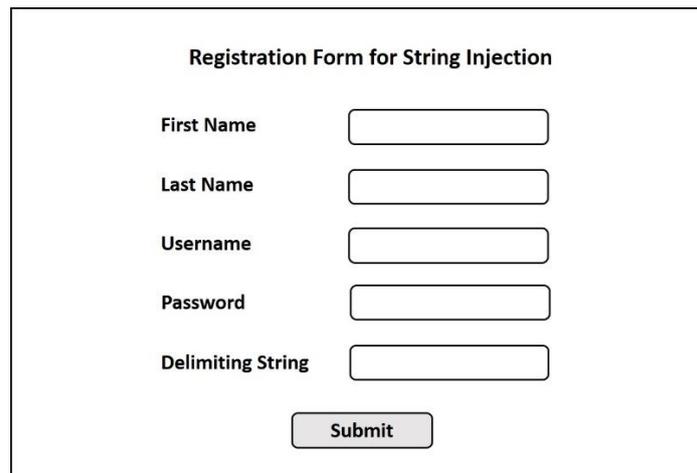
3. INTERMEDIARY INJECTION OF RANDOM STRINGS

Intermediary injection of random strings in a password at random position exponentially increases security against brute force attacks. When random strings of random length are injected in a password, original characters of the password are scattered all over the resulting password. Further, the resulting password can occupy multiple blocks of the encrypting key size. As a result, the attacker would not be in a position to ignore the remaining blocks of the ciphertext following the first block.

4. DIFFERENTIATING PASSWORD AND RANDOM STRINGS

The original password characters and the random injected strings can be differentiated by separating them with small delimiter strings of one or more characters specific to a particular user. Every user can select a delimiter string of his choice at the time of registration with the web application. The random injected string follows and is followed by the delimiter string of the user. The entire string resulting from injection of random strings in the password is encrypted by a public key and a ciphertext is generated which is passed over a network to its destination.

Fig. 3 below shows a registration form of a web application that implements unlimited length random passwords. The registration form contains an additional field named Delimiting String wherein the user enters one or more symbol characters from ~!@#%&^* found on the second row of buttons on a regular keyboard. A user has to remember this delimiter string along with his username and password and enter it in the login form at the time of Sign In.

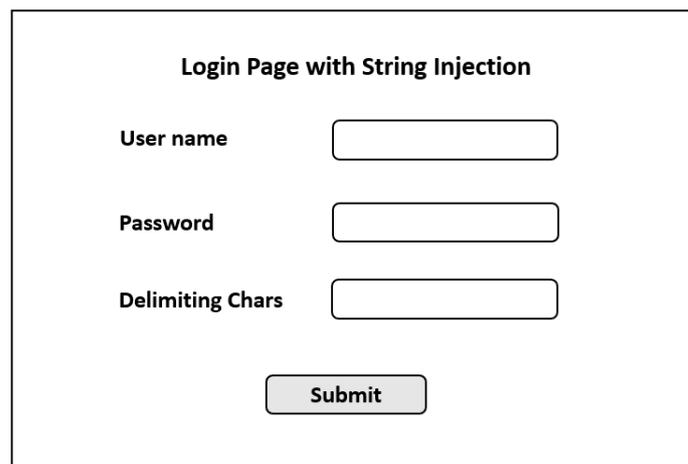


The image shows a registration form titled "Registration Form for String Injection". It contains five input fields: "First Name", "Last Name", "Username", "Password", and "Delimiting String". Each field is represented by a rectangular box. Below the input fields is a "Submit" button.

Fig. 3 A registration form on a website that implements random string injected passwords

One condition that the software code of the login page needs to verify is that it should check for occurrence of the delimiting string in the password that a user enters in the password box. The software program should alert the user not to include his delimiter string in his password.

Fig. 4 below shows a login page implementing random string injection in password.



The image shows a login form titled "Login Page with String Injection". It contains three input fields: "User name", "Password", and "Delimiting Chars". Each field is represented by a rectangular box. Below the input fields is a "Submit" button.

Fig. 4 A login page implementing random string injected password

During login a user enters his username, password and the delimiter characters he defined at the time of registration. Typically, a user could select two-character string "\$@" as his delimiter string. After entering these three values in the login form, the user clicks the submit button which will trigger a client-side scripting function that injects a random string of random length pre- and post-appended by the delimiter string at random positions in the password. Greater the length of the random strings injected, greater the security achieved against cracking. The random strings injected need not be the same size and in fact varying the length of random strings makes password cracking much more difficult. Once the random string injection process is complete, the randomized password string is encrypted with public key of the web application.

In an alternative approach to entering delimiter string in the login form, it can be fetched from the server and stored in a hidden field of the form. In this case the user enters only his username and

hits a Continue button on the login form which will submit the username to the web application, receive the user's delimiter string and store it in a hidden field of the form. At the same time, a password box hidden in the form is made visible. All this task can be performed in background without any page refresh, through an AJAX call. Once the user enters his password and clicks submit button, the delimiter string is picked by a client-side scripting function and random string injection is performed.

Fig. 5 below shows how random strings are injected in an original password before encryption.

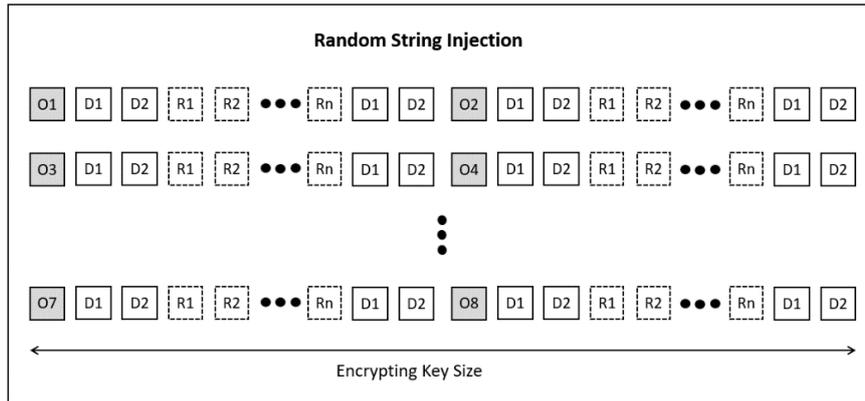


Fig. 5 A password string after random string injection

In the above figure O1, O2, ..., O8 are characters of the original user defined password whereas D1 and D2 are characters of the delimiter string that the user entered in the third input field of Fig. 3. R1, R2, ..., Rn are characters of the random string injected in the password. Multiple distinct sets of random characters are injected in the password as shown in the figure. These set are random in nature whereas D1 and D2 are always fixed for a given user.

When ciphertext of the random injected password reaches its destination server, the web application decrypts it with the corresponding private key. Subsequently, the random injected strings are identified by spotting the delimiter strings D1D2 and removed to extract the original password.

Fig. 6 below shows how random injected strings are identified and removed from the decrypted password.

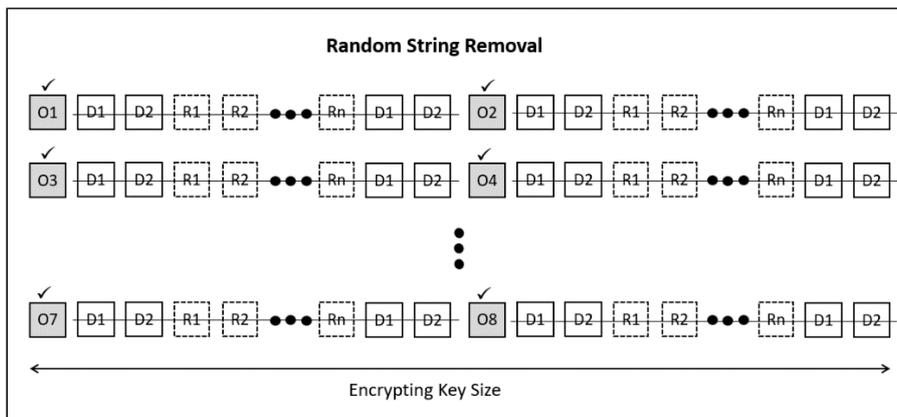


Fig. 6 Removing random injected strings and extracting original password

As shown in the above figure, the web application decrypts the ciphertext it received from the user, identifies the delimiter strings in the resulting plain text and removes them including the random string between them. The delimiter string for that particular user is retrieved from the user repository based on the username.

5. SECURITY AGAINST CIPHERTEXT ONLY ATTACKS (COA)

In a COA attack an attacker tries to discover the original plain text for the available ciphertext. Usually, hackers who have gained access to network traffic at intermediary routers on the Internet resort to such attacks. The attack is launched on public key encrypted ciphertexts. In this attack an attacker tries to encrypt each possible plain text and encrypt it with the public key and compare the resulting ciphertext with the available ciphertext, and the plain text is discovered whenever a match occurs. It is a kind of brute force attack on the plain text which is typically a password.

Random string injected passwords make their encryption exponentially stronger against COA attacks compared to the original password encryptions. The plain text combinations to be covered by an attacker in COA attacks will raise exponentially due to the insertion of random strings in the middle.

6. SECURITY FACTORS OF RANDOM INJECTED PASSWORDS

Random string injected password ciphertext received on a web server contains multiple blocks of the encrypting key size. The actual length of the password ciphertext received depends upon the size of random strings injected in the password before encryption. Larger the size of the total injected string, larger the size of password ciphertext. For example, a 1024 character random string injected in a password provides security equivalent to that of an 8192 bit public key encryption which is roughly four times the current NIST standard for RSA encryption. Today, achieving security of this level in real time is hardly possible with the existing computer processors. The current industry standard for public key encryption is 2048 bit RSA key.

When a 40-character random string is injected after every character of an eight character password, total number of characters in the complete plain string before encryption would be $8(40+2+1)$ which is equal to 344. When converted to a binary number it is 2752 bits long as each ASCII character requires eight bits in binary representation. Therefore, the attacker needs to try as many as 2^{2752} total number of combinations.

On the other hand, when a padded password without any random strings injected is encrypted with a 2048 RSA key, the ciphertext generated would be 2048 bits long and it would require the attacker to try 2^{2048} combinations, including padding, to crack the password.

$$\text{security factor} = 2^{2752} - 2^{2048} = 2^{704} = (10^{0.301})^{704} = 10^{212}$$

The above security factor indicates that encryption of a password with a 40-character random string injection with 2048-bit RSA key is 1×10^{212} times stronger compared to simple padded password encryptions with the same size key.

The following table provides hardness factors of cracking an eight-character password encrypted with 2048 bit key wherein random strings are injected before encryption. Hardness factors are tabulated for injected strings of different sizes.

Table. 1 Security Factors of an eight-character password with random string injection

Security Factors for different random string sizes and single character delimiter string with 2048-bit RSA Key		
Random String Size (in chars)	Security Factor Computation	Security Factor (Simplified)
40	$1 \times 10^{0.301 * [8 * 8 * (40 + 2 + 1) - 2048]}$	1×10^{212}
45	$1 \times 10^{0.301 * [8 * 8 * (45 + 2 + 1) - 2048]}$	1×10^{308}
50	$1 \times 10^{0.301 * [8 * 8 * (50 + 2 + 1) - 2048]}$	1×10^{405}
55	$1 \times 10^{0.301 * [8 * 8 * (55 + 2 + 1) - 2048]}$	1×10^{501}
60	$1 \times 10^{0.301 * [8 * 8 * (60 + 2 + 1) - 2048]}$	1×10^{597}

Security factors tabulated above are exponential figures and therefore guarantee enormous security to passwords. The security factor 1×10^{212} against 40 in the table implies that if it takes one day to crack a padded password using a super computer, it would take 1×10^{212} years to crack the same password on the same super computer if injected with random strings of 40 characters before encryption.

The above security factors are computed for a single character delimiter string. The length of delimiter string may vary user to user which, if considered, would result in even higher security factors. Another assumption made in the computations is that size of random string injected after each character of the password is constant, which need not be true in a real implementation. When the injected random string size varies in the implementing program, it would make the attack much more difficult.

7. CONDITIONS TO CHECK IN IMPLEMENTATION

The software implementation program needs to check the certain condition while injecting random string in password in order to make it strong and fail proof. Following are some of the condition to verify:

- Do not include delimiter string in password
this will avoid misidentification of delimiter string during removal of random strings.
- Do not include delimiter string in random string
this will ensure complete removal of random strings injected which would otherwise may leave part of the random string
- Adjust the length of the last block password resulting after random string injection to key size
this will make sure the last block of ciphertext is not vulnerable to easy cracking due to a short length of its plain text

8. APPLICABILITY TO CARDS AND CRYPTOCURRENCY DATA

Random strings can be injected in other secret data such as credit and debit card details, social security numbers and cryptocurrency details. Today, credit and debit card numbers are sixteen digits long. Inserting random strings after each digit of these numbers will exponentially increase security against brute force attacks.

Credit and debit cards details and cryptocurrency data are not hashed on client side as they are stored encrypted on server side and hence random string injection method may be applied directly on such data.

9. RANDOM INJECTING STRINGS IN PASSWORD HASH

Nowadays most servers store passwords in hash form in order to avoid theft by internal trust breaching elements. While a hash is irreversible it is also vulnerable to brute force attacks. The concept of encrypting secret data after injecting random strings can be added as an additional wrapper over hashing.

Once a hash of secret data is computed on client side, random strings can be inserted in the hash and further encrypted. The ciphertext of the hash generated after encryption can be passed over to the web application. The web application on server side can decrypt the ciphertext to produce the random string injected hash from which random strings can be removed to obtain the original hash.

10. APPLICABILITY TO SYMMETRIC, ASYMMETRIC AND HYBRID ENCRYPTIONS

Random string injection concept works well with both symmetric encryption schemes such as AES and asymmetric encryption schemes such as RSA and ECC. These schemes complement each other to make encryption more efficient and faster. In a hybrid scheme wherein both the schemes are used, a private key of a symmetric encryption scheme is generated, encrypted by a public key of asymmetric key and passed to the client. All subsequent communication is encrypted with the shared symmetric key for the entire session.

In case the server passes an encrypted symmetric key to its client, it needs to send the delimiter string used in random string injection as an additional parameter with the ciphertext. The client's software program can identify the random strings and remove them to extract the original symmetric key.

11. PRACTICAL IMPLEMENTATION

Random string injection in plain text before encryption may be implemented as part of the standard protocols such as the Transport Layer Security (TLS) or any encryption schemes such as RSA, ECC and AES. These implementations may use the same delimiter string irrespective of users. Alternatively, it can also be implemented by web applications that provide a login page for users. A separate delimiter string may be defined per user at the time of registration and asking the user to enter his delimiter string in the login form. Random string injection may be implemented in the login page using a browser side scripting language such as JavaScript which will execute before the form is submitted for TLS encryption.

12. CONCLUSION

Passwords are shorter in size compared the standard size of blocks used by block cipher encryption schemes. Strong encryption of passwords is mandatory to defeat password stealing by hackers. Passwords encrypted as is are vulnerable to simple brute force attacks. Padding passwords at the end is a conventional approach adopted by all most all encryption schemes.

Raising above the convention of padding passwords for security, a new method of exponentially fortifying the security of passwords against brute force attacks is proposed in this paper. The proposed method injects random strings of random length at random positions in the password

before encryption. A delimiter string is concatenated before and after the random strings injected in order to identify them at the destination end. The random strings injected are identified by the delimiter string and removed to obtain the original passwords string after decryption.

Random string injected passwords offer exponential security against brute force attacks and ciphertext attacks as the attacker needs to sift through a very wide space of strings. Security factors achieved on implementing the proposed method are reported. Also discussed is the applicability of the method to other secret information such as credit and debit card details, social security numbers and cryptocurrency data wallet passwords and seed phrases. Further, explained as to how the method can be applied to password hash in order to make the hash more secured against brute force attacks.

A future work recommendation is that the random string injection scheme may be developed as an easily pluggable module of code both for browser side injection of random strings and server-side removal of the injected string. Another recommendation is that the teams working with the TLS protocol enhance it to include random string injection on client side and its removal on server side.

REFERENCES

- [1] Kimberly Rallo, "Clear Text Password Risk Assessment Documentation", <https://www.sans.org/reading-room/whitepapers/authentication/clear-text-password-risk-assessment-documentation-113>
- [2] University of Cincinnati web page, "How to Choose a Password", <https://www.uc.edu/infosec/password/choosepassword.html>
- [3] Townsend Security Data Privacy Blog, "How Much Data Can You Encrypt With RSA Keys?", <https://info.townsendsecurity.com/bid/29195/how-much-data-can-you-encrypt-with-rsa-keys>
- [4] M. Bellare, P. Rogaway. Optimal Asymmetric Encryption -- How to encrypt with RSA. Extended abstract in Advances in Cryptology - Eurocrypt '94 Proceedings, Lecture Notes in Computer Science Vol. 950, A. De Santis ed, Springer-Verlag, 1995
- [5] Dan Boneh, "Simplified OAEP for the RSA and Rabin Functions", Advances in Cryptology – CRYPTO 2001, volume 2139 of Lecture Notes in Computer Science, pages 275–291. Springer, 2001
- [6] Liu Chengxia, "Discussion of New Padding Method in DES Encryption", Journal of Software Engineering and Applications, 2012, 5, 20-22
- [7] Gilles Barthe, David Pointcheval, Santiago Zanella Béguelin "Verified Security of Redundancy-Free Encryption from Rabin and RSA", proceedings of the 19th ACM Conference on Computer and Communications Security, CCS 2012. ACM Press, 2012
- [8] D. Prabavathi and Dr. M. Prabakaran, "An Improving Cloud Data Security Standard Using Prime Padding Attribute Based Encryption With Supportive Service Level Dynamic Auditing in Cloud Environment", International Journal of Scientific Research and Review - Volume 8, Issue 4, 2019

Author

Chemana Shaik is a Research & Development professional in Computer Science and Information Technology for the last twenty years. He has been an inventor in these areas of technology with eight U.S Patents for his inventions in Cryptography, Password Security, Codeless Dynamic Websites, Text Generation in Foreign Languages, Anti-phishing Techniques and 3D Mouse for Computers. He is the pioneer of the Absolute Public Key Cryptography in 1999. He is well known for his Password Self Encryption Method which has earned him three U.S Patents. He has published research papers in IJCSSEA and the proceedings of EC2ND 2006 and CSC 2008.

