Network And Information Security (ITE4001)

Fall Semester 2020-21

Project Review 3

Title:

Preventing Shoulder Surfing Attacks with Dynamic Graphical Input System

Team Members

Udit Agarwal 18BIT0159

Arihant Saxena 18BIT0133

Shantanu Dixit 18BIT0306

Submitted To: Prof. Jasmine Norman

****

**ABSTRACT**

In computer security, shoulder surfing is a type of social engineering technique use to obtain information such as Personal Identification Number (PIN), login passwords and other confidential data by looking over the victim’s shoulder, either from keystrokes from a device or sensitive information being spoken and heard, also known as eavesdropping. This attack can be performed either at close range (by directly looking over the victim's shoulder) or from a longer range, for example by using a pair of [binoculars](https://en.wikipedia.org/wiki/Binoculars) or similar hardware. To implement this technique attackers do not require any technical skills; keen observation of victims' surroundings and the typing pattern is sufficient.

A user enter password based on the type of input system. If the input system is pattern based, then user enters a pattern to login. If the input system is code-word based than user enter a code word to login. In this project, we intend to completely remove the chances of any type of Shoulder surfing attack. We introduce a new algorithm, which is effortless to implement but very effective. The algorithm along with the way the information is stored and fetched from the database (without any encryption and hashing method) makes it so much secure, that even if a user tells the person his/her login password, he will never be able to log in to the system.

We will be forming a dynamic graphical input system i.e. A image button input system which will be shuffling every session, someone tries to login. This type of system accepts a pattern based input that is entered in a code word form. The algorithm is based on simple mathematics and can be easily extended to add more features.

For this project, system will be implemented in pythonv3.6, and any type of SQL database can be used in the system, such as MySQL, Oracle etc. We are using MySQL here.

**FEATURES:**

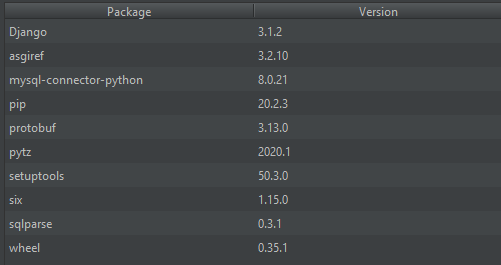
1. Completely safe from key-loggers, because what is recorded and what is viewed is completely different.

2. Since, every time a different pattern can be used from the set, attacker will record and enter a different wrong password every time.

3. Total combination that can be made is 16C4 = 1820 and each time 4! are possible, therefore total combinations = 1820 \* 24 = 43680. In development, we are providing 2 type of patterns, therefore total possibilities = 43680\*2 = 87360. Total possible actions brute force attack will take is 2^87360, which is huge. So practically, safe from brute forcing too.

4. Easy to implement and features can be extended, such as adding a CAPTCHA for additional security.

**EXTERNAL LIBRARIES USED**



**ALGORITHM**

REGISTRATION PHASE

🡪Input username (should be unique), email and password (should be 8 digit numerical password)

🡪Interlink the password

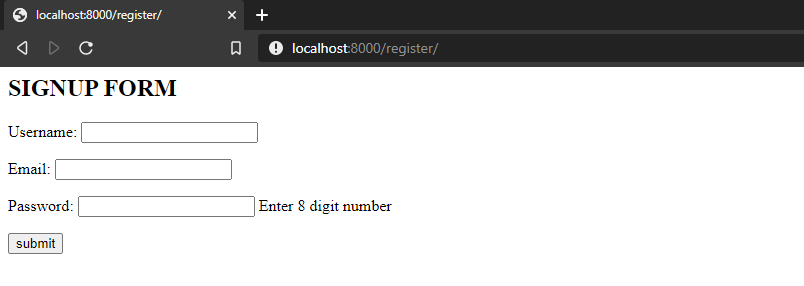
🡪Convert password into corresponding string. Below is the list of conversions:

[0-5: ‘a’, 6-11: ‘b’, 12-17: ‘c’, 18- 23: ‘d’, 24-29: ‘e’, 30-35: ‘f’, 36-41: ‘g’, 42-47: ‘h’, 48-53: ‘i’, 54-59: ‘j’, 60-65: ‘k’, 66-71: ‘l’, 72-77: ‘m’, 78-83: ‘n’, 84-91: ‘#’, 91-99: ‘\*’]

🡪Compute every permutation of the string

🡪Generate hash (md5) of each permutation

🡪Store the username, email, original password and each hash in the database.



**(Registration Page)**

**localhost:8000/register**

LOGIN PHASE

We are using three layer of verification in our security system.

First layer: The user will enter the Phone Number of the mobile device, which is registered, to the system. The Mobile device should be in sync with the system.

Second Layer: If the Phone number entered is correct, then a Question will be sent on the registered device. Various Tech Giant such as Google and IBM uses similar techniques these days. This way provides two additional sub level of security and an additional feature.

First level: Because only the genuine user will have the registered mobile device, we will be sure that , it is it real user not the attacker who is accessing the system.

Second level: In a case, the registered mobile device gets lost. In such situation, what a normal user does is deactivate the SIM card and device remotely. If that happens, The Authentication question will not be send on the device and the attacker will not be able to reach the final level of Login.

In that way, the system is protected at second Layer.

Additional Feature 🡪 The loophole in the Static Login system is that what if the attacker recognizes and remembers the pattern. The obvious solution is changing the pattern in each Login Session. That gives rise to another problem. We need to inform the user about that current session pattern. We cannot flash it directly onto the screen.

This problem is solved by the Second Layer above mentioned. When the user gets the authentication question on the registered mobile device, there will also be an additional piece of information telling the user about the pattern. The user remembers the pattern and enters the password in accordance to that.

Even if the user forgets the pattern, the whole information is there on the email send to him during the Registration Phase.

Third Layer.

The user will be asked to enter the details – Username and password through a unique input system. The user can enter any one password from the set as a code-word but actually, it is pattern. For demo purpose, we included only two patterns

|  |  |  |  |
| --- | --- | --- | --- |
| R1C1 (a) **0** | R1C2 (b) **1** | R1C3 (c) **2** | RC14 (d) **3** |
| R2C1 (e) **4** | R2C2 (f) **5** | R2C3 (g) **6** | R2C4 (h) **7** |
| R3C1 (i) **8** | R3C2 (j) **9** | R3C3 (k) **10** | R3C4 (l) **11** |
| R4C1 (m) **12** | R4C2 (n) **13** | R4C3 (#) **#** | R4C4(\*) **\*** |

(Horizontal Pattern)

|  |  |  |  |
| --- | --- | --- | --- |
| R1C1 (a) **0** | R1C2 (b) **4** | R1C3 (c) **8** | RC14 (d) **12** |
| R2C1 (e) **1** | R2C2 (f) **5** | R2C3 (g) **9** | R2C4 (h) **13** |
| R3C1 (i) **2** | R3C2 (j) **6** | R3C3 (k) **10** | R3C4 (l) **#** |
| R4C1 (m) **3** | R4C2 (n) **7** | R4C3 (#) **11** | R4C4(\*) **\*** |

(Vertical Pattern)

🡪Take the string of input buttons

🡪Generate the pattern from the input string.

🡪Query the database for the existing pattern.

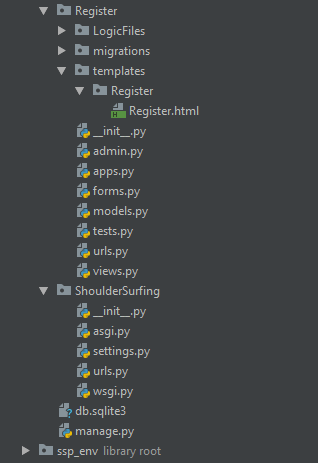
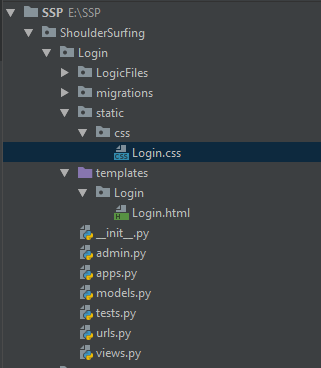
🡪If the pattern matches: Login Successful

🡪Next Login Session, Change the display values of the grid.

**Link to the Project: https://github.com/UditAgarwal27/SurfSecurity**

**CODE SCREENSHOT**

**PROJECT TREE**

****

**REGISTER USER CODE**

from .connectToDatabase import \*  
from .functionFile import \*  
  
integerList = []  
chunkList = []  
defaultIteration = []  
everyPermutationList = []  
everyHashedPermutationList = []  
  
def check(username):  
 checkSqlQuery = "SELECT \* FROM users WHERE username = %s"  
 value = (username,)  
 cursor.execute(checkSqlQuery, value)  
 result = cursor.fetchall()  
 if(len(result)>0):  
 return True  
 else:  
 return False  
  
def register(userdata):  
  
 dummyUserName = userdata['username']  
 dummyPassword = userdata['password']  
 dummyEmail = userdata['email']  
  
 isPresent = check(dummyUserName)  
  
 if(isPresent):  
 print("Username Already Exist")  
 else:  
 if (len(dummyPassword) % 2 == 0):  
  
 integerList = [int(x) for x in dummyPassword]  
  
 for i in range(0, int(len(integerList)), 2):  
 chunck = (integerList[i] \* 10) + integerList[i + 1]  
 chunkList.append(chunck)  
  
 defaultIteration = createDefaultIteration(chunkList)  
 print(defaultIteration)  
  
 defaultString = convertToString(defaultIteration)  
 print(defaultString)  
  
 everyPermutation = permutations(defaultString)  
  
 everyPermutationList = convertEachPermutationListToString(everyPermutation)  
 print(everyPermutationList)  
  
 everyHashedPermutationList = hashEveryPermutation(everyPermutationList)  
 print(everyHashedPermutationList)  
  
 everyHashedPermutationList.insert(0, dummyPassword)  
 everyHashedPermutationList.insert(0, dummyUserName)  
 print(everyHashedPermutationList)  
  
 insertSQLQuery = "INSERT INTO users (username, originalPass, pass1, pass2, pass3, pass4, pass5, pass6, pass7, " \  
 "pass8, pass9, pass10, pass11, pass12, pass13, pass14, pass15, pass16, pass17, pass18, pass19, " \  
 "pass20, pass21, pass22, pass23, pass24, email) VALUES("  
  
 for element in everyHashedPermutationList:  
 insertSQLQuery += "'"  
 insertSQLQuery += element  
 insertSQLQuery += "'"  
 insertSQLQuery += ','  
  
 insertSQLQuery += "'"  
 insertSQLQuery += dummyEmail  
 insertSQLQuery += "'"  
 insertSQLQuery += ')'  
  
 print(insertSQLQuery)  
 cursor.execute(insertSQLQuery)  
  
 mydb.commit()  
 print("User Registered successfully")  
  
 else:  
 print("Password is odd LENGTH")

**REGISTER VIEW CODE**

from django.shortcuts import render  
from .forms import SignUpForm  
from .LogicFiles.addUser import register  
  
def RegisterUser(request):  
  
 context = {}  
  
 data = SignUpForm(request.POST or None)  
 if(data.is\_valid()):  
 userdata = data.cleaned\_data  
 register(userdata)  
  
 context['form'] = SignUpForm()  
  
 return render(request, "Register/Register.html", context)

**LOGIN USER CODE**

from .connectToDatabase import \*  
from .functionFile import \*  
  
def signin(usernameLogin, passwordLogin):  
  
 searchSQLQuery = "SELECT \* from users WHERE username = %s"  
 usernameLogin = (usernameLogin,)  
 cursor.execute(searchSQLQuery, usernameLogin)  
  
 result = cursor.fetchall()  
  
 passwordLogin = hashLoginPassword(passwordLogin)  
  
 count = 0  
 for row in result:  
 for element in row:  
 if (element == passwordLogin):  
 count = 1  
 break  
 if(count == 1):  
 print("Login Success")  
 else:  
 print("Login Fail")

**LOGIN VIEW CODE**

from django.shortcuts import render, redirect

import random

from .LogicFiles.loginUser import signin

current = ''

username = ''

def dashboard(request):

    context = {}

    context['username'] = username

    if request.GET.get("logout"):

        return redirect("/login/")

    return render(request, "Login/dashboard.html", context)

def LoginUser(request):

    context = {}

    displayValue = ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9', '10', '11', '12', '13', '#', '\*']

    random.shuffle(displayValue)

    choiceList = [0, 1]

    choice = random.choice(choiceList)

    context['displayvaluebuttonone'] = displayValue[0]

    context['displayvaluebuttontwo'] = displayValue[1]

    context['displayvaluebuttonthree'] = displayValue[2]

    context['displayvaluebuttonfour'] = displayValue[3]

    context['displayvaluebuttonfive'] = displayValue[4]

    context['displayvaluebuttonsix'] = displayValue[5]

    context['displayvaluebuttonseven'] = displayValue[6]

    context['displayvaluebuttoneight'] = displayValue[7]

    context['displayvaluebuttonnine'] = displayValue[8]

    context['displayvaluebuttonten'] = displayValue[9]

    context['displayvaluebuttoneleven'] = displayValue[10]

    context['displayvaluebuttontwelve'] = displayValue[11]

    context['displayvaluebuttonthirteen'] = displayValue[12]

    context['displayvaluebuttonfourteen'] = displayValue[13]

    context['displayvaluebuttonfifteen'] = displayValue[14]

    context['displayvaluebuttonsixteen'] = displayValue[15]

    if (choice == 0):

        context['pattern'] = 'Pattern Horizontal'

        context['buttononevalue'] = 'a'

        context['buttontwovalue'] = 'b'

        context['buttonthreevalue'] = 'c'

        context['buttonfourvalue'] = 'd'

        context['buttonfivevalue'] = 'e'

        context['buttonsixvalue'] = 'f'

        context['buttonsevenvalue'] = 'g'

        context['buttoneightvalue'] = 'h'

        context['buttonninevalue'] = 'i'

        context['buttontenvalue'] = 'j'

        context['buttonelevenvalue'] = 'k'

        context['buttontwelvevalue'] = 'l'

        context['buttonthirteenvalue'] = 'm'

        context['buttonfourteenvalue'] = 'n'

        context['buttonfifteenvalue'] = '#'

        context['buttonsixteenvalue'] = '\*'

    elif(choice ==1):

        context['pattern'] = 'Pattern Vertical'

        context['buttononevalue'] = 'a'

        context['buttontwovalue'] = 'e'

        context['buttonthreevalue'] = 'i'

        context['buttonfourvalue'] = 'm'

        context['buttonfivevalue'] = 'b'

        context['buttonsixvalue'] = 'f'

        context['buttonsevenvalue'] = 'j'

        context['buttoneightvalue'] = 'n'

        context['buttonninevalue'] = 'c'

        context['buttontenvalue'] = 'g'

        context['buttonelevenvalue'] = 'k'

        context['buttontwelvevalue'] = '#'

        context['buttonthirteenvalue'] = 'd'

        context['buttonfourteenvalue'] = 'h'

        context['buttonfifteenvalue'] = 'l'

        context['buttonsixteenvalue'] = '\*'

    logindata = request.POST or None

    if request.POST.get("pass"):

        global current

        current = current + logindata['pass']

    if request.POST.get("save"):

        print(logindata['username'])

        print(logindata['password'])

        global username

        username = logindata['username']

        signin(logindata['username'], logindata['password'])

        return redirect("/login/dashboard")

    return render(request, "Login/Login.html", context)

def firstLayerAuthentication(request):

    firstLayerData = request.POST or None

    if request.POST.get("firstLayerSave"):

        answer = firstLayerData['answer']

        print(answer)

        if(answer == "Yes"):

            return redirect('/login/actualLogin')

    return render(request, "Login/firstLayer.html")

def phoneLayer(request):

    phoneLayerData = request.POST or None

    if request.POST.get("phoneLayerSave"):

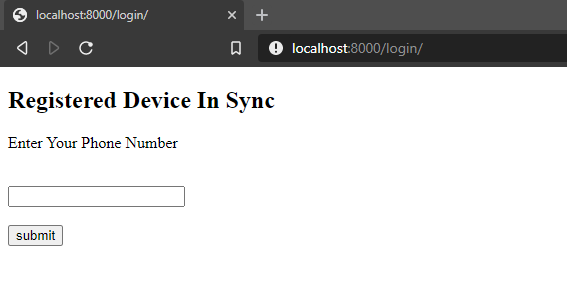
        phoneNo = phoneLayerData["phone"]

        print(phoneNo)

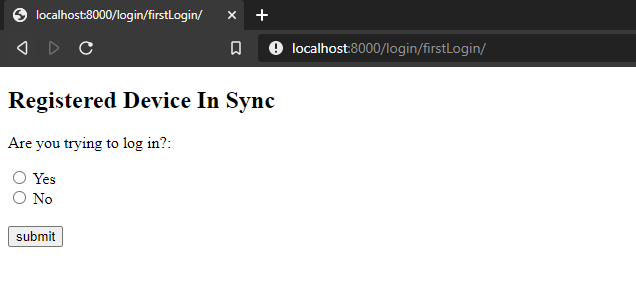
        return redirect('/login/firstLogin')

    return render(request, "Login/phoneLayer.html")

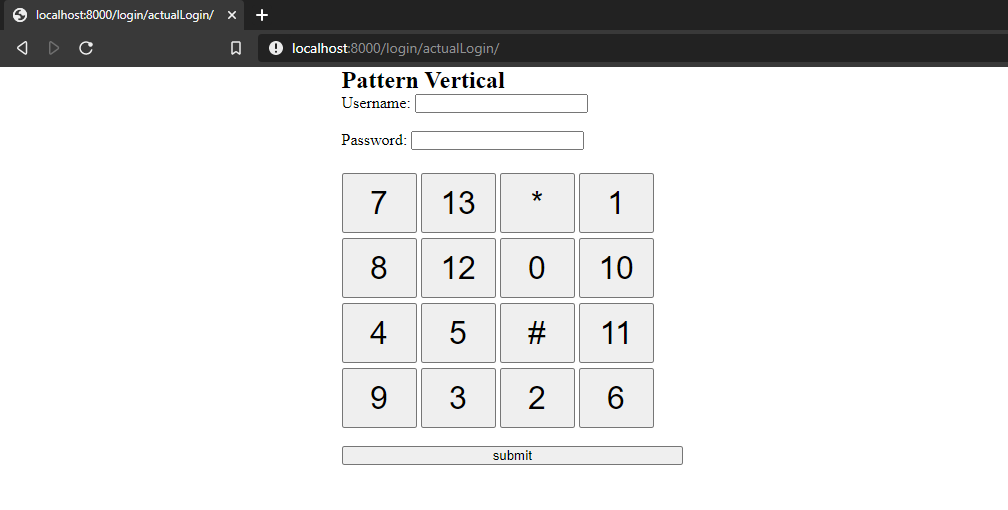
**localhost:8000/login**



First Layer: Entering the Registered Mobile Device Number

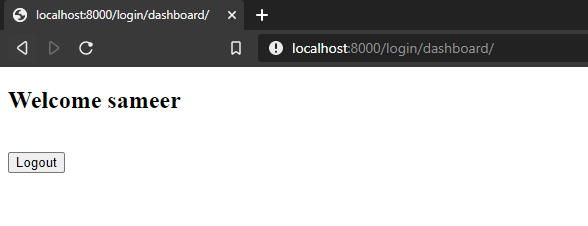


Second layer: Asking authentication question on the registered device in Sync.

****

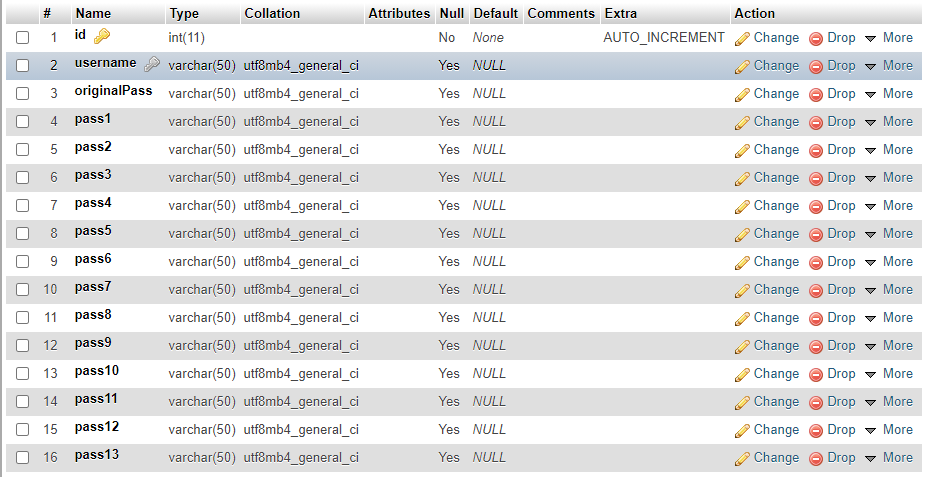
Actual Login Window: The current Session incorporates Vertical Type Pattern.

(This information of pattern will be sent to the user on the registered device , not on the desktop’s Browser)



Dashboard with final Logout button.

**localhost/phpmyadmin/ssp.users**



**LITERATURE RIVIEW**

1. Y. Chen, W. Ku, Y. Yeh and D. Liao, "A simple text-based shoulder surfing resistant graphical password scheme," 2013 International Symposium on Next-Generation Electronics, Kaohsiung, 2013, pp. 161-164, doi: 10.1109/ISNE.2013.6512317.

**ABSTRACT**: As most users are more familiar with textual passwords than pure graphical passwords, text-based graphical password schemes have been proposed. In this paper, the author propose an improved text-based shoulder surfing resistant graphical password scheme by using colors. In the proposed scheme, the user can easily and efficiently login system. Next, they analyze the security and usability of the proposed scheme, and show the resistance of the proposed scheme to shoulder surfing and accidental login.

2. H. Sun, S. Chen, J. Yeh and C. Cheng, "A Shoulder Surfing Resistant Graphical Authentication System," in IEEE Transactions on Dependable and Secure Computing, vol. 15, no. 2, pp. 180-193, 1 March-April 2018, doi: 10.1109/TDSC.2016.2539942.

**ABSTRACT**: With web applications and mobile apps piling up, people can access these applications anytime and anywhere with various devices. Attackers can observe directly or use external recording devices to collect users' credentials. To overcome this problem, the author proposed a novel authentication system PassMatrix, based on graphical passwords to resist shoulder surfing attacks. With a one-time valid login indicator and circulative horizontal and vertical bars covering the entire scope of pass-images, PassMatrix offers no hint for attackers to figure out or narrow down the password even they conduct multiple camera-based attacks.

3. H. Gao, Z. Ren, X. Chang, X. Liu and U. Aickelin, "A New Graphical Password Scheme Resistant to Shoulder-Surfing," 2010 International Conference on Cyberworlds, Singapore, 2010, pp. 194-199, doi: 10.1109/CW.2010.34.

**ABSTRACT**: In this paper, the author propose and evaluate a new shoulder-surfing resistant scheme which has a desirable usability for PDAs. Their inspiration comes from the drawing input method in DAS and the association mnemonics in Story for sequence retrieval. The new scheme requires users to draw a curve across their password images orderly rather than click directly on them. The drawing input trick along with the complementary measures, such as erasing the drawing trace, displaying degraded images, and starting and ending with randomly designated images provide a good resistance to shoulder-surfing.

4. H. Zhao and X. Li, "S3PAS: A Scalable Shoulder-Surfing Resistant Textual-Graphical Password Authentication Scheme," 21st International Conference on Advanced Information Networking and Applications Workshops (AINAW'07), Niagara Falls, Ont., 2007, pp. 467-472, doi: 10.1109/AINAW.2007.317.

**ABSTRACT**: Users tend to pick short passwords or passwords that are easy to remember, which makes the passwords vulnerable for attackers to break. In this paper, the author propose a Scalable Shoulder-Surfing Resistant Textual-Graphical Password Authentication Scheme (S3PAS). S3PAS seamlessly integrates both graphical and textual password schemes and provides nearly perfect resistant to shoulder-surfing, hidden-camera and spyware attacks. It can replace or coexist with conventional textual password systems without changing existing user password profiles. Moreover, it is immune to brute-force attacks through dynamic and volatile session passwords. S3PAS shows significant potential bridging the gap between conventional textual password and graphical password..

5. A. P. Sabzevar and A. Stavrou, "Universal Multi-Factor Authentication Using Graphical Passwords," 2008 IEEE International Conference on Signal Image Technology and Internet Based Systems, Bali, 2008, pp. 625-632, doi: 10.1109/SITIS.2008.92.

**ABSTRACT**: In this paper, the author present a series of methods to authenticate a user with a graphical password. To that end, they employ the users personal handheld device as the password decoder and the second factor of authentication. In their methods, a service provider challenges the user with an image password. To determine the appropriate click points and their order, the user needs some hint information transmitted only to her handheld device. They show that their method can overcome threats such as key-loggers, weak password, and shoulder surfing.

6. N. Wakabayashi, M. Kuriyama and A. Kanai, "Personal authentication method against shoulder-surfing attacks for smartphone," 2017 IEEE International Conference on Consumer Electronics (ICCE), Las Vegas, NV, 2017, pp. 153-155, doi: 10.1109/ICCE.2017.7889266.

**ABSTRACT**: They propose a personal authentication method against shoulder-surfing attacks for smartphone using secondary channel which is information transmitted to users only. Security level and usability of the proposed method is evaluated.

7. A. Papadopoulos, T. Nguyen, E. Durmus and N. Memon, "IllusionPIN: Shoulder-Surfing Resistant Authentication Using Hybrid Images," in IEEE Transactions on Information Forensics and Security, vol. 12, no. 12, pp. 2875-2889, Dec. 2017, doi: 10.1109/TIFS.2017.2725199.

**ABSTRACT**: They address the problem of shoulder-surfing attacks on authentication schemes by proposing IllusionPIN (IPIN), a PIN-based authentication method that operates on touchscreen devices. IPIN uses the technique of hybrid images to blend two keypads with different digit orderings in such a way, that the user who is close to the device is seeing one keypad to enter her PIN, while the attacker who is looking at the device from a bigger distance is seeing only the other keypad. The user's keypad is shuffled in every authentication attempt, since the attacker may memorize the spatial arrangement of the pressed digits. To reason about the security of IPIN, they developed an algorithm which is based on human visual perception and estimates the minimum distance from which an observer is unable to interpret the keypad of the user. They tested their estimations with 84 simulated shoulder-surfing attacks from 21 different people. None of the attacks was successful against their estimations. In addition, they estimated the minimum distance from which a camera is unable to capture the visual information from the keypad of the user. Based on their analysis, it seems practically almost impossible for a surveillance camera to capture the PIN of a smartphone user when IPIN is in use.

8. M. Lee, "Security Notions and Advanced Method for Human Shoulder-Surfing Resistant PIN-Entry," in IEEE Transactions on Information Forensics and Security, vol. 9, no. 4, pp. 695-708, April 2014, doi: 10.1109/TIFS.2014.2307671.

**ABSTRACT**: Unfortunately, the conventional PIN-entry method is vulnerable to shoulder-surfing attacks. In this paper, the author propose new theoretical and experimental techniques for quantitative security analysis of PIN-entry methods. They first present new security notions and guidelines for secure PIN-entry methods by analyzing the existing methods under the new framework. On the basis of these guidelines, they developed a new PIN-entry method that effectively obviates human shoulder-surfing attacks by significantly increasing the amount of short-term memory required in an attack

9. Kwon, Taekyoung, and Jin Hong. "Analysis and improvement of a pin-entry method resilient to shoulder-surfing and recording attacks." *Ieee transactions on information forensics and security* 10.2 (2014): 278-292.

ABSTRACT: In this paper, the author explore shoulder surfing defence for recall-based graphical password systems such as Draw-A-Secret and Background Draw-A-Secret, where users doodle their passwords (i.e. secrets) on a drawing grid. They proposed three innovative shoulder surfing defence techniques, and conduct two separate controlled laboratory experiments to evaluate both security and usability perspectives of the proposed techniques. One technique was expected to work to some extent theoretically, but it turned out to provide little protection. One technique provided the best overall shoulder surfing defence, but also caused some usability challenges. The other technique achieved reasonable shoulder surfing defence and good usability simultaneously, a good balance which the two other techniques did not achieve. Their results appear to be also relevant to other graphical password systems such as Pass-Go.

10. Kumar, Manu, et al. "Reducing shoulder-surfing by using gaze-based password entry." *Proceedings of the 3rd symposium on Usable privacy and security*. 2007.

ABSTRACT: The author presents EyePassword, a system that mitigates the issues of shoulder surfing via a novel approach to user input.

With EyePassword, a user enters sensitive input (password, PIN, etc.) by selecting from an on-screen keyboard using only the orientation of their pupils (i.e. the position of their gaze on screen), making eavesdropping by a malicious observer largely impractical. They present a number of design choices and discuss their effect on usability and security. Their results demonstrate that gaze-based password entry requires marginal additional time over using a keyboard, error rates are similar to those of using a keyboard and subjects preferred the gaze-based password entry approach over traditional methods.

11. P. Shi, B. Zhu and A. Youssef, "A PIN Entry Scheme Resistant to Recording-Based Shoulder-Surfing," 2009 Third International Conference on Emerging Security Information, Systems and Technologies, Athens, Glyfada, 2009, pp. 237-241, doi: 10.1109/SECURWARE.2009.43.

**ABSTRACT**: In this paper, the author propose a new PIN entry scheme which is resistant against shoulder-surfing attacks conducted by shoulder-surfers with normal cognitive capabilities. Additionally, this scheme offers a relatively good level of security when the shoulder-surfer can record the entire login procedure for one or two times with a video device. Mathematical analysis of the proposed scheme is also presented.

12. A. Aratani and A. Kanai, "Authentication method against shoulder-surfing attacks using secondary channel," 2015 IEEE International Conference on Consumer Electronics (ICCE), Las Vegas, NV, 2015, pp. 430-431, doi: 10.1109/ICCE.2015.7066474.

**ABSTRACT**: The author propose an authentication method against shoulder-surfing attacks using two kinds of channel, which are a primary channel on which there is a possibility of shoulder-surfing attacks and a secondary channel on which there is information only recognized by the user. Security and usability of the proposed method is evaluated.

13. Roth, Volker, Kai Richter, and Rene Freidinger. "A PIN-entry method resilient against shoulder surfing." *Proceedings of the 11th ACM conference on Computer and communications security*. 2004.

**ABSTRACT**: In this paper, the author present alternative PIN entry methods to which they refer as cognitive trapdoor games. These methods make it significantly harder for a criminal to obtain PINs even if he fully observes the entire input and output of a PIN entry procedure. They also introduce the idea of probabilistic cognitive trapdoor games, which offer resilience to shoulder surfing even if the criminal records a PIN entry procedure with a camera. They studied the security as well as the usability of their methods, the results of which we also present in the paper.

14. N. Chakraborty and S. Mondal, "Color Pass: An intelligent user interface to resist shoulder surfing attack," Proceedings of the 2014 IEEE Students' Technology Symposium, Kharagpur, 2014, pp. 13-18, doi: 10.1109/TechSym.2014.6807906.

**ABSTRACT**: In this paper, the author propose an intelligent user interface, known as Color Pass to resist the shoulder surfing attack so that any genuine user can enter the session PIN without disclosing the actual PIN. The Color Pass is based on a partially observable attacker model. The experimental analysis shows that the Color Pass interface is safe and easy to use even for novice users.