NTNU – Trondheim
Norwegian University of
Science and Technology

Design and Analysis of a Password Management System

Mats Birkeland Sandvoll

Electronics System Design and Innovation
Submission date: June 2014
Supervisor: Bjørn B. Larsen, IET
Co-supervisor: Colin Boyd, ITEM

Norwegian University of Science and Technology
Department of Electronics and Telecommunications

Problem Description

This thesis is based on the paper “Naturally Rehearsing Passwords” by J. Blocki, M. Blum and A. Datta [1], proposing a mathematical model for password management called Shared Cues.

The overall research problem of this thesis is to determine whether the Shared Cues model can be designed and implemented in software in a manner that is both usable and secure. The thesis should address the main challenges with password management in terms of usability and security, and further present the Shared Cues password management model. The work should include the design, implementation and evaluation of a password management system based on Shared Cues. The following should be addressed:

- How can the Shared Cues password management model be designed and implemented in software?
- Evaluate the implemented password management system in terms of usability, security and utilized hardware resources.
- Can the implemented password management system be used when logging in to a system?

Abstract

Managing passwords is a significant problem for most people in the modern world. In this thesis, a password management system has been designed and implemented as an iOS application called PassCue. PassCue is based on the Shared Cues password management model, proposed by J. Blocki, M. Blum and A. Datta in “Naturally Rehearsing Passwords”. The design and implementation choices, as well as parameter evaluation, were important in order to create a usable and secure system. PassCue uses cues to share secrets across multiple accounts in order to achieve the competing usability and security goals.

PassCue provides higher security than many of the popular password management schemes without significant reduction in usability. The probability that an attacker will compromise an account in an online attack is 1.47656×10^{-16} for PassCue (9,4,3) and (43,4,1), and 3.69140×10^{-21} for PassCue (60,5,1). In an offline attack with no previous plaintext leaks, cracking the PassCue (9,4,3) and (43,4,1) password will take over 38 years and cost over $\$700,000$. Cracking the PassCue (60,5,1) password would take over 1.5 million days and cost $\$2.84442 \times 10^{10}$ using technology known today. PassCue (9,4,3) does not require the user to invest additional time in order to maintain the passwords in memory, but in PassCue (43,4,1) and PassCue (60,5,1) the user must perform 11 and 20 extra rehearsals respectively.

The PassCue design and implementation can easily be customized to support different usability and security needs. The PassCue application utilizes a low percentage of the CPU and memory of an iPhone 5, and uses less then 1% of the CPU and 5.9MB of memory in idle state.

Sammendrag

Håndtering av passord er et betydelig problem i den digitale hverdagen. I dette arbeidet har et håndteringssystem for passord blitt designet og implementert som en iOS applikasjon kalt PassCue. PassCue er basert på en passordhåndteringsmodell kalt “Shared Cues”, utarbeidet av J. Blocki, M. Blum and A. Datta i “Naturally Rehearsing Passwords”. Design- og implementeringsvalg, i tillegg til evaluering av parametere, har vært viktig for å utvikle et system som både er brukervennlig og sikkert. PassCue bruker hint til å dele hemmeligheter på tvers av kontoer for å oppnå de motarbeidende målene om sikkerhet og brukervennlighet.

PassCue gir høyere sikkerhet enn mange av de mest populære håndteringssystemene uten å redusere brukervennligheten. Sannsynligheten for at en angriper klarer å bryte seg inn på en konto ved hjelp av et online angrep er 1.47656×10^{-16} for PassCue (9,4,3) og (43,4,1), mens 3.69140×10^{-21} for PassCue (60,5,1). Det vil ta over 38 år og koste over $\$700,000$ for å knekke et PassCue (9,4,3) eller et PassCue (43,4,1) passord i et offline angrep. Et offline angrep for å knekke et PassCue (60,5,1) passord vil ta over 1.5 millioner år og koste $\$2.84442 \times 10^{10}$ med dagens teknologi. PassCue (9,4,3) krever ingen ekstra øvinger, men for PassCue (43,4,1) og PassCue (60,5,1) må brukeren utføre henholdsvis 11 og 20 ekstra øvinger for å opprettholde passordene i minnet.

PassCue er utviklet for å enkelt kunne tilpasses til forskjellig sikkerhets- og brukervennlighetsbehov. PassCue er svært ressursgjerrig, og bruker mindre enn 1% av prosessorkraften til en iPhone 5 i idle tilstand, og kun 5.9 MB minne.

Preface

This master's thesis has been carried out at the Department of Electronics and Telecommunications at NTNU during the spring of 2014. The thesis has been written as the final part of the master program. The research problem of this thesis was proposed by Colin Boyd and involves the exploration and analysis of a new password management scheme.

The overall research goal of this thesis was to design, implement and analyse a password management system based on a theoretical model. A goal I personally think has been achieved. The combination of designing, implementing and analysing made the work varying, interesting and challenging.

When starting the project work, I did not have any experience with password management and limited knowledge of iOS programming. I spend much time studying the theoretical password management model, and how this could be implemented in software. The implementation debugging has been very time consuming. This work is not directly described in the thesis, but it is a big part of the project.

I would like to thank my supervisor Professor Colin Boyd, and Associate Professor Bjørn B. Larsen for inspiring guidance during this project.

- *Mats Birkeland Sandvoll*
Trondheim, 12.06.2014

Contents

List of Figures

List of Tables

List of Algorithms

List of Abbreviations

PSM	Password Strength Meters
APSM	Adaptive Password Strength Meters
NIST	National Institute of Standards and Technology
PAO	Person-Action-Object
CRT	Chinese Remainder Theorem
PRNG	Pseudo-Random Number Generator
TRNG	True Random Number Generator
CR	Constant Rehearsal Assumption
ER	Expanding Rehearsal Assumption
SQ	Squared Rehearsal Assumption
IDE	Integrated Development Environment
SDK	Software Development Kit
SC	Shared Cues
PCP	Password Composition Policy
API	Application Programming Interface
GPU	Graphical Processing Unit
CPU	Central Processing Unit
GPGPU	General-Purpose Computing on Graphics Processing Units
MVC	Model-View-Controller
EC2	Elastic Compute Cloud
MB	Megabyte
KB	Kilobyte

Chapter 1

Introduction

The average user logs on to several systems each day by using passwords. Passwords are used to secure valuable data. Online services including banking, voting, mail, social networks, commerce and enterprise resources depend on passwords in order to maintain secure. The typical user has multiple password protected accounts and needs to manage each password. As the number of accounts increases, the management of passwords gets complicated. As a consequence, many users tend to adapt weak password management schemes which can significantly reduce the security of the system.

This thesis is based on the paper "Naturally Rehearsing Passwords" by J. Blocki et al. [1] which proposes a mathematical model for password management called Share Cues. Shared Cues is a password management model which incorporates issues concerning both security and usability.

1.1 Problem Description

The overall research problem of this thesis is to determine whether the Shared Cues model can be designed and implemented in software in a manner that is both usable and secure. The thesis should address the main challenges with password management in terms of usability and security, and further present the Shared Cues password management model. The work should include the design, implementation and evaluation of a password management system based on Shared Cues. The following should be addressed:

- How can the Shared Cues password management model be designed and implemented in software?
- Evaluate the implemented password management system in terms of usability, security and utilized hardware resources.
- Can the implemented password management system be used when logging in to a system?

1.2 Outcome

The main object of this thesis is to determine how Shared Cues can be designed and implemented in software in a manner that is both usable and secure. The overall outcome consist of different parts which can be given as:

- Design of a password management system based on Shared Cues. The system design include selection and evaluation of parameters such as sharing set, association size, rehearsal schedule, public cues and evaluating how to cope with password composition policies. The database architecture is a major part of the overall system design.
- Implementation of the designed password management system as an iOS application and a presentation of its use. The application development is based on the system design and specific implementation choices. The presentation of the application include the initialization and how to use the application to log on to a system.
- Usability, security and implementation analysis of the implemented system. The usability analysis measure the effort the user must invest in order to use the system. The security analysis measure password entropy and resistance against common attacks. The implementation analysis covers the CPU utilization and memory consumption of the implemented iOS application.

1.3 Outline

A brief overview:

- Chapter 2 presents the theory including the security and usability challenges with password management.
- Chapter 3 presents the Shared Cues password management model.
- Chapter 4 covers the design of a password management system based on Shared Cues.
- Chapter 5 presents the implementation of the designed system.
- Chapter 6 contains the analysis and evaluation of the implemented system.
- Chapter 7 concludes the thesis and presents suggestions for future work.

Chapter 2

Theory

Password is one of the most widely used authentication techniques today. The average internet user has many online accounts and needs to manage multiple passwords on a daily basis. Password breaches [7, 8, 9, 10, 11] have shown that people tend to choose poor passwords and adopt weak password management techniques, caused by the inherent trade-off between security and usability.

This chapter presents the theory and the central principles that this thesis and the Shared Cue password management model is built upon. The first part of this chapter covers password authentication in general, password composition policies and five password management schemes. Password management tools and methods will also be briefly introduced in the first part. The second part of the chapter presents the security aspect of password management, including possible attacks, vulnerabilities and how to measure security. The last part of the chapter covers usability by presenting the psychological aspect of the human memory and different mnemonic techniques.

2.1 Password Authentication

The password is one of the most used authentication techniques [12] in computer systems. New technology facilitates for the use of biometrics and tokens in authentication, but the password is still a frequently used authentication method. Passwords do not require additional hardware, can easily be implemented and are simple to use. Millions of people use password as a method to protect their data, and the method dates back to the Roman empire [13].

Most internet users have many password protected online accounts and need to use a password management scheme. A password management scheme or password scheme is any method for creating and retrieving passwords. In 2007, it was estimated that the average user has approximately 25 password protected accounts [14]. Password composition policies, presented in Section 2.1.1, can complicate the process of managing all the passwords. As a result, many users tend to adopt insecure password management schemes to increase usability. Writing down the passwords and selecting weak passwords are typical insecure password schemes. Password reuse tend to grown in popularity as the number of online accounts increases [12].

In December 2009, a major password breach led to the release of 32 million passwords from the rockyou.com user database [7]. The password analysis showed that about 30% of the passwords had a length equal to or below six characters [15] and 70% equal to or below eight characters. Nearly 50% of these passwords were names, slang words or dictionary words and the most popular passwords included "123456", "12345", "password", "iloveyou", "princess" and "abc123". The Sony Playstation privacy breach in April 2011 [8] exposed 77 million customer accounts showed similar results; the password length was between six and ten characters, less then 1% had a non-alphanumeric character [16] and approximately 30% were dictionary words. The same analysis showed that users tend to select passwords inspired by words of personal importance or memorable patterns and that truly random passwords were almost non-existent. In attempts to strengthen a password, users tend to follow predictable patterns such as deriving passwords from person's names (15%), place names (8%) and dictionary words (25%) [16].

2.1.1 Password Composition Policy

Password composition policies (PCPs) are used by system administrators in order to prevent the user from selecting weak passwords that can easily be guessed. Rockyou.com, introduced in Section 2.1, were enforcing a weak password composition policy [17] by restricting the use of special characters, no requirement for mixed-case, numbers or punctuation and a minimum length of only five characters.

It is commonly understood that password composition policies make the passwords harder to guess, but that is not always the case [12, 18]. This is because a password composition policy does not only affect the resulting password, but also the user behaviour. A password composition policy that ensures a complex password could also lead users to write down the password, adapt insecure management schemes [18] or be adverse to password changes. Websites differ in their password composition policies, but common guidelines are presented in Table 2.1.

The password composition policy varies from website to website and in some cases it is not possible to follow the guidelines stated in Table 2.1. The German bank Berliner Sparkasse and the financial services cooperation Fidelity are examples of websites that only allow alphanumerical characters. The Bank of Brazil does only allow numbers, and in Virgin Atlantic the passwords are not case-sensitive [19]. Websites can also put restrictions on which type of special characters that can be used and restrictions on maximum length. The banking company Charles Schwab has a password length limit of eight characters [20] and Outlook.com has a maximum password length of 16. Google allow for 200 characters [21] in

Table 2.1: Guidelines for a password composition policy [2, 3, 4, 5]

Password Composition Policy Guidelines
Use both lowercase and capital letters
Use a combination of letters, numbers and special characters
Use passwords with minimum length of 12 characters
Use randomly generated passwords if possible
Never use dictionary words, names or biographical information
Never use passwords based on repetition, letter or number sequence or usernames
Never use personal information
Never reuse passwords

the account password. Evernote do not allow spaces in the middle of passwords because, as CTO Dave Engberg says, “Adding support for spaces only in the middle of the password would make the regular expression defining them three times longer” [20].

2.1.2 Password Schemes

A password scheme or password management scheme, is any method for creating and retrieving a password. Strong passwords are hard for attackers to guess, but they are also hard for users to remember. This results in the inherent trade-off between usability and security [22]. Richard Smith notes, “the password must be impossible to remember and never written down” [23]. There are multiple schemes for creating and retrieving passwords. Many of the password management schemes are vague and unclear [24] in their description which may lead the user to select a weak password. It has been illustrated that security schemes can break down when humans behave in unexpected ways due to vague instructions [25].

Multiple password schemes exist and the password scheme is only restricted according to the users imagination. Wilderhain et al. [24] describe 15 different schemes and compare them in terms of usability and security. This thesis will focus on five password management schemes; *Reuse Weak*, *Reuse Strong*, *Lifehacker*, *Strong and Independent* and *Randomly Generated*. The five password schemes are presented in Table 2.2.

In the *Reuse Weak* scheme the user selects a dictionary word and reuse this password on all accounts. The full algorithm for the scheme is given in Algorithm 5 in Appendix B. *Reuse Strong* is when the user selects a strong password, e.g. combining four independent words, and reuses this password on all sites. The *Reuse Strong* scheme is given in Algorithm 6 in Appendix B.

In the password scheme *Lifehacker* the user selects a base password of three words and uses a derivation rule to derive a string from the account name. The resulting password is the base password with the derived account name appended. The full

Table 2.2: Password Schemes

Scheme	**Create**	**Use**	**Example**
Reuse Weak	Select a word w randomly from a dictionary containing 20000 unique words	Password=w for all accounts	Password=*horse*
Reuse Strong	Select four words $w_1w_2w_3w_4$ randomly from a dictionary containing 20000 unique words	Password=$w_1w_2w_3w_4$ for all accounts	Password=*apledoghorseblue*
Lifehacker	Select three words $w_1w_2w_3$ randomly from a dictionary containing 20000 unique words as base. Use a derivation rule $d()$ to derive a string from account name	Password=$w_1w_2w_3d(A_{name})$ unique for each account	A_{name}=facebook Password=*apledoghorsefak*
Strong Rand. & Ind.	Select four words $w_1w_2w_3w_4$ randomly from a dictionary containing 20000 unique words	Password=$w_1w_2w_3w_4$ unique for each accounts	Password=*apledoghorseblue*
Randomly Generated	Generate a random password using a password generator	Password = *random* unique for all accounts	Password=*bcxtabf2owale89n*

algorithm is shown in Algorithm 8 in Appendix B. A detailed example for the *Lifehacker* scheme is given below.

> *Example - Lifehacker Password Scheme:* The three random words are; w_1 =aple, w_2=tower and w_3=cup. The derivation rule d is in this case *the first two characters and the last character in the account name*, which means that $d(A_i)$ returns the first two characters and the last character of the account name A_i. The resulting password of the account A_1= facebook is p_1=apletowercupfak. For account A_2=amazon the password is p_2=apletowercupamn.

Strong Random and Independent password management scheme is when the user selects four random dictionary words and combine them to a password. Four random words are selected for each account. Algorithm 7 in Appendix B illustrates the *Strong Random and Independent* password scheme.

In the *Randomly Generated* scheme the password is randomly generated using a computer. Multiple password management tools offer a random password generator, which will be further discussed in Section 2.1.3. In the *Randomly Generated* scheme the computer randomly composes a 16-character string with lower-case letters and numbers.

2.1.3 Password Management

This section covers a range of different password management methods and tools. We saw in section 2.1.2 that password management scheme is any method for creating and retrieving passwords. This section presents management tools which can help the user with the task of creating and using password. The section also presents management tools which utilizes graphical, geographical or visual information to help the user manage passwords. The wide research in this field and the numerous amount of tools reflect the difficulty for the user to remember an increasing amount of secure passwords. This proves that there is clearly a need for tools to help humans cope with password-based authentication.

Password Management Tools

A password management tool is a program that stores the passwords for the user. With a password manager, the user does not need to memorize all the different complex passwords. The passwords are safely stored in the password manager database. The password manager is protected with a master password and the master password is the only password the user needs to remember. The master password restrict the access to the password manager, hence it must be a strong password. Compromise of the master password would reveal all the stored passwords. The password database is securely encrypted, normally with 256-bit AES, which is considered safe by National Institute of Standards and Technology (NIST) [26]. Bruce Schneier notes, "In fact, we cannot even imagine a world where 256-bit brute force searches are possible. It requires some fundamental breakthroughs in physics and our understanding of the universe" [27]. Some password management tools also support biometrics such as fingerprint reader, or hardware two-factor authentication tokens to increase the security of the password database. Most password managers offer secure cloud storage of the password database, random password generator [28] and cross-platform compatibility.

Graphical Password Management

The inherent trade-off between usability and security in password based authentication has motivated the research on graphical password schemes. Graphical passwords schemes are assumed to give a higher usability, because of the memorability, while providing sufficient security. Graphical password schemes leverage the human capability of visualizing information, in order to share a secrets to be used as evidence of identity. There is multiple graphical password schemes, but few are able to address the known problems related to text-based passwords. Much of the published research lacks consistency [29], which makes it hard to compare and reproduce results.

Geographical Password Management

A recent paper proposes a new way of managing passwords with the use of geographical data. The author states, "A GeoGraphical password is a password that has been constructed based on GeoGraphical information" [30]. The user selects a geographical area, for example, by drawing a circle around a pyramid or drawing a polygon around a lake, and the geographical information is used to form the geographical password. The scheme uses information such as longitude, latitude, altitude, areas, perimeters, sides, angles, radius for the geographical area to construct the password. The scheme enables the use of multiple geographical areas and the use of memorable or randomly generated stings to increase the security.

Diceware

Diceware is a method for selecting passphrases which consists of multiple words combined in a string [31]. The words are selected from a Diceware Word List using a dice. Each of the words in the Diceware Word List is assigned a unique five-digit number, where the digits range from one to six. The user must roll the dice five times and select the word that corresponds to the obtained sequence. The Diceware Word List consists of $6^5 = 7776$ different words. The number of words that is combined to a passphrase string depends on the security. Diceware recommends using five to nine words depending on the level of security needed. Five words can be cracked for approximately \$1000 [32], while six words can be broken by organizations with large budgets. Seven words and above are considered unbreakable with known technology.

Pixelock

Pixelock is a system for password management that enables the creation of secure and effective passwords [33]. The user selects specific points on an image and a unique algorithm generates a password. A password typically consists of four clicks, but increasing the number of clicks increases the password space and the password security. The user only needs to remember the selected points on the image and the order they were selected.

2.2 Usability

> The Principle of Psychological Acceptability: *"It is essential that the human interface be designed for ease of use, so that users routinely and automatically apply the protection mechanisms correctly. Also, to the extent that the user's mental image of his protection goals matches the mechanisms he must use, mistakes will be minimized. If he must translate his image of his protection needs into a radically different specification language, he will make errors."*
>
> (Jerome Saltzer and Michael Schroeder [34])

The Principle of Psychological Acceptability states that accessing a system should not be more difficult with a security mechanism then if the mechanism was not present. This raises a crucial problem concerning usability; difficult for whom? Usability applies differently to different types of users. A computer engineer may find a password management scheme easy and highly usable, but an infrequent home user could disagree. Security mechanisms are often implemented using the developers own impression of the system [35] and without taking into account the users of the system. This could be a reason why password composition policies in some cases result in a weaker password [12, 18], namely that users include personal words when trying to fulfill the password requirements.

Usability is an important factor when people adopt a password scheme. In order to have a working password authentication protocol the user must create and maintain the memory of the created password. The usability can be measured as the effort the user must invest in order to remember his passwords. Users tend to prioritize usability over security [14] when creating a password. As Bensinger notes, "Humans, in general, do not prefer to memorize characters and if they had to, they do it in the least possible effort" [36]. This can be related to the restrictions in the human memory, as Jeff Yen notes "Many of the deficiencies of password authentication systems arise form human memory limitations" [37]. A typical user has approximately 25 account that require passwords [14], and logs on to about eight systems on a daily basis. Password composition policies forces the use to select a strong password, but does not take into account the usability aspect. The effect of composition policies is also debatable [18].

There is a trade-off between security and usability. In an ideal system, the security and usability would both increase simultaneously. The challenge is to balance the the ability of a human to remember a password that is hard or impossible for an attacker to guess or crack [35]. A major reason why users select weak passwords is that different users have different ideas of what makes a password strong and secure [38]. A user received feedback from a password checker that names could not be used as password, and subsequently changed the password to "Barbara1" [35]. Another user had the Japanese word for security as password, and he was very surprised that his password could be guessed so quickly. He had never expected an attacker to use Japanese words. A developer, system administrator or a computer engineer has a reasonable idea of how resourceful an attacker can be and how to securely choose passwords, but that is not the case for an infrequent home user.

2.2.1 Human Memory

There are also physical restrictions on remembering different passwords. The human memory is temporally limited in remembering sequences of items [39]. The human memory has a short-term capacity of approximately seven plus minus two items [40]. It has also been shown that humans remember when the sequence of items is familiar chunks as words or familiar symbols. The human memory is semantic [1], so in order to facilitate for usability the number of chunks should be minimized. The human memory can be characterised as associative, which means that memories are associated with other memories. We are more capable at remembering information we can encode in multiple, redundant ways [37].

Baker-baker Paradox

If a person is asked to remember the two sentences; “My name is Baker” and “I am a baker”, several studies have shown that most people remember the occupation baker rather then the name, even if the word is the same. This is known as the Baker-baker paradox [41]. A likely reason for the Baker-baker paradox is that the occupation baker is automatically associated with different memories and connected to different concepts tied to the profession. The name Baker, in the opposite, is much likely to stand alone without any associations or connections resulting in a weak memory. The strength of a specific memory may be measured according to the number of associations it triggers in the brain.

2.2.2 Mnemonic Techniques

Mnemonic techniques are methods to help retaining memories, and many mnemonic techniques exist. The effectiveness of a mnemonic technique is individual, and common mnemonic devices include music, names, pictures, expressions, words and models. It has been shown that users can exploit mnemonic strategies in order to remember passwords [42]. Blocki notes, “Competitors in memory competitions routinely use mnemonic techniques which exploit associative memory”[1].

There is multiple mathematical models for the human memory [43, 44, 45]. These models differ in some details, but all of them model an associative memory [1] with cue-association pair. Cues are the context in which a memory is created. The cue can be a sound, surrounding environment, a person, an object or anything that is associated with a specific memory. For the user to remember the password p he associates the password memory with a cue. The human memory is lossy, so the cue strength is often reduced as time passes. If a memory is not refreshed or a password rehearsed, it will be forgotten. To prevent the user from forgetting the cue-association pair, it is essential to create strong associations and maintain them over time through rehearsal. The rehearsal process should be as natural as possible, and as a part of the users normal activities so the usability is not decreased.

Person-Action-Object

Person-Action-Object (PAO) Story is a mnemonic technique that consist of a picture of a person, an action and an object. The person could, for example, be Elvis Presley with the action *shooting* and the object *banana*. The resulting PAO story would be; “Elvis Presley is shooting a banana”. The PAO pattern is very simple and with randomly generated persons, actions and objects, the story can be quite surprising [1]. It has been shown that memorable sentences use strange combination of words in a common pattern [46]. A modified version of the PAO mnemonic technique is used in the Shared Cues password management model. The modified version also includes a picture of a place or a background type image.

Memory Palace

Memory Palace, also known as the Method of loci [47], is a mnemonic technique that is presented in ancient Roman and Greek literature. In this technique the person must memorize and visualize the layout of a building, e.g. the subjects home, the arrangement of several houses in a street or any geographical location that includes a number of different places. This is known as the memory palace. In order to memorize items, the person need to visualize that he walks through his memory palace while imaginary placing the items in different places in the memory palace. Placing items is done by connecting a feature of the particular place in the memory palace to an image of the item. The more strange and funny story the person creates, the easier is it to recall the memory. In order to retrieve the items, the person visually walks through his memory palace and the different places will activate the memory of the corresponding item [48]. This method has shown to be quite effective and is used by many memory athletes today [49].

2.3 Security

The main goal of password-based authentication is to protect important assets. It is necessary to be familiar with the security threats in order to be able to defend against them. There are different threats that could compromise the security of a system, and there are numerous real-world examples of passwords breaches [1, 7, 8].

2.3.1 User Knowledge and Behavior

The users knowledge and behavior is a possible threat to the security of a system. An attacker can capture a password simply by watching the user input the password. This is called shoulder surfing and the user should be aware of this threat when using a system in public. Most internet sites display the password as asterisk when the user is typing the password, but a trained attacker can get the password by monitoring the keystrokes.

Passwords can also be monitored by the use of a keylogger. A keylogger is a type of malware [50], which monitors every key the user presses and reports it to an attacker. To mitigate this threat it is important to update software regularly, use anti malware tools, prevent software from running in root-mode and have caution when using public computers to ensure that passwords are not stored.

Social Engineering

A user can reveal his password as a result of social engineering. An attacker could call a user, pretending to be customer service, and ask the user to provide the password in order to fix a problem. Phishing attack is a social engineering technique. In a phishing attack the attacker masquerades as a trustworthy entity such as banking sites, government sites or social web sites. The user is tricked to enter personal information on fake websites or fake programs as they are seemingly identical to the real website or program. Phishing emails can redirect the user to a malicious site [50] or deceive the user to provide personal information and credit card numbers. A phishing attack could result in an economic loss and a compromise of the privacy.

2.3.2 Password Storage

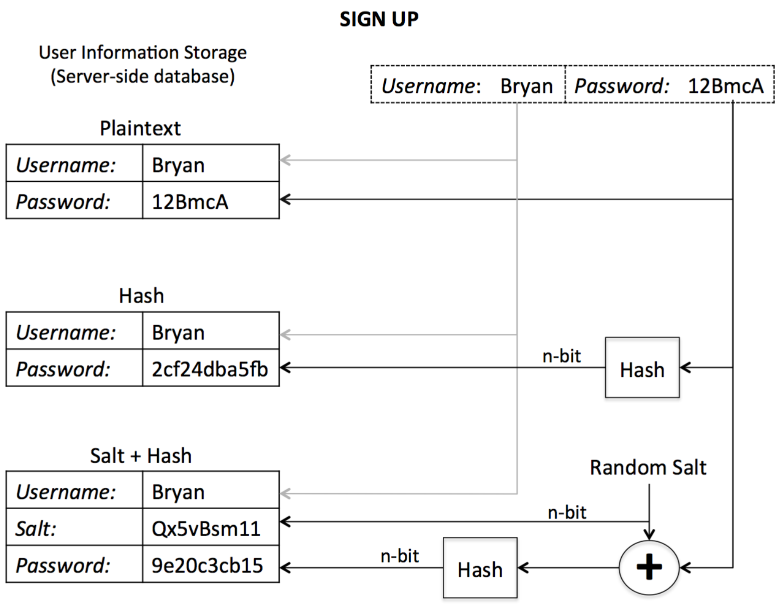


Figure 2.1: Sign up for an account

When a user signs up for an account, the online service saves the user credentials in a database on an external server. How the user credentials are kept on the server is out of control for the user, but it highly affects the security. Figure 2.1 shows the three most common ways of saving user information. The first method is to store the user password in plaintext. This should never be performed, but as we saw with rockyou.com in Section 2.1 and LinkedIn [11], this is not always the case. Saving the password in plaintext increases the probability for a plaintext leak attack, which will be further presented in Section 2.3.3.

Hashing the password before storing is the second method for saving user credentials. There are multiple hash functions available and among them popular used

MD5 and SHA. A hash function is a one-way function which takes a message of arbitrary length and produces a hash with fixed length. If the password hash is leaked, a potential attacker can perform an offline attack. Offline attack is a guessing, or cracking, technique. Offline attacks are not as trivial as plaintext leak attacks and will be further discussed in Section 2.3.3.

As presented in Figure 2.1, salting and hashing is the third way of storing the user password. When the user sign up for an account, a salt is randomly created and appended/prepended the password before hashing. A salt is a random value, which is concatenated with the password. A new and random salt must be generated for each password, which ensures that two identical passwords will have different hash. The salt should also be of equal length as the password hash. If the salt is too short, a potential attacker can compute a lookup table for every salt. Salting the password before hashing makes it more difficult to perform an offline attack if the password hash is leaked. This is because the attacker cannot use a cracking technique known as rainbow tables. This is further presented in Section 2.3.3. The salt must also be stored in order to successfully perform the user authentication. How the user is authenticated using the three password storage methods is presented in Figure 2.2.

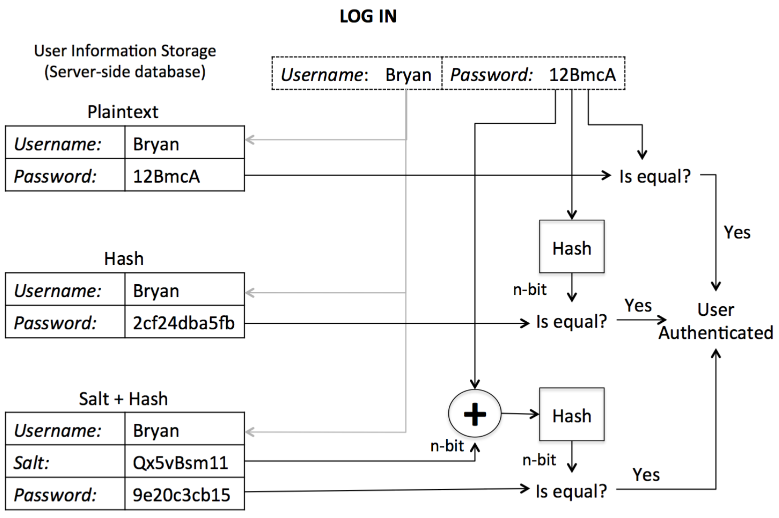


Figure 2.2: Log in to existing account

Salting and hashing the password before storing is considered safe password storing, but it is still possible to perform an offline attack. Encrypting the hash with a cipher such as AES [51] or using a keyed hash function such as HMAC before storing would create an impossible to crack hash. In order to do so, the user and the online service must share a secret key.

2.3.3 Attacks

There are multiple attacks that can be performed on a system which does not directly relate to the user. The users choice of password is highly relevant for the success of these attacks.

Online Attack

Online attack is a type of guessing attack. If the attacker has access to the authentication interface, the attacker can simply try as many guesses as the site permits. Online attack can be performed as a brute force, dictionary or hybrid attack. Brute force attack is when the attacker tries all possible combinations of characters assuming a fixed password length [50]. The brute force attack is resource demanding and will most likely require a large amount of time. Dictionary attack is when the attacker tries to guess the password using a list of possible passwords and not an exhaustive list of all possible combinations. Hybrid attack is a combination of dictionary and brute force attack. The attacker uses a list of possible passwords and then adds variations to the possible passwords in a brute force way in order to create new possibilities.

The attacker could also try to retrieve background information about the victim because words, dates and names of personal importance are often used to derive passwords [16]. Many internet sites have implemented a k-strike policy [52, 1] which limits the amount of password guesses to k guesses. The account will be locked for a period of time after k wrong guesses.

Plaintext Leak Attack

Plaintext leak attack is when the password is leaked in plaintext and can be directly used by the attacker. Plaintext leak attack can occur as a result of a phishing attack [1] or misconfiguration of the online server. Online services should never save the password in plaintext, only the salt and the computed hash of the password as we saw in Figure 2.1. This is not always the case as some services do not use salt [11] and some do not use hashing at all[7, 53]. If the password management scheme *Reuse weak* or *Reuse strong* is used, a plaintext leak attack can result in serious consequences.

Offline Attack

Offline attack is a type of cracking attack and requires the attacker to be in possession of the cryptographic hash of the victims password. Many online services have experienced that their database have been hacked [9, 10, 11] and the users password hash has been revealed. The attacker can compute the hash of a combination of characters till he finds a hash that matches the leaked password hash.

The offline attack can be performed as a brute force, dictionary or a hybrid attack. Offline attacks can be used together with rainbow tables [54]. Rainbow tables are lookup tables that contains pre-computed password hashes. With rainbow tables the attacker does not need to generate the hash, but simply iterate through the rainbow table and look for a hash that matches the leaked password hash [50]. As a consequence, rainbow tables can be used to speed up the offline cracking time. Rainbow tables are available online [55] and can easily be downloaded.

As we saw in Section 2.3.2, salting is the method of adding a random value to the password before generating the hash. Salting makes it impossible to use rainbow tables as every passwords needs to be added a random value before being hashed. Even though salting prevents the use of rainbow tables, it does not prevent dictionaries attacks and brute-force attacks. Salting the password can slow down brute-force and dictionary attacks, but modern cracking tools have optimization techniques [56], which can diminish the effect of salting. Advances in computing power and General-Purpose Computing on Graphics Processing Units (GPGPU) has exponentially increased the number of hash computations per second. It is in many cases less resource demanding to brute-force the password [57] rather then comparing the password hash with terabytes of precomputed hashes in a rainbow table.

Hashcat is the “worlds fastest password cracker” and the “worlds first and only GPGPU based rule engine” [58]. Hashcat is a free tool for advanced password recovery and is able to run millions of hash calculation every second due to its utilization of GPUs. Hashcat supports over 100 algorithms and different attack modes, such as dictionary attack, brute-force, hybrid or combination attack. Hashcat enables the user to specify rules in order to optimize the cracking.

As cracking tools get more sophisticated and computing hardware follows Moore’s law, the online services must carefully select which hashing algorithm to use when storing user passwords. The hashing algorithm should be slow, in the sense that it requires time to generate the hash. A slow hash algorithm will dramatically decrease the number of guesses per second for cracking software. A typical slow hash algorithm uses a technique called key stretching, which is a CPU-intensive hash function. Commonly used slow algorithms are bcrypt, scrypt and PBKDF2, while commonly used MD5 and SHA-1,2 and 3 is quite fast [59, 60, 61, 58, 57]. The bcrypt algorithm allows the user to specifically select the computation to be slower [57], hence increasing the security. The cost of password cracking will be presented in Section 2.3.4.

2.3.4 Measuring Security

Users have a very different understanding of security. A computer engineer or a system administrator might have a clear picture on how to create a strong password compared to the infrequent home user who uses a computer a couple of times a week. It is necessary to measure the password security in order to educate the user and give proper feedback upon account creation. Password strength represents how effective a password is against guessing and brute-forcing attacks. The password strength is related with the length, complexity and unpredictability of the password [3]. A secure password can lower the risk for a security breach in a system, but the overall security is also dependent on the implementation of the system in terms of authentication, communication and storage [22]. As we saw in Section 2.3.3, numerous attacks can occur which can compromise the system. The password strength is a measure on how difficult it is for these attacks to succeed.

Password Strength Meters

Proactive password checking or password strength meters (PSM) is a method that forces the user to choose a complex password. A PSM takes a password, a string, as input over an alphabet, and outputs a score [62] on how secure the string is as password. The score is usually a real number, which indicates the effort for an attacker to guess the password. The PSM can be used to advise the user or to enforce a password composition policy. The PSM uses certain rules, similar to the guidelines in Table 2.1, to exclude weak passwords. The effect PSM has on the password strength is debatable [63] and it could encourage people to use non-dictionary words of personal origin [22], such as person names and dates. Traditional PSMs base their score on simple metrics such as password length, non-alphabetic and capitalization, and does not take into account advanced cracking techniques [22, 64, 65].

NIST proposed a PSM [66, 62] that became quite influential. The NIST PSM calculates the entropy of a password based on the password length and gives special bonuses if a password matches special conditions, such as contains numbers or capitalization. The Microsoft PSM [5] gives out an integer between zero and four. Four is the strongest and it is given if the password pass a dictionary test, is longer then 14 characters and contains three types of characters. The Google PSM is implemented on the server-side [62], but it outputs an integer between zero and four. The Microsoft and Google PSM will be further discussed in Section 6.2.

A PSM can provide useful feedback when the user selects a weak password, but can also give the user a false sense of security. The PSM does not know if the user has used the password on other sites or if the password is based on personal information.

Adaptive password strength meters (APSM) works as regular PSM, but it also bases the score according to the specific site. The APSM takes the password and the password file of the site as inputs, and estimates the password score. The password file is a noisy model of the password file in order to preserve the secrecy of the password database. The APSM can be more effective than PSM as password distribution tend to be different for different sites [62].

Password Entropy

In 1948 Claude Shannon specified the use of the term entropy in information theory [67], and applied the term to express the amount of actual information in English text [66]. Shannon said, "The entropy is a statistical parameter which measures in a certain sense, how much information is produced on the average for each letter of a text in the language. If the language is translated into binary digits (0 or 1) in the most efficient way, the entropy H is the average number of binary digits required per letter of the original language." [68]. Entropy is commonly used together with guessing entropy and min-entropy to estimate the password strength. The guessing entropy is an estimate on the amount of work that is required to guess the password of a specific user, and the min-entropy measures the difficulty of guessing the easiest single password to guess in a population [66]. The entropy, guessing entropy and min-entropy is equal for randomly created passwords. For truly random passwords, the entropy H in bits is given i equation 2.1 were l denotes the password length and b the number of possible characters.

$$H = \log_2(b^l) \tag{2.1}$$

Example: Assume that the password $p_1 =$ bcxtabf2owale89n has been created using the *Randomly Generated* scheme from Section 2.1.2. $l = 16$ and $b = (26 + 10) = 36$, assuming 26 lower case letters and 10 (0-9) numbers. By applying the equation above the entropy of the password $p_1 =$ bcxtabf2owale89n is $H = \log_2(36^{16}) = 82.71880$ bits.

Cost of Cracking

As we saw in Section 2.3.3 password guessing and cracking can be very resource demanding for an attacker. Expressing the security in economic terms could make people more aware of the actual security associated with their password. The attacker needs to compute a lot of hashes to be able to crack the password. In order to put this in an economic context it is necessary to include cost in electricity and equipment. The cost per computed hash, C_g, can be estimated by assuming that the attacker rents computing time on Amazons cloud EC2 [69]. It is assumed that the attacker rents the *cg1.4xlarge* GPU instance [70]. C_g is given in equation 2.2, where C_{cg1} denotes the cost of renting the *cg1.4xlarge* GPU instance on Amazons EC2 and F_H denotes the number of guesses per hour.

$$C_g = \frac{C_{cg1}}{F_H} \tag{2.2}$$

Renting the *cg1.4xlarge* GPU instance on EC2 costs \$2.1 per hour. Hashcat, presented in Section 2.3.3, in one of the most advanced password recovery tool. Multiple studies of Hashcat on EC2 have revealed that Hashcat can run 2100 million MD5 computations per second which gives $F_H = 2100 \times 10^6 \times 3600$[71, 72]. We saw in Section 2.3.3 that MD5 is one of the fastest hashing algorithms, so the cost per guess, C_g, in USD (\$) given below is the minimum cost per guess. Equation 2.2 gives the following;

$$C_g = \frac{2.1}{2100 \times 10^6 \times 3600} = 2.77778 \times 10^{-13}$$

2.3.5 Randomness

Random numbers are applied in different fields, ranging from cryptography and gambling, to art. The methods of generating random numbers are many and some of them are well know, such as rolling dice, flipping a coin or shuffling cards. The increased need of random numbers led to new ways of generating randomness using computers. Computers follows pre-set instructions, which make it challenging to make a computer produce something random. This challenge resulted in two different methods of generating random numbers using a computer [73]; Pseudo-Random Number Generators (PRNGs) and True Random Number Generators (TRNGs).

PRNGs uses mathematical formulas, cryptographic methods or predetermined tables to create numbers that appear random, but is not truly random. PRNGs are efficient, deterministic and periodic [73], but the periodicity can be negligible.

TRNGs use randomness from physical phenomena as input to generate random numbers. The physical source could be background noise, strokes on keyboard, radio active source and atmospheric noise. TRNGs are not very effective, aperiodic and nondeterministic. Humans tend to have problems with selecting random characters when creating a password [1] because they are afraid of forgetting. It has been shown that humans have difficulties generating random numbers even if it is not a memorable sequence. Even though humans may generate poorly random numbers, it could provide a weak source of entropy that can be used by a TRNG [1].

2.4 Summary

In this chapter we saw that people tend to adopt weak password practices as the number of accounts increases. Several password breaches have revealed very poor password selection. We saw that password composition policies are used to force the user to select a good password and we were introduced to five different password management schemes. Knowing how people tend to select passwords and prioritize usability and security is essential for designing and implementing a password management system. We now know that password composition policies must be considered when designing a password management system. We learned that typical password attacks include *plaintext leak* attacks, *online* attacks and *offline* attacks, and the attacks must be taken into account when designing and evaluating a password management system. We also saw that there is a trade-off between usability and security and that people tend to prioritize usability over security. The human memory was briefly presented alongside with different mnemonic techniques such as PAO-stories. Knowing how people understand and store memories is crucial when designing a usable password management system. In the next chapter we will be introduced to the Shared Cues password management model and see how Shared Cues cope with known attacks, and how it uses PAO-stories to create secrets.

Chapter 3

Shared Cues

Empirical studies on password selection and studies on password habits have revealed significant security issues resulting from poor password selection [14, 15, 16, 1]. Studies have shown that users tend to prioritize usability over security and often adapt weak password management schemes. Shared Cues is a password management scheme that tries to balance the usability and security factors. The goal with the scheme is to provide a usable way of managing multiple passwords while keeping the passwords strong and preventing password reuse.

The scheme Shared Cues is proposed by J. Blocki, M. Blum and A. Datta at the Carnegie Mellon University in the paper "Naturally Rehearsing Passwords". The paper has been published in two versions; short version [1] and full version [6]. This chapter presents the Shared Cues password management model. The first part of the chapter covers issues on usability and security followed by model definitions and notation. The second and third part of the chapter presents the usability and security model, respectively. The chapter is concluded by a technical presentation of the scheme and a usability and security analysis.

3.1 Usability and Security

The average user has normally two types of memory available; his own human memory and persistent memory. The human memory, presented in Section 2.2.1, is associative and private, but also lossy. The persistent memory, such as a note or a file, is reliable, but not private. Shared cues makes use of cue-association pairs. Cue-association pairs, as showed in Section 2.2.2, can strengthen the memory of a password by creating a strong context, a cue, to the password. Since the human memory is lossy the cue-association pairs must be regularly rehearsed otherwise the password is lost. The password is only stored in the user associative memory and never written down.

The scheme provides a level of protection against online attacks, offline attacks and plaintext leak attacks. This will be further elaborated in Section 3.6. As presented in Section 2.3, *keyloggers*, social engineering techniques and *shoulder surfing*, are all threats the user must be aware off, but they are not directly connected to the security of the password management scheme. Hence, Shared Cues along with other available password management schemes does not protect against these threats.

Shared Cues is based on a usability assumption. The usability assumption is an assumption about the human memory. The assumption is that a user who follows a specific rehearsal schedule will maintain the corresponding memory. This assumption has been proven successful in different studies by using various rehearsal schedules [74, 42, 1].

In Shared Cues, a predefined visitation schedule and a given rehearsal requirement is used to approximate the total number of extra rehearsals required to maintain the memory. The total amount of extra rehearsals is used as a measure of the usability of the scheme. To measure the security of the scheme, Shared cues uses a model of a resource bound attacker which performs online, offline and plaintext attacks.

3.2 Definitions

The human memory is associative and Shared Cues uses cue-associations pairs to provide a context, a cue, for the association. We saw in Section 2.2.2, that Cue-association pairs are often used a mnemonic technique because its effectiveness. Table 3.1 shows the Shared Cues notation as defined in [1]. As presented in Table 3.1, $\hat{c} \in \mathrm{C}$ is used to denote a cue and $\hat{a} \in \mathrm{AS}$ is used to denote the corresponding association in a cue-association pair $(\hat{c}, \hat{a})$. The password management scheme stores m sets of cues $c_1, \ldots, c_m \subset \mathrm{C}$ in persistent memory to help the user remember the corresponding password.

The rehearsal schedule is necessary for keeping the cue-association pair $(\hat{c}, \hat{a})$ in associative memory. A rehearsal schedule is sufficient if the user maintain the cue-association $(\hat{c}, \hat{a})$ by following the rehearsal schedule [1]. It was shown in Section 2.2.2 that the time interval $[t_i^{\hat{c}}, t_{i+1}^{\hat{c}})$ used to rehearse the cue-association pair $(\hat{c}, \hat{a})$ could depend on the mnemonic technique as well as prior rehearsals, i. The function $\mathrm{R} : \mathrm{C} \times \mathbb{N} \rightarrow \mathbb{R}$ specifies the rehearsal requirements and R is used to denote a set of rehearsal functions.

Definition 1 ([1]) *A rehearsal schedule for a cue-association pair* $(\hat{c}, \hat{a})$ *is a sequence of times* $t_0^{\hat{c}} < t_1^{\hat{c}} < \ldots$ *For each* $i \geq 0$ *we have a rehearsal requirement. The cue-association pair* $(\hat{c}, \hat{a})$ *must be rehearsed at least once during the time window* $[t_i^{\hat{c}}, t_{i+1}^{\hat{c}}) = \{x \in \mathbb{R} \,| t_i^{\hat{c}} \leq t_{i+1}^{\hat{c}}\}$.

Table 3.1: Shared Cues Notation

Symbol	**Denotes**
P	Space of possible passwords
$p_1, \ldots, p_m \in \mathrm{P}$	Set of all m passwords
$\hat{c} \in \mathrm{C}$	One cue
$c \subset \mathrm{C}$	One set of cues
$\mathrm{C} = \bigcup_{i=0}^{m} c_i$	The set of all cues
$n = \lvert\mathrm{C}\rvert$	Number of cue-association pairs the user must remember
$\hat{a} \in \mathrm{AS}$	One association
$(\hat{c}, \hat{a})$	One cue-association pair
$R : \mathrm{C} \times \mathbb{N} \to \mathbb{R}$	Rehearsal requirements
$\mathrm{R} \in R$	One rehearsal schedule
A_i	Account i
G_m	Password generator
$k \in \mathrm{K}$	Users knowledge
$b \in \{0,1\}*$	Random bits
$\tau_0^i < \tau_1^i < \ldots$	Visitation schedule
$X_{t,\hat{c}}$	Number of extra rehearsals for a $(\hat{c}, \hat{a})$-pair in time interval $[t_i^{\hat{c}}, t_{i+1}^{\hat{c}})$
$\sum_{\hat{c} \in C} X_{t,\hat{c}}$	Total number of rehearsals for all $(\hat{c}, \hat{a})$-pairs in time interval $[0,t]$
σ	Association strength
$\hat{\lambda}$	Visitation schedule
U	User
$O_{S,q}$	guess-limited oracle
(n, l, γ)	n=total number of cues l=number or cues for each account γ=maxium number of cues shared between accounts
h	Offline attack
s	Online attack
r	Plaintext leak attack
A	Attacker
q	Number of guesses
δ	Probability that the attacker wins

> *Example:* A rehearsal schedule with exponentially decreasing rehearsal frequency could be given as the following time-interval[day 1, day 2), [day 2, day 4), [day 4, day 8), [day 8, day 16). The cue-association pair $(\hat{c}, \hat{a})$ must be rehearsed during each of these intervals in order to maintain the association.

Visitation schedule for the account A_i is a sequence of numbers $\tau_0^i < \tau_1^i < \ldots$ which represents the times account A_i was visited by the user. The visitation schedule is modelled using a random process with a known parameter, λ_i, based on the average time between visits to account A_i, $\mathrm{E}[\tau_{j+1}^i - \tau_j^i]$.

Definition 2 ([1]) *The rehearsal requirement* $[t_i^{\hat{c}}, t_{i+1}^{\hat{c}})$ *is naturally satisfied by a visitation schedule* $\tau_0^i < \tau_1^i < \ldots$ *if* $\exists j \in [m], k \in \mathbb{N} s.t. \hat{c} \in c_j \, and \, \tau_k^j \in [t_i^{\hat{c}}, t_{i+1}^{\hat{c}})$*.*

$$X_{t,\hat{c}} = |\{i | t_{i+1}^{\hat{c}} \leq t \wedge \forall j, k, (\hat{c} \notin c_j \vee \tau_k^j \notin [t_i^{\hat{c}}, t_{i+1}^{\hat{c}}))\}|$$

$X_{t,\hat{c}}$ *denote the number of rehearsal requirements that are not naturally satisfied by the visitation schedule during the time interval* $[0, t]$*.*

> *Example:* Considering the following rehearsal requirement for a cue-association pair $(\hat{c}, \hat{a})$; [day 1, day 2), [day 2, day 4), [day 4, day 8), [day 8, day 16). If the visitation schedule states that the user will log into account A_i which include the cue-association pair $(\hat{c}, \hat{a})$ one time or more in all the four time intervals, $X_{t,\hat{c}} = 0$. If the visitation schedule assumes a log in to account A_i in two of the four intervals, $X_{t,\hat{c}} = 2$.

A rehearsal requirement $[t_i^{\hat{c}}, t_{i+1}^{\hat{c}})$ can be satisfied naturally if the user visits account A_i. If the cue-association pair $(\hat{c}, \hat{a})$ is not naturally rehearsed during the interval $[t_i^{\hat{c}}, t_{i+1}^{\hat{c}})$, the user must perform an extra rehearsal. Shared Cues uses rehearsal requirements and visitation schedules to quantify the usability of the password management scheme by measuring the number of extra rehearsals the user need to perform. $X_{t,\hat{c}}$ is used to denote the total number of extra rehearsals required to maintain the cue-association pair $(\hat{c}, \hat{a})$ during the interval $[t_i^{\hat{c}}, t_{i+1}^{\hat{c}})$. $X_t = \sum_{\hat{c} \in C} X_{t,\hat{c}}$ is used to denote the total number of extra rehearsal during the time interval $[0, t]$ to maintain all cue-association pairs.

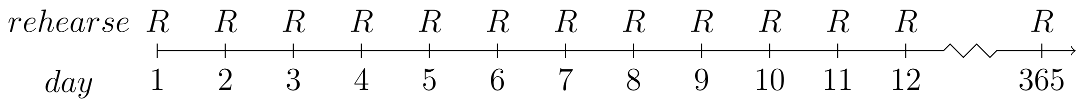
Shared Cues password management scheme includes a generator G_m and a rehearsal schedule $\mathrm{R} \in R$. The generator $G_m(k, b, \vec{\lambda}, \mathrm{R})$ utilizes the users knowledge $k \in \mathrm{K}$, random bits $b \in \{0,1\}^*$ to generate passwords $p_1, \ldots, p_m$ and public cues $c_1, \ldots, c_m \subseteq \mathrm{C}$. The rehearsal schedule $\mathrm{R} \in R$ is at initial start-up equal for all cues, but the rehearsal schedule will adapt for each cue according to the account logins. G_m may take use of the rehearsal schedule R and visitation schedule $\vec{\lambda} = (\lambda_1, \ldots, \lambda m)$ of each site to minimize $E[X_t]$. The code for the generator G_m and the cues $c_1, \ldots, c_m$ are public and stored in persistent memory. The passwords $p_1, \ldots, p_m$ are private and must be memorized and rehearsed using rehearsal schedule R in order to maintain the cue-association pair $(\hat{c}, \hat{a})$ in the associative memory of the user.

Definition 3 ([1]) *A password management scheme is a tuple* (G_m, R)*, where* G_m *is a function* $G_m : K \times \{0,1\} \times \mathbb{R}^m \times R \rightarrow (P \times 2^C)^m$ *and a* $\mathrm{R} \in R$ *is a rehearsal schedule which the user must follow for each cue.*

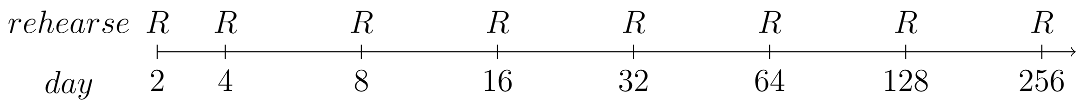
3.3 Usability Model

Users tend to prioritize usability over security [14] when choosing a password management scheme. Share Cues measures the additional effort the user must invest in rehearsing the passwords. The usability depend on the rehearsal requirement for each cue, visitation schedule for each site and the number of cues the user need to maintain. The usability of the password scheme is measured in numbers of extra rehearsals, $X_{t,\hat{c}}$, the user needs to perform to maintain the associations in memory. Shared Cues consider three rehearsal assumptions; Constant Rehearsal Assumption (CR), Expanding Rehearsal Assumption (ER) and Squared Rehearsal Assumption (SQ). CR, ER and SQ are defined in definition 4, 5 and 6 respectively. The constant, σ, is used to denote the strength of the mnemonic technique used to memorize the cue-association pair $(\hat{c}, \hat{a})$.

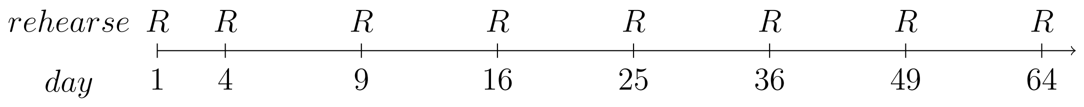
Definition 4 ([1]) *Constant Rehearsal Assumption (CR): The rehearsal schedule is given by* $R(\hat{c}, i) = i\sigma$


rehearse R R R R R R R R R R R R R
day 1 2 3 4 5 6 7 8 9 10 11 12 365

Definition 5 ([1]) *Expanding Rehearsal Assumption (ER): The rehearsal schedule is given by* $R(\hat{c}, i) = 2^{i\sigma}$



Definition 6 ([6]) *Squared Rehearsal Assumption (SQ): The rehearsal schedule is given by* $R(\hat{c}, i) = i^2\sigma$



The CR assumes that memories are not strengthen every time the user rehearses, hence the user must rehearse every σ days. The ER and SQ in contrast, assume that memories are strengthen every time the user rehearses and therefore the number of rehearsals decreases over time. Studies have shown that the availability of a memory is dependent on recency [75] and the pattern of previous rehearsals. The difference between SQ and ER is not significant, but ER is consistent with known memory studies [74, 76]. For ER this means that a password which has been rehearsed i times does not need to be rehearsed again for $2^{i\sigma}$ to satisfy the requirement $[2^{i\sigma}, 2^{i\sigma+\sigma})$. In SQ the password must be rehearsed with an $i^2\sigma$-interval. Table 3.2 and Table 3.3 shows the ER and CR with various association strength, σ, and natural rehearsal rate, λ.

Table 3.2: Expanding Rehearsal Assumption: $X_{365,c}, \lambda_c, \sigma$ [6]

λ $(\frac{visits}{days})$	2	1	$\frac{1}{3}$	$\frac{1}{7}$	$\frac{1}{31}$
$\sigma = 0.1$	0.686669	2.42166	5.7746	7.43555	8.61931
$\sigma = 0.5$	0.216598	0.827594	2.75627	4.73269	7.54973
$\sigma = 1$	0.153986	0.521866	1.56788	2.61413	4.65353
$\sigma = 2$	0.135671	0.386195	0.984956	1.5334	2.57117

Table 3.3: Constant Rehearsal Assumption: $X_{365,c}, \lambda_c, \sigma$ [6]

λ $\left(\frac{visits}{days}\right)$	2	1	$\frac{1}{3}$	$\frac{1}{7}$	$\frac{1}{31}$
$\sigma = 1$	49.5327	134.644	262.25	317.277	354.382
$\sigma = 3$	0.3024	6.074	44.8813	79.4756	110.747
$\sigma = 7$	0.0000	0.0483297	5.13951	19.4976	42.2872
$\sigma = 31$	0.0000	0.0000	0.0004	0.1432	4.4146

The visitation schedule for a site is dependent on the user. Shared Cues uses a Poisson process with parameter λ_i to model the visitation schedule for a site. Shared Cues defines four types of users and λ_i is used to denote how often a site is visited. The visitation schedule is illustrated in Table 3.4 and it presents the number of sites visited with frequency λ for different user types.

Table 3.4: Visitation Schedules [1]

Schedule - $\lambda(\frac{visits}{days})$	$\frac{1}{1}$	$\frac{1}{3}$	$\frac{1}{7}$	$\frac{1}{31}$	$\frac{1}{365}$
Very Active	10	10	10	10	35
Typical	5	10	10	10	40
Occasional	2	10	20	20	23
Infrequent	0	2	5	10	58

Shared Cues uses the number of extra rehearsals as a measure of usability of the scheme. This can be computed for other password management scheme in order to compare the usability based on the usability assumptions defined. Theorem 1 define the expected value of extra rehearsals needed to maintain all the cue-association pairs and it follows the linearity of expectations as stated in Lemma 1. Proof of Theorem 1 and Lemma 1 can be found in Appendix A in [6].

Theorem 1 ([1]) *Let* $i_{\hat{c}\star} = (argmax_x t^{\hat{c}x} < t) - 1$ *then*

$$E[X_t] = \sum_{\hat{c} \in C} \sum_{i=0}^{i_{\hat{c}\star}} exp(-(\sum_{j:\hat{c} \in C_j} \lambda_j)(t_{i+1}^{\hat{c}} - t_i^{\hat{c}}))$$

Lemma 1 ([1]) *Let* $S_{\hat{c}} = \{i | \hat{c} \in c_i\}$ *and* $\lambda_{\hat{c}} = \sum_{i \in S_{\hat{c}}} \lambda_i$ *then the probability that the cue* $\hat{c}$ *is not naturally rehearsed during time interval* $[a, b]$ *is* $exp(-\lambda_{\hat{c}}(b - a)))$.

Table 3.5 presents the number of expected extra rehearsals over the first year for both CR and ER. L represents the *Lifehacker* password management scheme and SRI represents *Strong Random and Independent*, which were presented in Section 2.1.2.

Table 3.5: $E[X_{365}]$: Expected extra rehearsals over the first year [1]

Assumption	CR ($\sigma = 1$)		ER ($\sigma = 1$)		SQ ($\sigma = 1$)	
Scheme	L	SRI	L	SRI	L	SRI
Very Active	≈ 0	23.396	0.023	420	≈ 0	794.7
Typical	0.014	24.545	0.084	456.6	≈ 0	882.8
Occasional	0.05	24.652	0.12	502.7	≈ 0	719.02
Infrequent	56.7	26.751	1.2	564	0.188	1176.4

3.4 Security Model

The values G_m, k and $\mathrm{C} = \bigcup_{i=0}^{m} c_i$ are assumed to be available for a potential attacker. Many breaches occur because users choose to put personal information in their password, such as hobbies and birth dates, assuming that this information is private. Shared Cues assume that the adversary has background information, $k \in K$, about the user. The public cues $\mathrm{C} = \bigcup_{i=0}^{m} c_i$ are stored in persistent memory and the password management scheme G_m is assumed known to the attacker. The secrecy of the scheme lies in the random string b used by G_m to generate $p_1, \ldots, p_m$.

Measuring the security and password strength can be performed in different ways. We saw in Section 2.3.4 that min-entropy and password meters, which is often used to measure password strength, have in several studies been proven to be a weak measure of password security [1, 62]. Shared Cues measure the security of the password scheme by estimating the cost of guessing, cracking, the password by a potential attacker. Shared Cues model three types of attacks; online attack, offline attack and plaintext leak attack, presented in Section 2.3.3. Shared Cues model online and offline attacks by using a guess-limited oracle. Let $S \subseteq [m]$ be a set of indexes which represent each account. It is assumed that an attacker can perform an offline attack for accounts $\{A_i | i \in \mathrm{S}\}$ if he is able to steal the password hashes. The guess-limited oracle $O_{S,q}$ is a black-box with the following behaviour;

- $O_{S,q}$ stops answering queries after q queries.
- $\forall i \notin \mathrm{S}, O_{S,q}(i,p) = \perp 3$
- $\forall i \in \mathrm{S}, O_{S,q}(i,p_i) = 1$
- $\forall i \in \mathrm{S}, p \neq p_i, O_{S,q}(i,p) = 0$

Shared Cues presents a game based definition of security. The user U starts with knowledge $k \in K$, visitation schedule $\hat{\lambda} \in \mathbb{R}^m$, random sequence of bits $b \in \{0,1\}^\star$ and a rehearsal schedule $\mathrm{R} \in R$. The user runs $G_m(k, b, \hat{\lambda}, \mathrm{R})$ to obtain m passwords $p_1 \ldots p_m$ and public cues $c_1 \ldots c_m$ for accounts $A_1 \ldots A_m$. The attacker is given $k, G_m, \hat{\lambda}$ and the public cues $c_1 \ldots c_m$. Online attack is when the attacker is given black-box access to the guess-limited oracle $O_{\{i\},s}$ for each $i \in [m] - \mathrm{S}$. Offline attack is when the attacker adaptively selects a set $\mathrm{S'} \subseteq [m] s.t. |\mathrm{S'}| \leq h$ and is given black-box access to the guess-limited offline oracle $O_{S',q}$. Plaintext leak attack is when the attacker selects a set $\mathrm{S} \subseteq [m] s.t |\mathrm{S}| \leq r$ and receives p_i for each

$i \in \mathrm{S}$. The attacker wins if he is able to obtain (j, p) where $j \in [m] - \mathrm{S}$ and $p = p_j$. $\mathbf{AttWins}(k, b, \hat{\lambda}, G_m, A)$ is used to denote the event that the attacker wins. It was illustrated in Section 2.3.4 that the security can be measured by putting an economic upper bound on q, which limits the computational resources available for an attacker. Shared Cues use $q_{\$10^6}$ in the security analysis, which defines the number of possible guesses an attacker can try if he invest $\$1,000,000$ in cracking the password.

Definition 7 ([1]) *A password management scheme G_m is $(q, \delta, m, s, r, h)-$secure if for every $k \in K$ and attacker A we have $Pr_b[\boldsymbol{AttWins}(k, b, \hat{\lambda}, G_m, A)] \leq \delta$.*

We saw in Section 3.3, supported by Theorem 2, that sharing cues across accounts improves the usability because the number of cue-associations is reduced and the rate of natural rehearsals increases. Theorem 2 shows that public cues can be securely shared across accounts if the public cues $\{c_1, \ldots c_m\}$ are a (n, l, γ) sharing set family. Proof of Theorem 2 can be found in appendix A in [6].

Definition 8 ([1]) *A set family $\mathrm{S} = \{\mathrm{S}_1 \ldots \mathrm{S}_m\}$ is a (n, l, γ) sharing set family if $|\bigcup_{i=1}^{m} \mathrm{S}_i| = n$, $|\mathrm{S}_i| = l$ for each $\mathrm{S}_i \in S$ and $|\mathrm{S}_i \cap \mathrm{S}_j| \leq \gamma$ for each pair $\mathrm{S}_i \neq \mathrm{S}_j \in S$.*

Theorem 2 ([1]) *Let $\{c_1, \ldots, c_m\}$ be a (n, l, γ) sharing set of m public cues produced by G_m. If each $a_i \in \mathrm{AS}$ is chosen uniformly at random then G_m satisfies (q, δ, m, s, r, h)-security for $\delta \leq \frac{q}{|\mathrm{AS}|^{l-\gamma r}}$ and any h.*

3.5 The Scheme

Shared Cues uses the Chinese Remainder Theorem (CRT) to construct (n, l, γ) sharing set families. As lemma 2 states, the input $n_1, \ldots, n_l$ to Algorithm 1 must be co-prime in order to use the CRT.

Algorithm 1 CRT $(m, n_1, \ldots, n_l)$ [1]

```
1: Input: $m$ and $n_1, \ldots, n_l$          ▷ $n_1, \ldots, n_l$ must be pairwise co-prime
2: for $i = 1 \rightarrow m$ do                ▷ $m$ is the number of passwords
3:     $\mathrm{S}_i \leftarrow \emptyset$
4:     for $j = 1 \rightarrow l$ do
5:         $\mathrm{N}_j \leftarrow \sum_{i=1}^{j-1} n_j$
6:         $\mathrm{S}_i \leftarrow \mathrm{S}_i \cup \{(i \, mod \, n_j) + \mathrm{N}_j\}$
7: Return: $\{\mathrm{S}_1, \ldots, \mathrm{S}_m\}$
```

Lemma 2 ([1]) *If the numbers $n_1 < \ldots < n_l$ are pairwise co-prime and $m \leq \prod_{i=0}^{\gamma+1} n_i$ then Algorithm 1 returns a $(\sum_{i=1}^{l} n_i, l, \gamma)$-sharing set of public cues.*

Algorithm 2 divide each PAO story in two parts. Each cue $\hat{c}$ consists of two pictures and the corresponding two persons with the a label (action or object). In this way

Algorithm 2 CreatePAOStories [1]

```
1: Input: $n$, random bits $b$, images $\mathrm{I}_1, \ldots, \mathrm{I}_n$, names $\mathrm{P}_1, \ldots, \mathrm{P}_n$
2: for $i = 1 \rightarrow n$ do
3:     $a_i \leftarrow$ ACT                                  ▷ Using random bits
4:     $o_i \leftarrow$ OBJ
5: for $i = 1 \rightarrow n$ do                ▷ Split PAO stories to optimize usability
6:     $\hat{c}_i \leftarrow ((\mathrm{I}_i, \mathrm{P}_i, \text{'ACT'}), (\mathrm{I}_{i+1modn}, \mathrm{P}_{i+1modn}, \text{'OBJ'})$
7:     $\hat{a}_i \leftarrow (a_i, o_{i+1modn})$
8: Return: $\{\hat{c}_1, \ldots, \hat{c}_n\}$, $\{\hat{a}_1, \ldots, \hat{c}_a\}$
```

the user will rehearse both the i'th and the $i+1$'th PAO story, but does only need to input either the action or the object associated with the picture-person pair.

3.6 Security and Usability Analysis

Shared Cues is analysed with three different configurations. SharedCues-0 uses a (9,4,3)-sharing set family of public cues, with $m = \binom{9}{4} = 126$ subsets of size 4. SharedCues-1 uses a (43,4,1)-sharing set family of public cues, where $m = 90$ computed by Algorithm 1 using $(n_1, n_2, n_3, n_4) = (9, 10, 11, 13)$. SharedCues-2 uses a (60,5,1)-sharing set constructed by Algorithm 1 using $m = 90$ and $(n_1, n_2, n_3, n_4) = (9, 10, 11, 13, 17)$. The security results in Table 3.7 assume 140 actions and 140 objects, $|AS| = 140^2$, and that the attacker would invest \$1000000 in cracking the password. In the case of no offline attacks, $h = 0$, the security is computed using $m \leq 100$.

The security results assume that the bcrypt hashing algorithm is used and the guessing cost and number of computed hash calculations as given in Table 3.8 and Table 3.9 respectively. As we saw in Section 2.3.3 bcrypt is one of the slowest hash functions, hence the results are a best-case evaluation. Unfortunately, MD5 and SHA is more often used as hash function [59, 60, 61, 57]. Table 3.6 illustrate the number of extra rehearsals the first year for the three different configurations of Shared Cues.

Table 3.6: $E[X_{365}]$: Expected extra rehearsals over the first year for Shared Cues [1]

Assumption	CR ($\sigma = 1$)			ER ($\sigma = 1$)			SQ ($\sigma = 1$)		
Scheme	SC-0	SC-1	SC-2	SC-0	SC-1	SC-2	SC-0	SC-1	SC-2
Very Active	≈ 0	1,309	2,436	≈ 0	3.93	7.54	≈ 0	2.77	5.88
Typical	≈ 0.42	3,225	5,491	≈ 0	10.89	19.89	≈ 0	7.086	12.74
Occasional	≈ 1.28	9,488	6,734	≈ 0	22.07	34.23	≈ 0	8.86	16.03
Infrequent	≈ 723	13,214	18,764	≈ 2.44	199.77	173.92	2.08	71.42	125.24

Table 3.7: Shared Cues $(q_{\$10^6}, \delta, m, s, r, h)$-security [1]

Offline Attack	$h = 0$			$h > 0$		
(n, l, γ)-sharing	$r = 0$	$r = 1$	$r = 2$	$r = 0$	$r = 1$	$r = 2$
$(n, 4, 3)$ - SC-0	2×10^{-15}	0.011	1	3.5×10^{-7}	1	1
$(n, 4, 1)$ - SC-1	2×10^{-15}	4×10^{-11}	8×10^{-7}	3.5×10^{-7}	0.007	1
$(n, 5, 1)$ - SC-2	1×10^{-19}	2×10^{-15}	4×10^{-11}	1.8×10^{-11}	3.5×10^{-7}	0.007

Table 3.8: Computed hash given \$1000000 [6]

Hash Function	\$1000000
SHA1	10^{16}
MD5	9.1×10^{15}
BCRYPT (L12)	5.2×10^{10}

Table 3.9: Guessing costs [6]

Hash Function	$F_H(guesses/hour)$	$C_g(\$)$
SHA1	$\approx 576{\times}10^6$	1×10^{-10}
MD5	$\approx 561{\times}10^6$	1.1×10^{-10}
BCRYPT (L12)	$\approx 31{\times}10^3$	1.94×10^{-5}

Moore's law and modern cracking techniques have outdated the numbers in Table 3.8 and Table 3.9. The table is included in order to understand the security results and analysis of Shared Cues given Table 3.7. As presented in Section 2.3.4, Hashcat can run 2100 million MD5 guesses per second. This will be further discusses in Section 6.2.

3.7 Summary

In this chapter we were introduced to the Shared Cues password management model proposed by J.Blocki et al. in the paper "Naturally Rehearsing Passwords". Shared Cues is built on the usability assumption that a user who follows a specific rehearsal schedule will successfully maintain the corresponding memory. Shared Cues uses this usability assumption to create rehearsal schedules, which balances usability and security. We now know that the Chinese Remainder Theorem can be used to securely share cues across multiple accounts to improve usability. We saw that Shared Cues uses PAO-stories in order to create secrets which will be saved in associative memory and used to derive passwords. In the next chapter we will see that the designed password management system is built on many of the same principles and assumptions as Shared Cues, and that it uses rehearsal schedules, PAO-stories and sharing sets to balance the usability security trade-off.

Chapter 4

Design

PassCue is a password management system that is based on the Shared Cues password management model. This chapter presents the PassCue design including design choices, design specification and parameter evaluation. The first part of this chapter presents the PassCue functionality and how PassCue can be used to create passwords. The second part covers the design choices, which forms the basis for the PassCue design requirements. The PassCue design including system flow and underlying database structure is presented in the third part of the chapter.

4.1 Functionality

This section presents the functionality of PassCue and how PassCue works. Important parameters will be introduced in order to highlight the factors which affect the security and usability of PassCue.

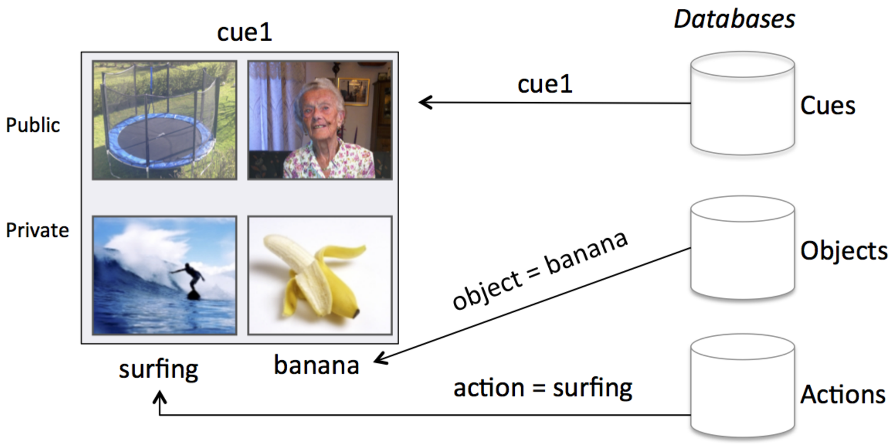


Figure 4.1: Creating a PAO-story

The PassCue system consist of n public cues supplied by the user. Each of the public cues include a picture of a known person and a picture of a known place/location/background. One action and one object are randomly selected from a large set and assigned to the public cue. The example in Figure 4.1 illustrates how a public cue is assigned an association. In the example, the user selected a public cue that consists of a trampoline and his grandmother. The randomly selected action is *surfing* and the object is *banana*. The action and object represents the association of the cue. The user should imagine his grandmother surfing a banana on the trampoline in his garden. The association is private and only displayed the first time a public cue is used. After initialization, the association is deleted and non-retrievable. The user must keep the association in his associative memory. The next time this particular cue is used, only the public cue with a picture of a trampoline and his grandmother will be visible. The user must remember what his grandmother did on the trampoline.

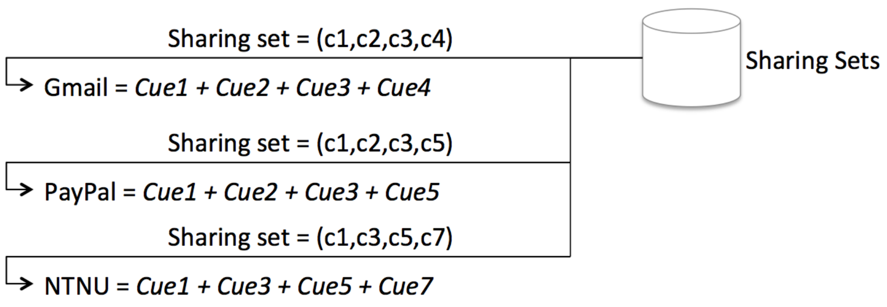

Sharing set = (c1,c2,c3,c4)
Gmail = Cue1 + Cue2 + Cue3 + Cue4
Sharing set = (c1,c2,c3,c5)
PayPal = Cue1 + Cue2 + Cue3 + Cue5
Sharing set = (c1,c3,c5,c7)
NTNU = Cue1 + Cue3 + Cue5 + Cue7
Sharing Sets

Figure 4.2: Sharing Set Distribution

The distribution of the public cues to each account is performed using (n, l, γ)-sharing sets. n is the total number of public cues, l is how many cues used for each account, and γ is the maximum number of cues that can be shared between two accounts. A sharing set is selected for each new account. The sharing set defines which cues the account should use and the number of cues, l. Figure 4.2 shows that the account *Gmail* uses the four cues; cue 1, cue 2, cue 3 and cue 4. The account *PayPal* uses cue 1, cue 2, cue 3 and cue 5, while *NTNU* uses cue 1, cue 3, cue 5 and cue 7. *Gmail* and *PayPal* share cue 1, 2 and 3, which means that a sharing set with $\gamma = 3$ is used. *NTNU* share cue 1 and cue 3 with the two accounts. Assuming that account *Gmail* was the first account to use cue 1, 2, 3 and 4, the association for all the public cues will be displayed. For account *Paypal*, the association for cue 1, 2 and 3 will not be displayed as they were displayed earlier for the *Gmail* account.

When a cue is used for the first time, a rehearsal schedule is created. The objective of the rehearsal schedule is to assure that the association assigned to the public cue is kept in the associative memory of the user, and not forgotten. The user is notified according to the rehearsal schedule when he needs to rehearse a specific cue-association in order to maintain the memory of the association.

Figure 4.3 shows how the private associations can be used to derive a password. The associations for each of the public cues are only stored in associative memory and are not available for a potential attacker. Account *Gmail* uses cue 1, 2, 3

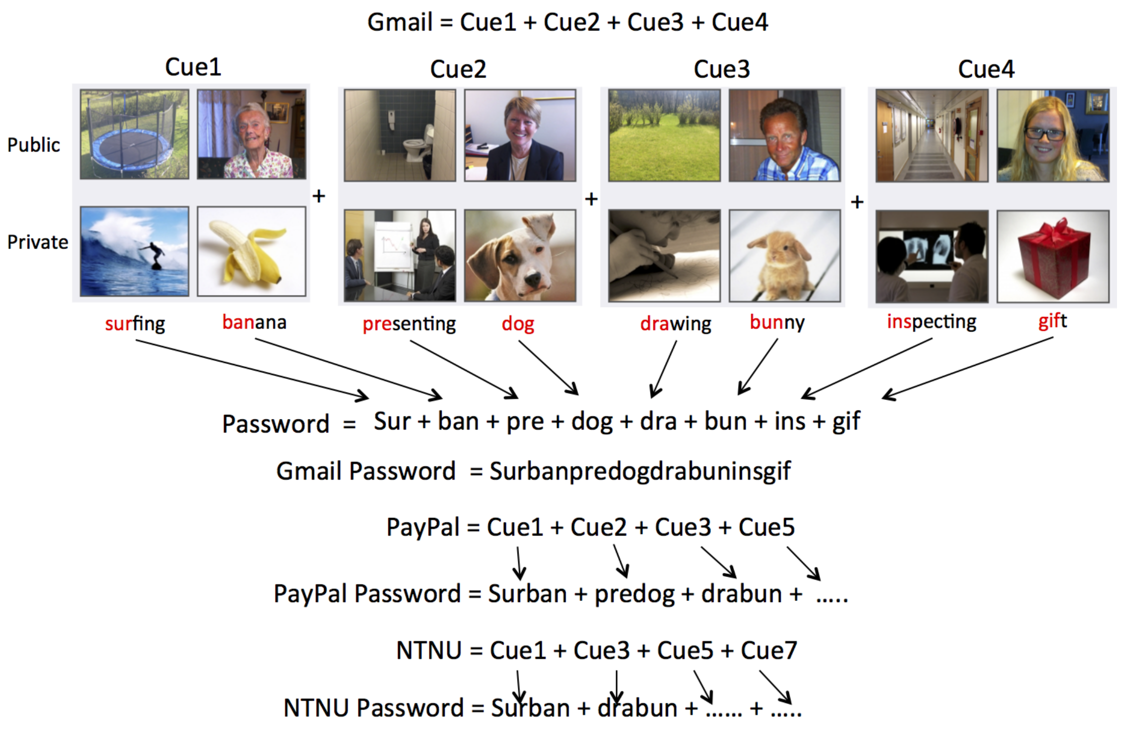

Gmail = Cue1 + Cue2 + Cue3 + Cue4
Cue1
Cue2
Cue3
Cue4
Public
Private
+
surfing
banana
presenting
dog
drawing
bunny
inspecting
gift
Password = Sur + ban + pre + dog + dra + bun + ins + gif
Gmail Password = Surbanpredogdrabuninsgif
PayPal = Cue1 + Cue2 + Cue3 + Cue5
PayPal Password = Surban + predog + drabun + ...
NTNU = Cue1 + Cue3 + Cue5 + Cue7
NTNU Password = Surban + drabun + ... + ...

Figure 4.3: Deriving password from actions and objects

and 4. All the cues consist of two public pictures and a private association of two pictures. In cue 1, the user selected a picture of a trampoline and his grandmother as public pictures. For cue 2, he selected a picture of a toilet and his mother. In cue 3, a picture of his garden was selected as background picture and his father as person picture. Cue 4 uses a picture of a hallway and his sister. The user has chosen to use the first three letters of each action and object to create a password for account *Gmail*.

The associations are only displayed when a cue is initialized and the user must use the public pictures to retrieve the associations from memory. For the first part of the *Gmail* password the user must imagine his grandmother on the trampoline, and retrieve the memory *surfing* and *banana*. The second part is his mother on the toilet *presenting* a *dog*. The third cue is his father in the garden *drawing* a *bunny*. For the last part of the *Gmail* password, cue 4 displays a picture of the user's sister and a hallway. The user must ask himself; "What did my sister do in the hallway?". The answer from associative memory is *inspecting* a *gift*. By combining the three first letters in the action and object for each of the four cues, the *Gmail* password is derived as "Surbanpredogdrabuninsgif". Since account *Gmail* and *PayPal* share cue 1, 2 and 3, the 18 first letters are the same and the last six differ. For account *NTNU*, the first six is similar, but the 13th to the 18th letter in *Gmail* is letter seven to twelve in the *NTNU* password.

The password is created using the first three letters of the action and object for each of the associations. The first character should be capitalized in order to cope with common password composition policies, as we saw in Section 2.1.1. This will be further discussed in Section 4.2.5. The whole action/object word could have been used and resulted in the same entropy, see the security analysis in Section 6.2. The reason why this is not recommended is that the password can be rejected because many PCPs restrict the use of dictionary words. If the online service has a restriction on maximum password length, the user might need to choose two instead of three letters for each action/object. How maximum password length affects the password security is discussed in Section 6.2.

4.2 Design Choices

This section covers the design choices and the resulting parameters when designing the PassCue application. In order to design a system with the functionality as defined in the previous section, several design choices had to be made. It was necessary to explore the following:

1. How should the public cues be created?
2. What type of sharing set should be used?
3. Which rehearsal schedule should be used?
4. What is the targeted platform for the design?
5. How can the password management system be designed to cope with PCPs?
6. How to determine the association set size?

4.2.1 Public Cues

The Shared Cues model is based on public cues, which trigger specific associations saved in the associative memory of the user. As illustrated in Section 2.2.1, the information saved in associative memory is only available for the user, hence not accessible for an attacker. A public cue consists of an image of a known person (e.g. your grandmother) and a background type of image (e.g. your trampoline). A background picture is included in each public cue for the purpose of creating a setting or an arena for the PAO-story. Each of the public cues are to be connected to a private picture of an action (e.g. surfing) and an object (e.g. a banana), together referred to as the association. This is a mnemonic technique that is based on the PAO technique presented in Section 2.2.2.

The resulting story for this specific cue will be “Your grandmother is *surfing* a *banana* on the trampoline”. An attacker will only have access to the public cue which is the picture of your grandmother and the trampoline in your garden. The attacker has no way of finding the secret association (*surfing* and *banana*), which is stored in the user’s associative memory.

Since the association is only stored in the associative memory of the user, it is important that the associations are strong in order to prevent the user from forgetting the association. We saw in Section 2.2.1 that the human memory is temporally limited in remembering sequences of items, hence it is important to create strong associations. Studies have shown that people tend to create stronger associations when they see pictures of people they know and a place they can imagine an action taking place at [49]. As a consequence, it is essential that the pictures that represent the public cues are known to the user. In the PassCue application it is important that the user supply pictures from his phone when initializing the public cues. In order to create strong association the user is forced to use private pictures from the photo library.

The mnemonic technique called memory palace was presented in Section 2.2.2. By allowing the user to input personal images, PassCue can be used together with the memory palace technique. The use of memory palace can increase the usability further and help to create strong associations. When the user initializes PassCue, the user takes pictures of places/locations inside his house. PassCue will generate absurd stories based on the association set, the person picture and the picture of a location inside the house. The absurd stories will be used to derive password to different accounts.

4.2.2 Sharing Set

We learned in Section 3.5 that the Shared Cues model uses sharing sets of public cues in order to share cues between accounts. The security and usability is dependent on how the sharing sets are constructed. As presented in Section 4.2.1, the public cues should be supplied by the user in order to create strong associations. A (n, l, γ)-sharing set consists of n public cues, l public cues are used to create a password for each account, and maximum γ public cues can be shared between two accounts. The Shared Cues security results in Section 3.6 and Theorem 2, illustrate that the security is dependent on the choice of l and γ.

The PassCue strive to provide a usable and secure way to manage passwords. There is a clear usability-security trade-off when defining l and γ. l defines the number of cues used to derive a password for each account. A large l require the user to invest a lot of effort every time he is deriving a password and logging in to a system. As the usability decreases, the security increases, and a large l provides higher security. Minimizing l would also affect the usability and security, as well as the number of possible passwords generated by PassCue. PassCue is designed with $l = 4$ because it provides reasonable security and usability as we saw in Table 3.6 and Table 3.7.

The sharing of the public cues between the accounts is essential in order to maintain the required associations without forcing the user to invest additional time. γ should be close to l in order to utilize the sharing property and increase usability. We saw in Section 3.3 that the usability can be measured in calculating the number of extra rehearsals using Theorem 1. In order to minimize the number of extra rehearsals, the PassCue application is designed with $\gamma = 3$. In order to preserve the security, the number of cues must be larger than the maximum number of cues two accounts can share. This is to avoid two accounts having identical passwords.

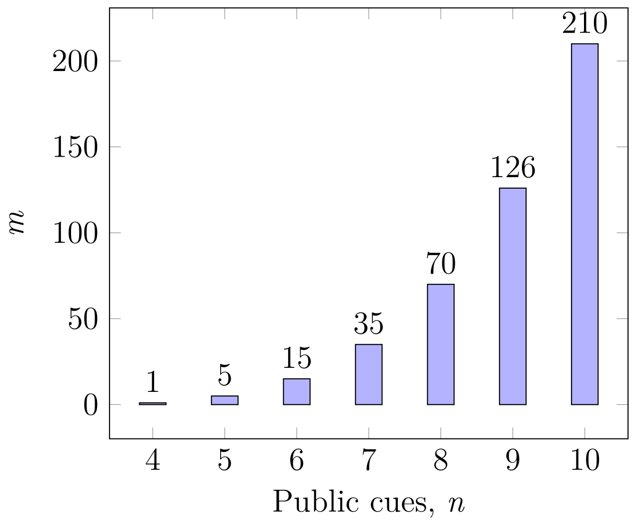


Figure 4.4: m assuming $l = 4$ and $\gamma = 3$

The number of public cues, n, does not affect the security, but it determines the number of accounts/passwords that can be generated with PassCue. Considering $l = 4$ and $\gamma = 3$, the number of unique passwords PassCue can generate is given in Figure 4.4. Choosing $n = 7$, enables PassCue to generate 35 passwords. $n = 8$ enables the generation of 70 different passwords. This may be sufficient for the everyday user, but as the technology develops the need for new accounts increases. By increasing the number of cues with one, $n = 9$, increases the account potential with over 50 accounts. Choosing $n = 9$ enables PassCue to generate 126 unique passwords, which should be enough even for the most active user. PassCue is designed with a $(9, 4, 3)$-sharing set, where $n = 9$, $l = 4$ and $\gamma = 3$.

In the usability and security results in Section 3.6 we saw that the (9,4,3)-sharing set have proved sufficient security and high usability. It is reasonable to assume that the user has nine pictures of a person and nine pictures of a background on his phone, or that it is obtainable using the phone's camera. The (9,4,3)-sharing set is incrementally designed. This means that the first six accounts all share cue 1, 2 and 3. After creating six accounts in PassCue, the user already knows all the cue-association pairs. The next 120 passwords generated by PassCue uses cues the user already knows. This makes it easier for the user to start using PassCue and it significantly increases the usability. This is further discussed in Section 6.1. The full (9,4,3)-sharing set given in Appendix A.

We saw in Section 3.5 that the sharing set must be generated with Algorithm 1 in order to invoke the Chinese Remainder Theorem. In the case where $l - \gamma = 1$, the sharing set can be generated by taking the $\binom{n}{l}$ subsets using Algorithm 3. This will be further discussed in Section 4.4.

4.2.3 Rehearsal Schedule

We saw in Section 3.3 that Shared Cues is built on a usability assumption. PassCue is built on the same usability assumption. The usability assumption is that memory is strengthen by rehearsal and a person who follows a specific rehearsal schedule will successfully maintain the corresponding memory. Section 3.3 defines three different rehearsal schedules; Constant Rehearsal Assumption (CR), Expanding Rehearsal Assumption (ER) and Squared Rehearsal Assumption (SQ). CR does not assume that memories are strengthen for each rehearsal.

Designing PassCue with the CR rehearsal schedule would force the user to practice all the cue-association pairs every day, and significantly reduce the usability of the system. We saw in Section 2.2.1 that the memory is strengthened by the number of associations triggered in the brain, and that the availability of a memory is dependent on recency and the pattern of previous rehearsals. The Baker-baker paradox from Section 2.2.1 confirms that memory strength can be measured by number of associations it triggers. If PassCue creates strong cue-association pairs, it is reasonable to assume that daily rehearsal is not required in order to maintain the association. The difference between SQ and ER is not significant, but ER is consistent with known memory studies [74, 76] and was therefore chosen as the rehearsal schedule for PassCue.

4.2.4 Platform

The PassCue application will be used multiple times a day, hence should always be available to the user. The most available platform is arguably the mobile phone. The average user carries his phone close at any time during the day. The mobile platform is therefore the natural choice when selecting target platform for the PassCue application. Implementing PassCue as a smartphone application makes the application easy to install, available and highly distributable.

The mobile platform is primarily dominated by Google's Android, Microsoft's Windows Phone and Apple's iOS. According to NetMarketShare [77] iOS is the dominant operating system in first quarter of 2014 for mobile and tablet. iOS 7 was selected as the target platform because of its popularity and because the proper development tools and testing device was available for the developer/author. In order for the application to be available for the user at any time regardless of gsm reception and Internet connection, the application should be a standalone application with no external connections.

4.2.5 Password Composition Policies

We saw in Section 2.1.1 that administrators often use password composition policies to force the user to select strong passwords. Table 2.1 of common password guidelines presented eight criteria a password should fulfill. Studies have shown that PCPs are often different from website to website. Some sites require the user to select numbers and symbols in combination with lower-case and capital letters, while other sites restrict the use of symbols. Embedding numbers and special characters in the PassCue application could in some cases make the passwords generated by PassCue unusable.

As a consequence of the differences in PCP, PassCue is designed to have an account note field where the user can specify numbers, special characters and other information in order to fulfil the PCP of a specific site. This information is displayed in plaintext and is considered accessible by a potential attacker. How this affects the security is presented in Section 6.2, including how the security is affected by maximum password length.

4.2.6 Association Set Size

Section 3.4 and Theorem 2 clearly state that the security of Shared Cues model is dependent on the association set size. Larger set size means larger password space and higher security. The association size is also bound to the password creation strategy. The security calculations of PassCue assume that each of the actions and objects is uniquely encoded and that each of the actions and objects, given a password creation strategy, produce a unique string. If the user selects the two first letters from each action and object in the association, the maximum size is limited to $26^2 = 676$. If the user selects three letters from each action and object, the maximum size is $26^3 = 17576$. The realistic association size is much less then the maximum size as the English language lacks actions and object with certain letter combinations such as “xs” or “zk”.

All the actions and objects must be supplied with the application and each represented by an image. In order to provide reasonable security while not demand to much storage and system resources, the action and object size in PassCue is 200. How the association set size affect the security is presented in Section 6.2 and Figure 6.1. The association set size can easily be extended in cases where higher security is required, or lowered to save system resources.

4.3 Specification

The PassCue specification is based on the PassCue functionality from Section 4.1 and the design choices presented in Section 4.2. The design and implementation of PassCue should fulfil the specification given in Table 4.1.

Table 4.1: PassCue Specification

System
iOS application
Application data must be stored in persistent memory
A set of action and object with image must be included
Stand alone application with no external connection
Usability
(9,4,3)-sharing set
9 person images and 9 background images must be supplied by the user
4 cues is used to each account
Maximum 3 cues can be shared between two accounts
Support up to $m = 126$ accounts
Possible to remove accounts
The user must be notified when to rehearse according to the rehearsal schedule
Account notes to cope with password composition policies
Possible to reset a cue-association pair if association is forgotten
Natural rehearsing effects the rehearsal schedule
Use the ER rehearsal schedule
Security
Random numbers must be cryptographically secure
No cue-associations pairs are possible to retrieve after initialization
Action-Object set size must be 200

4.4 System Design

This section covers the system design of the PassCue application. The design is based on elements from the Shared Cues password management model presented in Chapter 3. The design is to cope with the specification given in Table 4.1, which is derived according to specific design choices presented in Section 4.2. The first part of this section covers the system overview and the second part presents the database architecture used for storing the application data.

4.4.1 System Overview

Figure 4.5 gives an overview of the system and the application flow. When the application starts, the application version is checked to see if the application has been

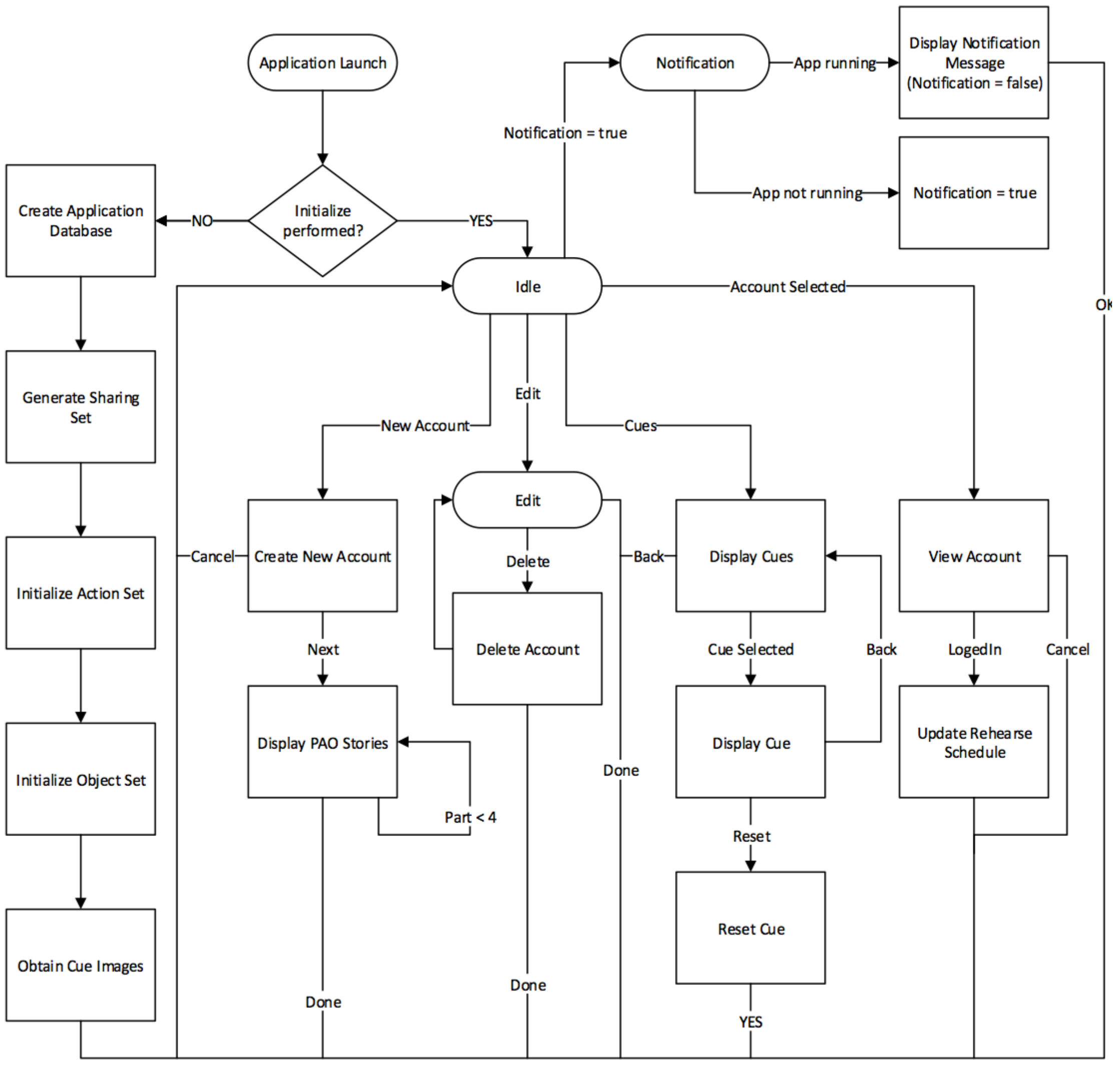


Figure 4.5: System Flowchart

launched before. If the application is launched for the first time, then initialization is performed. The application database is created with all the necessary tables as defined in Figure 4.6, and stored locally. A (9,4,3)-sharing set is generated using Algorithm 3 which returns all the $\binom{9}{4}$ subsets. The complete (9,4,3)-sharing set can be found in Appendix A. In cases where $l - \gamma \neq 1$, Algorithm 1 must be used in order to invoke the Chinese Remainder Theorem.

The system includes a set of action and object pictures which is inserted into the database with proper name and path. The rehearsal schedule for each cue is initialized by setting the number of prior rehearsals i to zero. The user is asked to supply a background image and a person image for each of the nine cues. The PAO-stories are created using Algorithm 4, derived from Algorithm 2, where random numbers are used to select an action and an object for each of the cues.

When the initialization is complete, the system makes a transition to idle state. In

Algorithm 3 Sharing Set Generation for (9,4,3)-Sharing Set

```
1: $n = 9$                                         ▷ $n$=number of public cues
2: $i = 1$                                         ▷ $i$=sharing set number
3: for $j = 1 \to n - 3$ do
4:     for $k = 1 \to n - 2$ do
5:         for $l = 1 \to n - 1$ do
6:             for $m = 1 \to n$ do
7:                 $SharingSet_i = (j, k, l, m)$
8:                 $i + +$
```

Algorithm 4 Generate PAO-stories

```
1: Input: $n$, random number $r1$ and $r2$, background images $\mathrm{B}_1, \ldots, \mathrm{B}_n$, person
   images $\mathrm{P}_1, \ldots, \mathrm{P}_n$
2: for $i = 1 \to n$ do
3:     $c_i \leftarrow (B_i, P_i)$
4:     $a_i \leftarrow \mathrm{ACT}$                 ▷ Select action from database using r1
5:     $o_i \leftarrow \mathrm{OBJ}$                 ▷ Select object from database using r2
6:     $ass_i \leftarrow (a_i, o_i)$
7: Return: $\{c_1, \ldots, c_n\}$, $\{ass_1, \ldots, ass_n\}$
```

idle state the system responds to different user inputs. If the user selects the *new account* button, the system creates a new account according to the user inputs, and displays the associated cues. If a cue is not initialized, the private associations are displayed and a rehearsal schedule for the cue is created. The rehearsal schedule for the cues are updated according to the current date and time before the system returns to idle state. The design uses the expanding rehearsal assumption we saw in Section 3.3. The rehearsal schedule is given by $R(\hat{c}, i) = 2^{i\sigma}$.

If the user selects one of the accounts, the cues and the account notes are displayed in a view account screen. The public cues are used to retrieve the associations from associative memory for each of the cues, and subsequently derive the password for the selected account. Once pressed the *LogedIn* button, the rehearsal schedule for the involved cues is updated. If the user selects the *edit* button, the application change to editing state, and the user is able to delete accounts. If an account is deleted, it is removed from the database and the system will not use the same sharing set for other accounts.

If the user selects the *Cues* button, a list of all the cues is presented. Each of the cues is presented with both the background and the person image, and information on how many accounts that use the cue. If the user selects one of the cues, a cue detail screen is displayed. The cue detail screen shows the two public images, next rehearsal time, the accounts that use the cue and an option for cue resetting. If one of the accounts is selected, the application makes a transition to the view account screen. If the cue is reset, all accounts that use the cue is deleted and the application returns to idle state.

Information text boxes can interrupt the application at any time to inform the user of a required rehearsal. If the application is not running, then the message will be displayed as a typical notification. This ensures that the user is always notified when a rehearsal is required.

4.4.2 Database Architecture

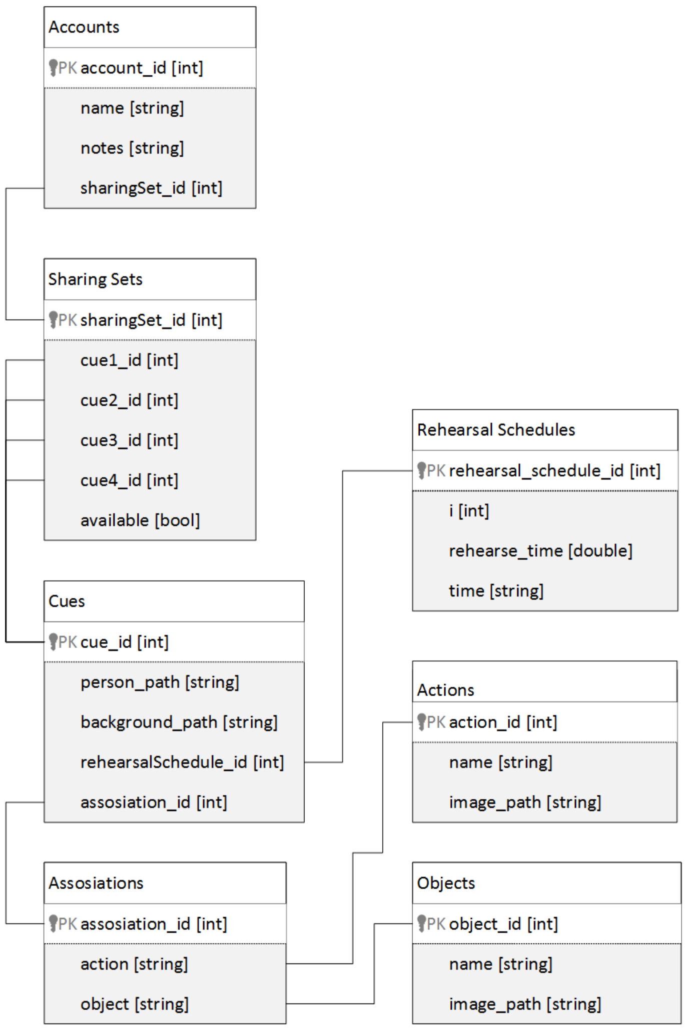


Figure 4.6: Database Architecture

Figure 4.6 presents the database architecture for the PassCue application. The database consists of seven tables. The *Accounts* table contains information of each account. For each account the account name, account notes and sharingSet_id, a foreign key to the *Sharing Sets* table, are stored. The sharingSet_id is used to retrieve the sharing set from the *Sharing Sets* table. The *Sharing Sets* table includes a foreign key, cue_id, to each of the four cues and a boolean value which indicates if the sharing set is available for use. Cue_id is used as primary key in the *Cues* table and it is used to retrieve information for a specific cue.

In the *Cues* table, the path to the background and person picture, a foreign key the *Rehearsal Schedules* and a foreign key to the *Associations* table is stored. The *Rehearsal Schedules* table stores information on number of prior rehearsals, i, the rehearse time and a date string. The *Associations* table consist of a foreign key to the *Actions* table and the *Objects* table. The *Action/Object* table stores name and image path information.

4.5 Summary

This chapter presented the design of the PassCue password management system based on Shared Cues. We were presented the PassCue functionality and how PassCue can be used to derive a password. PassCue uses a (9,4,3)-sharing set and the public cues are created using pictures supplied by the user. The ER rehearsal schedule is used by PassCue to ensure that the user maintains the cue-associations in memory. We saw that PassCue is designed with account notes in order to cope with modern PCPs. PassCue is designed for the iOS platform and has an associations size of 200. The chapter presented the full system design with system flow and underlying database structure. The design specification forms the basis for the implementation choices and specification in the next chapter. We will in the next chapter see how the PassCue design can be implemented for the iOS 7 platform and which implementation choices that must be made in order to implement PassCue as given in the design specification.

Chapter 5

Implementation

This chapter presents the PassCue implementation based on the design presented in the previous chapter. PassCue is implemented for the iOS platform using the Xcode IDE and Objective-C as implementation language. The first part of the chapter covers the implementation choices made in order to cope with the design requirements defined in previous chapter. The second part of the chapter presents the implementation specification followed by the application structure. In the fourth part, the navigation and interaction space of the application is presented. The chapter is concluded with a section presenting the application and illustrating how it can be used to log on to a system.

5.1 Implementation Choices

The PassCue design requirements defined in Table 4.1 is the basis for the iOS implementation. In order to cope with the defined requirement, several implementation choices had to be made. It was necessary to explore the following:

1. How should the application data be stored?
2. How can the actions and objects be randomly assigned to the public cues?
3. How should the associations be managed and displayed?
4. In what way should the cues be displayed to the user?
5. How can the application notify the user when rehearsal is required?
6. What happens if the user forgets one of the cues?

5.1.1 Data Storage

Data can be stored in numerous ways in iOS. The common practice of data management is using CoreData, SQLite or XML files [78]. CoreData is a powerful, full-feature data modeling framework. CoreData provides infrastructure to support basic activities such as save, edit and restore, and a large set of advanced activities. CoreData lets the developer define the application data in a graphical way, and uses a built-in SQLite data library with no need for additional database installation.

SQLite is a software library which implements an SQL database engine [79], and it is the most deployed SQL engine in the world. SQLite is a lightweight and powerful database engine which manipulates the database tables directly.

iOS provides a built in database based on XML files. Simple data structures such as user preference and application settings can easily be saved in XML files in the User Defaults database. The built in database is limited to simple data structures and provides only basic functionality.

The CoreData is a framework with much more functionality then what is needed for the PassCue application, and it can be time-consuming to configure. The database architecture given in Figure 4.6 illustrates the data structures that need to be saved in persistent memory. SQLite is a lightweight database engine, which supports all the needed functionality for PassCue. SQLite was chosen as the main data storage method because it is lightweight, has low setup time and provides all functionality needed for PassCue. The built in Users Database with XML files is used to store application settings, but it is not suited to be the main storage as it lacks functionality.

5.1.2 Random Numbers in iOS

There are multiple ways of generating random numbers in iOS. *Rand(3)* includes the following four functions; *rand, rand_r, srand, sranddev.* These functions require an initial seed. If seed is not specified, the number 1 is used as seed. If the seed is known, it is possible to retrieve the generated number. The functions defined in *Rand(3)* are considered to be bad random number generators [80] and should not be used for cryptographic use.

Random(3) includes the following functions; *initstate, random, setstate, srandom, srandomdev.* The functions in *Random(3)* creates random numbers of higher quality then *Rand(3).* The *Random(3)* functions are considered to be better random number generators [81], but the generated random numbers are not of cryptographic quality.

Arc4random() includes the following functions; *arc4random, arc4random_buf, arc4random_uniform.* These functions generate random numbers with higher quality than *Rand(3)* and *Random(3).* The original *arc4random()* function used the RC4 cipher [82], but it was replaced with the ChaCha20 cipher to increase the quality.

All the functions can be used to generate random numbers in iOS, but in order to be of cryptographic quality, Apple recommend to use the *Randomization Services* Application Programming Interface (API). The developer reference states; “Randomization Services is an API that generates cryptographically secure random numbers” [83]. The function *SecRandomCopyBytes* in *Randomization Services* generates an array of cryptographically secure random bytes by reading from the */dev/random* number generator.

The */dev/random* number generator collects environmental noise from device drivers and other sources into an entropy pool. Typical sources are disk and network activity or clock device interrupts. As stated in the Linux Programmer’s Manual; “When read, the /dev/random device will only return random bytes within the estimated number of bits of noise in the entropy pool. /dev/random should be suitable for uses that need very high quality randomness such as one-time pad or key generation. When the entropy pool is empty, reads from /dev/random will block until additional environmental noise is gathered” [84].

The *Randomization Services* API and the *SecRandomCopyBytes* function was used as PRNG because the generated random numbers are considered to be of high cryptographic quality. The random numbers determine the selection of action and object for each of the public cues. If an attacker can retrieve the random numbers, he can retrieve all cue-association pairs and decoding the password would be trivial.

We concluded in Section 2.3.5 that PRNGs are pseudo random and not truly random. Random.org is a service that offers a TRNG. Truly random numbers are accessible by using the Random.org API. Random.org uses atmospheric noise in order to create truly random numbers. The specification in Table 4.1 states that PassCue should be stand alone with no external connection, hence random.org was not used as TRNG. The *Randomization Services* API and the *SecRandomCopyBytes* is recommended by Apple as a cryptographically safe PRNG, and is sufficient for this design.

5.1.3 Associations

The random numbers, as presented in Section 5.1.2, are used to select an action and an object which constitutes the association for each of the public cues. It is important that the user easily understand the action and object in order to create strong associations. To facilitate for easy understanding, the action and object is illustrated with both image and text. The action and object elements must be carefully selected in order to give meaning to the user. The actions and objects should be commonly used words in order to be properly understood by the user.

As passwords are derived from the associations, the secrecy of the associations is crucial for the secrecy of the passwords. It is assumed that a potential attacker has access to the users phone, hence have access to the PassCue database. In order to preserve the secrecy of the passwords, no associations must be possible to retrieve after initialization. The association database, illustrated in Figure 4.6, is initially populated with random actions and objects picked from the actions and objects database. After a cue has been initialized, the connection to the association and the association itself is deleted and non-retrievable. A new association with a random action and object is created if a cue is reset, and the association will only be available for the initialization of the cue.

5.1.4 Cues Overview

It is important that the user has access to his cues in order to preserve the cue-associations in memory. The PassCue implementation has a cues overview screen which displays the public cues with their meta data. The cues overview shows all the cues with their respective person and background image, and the number of accounts that use the cue. If the user selects one of the cues, additional information such as which accounts that use the cue, and next rehearsal time is displayed.

The purpose with the cue overview is to facilitate for the rehearsal of the cues according to the rehearsal schedule. If the user is told to practice cue 3, the user must be able to easily see which accounts that are using cue 3, and subsequent log in to the account to practice cue 3. The application shows next rehearsal time for each of the cues. Knowing the next rehearsal makes it possible for the user to rehearse earlier if he unable to rehearse at the scheduled time.

5.1.5 Notifications

The rehearsal schedule is created in order to ensure that the cue-association pairs are maintained in the associative memory of the user. PassCue must notify the user according to the rehearsal schedule when a cue-association rehearse is required. The PassCue application creates notifications according to the rehearsal schedule and notifies the user with sound, banner and alert. This ensures that the user is always informed of a rehearsal.

5.1.6 Cue Resetting

If a user does not follow the rehearsal schedule, he might forget the cue-association pair. The user is able to reset a cue if a cue-association pair is forgotten. This functionality is added because the number of possible accounts is significantly affected if a cue is unusable. We learned in Section 4.2.2 that the (9,4,3)-sharing set can generate 126 passwords when all the nine cues are used. With eight cues the number is 70, 35 with seven cues and 15 with six cues. All accounts that use the cue will be deleted if a cue is reset.

5.2 Specification

The PassCue implementation specification is based on the implementation choices from Section 5.1 and the design specification in Table 4.1. The implementation specification for PassCue is given in Table 5.1.

Table 5.1: PassCue Implementation Specifications

System
Implemented for iOS 7 on iPhone 5
SQLite database as main data storage
The User Defaults database is used for application settings
A set of action and object with image must be included
No external communication
Usability
(9,4,3)-sharing set implemented using Algorithm 3
The user selects pictures from photo library using an image picker
The public cues are assigned associations using Algorithm 4
Text under each association to clarify picture
A screen with overview of all the cues
A screen with cue overview, accounts and restore function
Possible to see the next rehearsal time for each cue
Notifications are used to alert the user of a rehearse with sound, banner and message
All accounts that use the cue is deleted if the cue is reset
Account notes is a text field specific for each account
Security
Use the arch4random as PRNG
Associations are deleted from database after cue initialization
The included action and object set size is 200
The sharing set is unique for each account
The sharing set for a deleted account cannot be reused

5.3 Application Structure

The PassCue application was developed using the Model-View-Controller (MVC) design pattern. The MVC is a high-level pattern, which classifies objects according to the roles they play in the application. Applications developed using the MVC design pattern tends to be more adaptable to changing requirements [85]. This is because the objects in a MVC application are more reusable and the interface is better defined.

The design pattern comprise of the following three modules; the model, the view and the controller. The MVC pattern defines the roles of these modules and the communication flow between them. Figure 5.1 describes the relationship between the modules. The model communicates directly with the controller, while the controller-view communication is done through the outlet-action protocol [85].

- **Model:** The model contains the application data and the logic that manipulates the data. The data is stored in model objects and the model objects contains information and functions tied to a specific problem domain, making them reusable. The model objects do not consider how the data they represent is displayed and presented to the user. The presentation of the data is done by the view, hence the model never communicates with the view.
- **View:** The view is responsible for presenting and displaying the information to the user. The view does not consider the storing of information, only how the application data/model should be displayed. The view may also allow the user to edit the application data.
- **Controller:** The controller acts as a communication barrier between the model and the view. The controller ensures that the view has access to the model objects it needs to display and that changes to view objects are updated in the models.

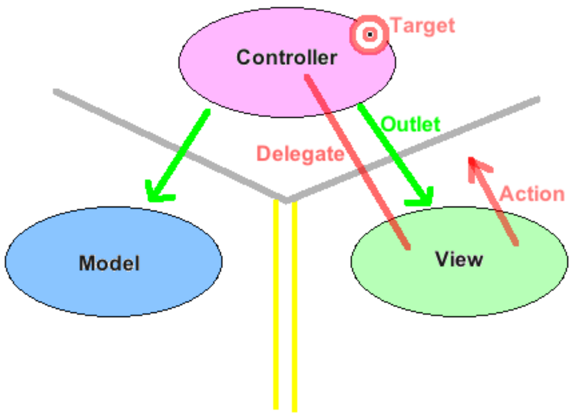


Figure 5.1: MVC Design Pattern

Table 5.2 presents the application structure. The views and controllers are in the same implementation, hence called view controllers, which is the standard way of implementing a MVC application. The view controllers include the functionality of both the view and the controller as presented in the previous list.

Table 5.2: PassCue Application Structure

Name	**Functionality**
Model	
Account	Object for account information
Action	Object for action information
Association	Object for association information
Cue	Object for cue information
DBManager	A global object for all communication with the database
Object	Object for object information
RehearsalSchedule	Object for rehearsal schedule information
SharingSet	Object for sharing set information
View Controller	
CueViewController	Retrieving and displaying cue information
CuesViewController	Retrieving and displaying information for all cues
ImagePickerViewController	Responsible for obtaining cue images
InitAccountController	Initializes and creates new accounts
InitPAOController	Generates PAO-stories and displays associations if required
MainViewController	Displaying all accounts and root view controller
PassCueAppDelegate	Head of all view controllers
ViewAccountController	Retrieves and displays account information

5.4 Navigation Space

Figure 5.2 illustrates the navigation space for the PassCue application, and which interactions that trigger navigation. The *mainVC* can perform navigation to *cuesVC*, *initAccount* or *viewAccount*, depending on the interaction performed. The *edit* button in *mainVC* does not trigger navigation, but it puts *mainVC* in edit mode, and the user can remove accounts. If the user selects one account in *mainVC*, the application makes a transition to *viewAccount*. If the user selects the *cues* button, it triggers navigation to the *cuesVC*. In the *cuesVC* screen the user is able to select a cue in order to get more information in the *cueVC*. The *+* button trigger navigation to *initAccount* where a new account is created, followed by the PAO-stories in *initPAO*.

As shown in Figure 5.2 all the view controllers have a *back* button, which triggers navigation to the previous screen, or a *cancel* button which navigates to the *mainVC*. The initialization process, where the user supply public cue images, is not included in the navigation space, but will be presented in Section 5.5.

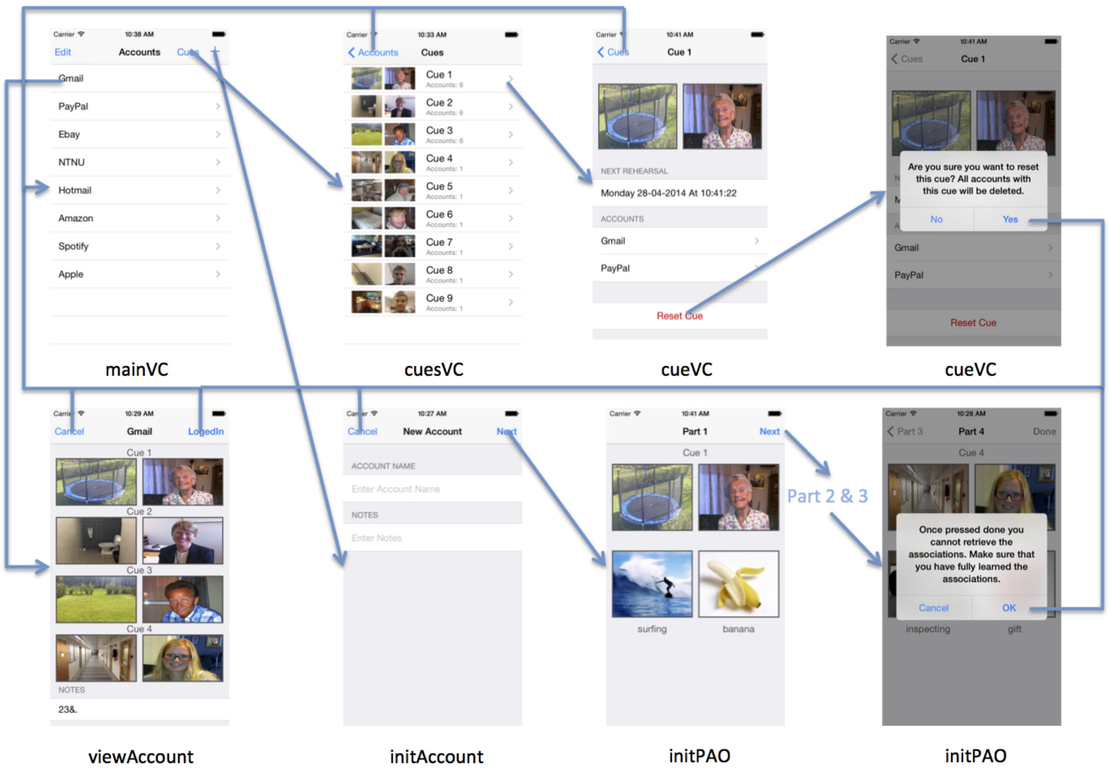


Figure 5.2: Navigation Space

5.5 PassCue

PassCue is designed and implemented based on the Shared Cues password management model. This section presents the application, implementation details and how PassCue can be used when logging on to a system. Appendix C provides an overview of the PassCue source files. The public pictures in the following example figures is taken by the author and used with persons' permission. The private action and objects pictures are used with permission from morguefile.com.

The PassCue implementation reflects the implementation specification in Table 5.1. Figure 5.3 shows the application screens for the initialization process. The initialization process is only required the first time the application is launched. The user is told to select a background image and select a person image for the nine required cues. The user is able to select pictures from the photo library or downloaded images. When the user push the *Select Background Image* button, an image picker screen is displayed and the user can select the appropriate picture. The user can quit the application in order to obtain the images or take the images with the camera on the phone.

The cue pictures are saved within the document directory in the application, and the path is saved in the database. If the images where to be deleted from the photo library, it will not affect the application. In this example the user selects a picture of the trampoline in his garden and a picture of his grandmother as the

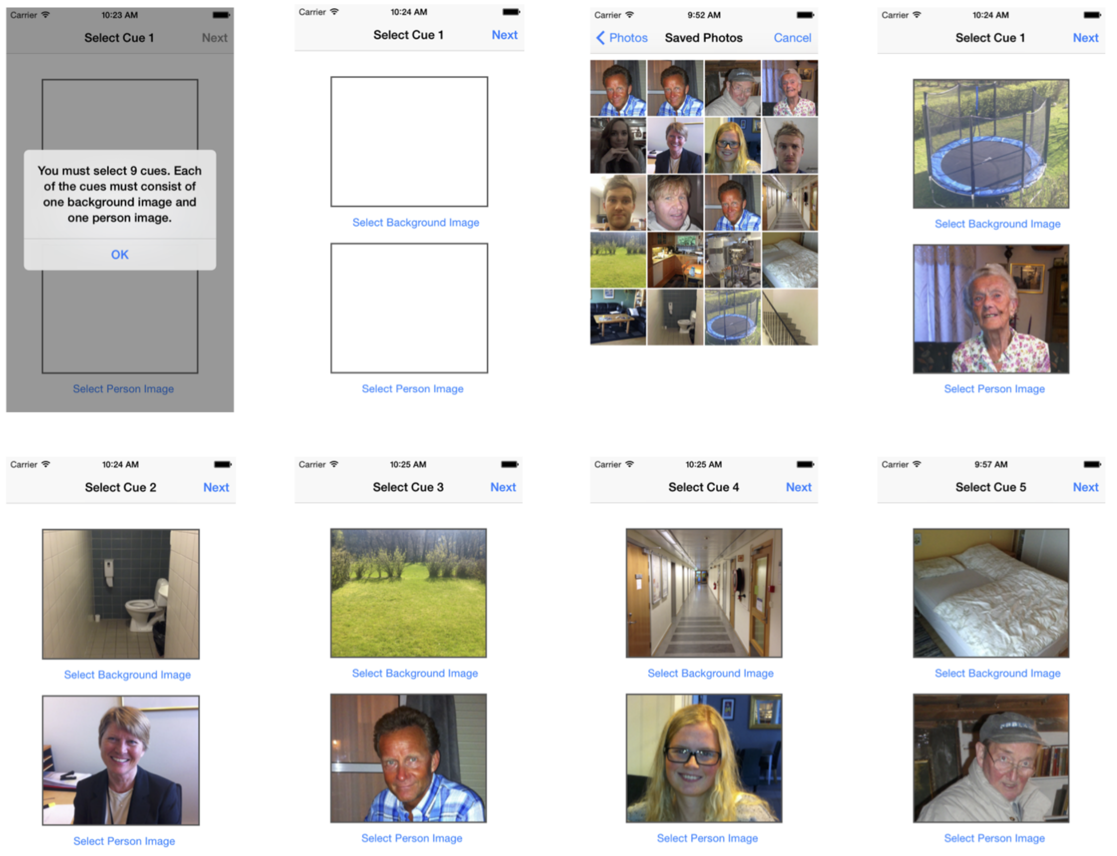
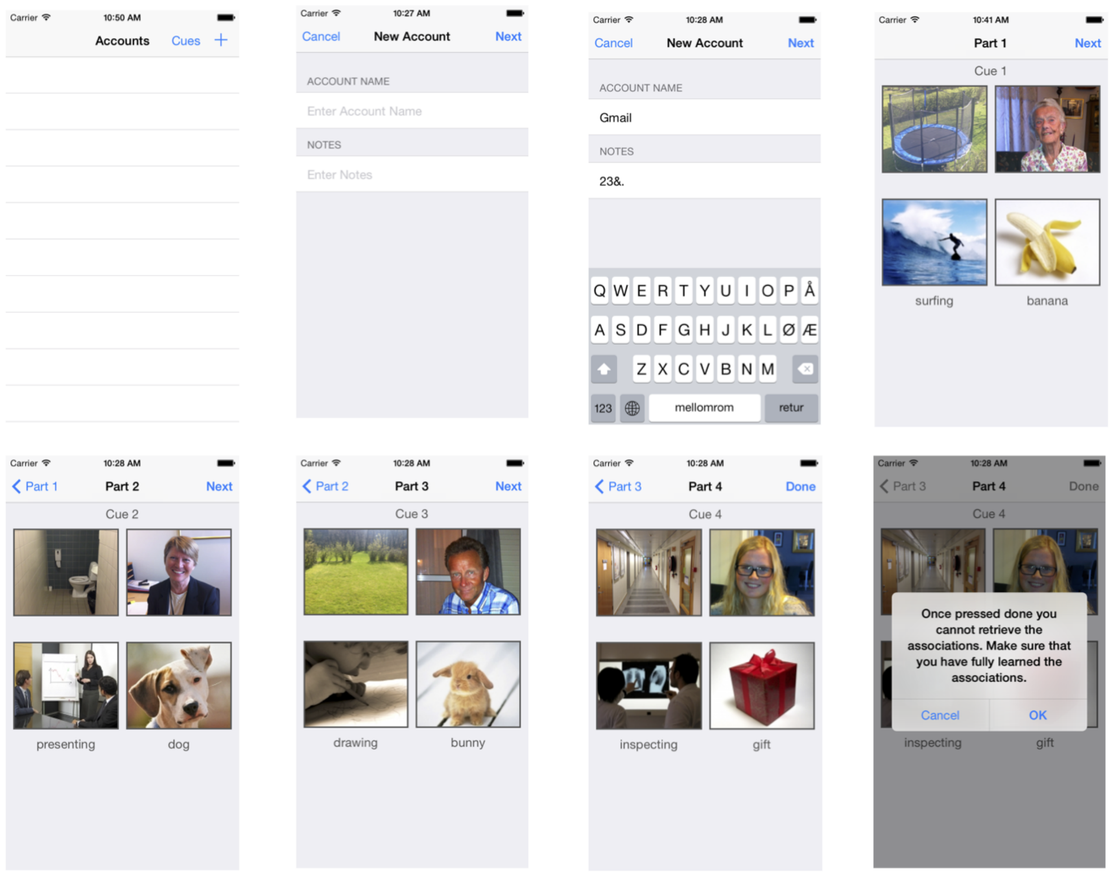

You must select 9 cues. Each of the cues must consist of one background image and one person image.

Figure 5.3: Select Cue Images

first cue. Once pressed the *Next*-button, the user can select images for cue number two. The user must continue the process until cue nine is initialized.

When the user has selected images for all nine cues the top left screen in Figure 5.4 is displayed. The user can add an account by pushing the + button. The user must select an account name and write account notes if desired. As we saw in Section 2.1.1, many sites puts restrictions on the password selection in order to force the user to select a strong password. In this case, for the *Gmail* account, Google recommends using a mix of letters, numbers and symbols in the password [86]. The user inputs "23&." in the account notes field, and will use this when deriving the password. The account notes are displayed in plaintext and are assumed to be accessible to an attacker. How this affects the security is detailed explained in the security analysis in Section 6.2.

When the *Next* button is pushed, the first cue and the randomly selected association is displayed. In the example in Figure 5.4 the user must imagine the following setting; "My grandmother is *surfing* a *banana* on the trampoline". *Surfing* and *banana* is the private part of the cue and will never be displayed after the cue initialization. *Surfing* and *banana* will be used to create the password. The public picture of the user's grandmother and his trampoline will later be used to trigger the association of *surfing* and *banana* from the users associative memory. In cue



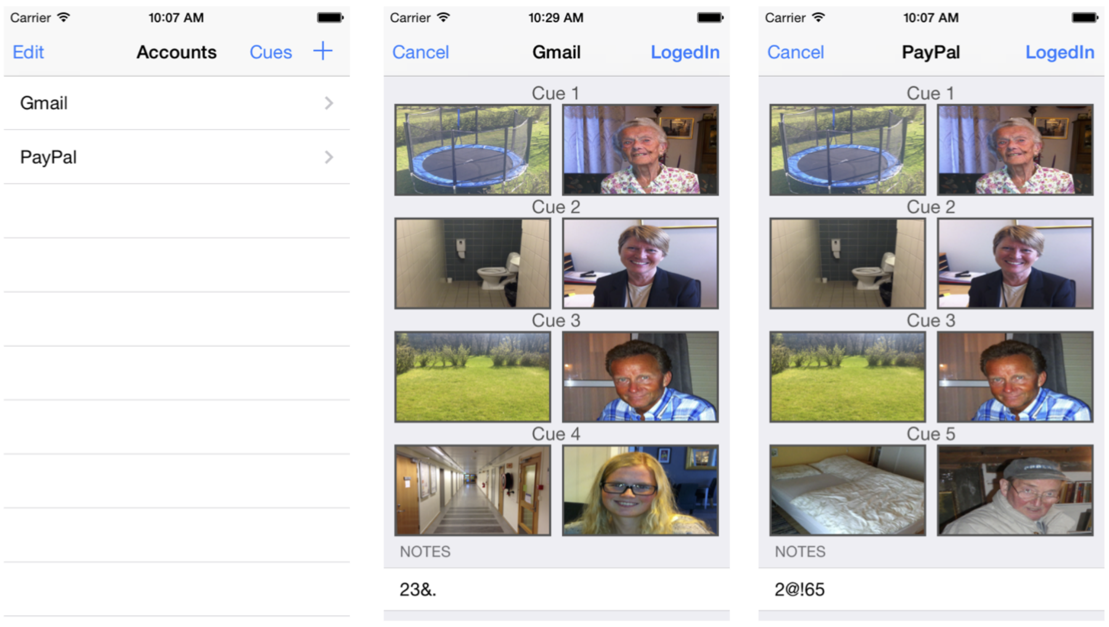
Once pressed done you cannot retrieve the associations. Make sure that you have fully learned the associations.

Figure 5.4: Create New Account

two the user must reflect over the following story; "My mother is *presenting* a *dog* on the toilet". Cue three gives the following story; "My father is *drawing* a *bunny* in the garden". In cue four the user must imagine the following; "My sister is *inspecting* a *gift* in the hallway".

Once the user presses the *Done* button in part 4, a warning message alerts the user that the associations are non-retrievable after this step. A rehearsal schedule is created for cue 1, 2, 3 and 4 using the calculation from Section 4.4.1. This is performed to ensure that the user does not forget the actions and objects associated with the cues.

Figure 5.5 shows how PassCue can be used to log in to a system. In this example, PassCue holds two accounts, *Gmail* and *PayPal*. If the user is to log in to the *Gmail* account, he selects the *Gmail* account and the account cues and notes are displayed. The user will use the cues in order to retrieve the associations from associative memory. The user must ask himself; "What did my grandmother on the trampoline?" and should remember that she was indeed "*surfing* a *banana*!". The next cue retrieves the association *presenting* and *dog*. Cue three reveals that "My father was *drawing* a *bunny* in the garden". The last cue was "My sister is *inspecting* a *gift* in the hallway".



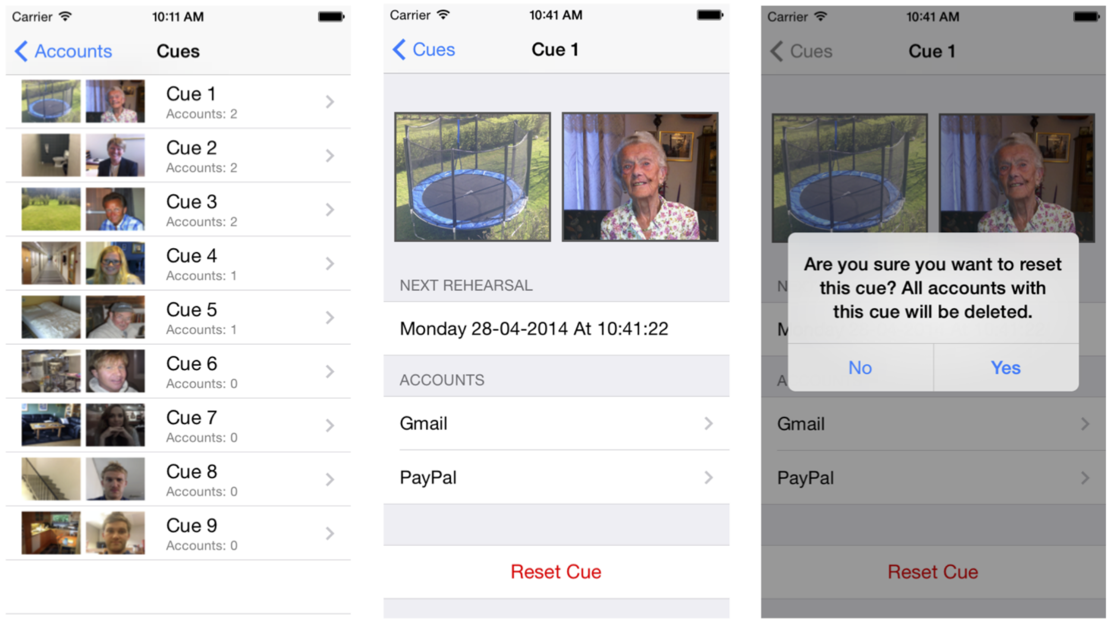
23&.

2@!65

Figure 5.5: Log in to *Gmail* and *PayPal*

In this example the user always uses the account notes as the first part of the password, and uses the three first letters from each action and object with capital first letter for all the action derived letters. The password for *Gmail* will be "23&.SurbanPredogDrabunInsgif". The user must press the *LogedIn* button for the rehearsal schedule to be updated. Once pressed *LogedIn* the application calculates a new rehearsal time for the involved cues according to the rehearsal schedule. The *LogedIn* button is not connected to the *Gmail* online interface or any other online interface. The *Logedin* button is solely for use in the PassCue application in order to manage the rehearsal schedules. Pressing the *Logedin* button does not provide automatic login to the specified account. The user must derive the password using the cues, and press *LogedIn* for the application to update the rehearsal schedule. Cue five in Figure 5.5 gives the following setting; "My grandfather is *kicking* an *elephant* in the bed". Since *Gmail* and *PayPal* share the first three cues, parts of the password is identical. By using the same derive method as for the *Gmail* account, the password for *PayPal* is "2@!65SurbanPredogDrabunKicele".

If the user selects the *Cues* button from the main screen, he is able to see information about the cues. The first screen in Figure 5.6 shows the overview of the cues with pictures, and number or accounts. The user is able to choose one of the cues for more information. The cue information shows the next time for cue rehearsal, which accounts the cue is used in, and an option to reset the cue. The user can select one of the displayed accounts and the application will make a transition to the log in screen for the account, as shown in Figure 5.5. This is illustrated in Figure 5.5. In the event that the user forgets the action and object associated to a cue, the user can reset the cue. If the cue is reset, all accounts that use the particular cue is deleted, and a new association and rehearsal schedule is created

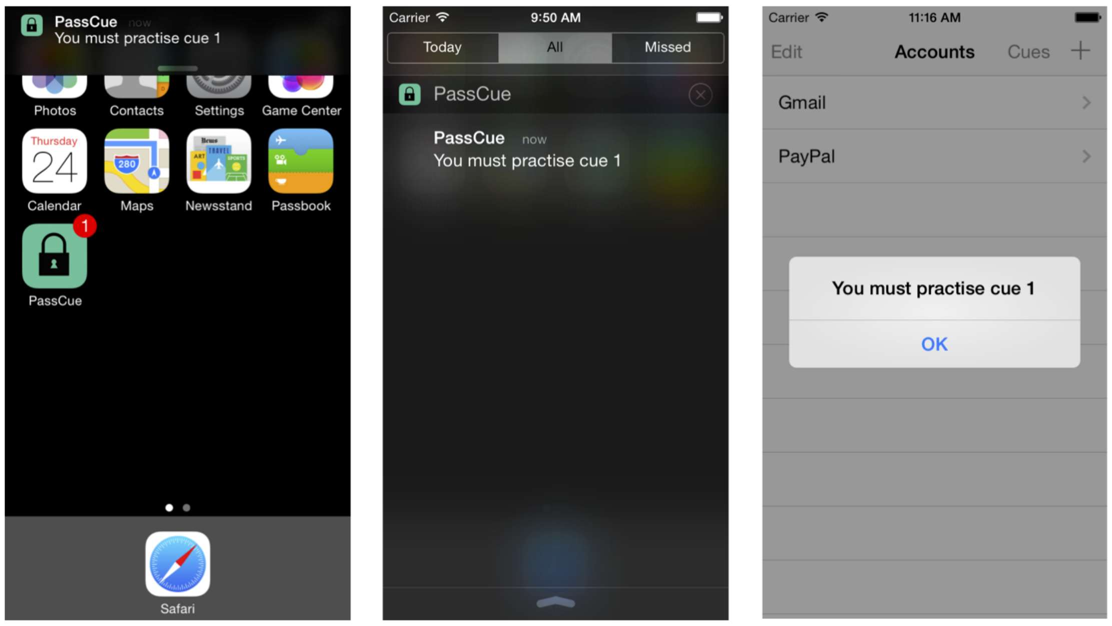

Carrier 10:11 AM
Accounts Cues
Cue 1 Accounts: 2
Cue 2 Accounts: 2
Cue 3 Accounts: 2
Cue 4 Accounts: 1
Cue 5 Accounts: 1
Cue 6 Accounts: 0
Cue 7 Accounts: 0
Cue 8 Accounts: 0
Cue 9 Accounts: 0

Carrier 10:41 AM
Cues Cue 1
NEXT REHEARSAL
Monday 28-04-2014 At 10:41:22
ACCOUNTS
Gmail
PayPal
Reset Cue

Carrier 10:41 AM
Cues Cue 1
Are you sure you want to reset this cue? All accounts with this cue will be deleted.
No Yes
Gmail
PayPal
Reset Cue

Figure 5.6: Cue information

for the cue.


PassCue now
You must practise cue 1

Carrier 9:50 AM
Today All Missed
PassCue
PassCue now
You must practise cue 1

Carrier 11:16 AM
Edit Accounts Cues +
Gmail
PayPal
You must practise cue 1
OK

Figure 5.7: Rehearsal notification

The main objective with the rehearsal schedule is to ensure that the user does not forget the action and object associated with each of the cues. This is done by notifying the user to rehearse the cue-association according to specific intervals. Figure 5.7 shows how PassCue notifies the user when rehearsal is required. The first screen shows that the application icon reflects a notification, and the next screen shows the notification in the notification center. If the user has the application open, or opens the application after a notification has been fired, the message is displayed in the application. In this example, the user must practice the cue-association for cue 1 in order to not forget the association.

The user rehearses the cue-association by logging in to one of the accounts that use the cue. The cue overview, shown in Figure 5.6, can be used to help the user see which account he must log in to for rehearse. In this example, both *Gmail* and *PayPal* are using cue 1, so the user can choose which account to log in to. If the user logs in to *Gmail* the rehearse schedule for not only cue 1, but cue 2, 3 and 4 is updated.

5.6 Summary

This chapter presented the PassCue implementation for the iOS platform. PassCue uses an SQLite database to store all application data, while the User Defaults database is used to save application settings. We saw that the *SecRandomCopyBytes* function in the *Randomization Services* API is used as PRNG and ensures that the actions and objects are randomly selected. The associations are only displayed when initializing the cue, and are not possible to retrieve after account creation. We learned that the cue overview shows information for each cue, and that it is possible to reset a cue if the user forgets the cue-association. PassCue is developed using the popular MVC pattern, which makes it reusable and flexible. The chapter presented the PassCue iOS application and how it is initialized and used. We saw how the user can use personal pictures, create new accounts and log in to existing accounts. We were presented to notifications and how it implements the rehearsal schedule and helps the user to maintain the cue-association in associative memory. In the next chapter we will evaluate the PassCue application in terms of usability and security using the concepts we discussed in chapter 2. We will also evaluate how the PassCue application utilizes the CPU and memory resources on an iPhone 5.

Chapter 6

Evaluation

This chapter covers the evaluation of the PassCue implementation. The first part of the chapter is an analysis of the usability of PassCue. The usability is evaluated by calculating the additional time the user must invest in rehearsing the cues. The second part of the chapter covers the security analysis of PassCue. In this part the password entropy is calculated and the resistance against plaintext leak attacks, online attacks and offline attacks is evaluated. The last part of the chapter presents the implementation analysis including how the PassCue application utilizes CPU and memory resources on an iPhone 5.

6.1 Usability Analysis

The most important part of the usability of PassCue is the additional time the user must invest in order to maintain all the cue-association pairs. We learned in Section 3.3 that PassCue is based on a usability assumption. The assumption states that users who follow a specific rehearsal schedule will successfully maintain the corresponding memory. As presented in Section 4.2.3, PassCue uses the Expanding Rehearsal Assumption as rehearsal schedule. A rehearsal schedule is created for each cue-association pair and updated every time the cue is used for log in. Given the ER rehearsal schedule from Definition 5, a cue-association pair must be rehearsed at least eight times per year in order maintain the cue-association pair in the user's associative memory. In Section 4.2.2 we saw that sharing sets are used to share cues between accounts to improve the usability and reduce the number of rehearsals. PassCue uses a (9,4,3)-sharing set, where maximum three cues can be shared between two accounts. This property significantly improves the usability, and reduces the extra amount of time the user must invest in rehearsing the cue-association pairs.

Table 6.1: Sharing set for the first 10 accounts

	Part 1	Part 2	Part 3	Part 4
Account	Cue	Cue	Cue	Cue
1	1	2	3	4
2	1	2	3	5
3	1	2	3	6
4	1	2	3	7
5	1	2	3	8
6	1	2	3	9
7	1	2	4	5
8	1	2	4	6
9	1	2	4	7
10	1	2	4	8

Table 6.1 shows the cue distribution for the first ten accounts in PassCue (9,4,3). The full (9,4,3)-sharing set can be found in appendix A. The sharing set has been incrementally designed in order to make it easy for the user to start using PassCue. As given in Table 6.1, the first six accounts use cue 1, cue 2 and cue 3. After the first account is created, the user only needs to remember one new cue each time a new account is created. After creating the sixth account, the user knows all the nine cue-association pairs. The next $126 - 6 = 120$ accounts use cues which the user already knows. This makes PassCue very usable and easy to extend as the number of accounts increases.

In Section 2.1 we saw that a normal user has at least 25 different accounts. In PassCue, creating a new account forces the user to rehearse some of the cues used in other accounts. Creating passwords for 25 accounts in PassCue would be sufficient yearly rehearsal for all the nine cues. It follows from the rehearsal schedule and sharing set in Section 4.2.3 and Section 4.2.2, when creating 25 accounts in PassCue, the cues will be rehearsed multiple times. Table 6.2 shows how many times each of the cue-association pairs are rehearsed when creating 25 accounts in PassCue. The number of extra rehearsals, additional time invested in PassCue, for a normal user is close to zero. This proves that PassCue (9,4,3) provides high usability.

PassCue (9,4,3) is implemented with a (9,4,3)-sharing set, but the sharing set can be easily changed and other sharing sets can be used. We concluded in Section 4.2.2 that (9,4,3)-sharing set was chosen because it provided high usability and is easy to initialize with user photos. The usability of PassCue with another sharing set can be evaluated using Theorem 1. The usability result for PassCue (9,4,3) is given in Table 6.3 together with the usability of PassCue (43,4,1) and PassCue (60,5,1), as defined in Shared Cues Table 3.6. The usability results for the five password management schemes, presented in Section 2.1.2, are also presented in Table 6.3. The usability results show the number of extra rehearsals the user must perform the first year in order to maintain all the cue-association pairs in memory.

Table 6.2: Number of cue rehearsals performed when creating 25 accounts

Cue	Times Rehearsed
1	25
2	21
3	10
4	10
5	7
6	7
7	7
8	7
9	6

An extra rehearsal is when the user must rehearse the password, e.g. log on to a system, without having a purpose of logging on to the system. The log on is primary for rehearsing the cue-association pair.

Table 6.3: Usability Results

Scheme	Extra Rehearsals
PassCue (9,4,3)	≈ 0
PassCue (43,4,1)	10.89
PassCue (60,5,1)	19.89
Reuse Weak	≈ 0
Reuse Strong	≈ 0
Lifehacker	≈ 0
Strong Random and Independent	456.6
Randomly Generated	-

The PassCue usability results in Table 6.3 show that PassCue (9,4,3) requires approximately zero extra rehearsal the first year. This result corresponds to what was shown in Table 6.2 and by evaluating the structure of the sharing set in Table 6.1 with the ER rehearsal schedule. PassCue provides high usability and should be easy to start using. The results show that PassCue provides the same usability as the *reuse weak*, *reuse strong* and *lifehacker* scheme.

For PassCue (43,4,1) a normal user would invest 12 extra rehearsals the first year in order to maintain all the cue-association pairs in associative memory, given the ER rehearsal schedule. PassCue (60,5,1) require 20 extra rehearsals the first year. The normal user is defined in Section 3.3 and Table 3.4. *The strong random and independent* requires 456.6 extra rehearsals the first year. This scheme provides extremely high security, but requires the user to invest a huge amount of time for rehearsal. The security results for these schemes are presented in Section 6.2.

The usability result in Table 6.3 for the *lifehacker* scheme and the *strong random and independent* scheme assume 75 accounts, $m = 75$. The usability of *randomly* *generated* is not defined as it can not be measured in number of extra rehearsals by Theorem 1. The *randomly generated* scheme require that a password management tool is used to store all the passwords. The usability for the *randomly generated* scheme is the additional time the individual user invests in using the management tool.

6.2 Security Analysis

The main goal of PassCue is to provide a user-friendly way of creating and managing secure passwords. As we discussed in Section 2.3, there are different threats that must be taken into account in order to evaluate the security of PassCue. This section evaluates the PassCue resistance against these threats.

6.2.1 Password Entropy

We learned in Section 2.3.4 that the password entropy H can be used as a measure of password strength. A password with entropy H means that there are 2^H possible values of the password. We can calculate the entropy of a PassCue password using Equation 2.1 for entropy defined in Section 2.3.4. The calculation assumes a (9,4,3)-sharing set and an action and object set size of 200. It is assumed that the attacker knows all the 200 actions and 200 objects in the association set.

$$H = \log_2\left((|Actions| \times |Objects|)^l\right) = \log_2\left((200^2)^4\right) = 61.15085 \tag{6.1}$$

Using Equation 6.1, the *Gmail* password "23&.SurbanPredogDrabunInsgif" derived in Section 5.5 has 61.15085 bits of entropy. The *PayPal* password "2@!65SurbanPredogDrabunKicele" has 61.15085 bits of entropy, since the password is generated using the same association set size and number of cues.

Table 6.4: Password Entropy

		Entropy (*bits*)		
Scheme	m	r=0	r=1	r=2
PassCue (9,4,3)	126	61.15085	15.28771	0
PassCue (43,4,1)	90	61.15085	45.86313	30.57542
PassCue (60,5,1)	90	76.43856	61.15085	45.86313
Reuse Weak	∞	14.28771	0	0
Reuse Strong	∞	57.15084	0	0
Lifehacker	∞	56.96445	0	0
Strong Random and Independent	∞	57.15084	57.15084	57.15084
Randomly Generated	∞	82.71880	82.71880	82.71880

Table 6.4 shows the password entropy for PassCue (9,4,3), (43,4,1), (60,5,1) and the five password schemes presented in 2.1.2. r is used to denote a plaintext leak attack. The entropy of PassCue (9,4,3) when $r = 0$ is 61.15085. The entropy of PassCue is dramatically lowered if one of the passwords is leaked in plaintext, $r = 1$. PassCue (43,4,1) maintain acceptable security when one password is leaked and PassCue (60,5,1) maintain high security for $r = 1$ and acceptable security for $r = 2$. We discussed in Section 2.2 that there is a trade-off between usability and security. The usability results from Table 6.3 show that (43,4,1) and (60,5,1) require that the user invests additional time to rehearse the cue-association pairs compared to (9,4,3).

In the *reuse weak* scheme, Algorithm 5 is used to create a password. Assuming that the string “password” is selected randomly from a dictionary of 20000 words, the entropy is $H = \log_2(20000) = 14.28771$ bits. As we saw in Section 2.1,“password” was one of the most popular passwords in the rockyou.com password breach in 2009.

The *reuse weak* provides a very low level of security even before one plaintext password is leaked. The *reuse strong* scheme assumes that Algorithm 6 returns the string “horsecakepaperdog”, composed of four randomly chosen dictionary words. The password entropy when $r = 0$ is $H = \log_2(20000^4) = 57.15084$ bits. The *reuse strong* provides high entropy when $r = 0$, but as the password is reused across multiple sites, the entropy is zero when a plaintext password leak occur.

The *lifehacker* scheme assumes the password derived in Section 2.1.2; “apletowercupfa11”. The entropy is given as $H = \log_2(20000^3) + \log_2(26^3) = 56.96445$ bits, assuming that the derivation rule produces a three character string of lower case letters. The *lifehacker* scheme provides high entropy for $r = 0$, but when a plaintext password leak attack occur, the base password and derivation rule is revealed and the security breaks down.

The *Strong Random and Independent* scheme assumes that the password is created using Algorithm 7. The scheme creates a password string of four words selected randomly from a dictionary of 20000 words, and a new password is created for each account. The password entropy for $r = 0$ is the same as for the *reuse strong*, but the *strong random and independent* scheme maintains the entropy even after multiple passwords are leaked in plaintext.

The *randomly generated* scheme creates passwords similar to the password “bcxtabf2owale89n” from Section 2.3.4 with entropy $H = \log_2 36^{16} = 82.71880$ bits. As the password is randomly generated for each account, the *randomly generated* scheme ensures high entropy regardless of plaintext password leaks. How the entropy can be compared to cracking time and cost is explained later in this section.

As given in Equation 6.1, the entropy of the PassCue generated passwords are depended on the association size and number of cues, l, used for each account. PassCue uses a (9,4,3)-sharing set and association set size of 200. Figure 6.1

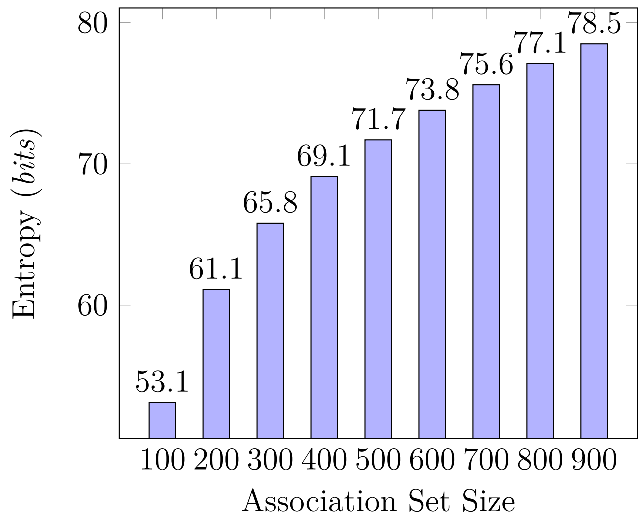


Figure 6.1: The PassCue password entropy with $l = 4$

presents how the association set size affects the entropy when assuming $l = 4$.

The association size entropy in Figure 6.1 shows that the largest increase in entropy is when the association set size is between 100 and 400. This is a result of the entropy being calculated with $\log_2$, as shown in Equation 6.1. The results show that entropy gained by increasing the association set size is not proportional with the increase in set size. Increasing the set size with 100 means adding 100 new actions and 100 new objects with pictures. There is a trade-off between entropy gain and resources needed to extend the association set. Extending the association set after 400 may not be defended with entropy gain.

As discussed in Section 4.2.6 the association set size is limited by the password creation strategy of the user. If the user selects the two first letters from the action and the two letters from the object in each of the associations, the association set size is limited to $26^2 = 676$. This is the maximum set size, but as the English language lacks actions and objects which begins with "xs" and "zk", a realistic set size is much less then 676. The entropy calculations assumes that each of the actions and objects in the association set is unique encoded and derived as presented in Section 4.1. If 50 actions and objects in a set of 200 all starts with "sa", the entropy is reduced with $H = 61.15085 - \log_2\left((150^2)^4\right) = 61.15085 - 57.83055 = 3.3203$ bits.

Figure 6.2 shows how l, the number of cues, affects the entropy assuming an association size of 200. The l is part of the sharing set and must be changed according to the rules for creating sharing sets. We learned in Section 3.5 and Section 4.2.2 that the sharing set must be created using Algorithm 1 if $l - \gamma \neq 1$. If $l - \gamma = 1$ it is sufficient to calculate the $\binom{n}{l}$ subsets using Algorithm 3. The entropy results in Figure 6.2 show that the password entropy increases with 15.3 bits for each additional cue.

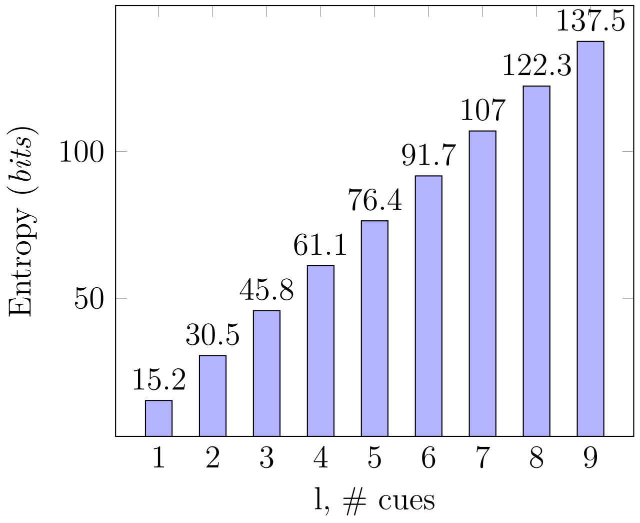
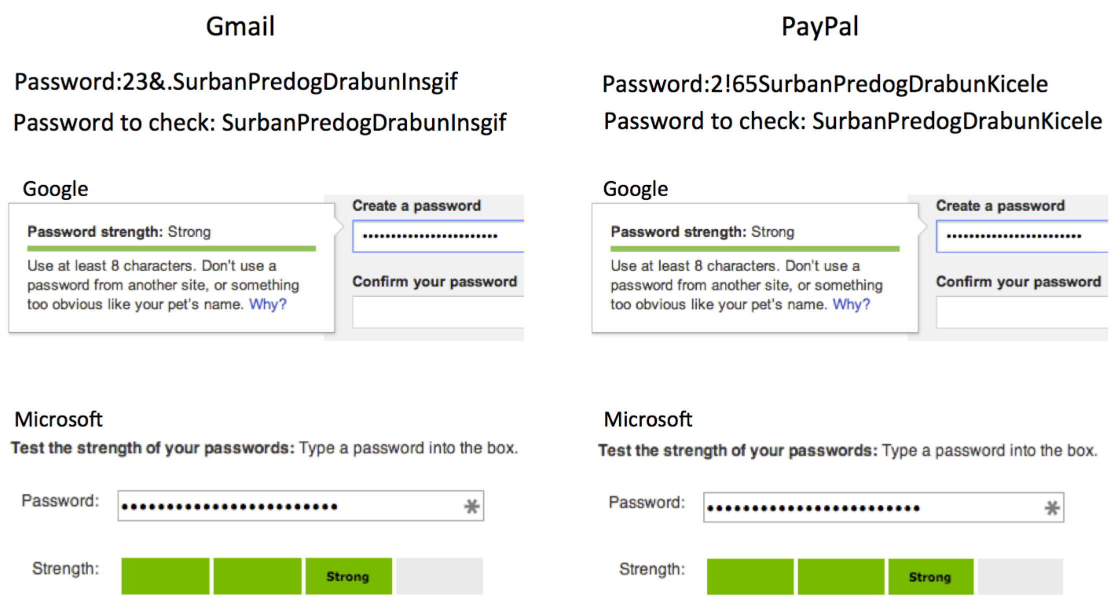


Figure 6.2: The PassCue password entropy with association size of 200

6.2.2 Password Strength Meters

Section 2.3.4 introduced password strength meters as another tool to measure the password strength. Figure 6.3 shows the evaluation of the 'Gmail' password "23&.SurbanPredogDrabunInsgif", and 'PayPal' password "2@!65SurbanPredogDrabunKicele", using both the Google [87] and Microsoft [88] PSM. The Google PSM ranks the *Gmail* password as strong, which is the highest rating. Using only the first eight characters of the password, "surbanpr", is sufficient for the password to be ranked as strong. This shows that Google does not check for symbols, numbers or capital letters when evaluating. A random eight-character string of lower-case letters will be ranked as strong. The entropy of a random eight character string of lower-case letters is $H = \log_2 (26^8) = 37.60351$ bits. We will see in Section 6.2.5 and Table 6.7 that a password with this entropy can be cracked in maximum 8.7 minutes with modern cracking tools. This emphasizes that the Google password meter cannot guarantee security, and is only meant as a reference. Entropy calculation is a more accurate measure.

As shown in Figure 6.3 the Microsoft PSM ranks both the *Gmail* and *PayPal* password as strong, which is the second highest ranking. As we saw in Section 2.3.4, the password must be at least 14 characters, contains both lower-case and upper-case letters, symbols and numbers, to receive the highest ranking. A password with theoretical unlimited entropy will still be ranked as strong, and not ranked as best. The password "kdjeuhfnsldmekrjfndjahtulemska" is a 30-character string with randomly chosen lower-case letters. The entropy is $H = \log_2 (26^{30}) = 141.01319$ bits, but is still not ranked as best in the Microsoft PSM, even though cracking the password would take over 4.24789×10^{25} years.


Gmail

Password:23&.SurbanPredogDrabunInsgif

Password to check: SurbanPredogDrabunInsgif

Google

Password strength: Strong

Use at least 8 characters. Don't use a password from another site, or something too obvious like your pet's name. Why?

Create a password

Confirm your password

Microsoft

Test the strength of your passwords: Type a password into the box.

Password:

Strength: Strong

PayPal

Password:2!65SurbanPredogDrabunKicele

Password to check: SurbanPredogDrabunKicele

Google

Password strength: Strong

Use at least 8 characters. Don't use a password from another site, or something too obvious like your pet's name. Why?

Create a password

Confirm your password

Microsoft

Test the strength of your passwords: Type a password into the box.

Password:

Strength: Strong

Figure 6.3: Google and Microsoft PSM

6.2.3 Maximum Length Restrictions

We learned in Section 2.1.1 that some online services include a restriction on maximum password length in the password composition policy. From a security point of view, there is no reason to put limitations on the password length. Some sites have a maximum password length in order to support legacy systems, because of ignorance, prevention of dos-attacks, reduce customer service issues or simply because "we have always done it that way".

If the online service enforces a PCP with maximum length, and the password must contain numbers and symbols, it can reduce the password entropy. This is because the numbers and symbols included in the password, to cope with the PCP, is displayed in plaintext in the account notes in PassCue. This is illustrated in Figure 5.5. We concluded in Section 5.5 that the user should always choose the three first letters of each action and object unless there is a restriction on maximum password length. Equation 6.2 must hold in order for the security calculations in Section 6.2 to be valid.

$$|chars\,password| + |chars\,PCP| \leq max\,password\,length \tag{6.2}$$

Equation 6.2 states that the number of characters in the derived PassCue password, added with the numbers of characters used to satisfy a PCP, must be lower or equal to the maximum password length. We know from Section 4.1 that the number of characters in the derived PassCue password is dependent of the number of characters from each action and object, and the number of cues l for each account. This gives $|chars\,password| = |chars\,each\,act/obj| \times (2 \times l)$, as each cue contains one action and one object, assuming that number of characters used is equal for actions and objects. Inserting this property in Equation 6.2 gives Equation 6.3 which defines the upper limit for the number of characters from each action and object. The lower limit is $|chars\,each\,act/obj| = 2$, assuming that each of the actions and objects in the set in uniquely encoded.

$$|chars\,each\,act/obj| = \frac{(|max\,password\,length| - |chars\,PCP|)}{2 \times l} \tag{6.3}$$

The security calculations in Section 6.2 are invalid if only one character from each action and object is used. The PassCue password entropy, assuming no plaintext leaks, would be $H = \log_2(26^8) = 37.60351$ bits. This is not a recommended password derive method and will not be further discussed.

Assuming that the online service has a PCP that requires a mix of letters, numbers and symbols, and has a maximum password length of 20, one character will be used for the number and one for the symbol. As the number and the symbol will be displayed in plaintext in the account notes, presented in Section 5.5, only 18 characters are available for the rest of the password. If the user selects three characters from each action and object, only the first three cue-association pairs will fit inside the 18-character limit. This will result in an entropy reduction of $\log_2(200^8) - \log_2(200^6) = 61.15085 - 45.86314 = 15.28771$ bits. If the user selects two characters from each action object, there will be no entropy reduction. As a result, the user should use two letters from each action and object if the maximum password length is < 26.

6.2.4 User Knowledge and Behavior

In Section 2.3.1 we discussed that user behavior can be a possible threat to the security of a system. The lack of knowledge can make the user vulnerable to shoulder surfing, keyloggers and social engineering. Like other password-based authentication methods PassCue is vulnerable to shoulder surfing and social engineering. If the user is logging on to one of his accounts in a public place using PassCue, the attacker could watch over the user's shoulder and see the password. The only way to mitigate this threat is to educate the user of this problem, and to avoid logging in to accounts in crowded or public places.

We learned in Section 2.3.1 that the attacker can utilize social engineering techniques in order to deceive the user to reveal his password. PassCue is, as any password based system, vulnerable to this threat. The only way to mitigate this threat is to inform and educate the user to never reveal his password, and to verify the identity of a website or a program before entering sensitive information.

PassCue is also vulnerable to keyloggers. A keylogger is similar to shoulder surfing, but the attacker infects the users computer with a malware that is monitoring the keyboard and input from the user. By using a keylogger, the attacker can easily obtain passwords when the user logs on to different accounts. To protect against keyloggers and malware in general, it is necessary to regularly update the software, use anti malware tools and to prevent software to run in root mode.

6.2.5 Attacks

We discussed in Section 2.3.3 that it is necessary to understand the threats in order to defend against them. This section evaluates the PassCue system by analyzing how resistant PassCue is against plaintext leak attacks, online attacks and offline attacks.

Plaintext Leak Attack

Plaintext leak attack is an attack where the user password is leaked in plaintext and can be directly used by the attacker. Plaintext leak attacks can occur as a result of a phishing attack or a misconfiguration of the online server. All password-based authentication methods, including PassCue, is vulnerable to plaintext leak attack. In order to mitigate this threat, the user must be educated, and be suspicious to emails regarding password request and strange links. As we discussed in Section 2.3.2, the user has no control of how the online service stores his password. The user he must have trust in the online service he signs up for. The rockyou.com password breach in 2009 led to the release of 32 million user passwords, and it revealed that rockyou.com saved all the passwords in plaintext [17]. In order minimize the risk of a plaintext attack the user must be sure that the online service manages password properly.

Online Attack

Online attack is an attack method where the attacker has access to the authentication interface of the online service and simply guesses the login details. We discussed online attack in Section 2.3.3 and we saw that most online services have a k-strike policy where the user is locked out after trying k wrong passwords. The number of guesses the attacker can perform is limited to $k\times m$, where m is the total number of passwords, accounts, in PassCue. Modifying Theorem 2 in Section 3.4, the probability that the attacker can successfully retrieve the password is given in Theorem 3. Proof of Theorem 3 and Theorem 2 can be found in Appendix A in [6].

Theorem 3 *Let $\{c_1, \ldots, c_m\}$ be a (n, l, γ) sharing set of m public cues produced by PassCue. If each action and object is chosen uniformly at random, the probability that the attacker retrieves the password using online attack is;*

$$P_r = \frac{km}{(|Actions| \times |Objects|)^{l-\gamma r}},$$

where r is the number of plaintext password leaks and k is the number of guesses permitted.

As given in Theorem 3 the PassCue resistance against online attacks is highly dependent on previous plaintext password leaks. That is because cues are shared between accounts, as discussed in Section 4.2.2. PassCue uses a (9,4,3)-sharing set. Assuming $k = 3, r = 0$ and $m = 126$ gives P(Attacker retrieves password) $= \frac{3\times126}{(200\times200)^{4-3\times0}} = 1.47656 \times 10^{-16}$. The probability that the attacker is able to guess the *Gmail* or *PayPal* password from Section 5.5, assuming no previous password leaks $r = 0$, is 1.47656×10^{-16}. For the other password management schemes, P(Attacker retrieves password) is calculated using Equation 6.4, where H is the password entropy calculated in Table 6.4, $k = 3$ and $m = 126$:

$$P_r = \frac{km}{2^H} \tag{6.4}$$

Table 6.5: Online Security Results

Scheme	m	r=0	r=1	r=2
PassCue (9,4,3)	126	1.47656×10^{-16}	0.00945	1
PassCue (43,4,1)	90	1.47656×10^{-16}	5.90625×10^{-12}	2.36250×10^{-7}
PassCue (60,5,1)	90	3.69140×10^{-21}	1.47656×10^{-16}	5.90625×10^{-12}
Reuse Weak	126	0.01890	1	1
Reuse Strong	126	2.36251×10^{-15}	1	1
Lifehacker	126	2.68833×10^{-15}	1	1
Strong Random and Independent	126	2.36251×10^{-15}	2.36251×10^{-15}	2.36251×10^{-15}
Random Generated	126	4.74954×10^{-23}	4.74954×10^{-23}	4.74954×10^{-23}

Table 6.5 presents the probability that an account is compromised as a result of an online attack for various password management schemes. The online security of PassCue (9,4,3), (43,4,1) and (60,5,1) is calculated using Theorem 3. PassCue (9,4,3) provides high resistance against online attacks even if one password is leaked in plaintext. The PassCue (9,4,3) online security breaks down if two plaintext passwords are leaked. PassCue (43,4,1) and (60,5,1) provides high security even in the case of multiple password leaks. As we saw in Table 6.3 the two sharing sets require the user to invest additional time in order to maintain the cue-association pairs in memory.

The probability that an account is compromised, assuming the *reuse weak* password scheme is used, is 1.89% for $r = 0$. Since the password is reused across all account, compromise of one account gives the attacker access to all the accounts. For the *reuse strong* scheme, it is very unlikely that the attacker can compromise an account for $r = 0$, but the security breaks down with the first password leak. The *lifehacker* scheme provides high online security for $r = 0$, but the attacker would learn both the password base and the derivation rule if one password is leaked. Both the *strong random and independent* scheme and *random generated* scheme provide high online security regardless of password leaks.

Offline Attack

We learned in Section 2.3.3 that the attacker can perform an offline attack if he can access the hash of the user's password. There are many examples of online services that have been hacked and the cryptographic hash released [9, 10, 11]. If the attacker has access to the password hash, he can guess the password, compute the hash and compare it to the leaked password hash. Modern password cracking tools can do millions of password guesses per second and can crack a weak password in seconds.

We saw in Section 2.3.3 that Hashcat is state of the art cracking software that is optimized for speed using the Graphical Processing Unit (GPU). The PassCue resistance against offline attack will be measured by estimating the time and cost of cracking the PassCue passwords using Hashcat. MD5 is considered the fastest of the common hash algorithms. The estimates assume that MD5 is used as hash function in order to estimate a worst case scenario of the security of PassCue. The estimates are based on renting computing capacity on the Amazon Elastic Compute Cloud (EC2), as presented in Section 2.3.4. The calculations assume that Hashcat can run 2100 million MD5 guesses per second on EC2 [71, 72]. Theorem 4 defines how to calculate the time required to crack a PassCue password. Theorem 4 is defined using Theorem 2, and proof can be found in Appendix A in [6].

Theorem 4 *Let $\{c_1, \ldots, c_m\}$ be a (n, l, γ) sharing set of m public cues produced by PassCue. If each Action and Object is chosen uniformly at random and r is the number of plaintext leaks, the estimated guaranteed password cracking time in seconds is given by*

$$P_{crack\,time} = \frac{(|actions| \times |objects|)^{l-\gamma r}}{guesses/sec}$$

Theorem 4 defines the guaranteed cracking time which is the time required to brute force though the whole password space. In some cases, the cracking time will be less then the guaranteed cracking time because the password is cracked before the whole space is searched. Guaranteed cracking time assures that the password will be cracked within the time. The guaranteed cracking time in seconds for a PassCue password hashed with MD5 using Hashcat with 2100 million guesses per second, assuming (9,4,3)-sharing set, association set of 200 and $r = 0$ is given as;

$$P_{crack\,time} = \frac{(200^2)^4}{2100 \times 10^6} = 1219047619$$

Renting the *cg1.4xlarge* GPU instance on EC2 costs \$2.1 per hour. Given this price, the estimated cracking cost in USD (\$) for a PassCue password is given as;

$$P_{crack\,cost} = P_{crack\,time} \times \frac{2.1}{3600} \tag{6.5}$$

Equation 6.5 defines how to calculate the cracking cost when renting computing power on Amazon EC2 assuming that the cracking time is known. The cracking cost for a PassCue (9,4,3) password if $r = 0$ is;

$$P_{crack\,cost} = 1219047619 \times \frac{2.1}{3600} = 711111$$

In order to crack the *Gmail* password or *PayPal* from Section 5.5, given no previous password leaks $r = 0$, the attacker must invest $\$711,111.1$ and it would take 14109 days, over 38 years, on a single GPU on Amazon EC2. The full analysis of the PassCue configurations is given in Table 6.6. PassCue (9,4,3) provides significantly higher offline security compared to the *reuse weak* scheme when $r = 0$. PassCue (9,4,3) provides almost twice the offline security compared to the *reuse strong*, *lifehacker* and *strong random and independent* scheme for $r = 0$. After one plaintext leak occur, the offline security for PassCue (9,4,3), *reuse strong* and *lifehacker* breaks down. PassCue (43,4,1) maintains a certain level of protection for $r = 1$ and PassCue (60,5,1) maintains very secure. The security of PassCue (43,4,1) breaks down when two plaintext leaks occur, while PassCue (60,5,1) still maintain a level of security. The *strong random and independent* scheme and the *randomly generated* scheme provide high security regardless of password leaks. Even if PassCue (43,4,1) and PassCue (60,5,1) does not provide as high security as the *strong random and independent* scheme, they are much more usable. For $r = 0, 1$ the security is also comparable.

Table 6.6: Offline Security Results

		r=0		r=1		r=2	
Scheme	m	Time (*days*)	Cost (*$*)	Time (*days*)	Cost (*$*)	Time (*days*)	Cost (*$*)
PassCue (9,4,3)	126	14109	711111.1	0.00002(sec)	≈ 0	≈ 0	≈ 0
PassCue (43,4,1)	90	14109	711111.1	0.35273	18	≈ 0	≈ 0
PassCue (60,5,1)	90	5.64373×10^8	2.84442×10^{10}	14109	711111.1	0.35273	18
Reuse Weak	∞	0.0000095(sec)	≈ 0	≈ 0	≈ 0	≈ 0	≈ 0
Reuse Strong	∞	881.82	44444.15	≈ 0	≈ 0	≈ 0	≈ 0
Lifehacker	∞	774.95	39057.6	≈ 0	≈ 0	≈ 0	≈ 0
Strong Ran. & Ind.	∞	881.82	44444.15	881.82	44444.15	881.82	44444.15
Random Generated	∞	4.38638×10^{10}	2.21073×10^{10}	4.38638×10^{10}	2.21073×10^{10}	4.38638×10^{10}	2.21073×10^{10}

Table 6.7: Entropy and Offline Security

Entropy (*bits*)	Time (*years*)	Cost(*$*)
40	0.000016	0.3
45	0.00053	0.7
50	0.01700	312.7
55	0.54403	10007.9
60	17.40900	320255.9
65	557.08801	1.02481×10^7
70	17826.81652	3.27942×10^8
75	570458.12892	1.04941×10^{10}
80	1.82546×10^7	3.35812×10^{11}

The security results given in Table 6.6 and Table 6.7 assume that the on-demand pricing model is used for Amazon EC2. In a real scenario an attacker would probably select another subscription in order to reduce the cost. Table 6.7 shows the relation between password entropy and cracking cost and time, assuming MD5 as hash function and Hashcat as cracking tool. The security results in Table 6.6 and 6.7 assumes guaranteed cracking time. A sophisticated attacker with a large budget could also rent multiple GPU's on Amazon EC2 and do the cracking in parallel, or buy optimized GPU cracking hardware [89].

PassCue (9,4,3), (43,4,1) and (60,5,1) have proven to provide acceptable security with no required rehearsals or a reasonable number of required rehearsals. PassCue (43,4,1) and (60,5,1) requires 43 and 60 public cues respectively. Each of the public cues consists of a person picture and a background picture, which must be supplied by the user. Gathering enough pictures for the public cues may for some users require additional time.

6.3 Implementation Analysis

This section covers the analysis of the PassCue implementation in terms of memory consumption and utilized CPU resources. The PassCue application has been analysed and tested on an iPhone 5 with iOS 7. The size of the PassCue application is 1300 KB, excluding the public cue pictures and the association set. The association set includes 200 actions and 200 objects. The actions and objects are 20 KB. The size of a picture taken with the main camera of an iPhone 5 is 3400 KB. The total size of the PassCue Application is $1300 + (400 \times 20) + (18 * 3400) = 70500$ KB = 70.5 MB. In idle, PassCue utilizes less than 1 % of the CPU and consumes 5.9 MB of memory. The following analysis shows how PassCue utilizes CPU and memory on an iPhone 5 for all possible application states.

Figure 6.4 shows the CPU and memory resources allocated for the initialization process. We learned in Section 5.5 and Figure 5.3 that the user must select 18 pictures from the photo library when the application launches for the first time. Nine person pictures and nine background pictures. The first CPU peak is the initialization of the application as presented in Section 5.5 and Figure 5.3. The initialization include the PassCue database creation, sharing sets generation and creation of the *Actions* and *Objects* table, presented in Figure 4.6. The initialization is followed by nine CPU peaks with a CPU utilization between 72% and 96%. This is when the user selects the required pictures. The supplied pictures are added to the document directory of the PassCue application, and the name and path is inserted to the *Cues* table in the PassCue database. Associations are randomly selected, added to the cues and inserted to the *Associations* table. Identical peaks can be found in the memory consumption. Nine memory consumption peaks of between 9.6 MB to 12.3 MB. Carefully inspection of both the CPU and memory analysis reveals two small CPU and memory peaks of 5% and 6.5 MB respectively, before each of the nine large peaks. This is when the image picker

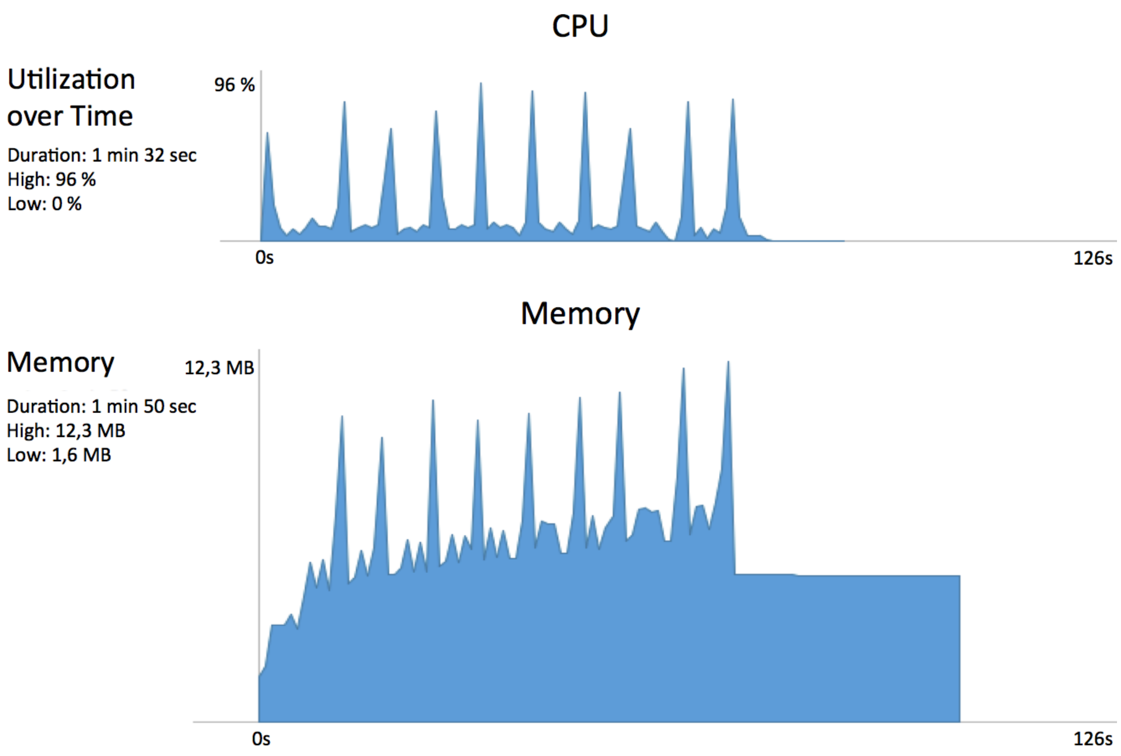


Figure 6.4: CPU utilization and memory consumption for the initialization

is loaded and enables the user to select two pictures from the photo library, as illustrated in Figure 5.3.

Figure 6.5 shows the CPU and memory allocation when the user creates a new account, as illustrated in Figure 5.4. The first CPU peak of 20 % is when the application is launched. A small CPU peak of 10% is when the user inputs account name and notes, which is inserted in the *Account* table. Four 30-33% CPU peaks follow after the first small peak. This is when the cues are displayed for the user, as seen in Figure 5.4. This is also where the rehearsal schedule is managed, and the notifications are scheduled according to the rehearsal schedules. The memory consumption analysis shows that approximately 10 MB is allocated for each of the cues that is displayed for the user. When the last cue is displayed, 45.4 MB of memory is consumed by PassCue. The memory is released when the user pushes the *Done* button. When the user presses the *Done* button, the associations connected to cues is deleted and completely removed, and the application makes a transition to idle state.

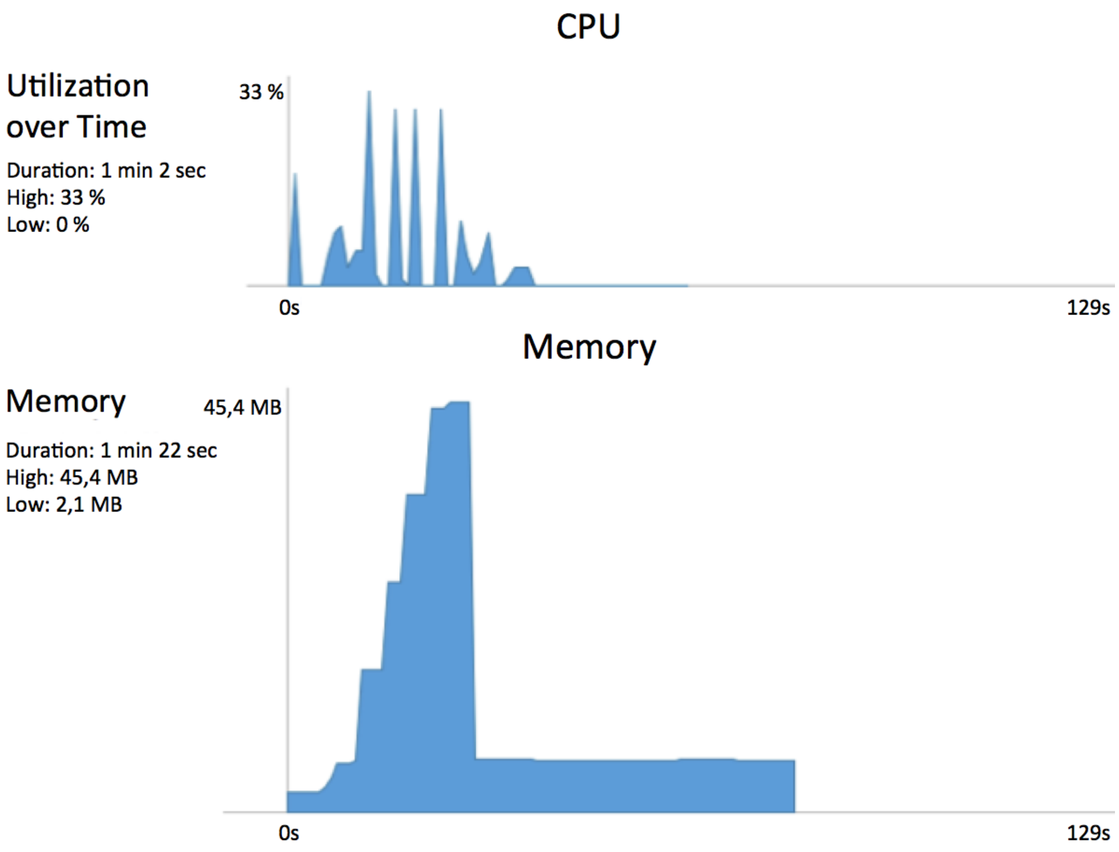


Figure 6.5: CPU utilization and memory consumption when creating a new account

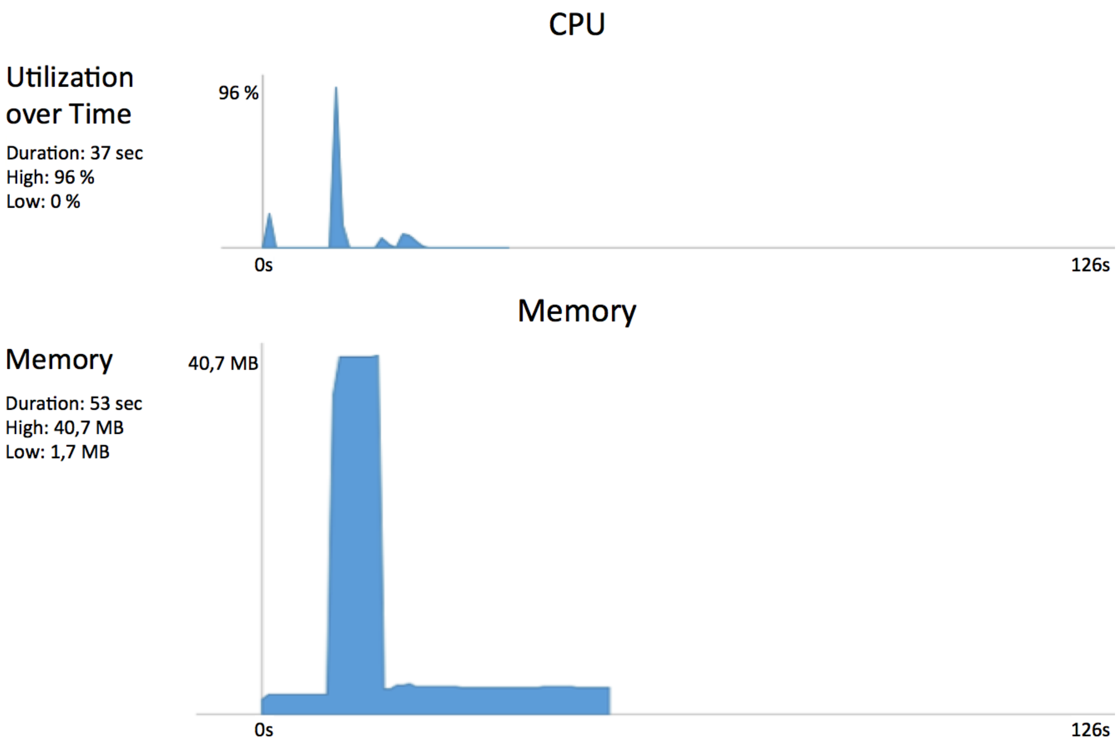


Figure 6.6: CPU utilization and memory consumption when logging on to a system

Figure 6.6 shows the CPU and memory utilization when the user logs on to a system. Figure 6.6 shows one minor and one major CPU peak when the user logs in to an account. The first peak of 20% is the start up process when the application is launched. The second peak of 96% is when the four account cues are retrieved from the database and displayed to the user, as illustrated in Figure 5.5. This is reflected in the memory consumption in the same figure. The memory analysis shows one major memory peak of 40.7 MB when memory is allocated to the four cues.

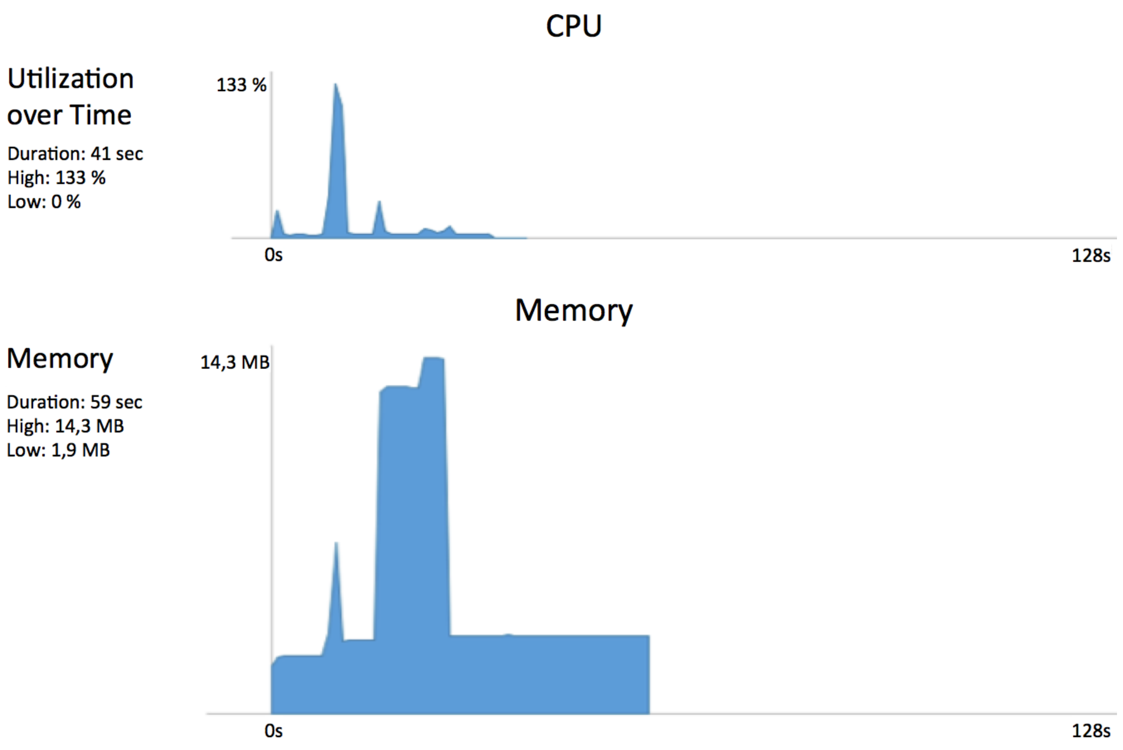


Figure 6.7: CPU utilization and memory consumption for the cue overview

Figure 6.7 shows the CPU and memory allocations when cue information is displayed. The first CPU peak of 20% is the start up process when the application is launched. The second peak of 133% CPU resources is when all the cues are displayed for the user. This is illustrated in Figure 5.6. The amount of CPU utilization is quite high. This is because the application must retrieve 18 pictures from the database and prepare them to be displayed in a specific pattern. The third CPU peak of 28% is when one of the cues is selected and cue information is displayed, as seen in Figure 5.6. The memory analysis shows that more memory is allocated for the cue information screen, 14.3 MB, compared to when all cues are displayed, 7.8 MB, and the application uses 133% of the CPU resources available. It is to be noted that the iPhone 5 has two CPU cores.

6.4 Summary

In this chapter we saw that PassCue (9,4,3) provides reasonable security while requiring no extra rehearsals. Unlike *reuse weak*, *reuse strong* and *lifehacker*, PassCue (9,4,3) provides high resistance against online attacks even if one plaintext password attack has occurred. The offline security of PassCue (9,4,3) is approximately twice as high as *reuse strong* and *lifehacker* for $r = 0$, but the offline security breaks down if one password is leaked in plaintext. PassCue (43,4,1) provides high online security, even if multiple password leaks occur, but the normal user must invest 11 extra rehearsals the first year.

PassCue (43,4,1) offers high offline security for $r = 0$, and if $r = 1$ the attacker must invest 8.5 hours to crack the password in an offline attack. PassCue (60,5,1) provides high offline security for $r = 1$ and acceptable security if two password plaintext leaks occur. PassCue (60,5,1) offers high resistance against online attacks, regardless of password leaks, but the user must invest approximately 20 extra rehearsals the first year.

The *strong random and independent* scheme provides high online and offline security regardless of plaintext password leak attacks, but it is not very usable. The scheme requires approximately 460 extra rehearsals the first year. Even if PassCue (43,4,1) and (60,5,1) are not as secure as *strong random and independent*, they are much more usable, and for $r = 0$ and $r = 1$ the security is also comparable. The *randomly generated* scheme provides very high security regardless of plaintext password leaks, but it requires additional tools and software. The tool must be available for the user at all time and the user is dependent on the tool in order to use any accounts.

We saw in the chapter that the PassCue application utilizes a low percentage of the CPU and the memory of an iPhone 5. PassCue uses less then 1% of the CPU and only 5.9 MB of memory in idle state.

Chapter 7

Conclusions

Managing passwords is a significant problem for most people in the modern world. It is challenging to use passwords that are difficult for attackers to guess. In this thesis, the Shared Cues model has been presented, and a password management system based on Shared Cues has been designed and implemented. The password management system is called PassCue and it uses cues in order to share secrets across multiple accounts. The cues are used to derive a unique password for each of the accounts. PassCue has been implemented as an iOS application, which can be used to log on to a system. PassCue uses a (9,4,3)-sharing set, ER as rehearsal schedule and account notes to cope with PCPs. PassCue has an association set size of 200, and the public cues are created using pictures supplied by the user. The PassCue application uses an SQLite database for local data storage. Notifications are used to ensure that the user follows the rehearsal schedules. The *SecRandom-CopyBytes* in the *Randomization Services* API provides cryptographically secure random numbers which ensure that the associations are randomly selected. The PassCue application support cue resetting and provides a detailed overview of each cue. The PassCue application utilizes a low percentage of the CPU and memory of an iPhone 5, and uses less then 1% of the CPU and only 5.9 MB of memory in idle state.

PassCue (9,4,3) proved to provide higher online and offline security compared to the popular used password management schemes *reuse weak*, *reuse strong* and *lifehacker*. In an offline attack, the attacker must invest over $\$700,000$ and it would take over 38 years to crack a PassCue (9,4,3) password on a single GPU on Amazon EC2, given no previous plaintext leaks. The probability that an attacker would retrieve the password in an online attack is 1.47656×10^{-16}, assuming $r = 0$. Unlike *reuse weak*, *reuse strong* and *lifehacker*. PassCue (9,4,3) provides a strong level of online security when $r = 1$, but the offline security of PassCue (9,4,3) breaks down when one password plaintext leak occur. PassCue (9,4,3) requires no extra rehearsals in order to maintain the cue-associations in the associative memory of the user.

PassCue (43,4,1) provides high online security even if multiple plaintext leak attack occurs. The offline security for PassCue (43,4,1) is equal to PassCue (9,4,3) for $r = 0$, but it would take the attacker 8.5 hours to crack a PassCue (43,4,1) password after one plaintext leak. PassCue (43,4,1) requires the user to rehearse the cue-association pairs approximately once a month the first year in order to keep them in associative memory.

PassCue (60,5,1) provides high online security and high security if the attacker performs an offline attack when $r = 1$. If no plaintext leaks have occurred, the attacker must invest \$2.84442 $\times 10^{10}$ in order to search through all possibilities in an offline attack. The offline security of PassCue (60,5,1) after one plaintext attack is the same as PassCue (9,4,3) and (43,4,1) for $r = 0$. The offline security of PassCue (60,5,1) for $r = 2$ is equal to PassCue (43,4,1) when $r = 1$. PassCue (60,5,1) requires the user to perform approximately 20 extra rehearsals the first year to maintain the passwords in associative memory. The offline security of PassCue (43,4,1) and (60,5,1) is not as high as the *strong random and independent* scheme, but it is much more usable. The *strong random and independent* scheme requires approximately 460 extra rehearsals the first year. The PassCue parameters including sharing set, association size, rehearsal schedule and public cues can be customized to support different security and usability needs.

Future Work

It would be useful to test the PassCue application on different types of users in order to improve the design and application experience. User feedback can provide valuable insight in how the application is used and it could reveal unknown needs or user difficulties. Identifying how users with different usability and security needs experience the PassCue application could provide valuable information. It would be interesting to explore how PassCue can be used with high usability for accounts of lower importance, and at the same time high security for important accounts, such as online banking.

User feedback can be a legitimate source to verify if the ER can be used to provide a sufficient rehearsal schedule. It could be possible to add support for multiple rehearsal schedules, and make it possible for the user to select and specify a rehearsal schedule according to personal needs. With ER, most of the the rehearsals occur close to the cue initialization and the intervals increase with time. It would be valuable to explore if it is possible to rehearse a certain number of cues initially, and add new cues proportional to the increase in accounts.

PassCue forces to choose pictures from the photo library or from downloaded pictures. It could be possible to embed a picture library in the PassCue application, and evaluate if the association strength over time is different from using personal pictures.

Differences in PCPs are solved with account notes in PassCue. Incorporating numbers and symbol directly in the password model would be highly valuable. As the PCPs differ, adding symbols and numbers in a password must be optional. The challenge is to find innovative ways to make the numbers and symbols create strong associations. It might be useful to explore some of the techniques used in graphical and geographical password management models, as presented in Section 2.1.3. Diceware and Pixelock introduced in the same section could also inspire new password management models.

PassCue is implemented for iOS and primarily iPhone. Extending the application to support iPad would increase the availability and it would require a redesign of the UI positioning. Implementing PassCue for other platforms, such as Android, Windows Phone, browser plug-in or as a desktop program would significantly increase the usability and accessibility of the application. Implementing a cloud infrastructure would make PassCue highly available, and it would make it possible to add synchronization and backup functionality. Synchronization functionality enables the PassCue cues to be shared seamlessly across all devices.

Bibliography

[1] Jeremiah Blocki, Manuel Blum, and Anupam Datta. Naturally rehearsing passwords. In Kazue Sako and Palash Sarkar, editors, *ASIACRYPT (2)*, volume 8270 of *Lecture Notes in Computer Science*, pages 361–380. Springer, 2013.

[2] Nasa. Password creation rules. `http://www.nas.nasa.gov/hecc/support/kb/Password-Creation-Rules_270.html`, 2014. Retrieved 05.02.2014.

[3] United States Computer Emergency Readiness Team. Choosing and protecting passwords. `https://www.us-cert.gov/ncas/tips/st04-002`, 2013. Retrieved 05.02.2014.

[4] Google. How secure is your password? `https://accounts.google.com/PasswordHelp`, 2014. Retrieved 05.02.2014.

[5] Microsoft. Create strong passwords. `https://www.microsoft.com/security/pc-security/password-checker.aspx`, 2014. Retrieved 05.02.2014.

[6] Jeremiah Blocki, Manuel Blum, and Anupam Datta. Naturally rehearsing passwords. *CoRR*, abs/1302.5122, 2013.

[7] Computerworld. Rockyou hack exposes names, passwords of 30m accounts. `http://www.computerworld.com/s/article/9142327/RockYou_hack_exposes_names_passwords_of_30M_accounts`, 2009. Retrieved 05.02.2014.

[8] The Age. Playstation privacy breach: 77 million customer accounts exposed. `http://www.theage.com.au/digital-life/games/playstation-privacy-breach-77-million-customer-accounts-exposed-20110427-1dvhf.html`, 2011. Retrieved 05.02.2014.

[9] The Verge. Evernote resets all passwords after user information is stolen in security breach. `http://www.theverge.com/2013/3/2/4056704/evernote-password-reset`, 2013. Retrieved 16.02.2014.

[10] USA Today. Zappos customer accounts breached. `http://usatoday30.usatoday.com/tech/news/story/2012-01-16/mark-smith-zappos-breach-tips/52593484/1`, 2012. Retrieved 16.02.2014.

[11] LinkedIn. An update on linkedin member passwords compromised. http://blog.linkedin.com/2012/06/06/linkedin-member-passwords-compromised/, 2012. Retrieved 16.02.2014.

[12] P.G. Kelley, S. Komanduri, M.L. Mazurek, R. Shay, T. Vidas, L. Bauer, N. Christin, L.F. Cranor, and J. Lopez. Guess again (and again and again): Measuring password strength by simulating password-cracking algorithms. In *Security and Privacy (SP), 2012 IEEE Symposium on*, pages 523–537, May 2012.

[13] Martin M.A. Devillers. Analyzing password strength. http://www.cs.ru.nl/bachelorscripties/2010/Martin_Devillers___0437999___Analyzing_password_strength.pdf, 2010. Retrieved 16.02.2014.

[14] Dinei Florencio and Cormac Herley. A large-scale study of web password habits. In *Proceedings of the 16th International Conference on World Wide Web*, WWW '07, pages 657–666, New York, NY, USA, 2007. ACM.

[15] Imperva. Consumer password worst practices. http://www.imperva.com/docs/WP_Consumer_Password_Worst_Practices.pdf, 2010. Retrieved 05.02.2014.

[16] Troy Hunt. The science of password selection. http://www.troyhunt.com/2011/07/science-of-password-selection.html, 2011. Retrieved 05.02.2014.

[17] Nik Cubrilovic. Rockyou hack: From bad to worse. http://techcrunch.com/2009/12/14/rockyou-hack-security-myspace-facebook-passwords/, 2009. Retrieved 07.04.2014.

[18] Saranga Komanduri, Richard Shay, Patrick Gage Kelley, Michelle L. Mazurek, Lujo Bauer, Nicolas Christin, Lorrie Faith Cranor, and Serge Egelman. Of passwords and people: Measuring the effect of password-composition policies. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '11, pages 2595–2604, New York, NY, USA, 2011. ACM.

[19] Defuse. Password policy hall of shame. https://defuse.ca/password-policy-hall-of-shame.htm. Retrieved 10.03.2014.

[20] Casey Johnston. Why your password can't have symbols—or be longer than 16 characters. http://arstechnica.com/security/2013/04/why-your-password-cant-have-symbols-or-be-longer-than-16-characters/, 2013. Retrieved 11.03.2014.

[21] Sophos. Outlook webmail passwords restricted to 16 chars - how does that compare with yahoo and gmail? http://nakedsecurity.sophos.com/2012/08/02/maximum-password-length-outlook-yahoo-gmail-compared/, 2012. Retrieved 11.03.2014.

[22] Matteo Dell'Amico, Pietro Michiardi, and Yves Roudier. Password strength: An empirical analysis. In *Proceedings of the 29th Conference on Information Communications*, INFOCOM'10, pages 983–991, Piscataway, NJ, USA, 2010. IEEE Press.

[23] Richard E. Smith. The strong password dilemma. Addison-Wesley, 2002. Ch. 6.

[24] Anne Wildenhain et al. Comparison of usability and security of password creation schemes. `https://www.cs.cmu.edu/~jblocki/Anne_Wildenhain_2012.htm`, 2012. Retrieved 07.02.2014.

[25] Kenneth Radke, Colin Boyd, Juan Gonzalez Nieto, and Margot Brereton. Towards a secure human-and-computer mutual authentication protocol. In *Proceedings of the Tenth Australasian Information Security Conference - Volume 125*, AISC '12, pages 39–46, Darlinghurst, Australia, Australia, 2012. Australian Computer Society, Inc.

[26] BlueKrypt. Cryptographic key length recommendation. `http://www.keylength.com/en/4/`, 2012. Retrieved 12.02.2014.

[27] Bruce Schneier. Crypto-gram newsletter. `https://www.schneier.com/crypto-gram-9902.html`, 1999. Retrieved 12.02.2014.

[28] Davey Winder. Password managers: Are they safe? which is the best? `http://www.pcpro.co.uk/features/380377/password-managers-are-they-safe-which-is-the-best`, 2013. Retrieved 08.02.2014.

[29] Robert Biddle, Sonia Chiasson, and P.C. Van Oorschot. Graphical passwords: Learning from the first twelve years. *ACM Comput. Surv.*, 44(4):19:1–19:41, September 2012.

[30] Ziyad S. Al-Salloum. GeoGraphical Passwords. *Int. J. Signal and Imaging Systems Engineering*, 1(1/4), March 2013.

[31] The Diceware Passphrase Home Page. What is diceware? `http://world.std.com/~reinhold/diceware.html`, 2014. Retrieved 29.04.2014.

[32] The Diceware Passphrase FAQ. How long should my passphrase be? `http://world.std.com/~reinhold/dicewarefaq.html#howlong`, 2014. Retrieved 29.04.2014.

[33] Pixelock. What is pixelock? `https://www.pixelock.com/`, 2014. Retrieved 29.04.2014.

[34] Jerome H. Saltzer and Michael D. Schroeder. The protection of information in computer systems, 1975.

[35] Lorrie Cranor and Simson Garfinkel. *Security and Usability.* O'Reilly Media, Inc., 2005.

[36] D. Bensinger. Human memory and the graphical password. *Passlogix White Paper*, 1998.

[37] Jeff Yan, Alan Blackwell, Ross Anderson, and Alasdair Grant. Password memorability and security: Empirical results. *IEEE Security and Privacy*, 2(5):25–31, September 2004.

[38] Anne Adams and Martina Angela Sasse. Users are not the enemy. *Commun. ACM*, 42(12):40–46, December 1999.

[39] G. J. Johnson. A distinctiveness model of serial learning. *Psychological Review*, 98(2):204–217, 1999.

[40] George A. Miller. The magical number seven, plus or minus two: Some limits on our capacity for processing information. *The Psychological Review*, 63(2):81–97, March 1956.

[41] Gillian Cohen. Why is it difficult to put names to faces? *British Journal of Psychology*, 81:287–297, 1990.

[42] Alan David Baddeley. *Human Memory: Theory and Practice.* Lawrence Erlbaum Associates, Hove, UK, 1990.

[43] Teuvo Kohonen. *Associative Memory: A System-Theoretical Approach.* Springer, Berlin; New York, 1977.

[44] D J Willshaw and J T Buckingham. An assessment of Marrs theory of the hippocampus as a temporary memory store. *Philos Trans R Soc Lond B Biol Sci*, 329(1253):205–15, 1990.

[45] John R. Anderson, Michael Matessa, and Christian Lebiere. Act-r: A theory of higher level cognition and its relation to visual attention. *Human-Computer Interaction*, 12(4):439–462, 1997.

[46] Cristian Danescu-Niculescu-Mizil, Justin Cheng, Jon M. Kleinberg, and Lillian Lee. You had me at hello: How phrasing affects memorability. *CoRR*, abs/1203.6360, 2012.

[47] Herbert Bloch. Harry caplan, ed. and trans., [cicero] ad c. herennium de ratione dicendi (rhetorica ad herennium). with an english translation. (loeb classical library.) cambridge, massachusetts: Harvard university press; london: William heinemann, 1954. pp. lviii, 433. *Speculum*, 32:150–151, 1 1957.

[48] Frances A. Yates. *The Art of Memory.* University Of Chicago Press, April 1966.

[49] J. Foer. *Moonwalking with Einstein: The Art and Science of Remembering Everything.* Penguin Books Limited, 2011.

[50] NIST. Guide to enterprise password management. `http://csrc.nist.gov/publications/drafts/800-118/draft-sp800-118.pdf`, 2009. Retrieved 12.02.2014.

[51] Crack Station. Impossible-to-crack hashes: Keyed hashes and password hashing hardware. https://crackstation.net/hashing-security.htm, 2014. Retrieved 06.05.2014.

[52] Twitter Help Center. I'm locked out after too many login attempts. https://support.twitter.com/entries/63510-i-m-locked-out-after-too-many-login-attempts. Retrieved 0303.2014.

[53] IEEE log. Data breach at ieee.org: 100k plaintext passwords. http://ieeelog.dragusin.ro/init/default/log, 2012. Retrieved 16.02.2014.

[54] Philippe Oechslin. Making a faster cryptanalytic time-memory trade-off. In Dan Boneh, editor, *CRYPTO*, Lecture Notes in Computer Science, pages 617–630. Springer.

[55] Free Rainbow Tables. Rainbow tables available. https://www.freerainbowtables.com/en/tables2/. Retrieved 16.02.2014.

[56] Dan Goodin. Anatomy of a hack: How crackers ransack passwords like 'qeadzcwrsfxv1331'. http://arstechnica.com/security/2013/05/how-crackers-make-minced-meat-out-of-your-passwords/2/, 2013. Retrieved 12.05.2014.

[57] Coda Hale. How to safely store a password. http://codahale.com/how-to-safely-store-a-password/, 2014. Retrieved 12.05.2014.

[58] Hashcat. Advanced password recovery. http://hashcat.net/oclhashcat/, 2014. Retrieved 26.04.2014.

[59] Jeff Atwood. Speed hashing. http://blog.codinghorror.com/speed-hashing/, 2012. Retrieved 26.04.2014.

[60] AgileBits. On hashcat and strong master passwords as your best protection. http://blog.agilebits.com/2013/04/16/1password-hashcat-strong-master-passwords/, 2013. Retrieved 26.04.2014.

[61] Lucas Kauffman. About secure password hashing. http://security.blogoverflow.com/2013/09/about-secure-password-hashing/, 2013. Retrieved 26.04.2014.

[62] Claude Castelluccia, Markus Dürmuth, and Daniele Perito. Adaptive password-strength meters from markov models. In *NDSS*. The Internet Society, 2012.

[63] Thomas D. Wu. A real-world analysis of Kerberos password security. In *NDSS*. The Internet Society, 1999.

[64] Jianxin Jeff Yan. A note on proactive password checking. In Victor Raskin, Steven J. Greenwald, Brenda Timmerman, and Darrell M. Kienzle, editors, *NSPW*, pages 127–135. ACM, 2001.

[65] Matt Bishop and Daniel V. Klein. Improving system security via proactive password checking. *Computers and Security*, 14(3):233–249, 1995.

[66] William E. Burr, Donna F. Dodson, Elaine M. Newton, Ray A. Perlner, W. Timothy Polk, Sarbari Gupta, Emad A. Nabbus, U.S. Department of Commerce, National Institute of Standards, and Technology. *Electronic Authentication Guideline: Recommendations of the National Institute of Standards and Technology - Special Publication 800-63-1.* CreateSpace Independent Publishing Platform, USA, 2012.

[67] Claude Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27:379–423, 623–656, July, October 1948.

[68] C. E. Shannon. Prediction and entropy of printed english. *Bell Systems Technical Journal*, pages 50–64, 1951.

[69] Amazon. Amazon ec2. `https://aws.amazon.com/ec2/`, 2014. Retrieved 16.02.2014.

[70] Amazon. Previous generation instances. `https://aws.amazon.com/ec2/previous-generation/`, 2014. Retrieved 20.04.2014.

[71] Matthew Bryant. Amazon ec2 gpu hvm spot instance password cracking – hashcat setup tutorial. `http://thehackerblog.com/amazon-ec2-gpu-hvm-spot-instance-cracking-setup-tutorial/#more-576`, 2013. Retrieved 26.04.2014.

[72] Andrew Dunham. Password cracking on amazon ec2. `http://du.nham.ca/blog/posts/2013/03/08/password-cracking-on-amazon-ec2/`, 2013. Retrieved 26.04.2014.

[73] Dr Mads Haahr. Introduction to randomness and random numbers. `http://www.random.org/randomness/`. Retrieved 26.02.2014.

[74] L. R. Squire. On the course of forgetting in very Long-Term-Memory. *J Exp Psychol Learn J Exp Psychol Learn*, 15(2):241–245.

[75] John R. Anderson and Lael J. Schooler. Reflections of the environment in memory. *Psychological Science*, 2(6):396–408, 1991.

[76] Woźniak and E. J. Gorzelańczyk. Optimization of repetition spacing in the practice of learning. *Acta neurobiologiae experimentalis*, 54(1):59–62, January 1994.

[77] Netmarketshare. Mobile/tablet operating system market share. `http://www.netmarketshare.com/mobile-market-share?qprid=8&qpmr=100&qpdt=1&qpct=3&qpcustomd=1&qpsp=60&qpnp=1&qptimeframe=Q`, 2014. Retrieved 10.04.2014.

[78] Apple Developer. Data management in ios. `https://developer.apple.com/technologies/ios/data-management.html`. Retrieved 22.04.2014.

[79] SQLite. About sqlite. https://sqlite.org/about.html. Retrieved 22.04.2014.

[80] iOS Developer Library. Rand(3) - bsd library functions manual. https://developer.apple.com/library/ios/documentation/System/Conceptual/ManPages_iPhoneOS/man3/srand.3.html, 1999. Retrieved 22.04.2014.

[81] iOS Developer Library. Random(3) - bsd library functions manual. https://developer.apple.com/library/ios/documentation/System/Conceptual/ManPages_iPhoneOS/man3/random.3.html#//apple_ref/doc/man/3/random, 1993. Retrieved 22.04.2014.

[82] iOS Developer Library. Randomization services reference. http://www.openbsd.org/cgi-bin/man.cgi?query=arc4random&sektion=3&arch=&apropos=0&manpath=OpenBSD+Current, 2013. Retrieved 22.04.2014.

[83] OpenBSD. Arc4random(3) - openbsd programmer's manual. https://developer.apple.com/library/ios/documentation/Security/Reference/RandomizationReference/Reference/reference.html#//apple_ref/doc/uid/TP40007281, 2013. Retrieved 22.04.2014.

[84] man7.org. Random(4) - linux programmer's manual. http://man7.org/linux/man-pages/man4/random.4.html, 2013. Retrieved 22.04.2014.

[85] iOS Developer Library. Model-view-controller. https://developer.apple.com/library/ios/documentation/general/conceptual/CocoaEncyclopedia/Model-View-Controller/Model-View-Controller.html. Retrieved 10.03.2014.

[86] Google. Creating a strong password. https://support.google.com/accounts/answer/32040?hl=en, 2013. Retrieved 26.04.2014.

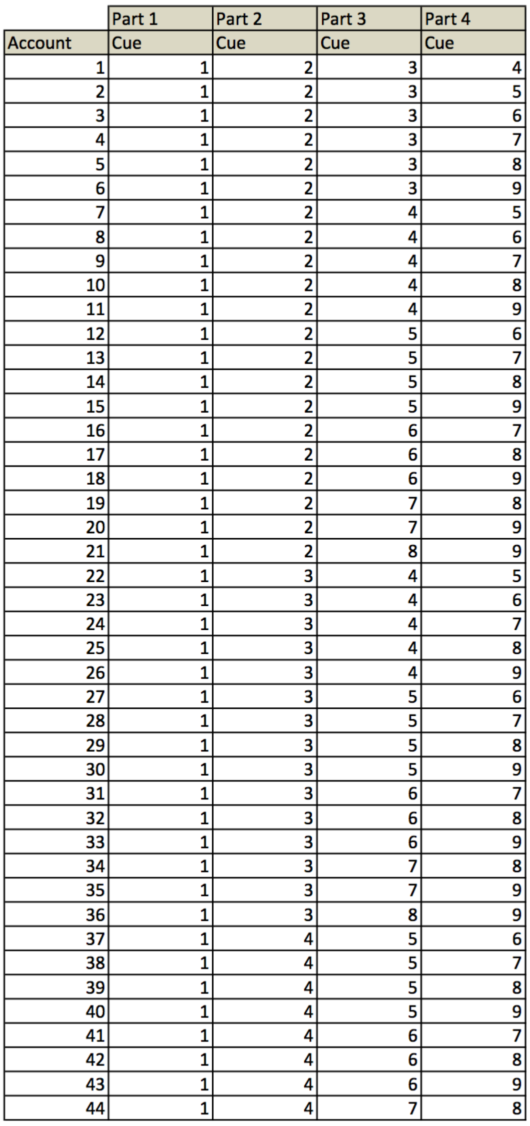
[87] Google. Create your google account. https://accounts.google.com/SignUp?service=mail&continue=https%3A%2F%2Fmail.google.com%2Fmail%2F&ltmpl=default&lp=1&hl=en, 2014. Retrieved 29.04.2014.

[88] Microsoft Security and Security Center. Check your password - is it strong? https://www.microsoft.com/en-gb/security/pc-security/password-checker.aspx, 2014. Retrieved 29.04.2014.

[89] Dan Goodin. 25-gpu cluster cracks every standard windows password in <6 hours. http://arstechnica.com/security/2012/12/25-gpu-cluster-cracks-every-standard-windows-password-in-6-hours/, 2012. Retrieved 20.04.2014.

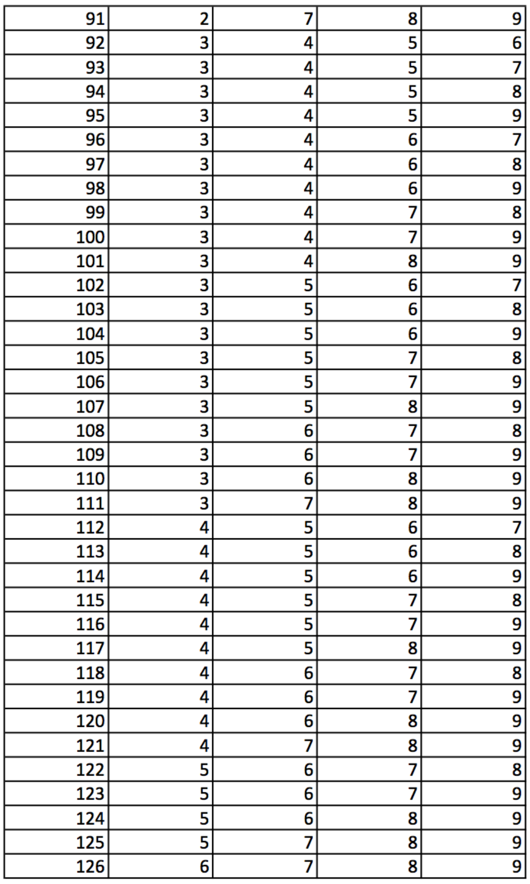
Appendix A

(9,4,3)-sharing set



	Part 1	Part 2	Part 3	Part 4
Account	Cue	Cue	Cue	Cue
1	1	2	3	4
2	1	2	3	5
3	1	2	3	6
4	1	2	3	7
5	1	2	3	8
6	1	2	3	9
7	1	2	4	5
8	1	2	4	6
9	1	2	4	7
10	1	2	4	8
11	1	2	4	9
12	1	2	5	6
13	1	2	5	7
14	1	2	5	8
15	1	2	5	9
16	1	2	6	7
17	1	2	6	8
18	1	2	6	9
19	1	2	7	8
20	1	2	7	9
21	1	2	8	9
22	1	3	4	5
23	1	3	4	6
24	1	3	4	7
25	1	3	4	8
26	1	3	4	9
27	1	3	5	6
28	1	3	5	7
29	1	3	5	8
30	1	3	5	9
31	1	3	6	7
32	1	3	6	8
33	1	3	6	9
34	1	3	7	8
35	1	3	7	9
36	1	3	8	9
37	1	4	5	6
38	1	4	5	7
39	1	4	5	8
40	1	4	5	9
41	1	4	6	7
42	1	4	6	8
43	1	4	6	9
44	1	4	7	8

45	1	4	7	9
46	1	4	8	9
47	1	5	6	7
48	1	5	6	8
49	1	5	6	9
50	1	5	7	8
51	1	5	7	9
52	1	5	8	9
53	1	6	7	8
54	1	6	7	9
55	1	6	8	9
56	1	7	8	9
57	2	3	4	5
58	2	3	4	6
59	2	3	4	7
60	2	3	4	8
61	2	3	4	9
62	2	3	5	6
63	2	3	5	7
64	2	3	5	8
65	2	3	5	9
66	2	3	6	7
67	2	3	6	8
68	2	3	6	9
69	2	3	7	8
70	2	3	7	9
71	2	3	8	9
72	2	4	5	6
73	2	4	5	7
74	2	4	5	8
75	2	4	5	9
76	2	4	6	7
77	2	4	6	8
78	2	4	6	9
79	2	4	7	8
80	2	4	7	9
81	2	4	8	9
82	2	5	6	7
83	2	5	6	8
84	2	5	6	9
85	2	5	7	8
86	2	5	7	9
87	2	5	8	9
88	2	6	7	8
89	2	6	7	9
90	2	6	8	9



91	2	7	8	9
92	3	4	5	6
93	3	4	5	7
94	3	4	5	8
95	3	4	5	9
96	3	4	6	7
97	3	4	6	8
98	3	4	6	9
99	3	4	7	8
100	3	4	7	9
101	3	4	8	9
102	3	5	6	7
103	3	5	6	8
104	3	5	6	9
105	3	5	7	8
106	3	5	7	9
107	3	5	8	9
108	3	6	7	8
109	3	6	7	9
110	3	6	8	9
111	3	7	8	9
112	4	5	6	7
113	4	5	6	8
114	4	5	6	9
115	4	5	7	8
116	4	5	7	9
117	4	5	8	9
118	4	6	7	8
119	4	6	7	9
120	4	6	8	9
121	4	7	8	9
122	5	6	7	8
123	5	6	7	9
124	5	6	8	9
125	5	7	8	9
126	6	7	8	9

Appendix B

Password Schemes

Algorithm 5 Reuse Weak [6]

```
1: Random Word: $w \leftarrow D_{20000}$    ▷ Select $w$ randomly from a dictionary of 20000 words
2: for $i = 0 \rightarrow m$ do    ▷ $m$ is the number of passwords
3:     $p_i \leftarrow w$
4:     $c_i \leftarrow 'cue'$
5: Return: $(p_1, c_1), ..., (p_m, c_m)$
```

Algorithm 6 Reuse Strong [6]

```
1: for $i \leftarrow 4$ do
2:     Random Word: $w_i \leftarrow D_{20000}$    ▷ Select $w$ randomly from a dictionary of 20000 words
3: for $i = 0 \rightarrow m$ do    ▷ $m$ is the number of passwords
4:     $p_i \leftarrow w_1 w_2 w_3 w_4$
5:     $c_i \leftarrow (('cue', j) | j \in [4])$
6: Return: $(p_1, c_1), \ldots, (p_m, c_m)$
```

Algorithm 7 Strong Random and Independent [6]

```
1: for $i = 0 \rightarrow m$ do    ▷ $m$ is the number of passwords
2:     for $j \leftarrow 4$ do
3:         Random Word: $w_i \leftarrow D_{20000}$    ▷ Select $w$ randomly from a dictionary of 20000 words
4:         $p_i \leftarrow w_1^i w_2^i w_3^i w_4^i$
5:         $c_i \leftarrow ((A_i, j) | j \in [4])$
6: Return: $(p_1, c_1), ..., (p_m, c_m)$
```

Algorithm 8 Lifehacker [6]

```
1: for $i \leftarrow 3$ do
2:     Random Word: $w_i \leftarrow D_{20000}$
                                  ▷ Select $w$ randomly from a dictionary of 20000 words
3: Derivation Rule: $d \leftarrow DerivRule$  ▷ $DerivRules$ is a set of derivation rules
   to map the name of a site $A_i$ to a string $d(A_i)$
4: for $i = 0 \rightarrow m$ do                  ▷ $m$ is the number of passwords
5:     $p_i \leftarrow w_1 w_2 w_3 d(A_i)$
6:     $c_i \leftarrow (('cue', j) | j \in [4]) \cup ('Rule')$
7: Return: $(p_1, c_1), \ldots, (p_m, c_m)$
```

Appendix C

Source Code Overview

Table C.1: Overview of the PassCue source files

Name	Type	Functionality
Account.h	C Header Source	Object for account information
Account.m	Objective-C Source	Object for account information
Action.h	C Header Source	Object for action information
Action.m	Objective-C Source	Object for action information
actions	Folder	Contains 10 action images
Association.h	C Header Source	Object for association information
Association.m	Objective-C Source	Object for association information
Cue.h	C Header Source	Object for cue information
Cue.m	Objective-C Source	Object for cue information
DBManager.h	C Header Source	A global object for all communication with the database
DBManager.m	Objective-C Source	A global object for all communication with the database
Default-568h@2x.png	PNG Image	The PassCue start up screen
icon@2x.png	PNG Image	The PassCue icon
Object.h	C Header Source	Object for object information
Object.m	Objective-C Source	Object for object information
objects	Folder	Contains 10 object images
PassCue.xcodeproj	Xcode Project	The PassCue Xcode project
PassCue-Info.plist	Property List	PassCue property list
PassCue.Prefix.pch	C Header Source	Precompiled header
RehearsalSchedule.h	C Header Source	Object for rehearsal schedule information
RehearsalSchedule.m	Objective-C Source	Object for rehearsal schedule information
SharingSet.h	C Header Source	Object for sharing set information
SharingSet.m	Objective-C Source	Object for sharing set information
CueViewController.h	C Header Source	Retrieving and displaying cue information
CueViewController.m	Objective-C Source	Retrieving and displaying cue information
CuesViewController.h	C Header Source	Retrieving and displaying information for all cues
CuesViewController.m	Objective-C Source	Retrieving and displaying information for all cues
ImagePickerViewController.h	C Header Source	Responsible for obtaining cue images
ImagePickerViewController.m	Objective-C Source	Responsible for obtaining cue images
InitAccountController.h	C Header Source	Initializes and creates new accounts
InitAccountController.m	Objective-C Source	Initializes and creates new accounts
InitPAOController.h	C Header Source	Generates PAO-stories and displays associations if required
InitPAOController.m	Objective-C Source	Generates PAO-stories and displays associations if required
main.m	Objective-C Source	Assigning the application control to the app delegate
MainViewController.h	C Header Source	Displaying all accounts and root view controller
MainViewController.m	Objective-C Source	Displaying all accounts and root view controller
PassCueAppDelegate.h	C Header Source	Head of all view controllers
PassCueAppDelegate.m	Objective-C Source	Head of all view controllers
ViewAccountController.h	C Header Source	Retrieves and displays account information
ViewAccountController.m	Objective-C Source	Retrieves and displays account information