

2007

Passwords for Everyone: Secure Mnemonic-based Accessible Authentication

Umut Topkara

Mercan Topkara

Mikhail J. Atallah

Purdue University, mja@cs.purdue.edu

Report Number:

07-008

Topkara, Umut; Topkara, Mercan; and Atallah, Mikhail J., "Passwords for Everyone: Secure Mnemonic-based Accessible Authentication" (2007). *Computer Science Technical Reports*. Paper 1672.

<http://docs.lib.purdue.edu/cstech/1672>

This document has been made available through Purdue e-Pubs, a service of the Purdue University Libraries. Please contact epubs@purdue.edu for additional information.

**PASSWORDS FOR EVERYONE: SECURE
MNEMONIC-BASED ACCESSIBLE AUTHENTICATION**

**Umut Topkara
Mercan Topkara
Mikhail J. Atallah**

**Department of Computer Science
Purdue University
West Lafayette, IN 47907**

**CSD TR #07-008
April 2007**

Passwords for Everyone: Secure Mnemonic-based Accessible Authentication

Umut Topkara Mercan Topkara Mikhail J. Atallah
Department of Computer Science
Purdue University
utopkara, mkarahan, mja@cs.purdue.edu

Abstract

In many environments, the input mechanism to a computer system is severely constrained. For example, a disabled person may only be capable of yes/no responses to prompts from the screen (by different nods of the head, eye movements, hand movements, or even by different thought patterns that are captured by a sensor). Alternatively, the user may not suffer from any impairment yet the environment precludes the use of a keyboard or keypad, as happens with tiny portable devices such as some of the smaller mp3 players, voice sensors at the doors of restricted-access areas, and hands-free situations such as construction work sites, operation of a motor vehicle, etc. Finally, a case can be made, in situations where shoulder-surfing is prevalent (such as in crowded cyber-cafes), for deliberately restricting the input to be a response that is hard to detect by a shoulder-surfer (e.g., left-click vs right-click), even though the user in such cases has a keyboard and is perfectly capable of using it. Requiring the user to remember a long random bit string and to authenticate by entering each bit in the yes/no available input mechanism, is completely impractical. This paper deals with the question of authentication in such environments where the inputs are constrained to be yes/no responses to statements displayed on the user's screen. We present PassWit, a mnemonic-based system for such environments that combines good usability with high security, and has many additional features such as (to mention a few) resistance to phishing, keystroke-logging, resistance to duress and physical coercion of the user, and compatibility with currently deployed systems and password file formats (hence it can co-exist with existing login mechanism). An important ingredient in our recipe is the use of a mnemonic that enables the user to produce a long enough (hence more secure) string of appropriate yes/no answers to displayed prompts (i.e., challenges). Another important ingredient is the non-adaptive nature of these challenges – so

they are inherently non-revealing to a shoulder-surfer or phisher. The mnemonic is a sentence or a set of words known only to the user and authenticating server (in the server they are stored in a cryptographically protected way rather than in the clear) – the users are never asked to enter their mnemonics to the system, they only use the mnemonic to answer the server's challenge questions. Our usage of text for mnemonics is not necessary but it is what we implemented for reasons of convenience and compatibility with existing login mechanisms; we could equally well have used speech, video, or pictures.

1 Introduction

In 1998, Congress amended the Rehabilitation Act to require Federal agencies to make their electronic and information technology accessible to people with disabilities [1]. A major focus of the accessibility research is to improve the design of electronic interfaces in a way that does not prevent users with disabilities (especially vision) from viewing them or navigating through them [7, 6]. There have not been many studies on electronic accessibility issues for users with motor disabilities [14].

The ability to securely use computer resources and the web provides more freedom for users with disabilities: It enhances their educational and entrepreneurial opportunities [14], as well as their ability to stay in touch with friends and family, to manage their finances, or shop online, all without having to rely on other humans to do it on their behalf (thereby improving their privacy as well).

Many of the deployed regular authentication systems are difficult (even impossible) to use under the below-mentioned environments, environments that our system is designed to handle.

- **Users with motor-disabilities:** Not only paralyzed patients (that have cerebral palsy, paraplegia, quadriplegia, etc.), but also users who have rheuma-

toid arthritis or hand tremor ¹, or who are temporarily unable to use their hands (e.g., due to a broken arm or repetitive stress injury).

- **Input constrained devices:** Authentication using tiny portable devices such as some of the smaller mp3 players, game consoles, home entertainment systems, voice sensors at the doors of restricted-access areas, hands-free situations such as construction work sites, operation of a motor vehicle, etc.
- **Inherently non-private environments:** Authentication when shoulder-surfing is unavoidable such as in crowded places (e.g Internet access points, coffee shops, cyber cafes), in places where there are many surveillance cameras (e.g., labs, airports, shopping malls, etc.), in tight spaces such as an airplane while sitting in economy class next to another passenger.

We designed a system, PassWit, that enables users to achieve security of truly random passwords by remembering just a mnemonic sentence (which they have several choices to pick from), and these users will be able to securely authenticate themselves by just answering a series of “yes/no” questions.

This can be achieved without requiring any special input device, or any computation at client site. The authentication questions are designed in a way that a short mnemonic sentence can encode a long password. There is no restriction on the size of the mnemonic sentence or the password, if desired the security (hence the length) of the passwords can be increased by requiring the user to remember more than one sentence. Another important ingredient is the non-adaptive nature of these challenges – so they are inherently non-revealing to a shoulder-surfer or phisher.

PassWit is safe against many attacks including shoulder surfing, phishing, and acoustic attack. We use several measures against each type of these attacks such as displaying one challenge at a time, or displaying the challenges in graphical CAPTCHA [21] ² format if the environment allows graphical display. We also achieve duress resistance, which we will discuss more in the following sections.

Our design can work side-by-side with text passwords in UNIX systems. It is also compatible with pure text based interfaces as well as other media interfaces that can represent the text mnemonics (e.g. speech, video, pictures, etc.).

Moreover, using text for mnemonics (as opposed to pictures, video or audio) brings more flexibility to our

¹According to International Essential Tremor Foundation, up to 10 million Americans have Essential Tremor

²stands for “Completely Automated Public Turing test to tell Computers and Humans Apart”

system, since it is compatible with text consoles and LED screens. There is no language restriction for our system, it can be implemented for languages other than English.

Before going into the details of our system, we will briefly mention the challenges we need to overcome and give a glimpse of how our system approaches them:

- The authentication system would be able to work in input-constrained environments. Users with motor disabilities can input through switches that can be activated by simple muscle movements (e.g. raising the eye brow or eye-lid) or brain signals [11, 5, 17]. In order to free these users from the need to ask the help of another person while authenticating themselves to these systems, one has to come up with a secure authentication system that is usable by anyone who can control such a switch. In our scheme, only yes/no answers are enough for authentication. Of course the ability to provide yes/no inputs makes it possible to transmit any random bit string but it does not help at all for remembering which bit string to transmit (even the luxury of writing the password on a sticky note may not be available to the disabled person). Our scheme, on the other hand, is mnemonic-based and makes it possible to securely remember a long random bit string, by remembering only a relatively short sentence.
- Mnemonics have to be easy-to-remember sentences. Users with motor disability can be elder people that do not have a very strong memory. PassWit encodes a long random password with a short mnemonic sentence. Besides that, we use the news headlines as our corpus for generating the mnemonic sentences. We used newspaper headlines since they summarize a story, thus form a connected discourse, which was experimentally shown [15] to be much easier to learn than same amount of nonsense. Automatically generating easy to remember mnemonics with video or images is a more challenging task.
- Shoulder-surfing attacks are very hard to avoid by disabled users, who are always in areas that are crowded (such as hospitals, or care centers) or always have a second person (e.g., day care nurses) around. PassWit has several measures for protecting the user against shoulder-surfing: i) using mnemonic based passwords, ii) not asking the user to enter the password directly, but performing authentication by asking the user a series of “yes/no” questions, iii) offering a different set of challenge questions at each time of authentication, iv) making the use of an input device that can be covered possible (such as a mouse or other haptic device).

- The authentication system should not require the user to carry an extra portable device (e.g. calculator) or even a paper and a pen. In PassWit the user just reads the challenge statements prompted to him/her at authentication time, and if the challenge contains one of the mnemonic words the user inputs “yes”, otherwise “no”.
- The password initialization and reset should be easy. In PassWit, initialization consists of asking the users on which topic they would like their mnemonic sentence to be, and later providing them many alternative mnemonic sentences. Note that the users can use only yes/no answers to input their choice of mnemonic sentence. At reset time, the user will go through the same process as the password initialization.
- Mnemonics have to be secure. They can not be a popular quote, or the lyrics of a well known song. We achieve this security by using a previously proposed password mnemonic system [18] to generate our mnemonic sentences.

It is easy to come up with yes/no mechanisms for authentication in constrained environments, that are conceptually simple but that suffer from a lack of security, poor usability, or both. This paper presents an authentication method that combines good usability with security. An important feature of our scheme includes full resistance to *replay attacks* by someone who can observe only the user’s sequence of yes/no responses (but not the mnemonics from which they are derived, and that are safely stored between the user’s two ears): Such an attacker acquires no advantage whatsoever over someone who did not observe the yes/no sequence for that login session (in fact the yes/no sequence is indistinguishable from a random binary sequence). Our scheme also offers full resistance to a *phishing attack*: Mnemonics are shared secrets between the user and the authenticating server and the users are never asked to enter their mnemonics to the system. The users can detect an adversary that does not know the shared secret. Otherwise an adversary does not gain any information about the password even if the user answers random phishing challenges.

Section 2 includes an overview of the system and how a user will interact with the system. Section 3 covers our adversary model. We will discuss the details of the system and our implementation in Section 4. The work on related literature will be summarized in Section 5.

2 System Overview

We propose PassWit, an authentication system that is based on mnemonic passwords [18], whose details will be described in the following subsections, where \mathcal{P} denotes the user’s previously existing (and securely generated) password bit string (for now we assume \mathcal{P} is 40 bits long, but we can accommodate any other length).

2.1 Password initialization step

1. The system generates a number of random sentences s_1, \dots, s_λ and displays them to the user. See Figure 1. Each sentence has a length of μ words (not counting functional words such as “the”, “a”, “with”). In our implementation we used $\mu = 10$. For example, s_2 could come from tracing a random left-to-right path along the columns of Table 1, using some of the password bits to select one word from each column. In this case, 4 password bits are used per column and first column shows the bit string encoded by the words in the same row. For example, if $\mathcal{P} = 0101100101010011111101001000101010001101$ then the resulting s_2 is *Angry union artists simply dismissed demand to forgive the laziness of the crazy mayor*. Each s_i is selected from a separate table like Table 1 which was derived from a different text source (e.g., one table from sports, one from stock markets, one from animals, etc). We did not use advanced natural language processing techniques in the generation of these tables, and this is an area for future extensions of this work.
2. The user selects one of the above s_i ’s, suppose it consists of the successive words m_1, m_2, \dots, m_μ .
3. The column from which word m_j was selected (call it C_j) contains what we call the equivalence class (in that table) of the word m_j . We use r to denote the size of an equivalence class; in our example $r = 16$. The user does not need to memorize the equivalence class (only m_j needs to be remembered).

2.2 Authentication step

For $j = 1, \dots, \mu$ in turn, the system asks the user, $\ell = \log_2 r$ questions ($\ell = 4$ in our example) about column C_j , as follows.

1. The system randomly permutes the entries of column C_j before creating the challenges at each session (which foils a replay attack). For the i th entry of C_j in the permuted order, let $b_{i,3}b_{i,2}b_{i,1}b_{i,0}$ be the

binary representation of i . For instance last column of Table 1 might be permuted as {leader, senator, enemy, foe, king, queen, president, chairman, children, mayor, friend, ally, associate, assistant, manager, supporter}.

2. The system creates 4 sets Q_3, Q_2, Q_1, Q_0 such that the i th word of the permuted C_j is included in Q_k if and only if $b_{i,k} = 1$. For the permutation of C_{10} in the previous step, Q_0 would be {senator, foe, queen, chairman, mayor, ally, assistant, supporter}, Q_1 would be {enemy, foe, president, chairman, friend, ally, manager, supporter}, Q_2 would be {king, queen, president, chairman, associate, assistant, manager, supporter} Q_3 would be {children, mayor, friend, ally, associate, assistant, manager, supporter}. Since the entry *leader* has index $i = 0$ in the permutation of this example session, it does not appear in any of the Q_k . See Figure 2 for the challenges of this session, which are displayed in random order (as opposed to alphabetical order) as CAPTCHAs for added security against sophisticated malicious software (the random re-ordering as well as the CAPTCHA representation are not needed if there is no threat of such an adversary).
3. For $k = 3, 2, 1, 0$ in turn, the system displays Q_k to the user who answers “Yes” if the mnemonic word m_j (corresponding to the current column C_j) is in Q_k , and answers “No” otherwise. The contents of each Q_k are displayed in random order each authentication round. (There is no need to randomly permute the ordering of the Q_k ’s to foil a replay attack, as they have already been implicitly permuted by the above-mentioned permutation of the column C_j .)

More on the rationale and security of the above is said later. For now we note that (i) the user’s answers uniquely identify to the server the mnemonic word in each column; (ii) the total number of questions is logarithmic in the size r of each column, so that password security can be increased by a factor of 2^μ by doubling the size of a column yet adding only 1 extra question per column (and, more importantly, without any increase in the size of the mnemonic, i.e., without further burdening the user’s memory); (iii) a shoulder-surfer adversary sees the questions but not the user’s yes/no answers (hence learns nothing); (iv) that a keystroke-logger sees the answers but cannot use them to authenticate itself or to obtain the passwords unless it can relate these answers to the challenges (which are preferably obfuscated as in the figure); (v) that a phisher adversary does not even know what questions to display, immediately alerting the user

to the user that something seriously phishy is going on (in fact even if the phisher got the perhaps-careless user to respond to very unfamiliar challenges, those responses are useless to such an attacker). The randomized display of the contents of a Q_k are aimed at foiling an adversary who is trying to decode the CAPTCHAs and might gain information if they appear in a predictable order (e.g., alphabetical).

3 Adversary Model

We assume that the information used by the system during the mnemonic creation is public and the adversary has equal (or more) computational power than our system.

We foresee five possible types of attacks on this scheme:

- **Shoulder-surfing:** This type of adversary is assumed to have the ability to physically record the authentication session of the user (possibly with a video recording device such as a cell phone). The only thing that appears on the screen is the challenge statements, the answers of the user are not displayed. After the user answers the current challenge, the system refreshes the screen to display the next challenge. It is possible to prevent a shoulder-surfing adversary from obtaining the challenges by displaying them in the form of graphical CAPTCHAs [21]. Even though the shoulder surfer can successfully record and resolve the content of the challenges, the only way she/he can learn the answers is by observing the user’s activities while inputting the answers. For defense against a shoulder surfer that can effectively observe and record the CAPTCHA challenges and the user’s activities at the same time, we recommend the use of input devices that can be operated easily under the table or another cover (such as remote control, mouse buttons or keyboard keys which can be covered by the other hand or other haptic devices).
- **Malicious Software (Spy-ware, keystroke-logger etc.):** Malicious software can record what is being sent by the authentication server as challenges and what is being sent by the user as response. For defense against this adversary our PassWit can be used in conjunction with CAPTCHAs.
- **Brute-force attack:** This type of adversary has access only to the encrypted password file (e.g., “/etc/passwd”). PassWit does not weaken the security of the existing authentication system and is based on mnemonic passwords, which lets the users

	leading	U.S.	couturiers	strongly	resist	pressure	regulate	thinness	popular	models
0000	peaceful	viking	tailor	alarmingly	welcome	attempt	modify	rent	passive	queen
0001	thoughtful	romanian	cartoonist	hardly	agree	haste	alter	wisdom	inept	leader
0010	rich	city	beekeeper	suddenly	reject	duress	cement	culture	able	senator
0011	uninterested	rural	realist	simply	embrace	pressure	manipulate	education	dull	supporter
0100	provoked	irish	firefighters	warily	resist	demand	secure	diligence	hot	king
0101	angry	suburban	artist	doubtfully	renounce	bid	fix	weakness	skilled	ally
0110	outraged	texan	architect	remarkably	submit	call	quantify	salary	adept	foe
0111	neutral	aussie	police	again	honor	ultimatum	measure	pension	dormant	manager
1000	furious	canadian	cubist	blindly	recognize	struggle	forgive	thinness	crazy	friend
1001	poor	union	farmer	suspiciously	allow	operation	change	obedience	gifted	president
1010	average	british	fantasist	delicately	accept	order	limit	laziness	bright	enemy
1011	determined	european	developer	fiercely	surrender	imperative	throttle	spirit	witless	children
1100	strong	downtown	farmer	repeatedly	tolerate	hurry	harness	tenuity	exhausted	associate
1101	calm	urban	goldsmith	reluctantly	permit	insistence	deregulate	slenderness	talented	mayor
1110	silent	italian	musician	discreetly	refuse	ban	restrict	citizenship	clumsy	chairman
1111	ordinary	french	drivers	slowly	dismiss	decree	fiddle	discipline	sharp	assistant

Table 1: The mnemonic generation table for the sentence “Leading U.S. couturiers are strongly resisting pressure to regulate the thinness of the popular models.” The order of words within a column is randomly determined.

to have random passwords that are secure against dictionary attacks [18].

- **Phishing:** Mnemonics are shared secrets between the user and the authenticating server, and the server has to use the knowledge of the mnemonics to generate the challenge prompts. Thus, PassWit inherently has full resistance to phishing attacks by an adversary who does not have this knowledge. The users are never asked to enter their mnemonics to the system; instead, they will be prompted with challenges that can be correctly answered only by someone who has full knowledge of the content of the mnemonic. Since the users will be looking for mnemonic words to appear in the challenge prompts, they will very soon (after seeing one or two challenges) realize that the authenticating server does not know the shared secret.
- **Physically armed attacker (“Duress”):** Duress codes are needed in case of an armed attack, where the user needs a way to alert the police, while obeying all requests of the attacker. Several home alarm systems already handle this situation by providing home owners a duress code together with the regular code. Duress code will turn off the alarm, but it will also send an alert to the police. Same idea can be deployed for any type of password protected account, where the duress code lets the user in, alerts the police and possibly asks the system to adjust the information accessible to a duress situation. PassWit is designed to have duress codes, that the user can enter to login under duress. See Section 4 for details on how duress codes are implemented.

4 Implementation Details

We assume that the environment enables the user to read (or hear, in the case of vision impairment) the challenges displayed on the screen and the user can input the yes/no answers through a switch activated by muscle movements or brain signals. The system includes a large set of tables, S , which are already populated offline. These tables, such as Table 1, are used for generating mnemonic sentences and challenges. Each table has a unique ID. Every table corresponds to a source sentence from a corpus, and these source sentences are stored in the first row of the table. Table 1 was generated using the example source sentence “Leading U.S. couturiers are strongly resisting pressure to regulate the thinness of the popular models.”

In Table 1, the first row includes the source sentence (since functional words are not used for mnemonic generation they are excluded). Every column in this table shows a possible candidate word set for replacing the original word in the first row.

4.1 Mnemonic Creation

At mnemonic-creation time, the system first generates a random password, \mathcal{P} , for the user (e.g. a random string of 40 bits), or the user’s existing password is used. Next step is generating the possible candidates for mnemonic sentences that will encode this password.

If the source sentence has 10 words as in our example sentence, and each of these words have 16 alternatives; a random password chooses one word out of each of these 16 alternative words, hence encodes 4 bits per word, 40 bits in total.

As mentioned in Section 2, candidate sentences are formed by tracing a left-to-right path along these columns guided by the \mathcal{P} . First column shows the bit string encoded by the words in the same row.

The system selects the words that encode \mathcal{P} . This process generates one candidate mnemonic sentence per such table. All of the candidate mnemonic sentences encode the same \mathcal{P} .

At the end, the user is provided with a set of candidate mnemonic sentences to pick from and the random password, \mathcal{P} , to use in a keyboard setting if needed.

Mnemonic creation concludes with the user’s selection of one of the candidate mnemonic sentences for remembering as a mnemonic. (see Figure 1)

Once the user selects which mnemonic sentence to use, the ID of the corresponding table that generated it is recorded in the least significant $\log_2 |S|$ bits of the salted hash of the password file entry.

Since we have many source sentences (say 1024 of them), the user can choose from 1024 different mnemonic sentences generated for each source sentence. However there might be psychological attacks [9] to such a flexible system; hence we advise only a small random portion of these possible mnemonics be given as choice to the users.

4.2 Mnemonic Usage

The mnemonic sentence is not stored in the system, instead the source table ID is stored in the salted password hash. The authentication involves a conversion of the yes/no answers of the user into a password.

We achieve this by generating the challenges in such a way that every yes/no answer narrows the search space by *one-bit*, similar to the idea behind the “20-Question Game”³. In our scheme, instead of looking for one object, we are searching for a password that is composed of concatenation of several substrings, each of which is encoded by a different word of the mnemonic sentence. Each mnemonic word is a member of an equivalence class, and we need to ask several questions that will deterministically find the exact mnemonic word within a class. We ask $\log_2 r$, (e.g., 4), questions to determine one mnemonic word, where r , (e.g., 16), is the number of words in an equivalence class.

The key idea behind generating each challenge is very similar to the idea behind *non-adaptive blood testing* technique [10]. The area of combinatorial group testing concerns itself with performing group tests on subsets of a given set to identify defective elements in that set: A

test for a subset tells whether that subset contains a defective element. If the set size is r and the number of defective elements is no more than d , then the goal is to pinpoint all the defective elements by making as few group tests as possible. This research area arose out of the need to test blood supplies for syphilis antigens during the last World War: Each test was expensive, and using one test for each of the r blood samples was unacceptable, hence the idea of group testing by using each test on a mixture of blood droplets from a subset of the r blood samples [10]. The original problem was adaptive in the sense that test $i + 1$ could be designed after the outcome of test i was known, thereby enabling a simple binary search for the defective element in the special case of $d = 1$. The non-adaptive version of the problem is when all the tests are done in a single round, with all the subsets to be tested determined in advance.

The analogy with our problem is as follows: For each mnemonic word m_j , the r “blood samples” are the r words in m_j ’s equivalence class. The mnemonic word is like the contaminated blood sample. The server presents the user with a subset of words from m_j ’s equivalence class, C_j , (possibly containing m_j) and the user is supposed to respond yes or no based on whether m_j is in that subset (i.e., whether that subset is “contaminated”). A shoulder-surfer type adversary sees the server’s questions, but does not see the user’s yes/no answers. A keystroke-logger type adversary sees the answers but not the questions. The server tests subsets in a manner that enables it to uniquely identify m_j , and then the server does a table lookup (local to the server) to derive the password bit string associated with m_j .

Does the adversary learn anything from the questions? To prevent the adversary from learning anything by using questions, it is imperative for the server to use a *non-adaptive technique* whereby *all* the questions have been pre-determined well in advance, as in non-adaptive combinatorial group testing. The questions are therefore independent of which item in the set is the “contaminated” one, and hence they reveal nothing to the adversary who sees them. Using adaptive group testing techniques (like binary search) for determining the questions would be lethal from the security point of view.

Our scheme will use $d = 1$, for which an $\ell = \lceil \log_2 r \rceil$ test non-adaptive solution is well known and in fact quite straightforward. We briefly sketch it, for the sake of making this paper self-contained. In what follows C_j is the equivalence class of word m_j , where $|C_j| = r$. (Recall the authentication step described in Section 2)

1. Let the words in C_j be listed in an order (which will be randomly changed at every authentication session) as w_1, \dots, w_μ .
2. For each word w_i , let the ℓ bit binary representation

³This game is based on asking the players to think of an object and answer the classification questions asked by the 20Q Artificial Intelligence Game device. See <http://www.20q.net/>

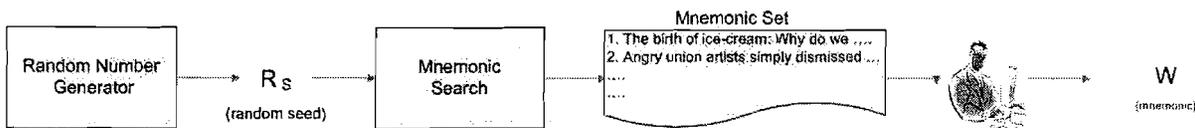


Figure 1: Mnemonic Generation and Selection: Initial selection of the mnemonic sentence involves the users, since memorability of a sentence depends on individual experiences and tastes.

of i be denoted as the bit string $b_{i,\ell-1}, \dots, b_{i,0}$.

3. For $k = 0, \dots, \ell - 1$ in turn, the server's question Q_k is constructed as follows: Every w_j whose $b_{i,k} = 1$ is included in Q_k .

The number of server asks ℓ questions, and each question is constructed without any dependency on which element of C_j is the “contaminated” one, m_j . The server can easily determine m_j : It is the only word of C_j such that all of the Q_k 's that contained it were answered with a “yes” by the user. Note that, this scheme is not restricted to inputting passwords using mnemonics, and it can be used to input plain text passwords when the challenges contain single ASCII symbols.

When the equivalence classes have a size of 16 as in our example, each challenge will have 8 words and the user will be asked 4 questions. An example question for finding which word is the mnemonic word in the last column of Table 1 would be as follows:

- “Does your mnemonic contain one of the following words?:”
 - “senator, foe, queen, chairman, mayor, ally, assistant, supporter”

The user answers 40 such questions in total (4 for each one of the 10 mnemonic words) with a “yes” or a “no” signal using the switch (equivalently with 1 for “yes” or 0 for “no”).

After the answers are collected, the system extracts each password substring encoded by the mnemonic words. These substrings are concatenated in the order corresponding to the order of words in the source sentence to form \mathcal{P} . The hash of \mathcal{P} with the salt is compared to the hash value kept in the password file (where the hash value is stored as in the regular UNIX password file). Note that the same password file can still be used with the ASCII passwords.

Our current password size of 40 bits falls short of the 52 bits commonly used in deployed systems, but we are confident that we will be able to exceed the 52 bits in the continuation and further refinement of this work. In the meantime, even in its present form our current implementation is suitable for use as a front-end to a 52-bit

password system: We would use our system for entering 40 of the 52 bits, and the missing 12 bits would be handled as private salt in a similar fashion to what was described in [13] – by the front-end essentially trying all 2^{12} possibilities for the remaining 12 bits. The password file would stay the same as before our system was deployed, as would the password: We act only as a front end, for special situations where normal keyboard entry is either impossible or risky.

4.3 Duress Codes

After the user picks a mnemonic sentence the system will randomly decide a position, o , on the password bit string, \mathcal{P} . This random position information is stored in the salted hash of the user. Our system provides two different duress codes to the user. We decided to provide at least two duress codes to protect the user even when the attacker is aware of the fact that the system has duress code and asks the user to write down two different passwords and show which one is the duress code, which one is the original password.

Duress password will differ only at one of the 2 positions that comes after position o . If the entered password differs from the correct password at any one (and only one) bit in this sequence, the system will interpret the login attempt to be performed under duress. It will let the authenticating server know about the duress situation, let the user authenticate and send an alarm to the police. Note that, if the entered sequence differs at more than one bit from the duress sequence, the system will interpret this entry as a wrong password and will deny the authentication. This restriction is required in order not to increase the chances of successful dictionary attacks.

For example, when the user's password is equal to $\mathcal{P} = 0101100101010011111101001000101010001101$ and o is 37. The mnemonic sentence is *Angry union artists simply dismissed demand to forgive the laziness of the crazy mayor*. The mnemonic word “mayor” encodes the last four bits, “1101”, and the sequence of bits marked by the duress marker (37) is “10”. Then “president” (encoding “1001”), and “assistant” (encoding “1111”) will be displayed to the user as the duress codes. The user is free to memorize all three or choose the two that are easier to remember.

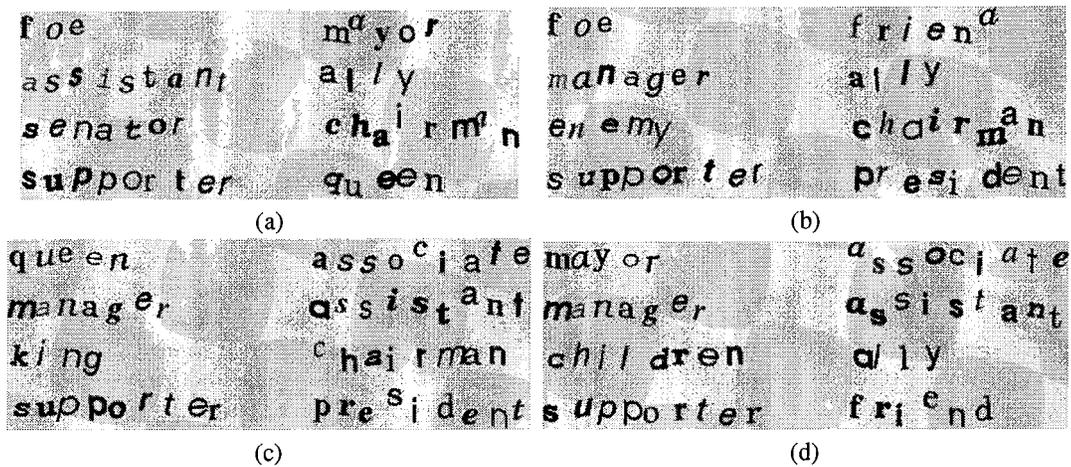


Figure 2: One of the possible set of challenges Q_0 to Q_3 that could be created for the last column of Table 1. The challenge words are presented in random order and in the form of CAPCHAs for added security against sophisticated key-loggers. In the absence of such concerns the challenges can be displayed as text images in alphabetical order. It is also possible to display them in plain text format on text-only consoles or LED displays.

5 Related Work

Previous studies state the requirements for increasing the accessibility of electronic resources for disabled users [1, 7, 6, 14, 2, 4] and suggest possible techniques (including both hardware and software solutions) to increase the bandwidth of input from these users [11, 5, 8]. There is also a body of work for providing access to web through smaller devices which have limited input capabilities [3, 19]. These two research areas have a considerable overlap in the design requirements such as assumption of low input bandwidth, emphasis on usability, and the need for platform independence. Trewin discusses the overlap between the accessibility requirements for desktop browsers for Web, and the requirements for a usable Mobile Web in [19].

Mankoff et al. discuss the needs of motor disabled users for accessing the web in [14]. They define this access as a “low bandwidth” access, due to the fact that these users can produce only one or two signals, when communicating with a computer. These users usually use a switch mechanism that can be controlled in a variety of ways, such as button activated switches, pneumatic switches, switches that are operated by a muscle movement such as raising an eye brow [11], or Brain Computer Interface (BCI) [20]. Mankoff et al. introduced the design requirements for accessibility of web pages by motor-disabled user, these requirements include making currently selected link highlighted, allowing the user access to the bookmarks as links or adding links for skipping unwanted text. They have implemented a proxy and a web browser that can render any given web page and convert it into a page that fulfills the requirements of low

bandwidth accessibility.

The closest work to ours is the “pass-thoughts” authentication system proposed by Thorpe et al. [17]. Pass-thoughts system is based on recognition of unique brain signals send by the users. This system benefits from the BCI technology [20] that can take a brain signal, extract its features and then translate or classify these features into a command. They list the following set of requirements for an authentication system: i) changeability (in case the old one is stolen); ii) shoulder-surfing resistance; iii) theft protection (e.g. acoustic recording of keyboard sounds, or brute force attacks on the password file); iv) protection from user non-compliance (such as sharing the password); v) usability (i.e. fast and easy authentication). In order to fulfill all of these requirements, the authors design an authentication scheme that is solely based on training a user to think about the same idea (e.g. a place, a thing, or a melody), and recording the repeatable parts of the brain signal features extracted from this “pass-thought”. In theory a password space based on pass-thoughts would be very large, since humans can generate many different pass-thoughts, however Thorpe et al. note that the BCI technology, that was available when that paper was published (September 2005), was not able to provide a communication channel with enough bandwidth that can carry a unique brain signal. The users were able to input approximately 25 bits per minute using a BCI device. Since such a low-bandwidth has limited the applicability of pass-thoughts as a high entropy authentication system, the authors provided a feasible pass-thoughts system, where they provide the user with a screen that has a grid of several characters and the user is asked to generate a sequence

of P300 potential spikes, hence highlight several characters one by one on this grid using a BCI device that allows disabled people to spell words. This BCI device records the evoked P300 potentials (generated by the brain 300ms after a surprising or an exciting event) from a user and highlights a random item on the screen for each one of the detected potential, in time the user is expected to learn how to control the P300 potentials and input the same sequence of P300 potential spikes (i.e. the pass-thought) to the system. The verification of the pass-thought is performed by comparing the hash of this input (recorded spikes) with the hash of previously recorded pass-thought.

In 2004, Yan et al. conducted a controlled experiment to compare the effects of giving three alternative forms of advice about password selection [22]. This trial involved 400 first-year students at Cambridge University. 100 of these students were given the classical instructions on how to pick a password: “Your password should be at least seven characters long and contain at least one non-letter.” 100 of them were given a paper that has the letters A-Z and integers 1-9 repeatedly on it, and they were asked to close their eyes and randomly pick symbols from this letter to generate a random password, later they were asked to write it down and carry that paper with them until they memorize the password. The other 100 students were given an instruction sheet that explains how to generate mnemonic passwords. The last 100 were not given any instructions at all. Yan et al. performed several well-known attacks on these passwords, as well as analyzing the statistical properties of these passwords (e.g. length) and the frequency of the users’ need for a password reset.

This study challenged several widely accepted beliefs about security and memorability of passwords:

1. It is confirmed that users have difficulty remembering random passwords (Many students continued to carry the written copy of their password for a long time, 4.8 weeks on the average.).
2. The results also confirmed that mnemonic passwords are indeed harder for an adversary to guess than naively selected passwords.
3. Contrary to the popular belief that random passwords are better than mnemonic passwords, this experiment showed that mnemonic passwords are just as strong as the random passwords.
4. This study also showed that it is *not* harder to remember mnemonic passwords, which are just as memorable as naively selected passwords.
5. Another interesting result of this study is that it is *not* possible to gain a significant improvement

in security by just educating users to use random or mnemonic passwords; both random passwords and mnemonic passwords suffered from a *non-compliance* rate of about 10% (including both too-short passwords and passwords not chosen according to the instructions).

The lack of compliance of users can also be explained with the *lack of incentive*. User incentive can be created by refusing non-compliant passwords or by providing an easy to use authentication scheme, or by bundling small incentives from several systems into a large incentive. In our system, we provide encouraging incentives to the users for complying with our instructions. Some of the user incentives we provide are as follows:

- for users with motor disability providing we provide a way to authenticate themselves without the help of another person, and for all users we provide a way to authenticate themselves when they are faced with an input-constrained environment either due to the lack of a keyboard or due to high risk of shoulder surfing.
- allowing the users to pick the mnemonic sentences that fit to their taste, hence are memorable to them, from a set of mnemonic sentences generated by the system.
- providing an easy password reset mechanism.

In 2005, Jeyaraman and Topkara proposed a system that automatically generates memorable mnemonics for a given random password [12]. This system is based on searching for a mnemonic that encodes the given password in a pre-computed database of mnemonics, which is generated by taking sentences from a text-corpus and producing syntactic and semantic variations of these sentences. In order to produce the variants of corpus sentences, they used linguistic transformations (e.g., synonym substitutions).

A recently introduced mnemonics based password authentication scheme by Topkara et al. [18] allows the users to maintain a multiplicity of truly random passwords, which are independently selected, by remembering only one mnemonic sentence. An adversary who breaks one of the passwords encoded in the mnemonic sentence does not gain information about the other passwords. A key idea is to split a password into two parts: One part is written down on a paper (helper card), another part is encoded in the mnemonic sentence. Both of these two parts are required for successfully reproducing the password, and the password reconstruction from these two parts is done using only simple table lookups. In this scheme changes to passwords do not necessitate

a change in the mnemonic sentence, only requirement is the generation of a new helper card.

Note that in both schemes, [12, 18], the mnemonics are used as an aid to remember text passwords, whereas the current paper enables the use of the mnemonic sentence to serve as the password itself. In the current paper our main challenge is to construct an authentication mechanism that can work in restricted environments. We present a suggested mode of use for other mnemonic password schemes that use other media mnemonics including graphics, and audio besides text. The scheme in this paper provides resistance to phishing, to keystroke-logging, to shoulder surfing as well as to dictionary attacks.

The study of Reverse Turing Tests in [16] suggests a method to ensure that it will take a pre-determined time to break a password with an automated attack if the adversary has to use the login system. This is achieved by judicious use of challenges by the system that require computational capabilities of a human (e.g., CAPTCHAs [21]). PassWit can be complemented with a similar system such that the adversary is even further limited in the time that it is required to break a password.

6 Conclusion and Further Remarks

We presented a password authentication system that is suitable for use in input-constrained environments, and that has many security and password-mnemonic advantages over existing keyboard-based schemes. Because of its compatibility with existing systems (to which it can act as a front-end), it can be used in an intermittent fashion alongside these existing systems: A user may prefer to use the normal keyboard entry most of the time (e.g., at home and in the office) but occasionally switch to using our system in certain situations, such as when the user fears the presence of shoulder-surfers or surveillance cameras, or has a temporary wrist injury that prevents the use of a keyboard, etc.

Yet another advantage of our scheme is that a truly random password does not place any more burden on the user's memory: The mnemonic sentence that our system generates is neither easier nor harder to remember for a strong password than for a weak password. Once users realize this fact, they will tend to make stronger password choices (or even use quality random number generators for that purpose). Making strong passwords more acceptable to users could perhaps turn out to be a greater advantage of our scheme than its suitability for input-constrained environments. Contrast this with what happens with currently deployed systems when an organization forces its staff to use truly random (hence hard to remember) passwords: The little yellow stickers tend to appear near computers, so that the janitorial staff and

perhaps even a sharp-eyed visitor gets to read the password.

Finally, our system is nowhere near its final form, and we will continue to actively work on enhancing it along many directions, including the use of more sophisticated natural language processing techniques than the simple ones we are using currently.

7 Acknowledgement

Portions of this work were supported by Grants IIS-0325345 and CNS-0627488 from the National Science Foundation, and by sponsors of the Center for Education and Research in Information Assurance and Security.

References

- [1] Information technology accessibility and workforce-division, section 508: The road to accessibility, 1998.
- [2] Australian banker's association inc., guiding principles for accessible authentication, Accessed on 4 December 2006.
- [3] W3c mobile web initiative, Accessed on January 10, 2006.
- [4] W3c web accessibility initiative, Accessed on January 10, 2006.
- [5] ADAMS, L., HUNT, L., AND MOORE, M. The 'aware-system' – prototyping an augmentative communication interface. In *Rehabilitation Engineering and Assistive Technology Society of North America Annual Conference* (2003).
- [6] BBC-NEWS. Most websites failing disabled, Published on 2006/12/05.
- [7] BROWN, C. Assistive technology computers and persons with disabilities. *ACM Communications* (1992).
- [8] COPESTAKE, A. Applying natural language processing techniques to speech prostheses. In *Working Notes of the 1996 AAAI Fall Symposium on Developing Assistive Technology for People with Disabilities* (1996).
- [9] DAVIS, D., MONROSE, F., AND REITER, M. K. On user choice in graphical password schemes. In *13th USENIX Security Symposium* (2004).
- [10] DORFMAN, R. The detection of defective members of large populations. *The Annals of Mathematical Statistics* (1943).
- [11] GRAUMAN, K., BETKE, M., LOMBARDI, J., GIPS, J., AND BRADSKI, G. Communication via eye blinks and eyebrow raises: video-based human-computer interfaces. *Universal Access in the Information Society* (2003).
- [12] JEYARAMAN, S., AND TOPKARA, U. Have the cake and eat it too – infusing usability into text-password based authentication systems. In *21st Annual Computer Security Applications Conference* (2005).
- [13] MANBER, U. A simple scheme to make passwords based on one-way functions much harder to crack. *Computers and Security* (1996).
- [14] MANKOFF, J., DEY, A., BATRA, U., AND MOORE, M. Web accessibility for low bandwidth input. In *5th International ACM Conference on Assistive Technologies* (2002).
- [15] MILLER, G. Human Memory and the Storage of Information. *Information Theory, IEEE Transactions on* 2, 3 (1956), 129–137.
- [16] PINKAS, B., AND SANDER, T. Securing passwords against dictionary attacks. In *ACM Computer and Security Conference* (2002).

- [17] THORPE, J., VAN OORSCHOT, P. C., AND SOMAYAJI, A. Pass-thoughts: Authenticating with our minds. In *Workshop on New Security Paradigms* (2005).
- [18] TOPKARA, U., ATALLAH, M. J., AND TOPKARA, M. Passwords decay, words endure: Secure and re-usable multiple password mnemonics. In *ACM Symposium on Applied Computing* (2007).
- [19] TREWIN, S. Physical usability and the mobile web. In *International Cross-disciplinary Workshop on Web Accessibility* (2006).
- [20] VAUGHAN, T., HEETDERKS, W., TREJO, L., RYMER, W., WEINRICH, M., MOORE, M., KUBLER, A., DOBKIN, B., BIRBAUMER, N., DONCHIN, E., WOLPAW, E., AND WOLPAW, J. Brain-computer interface technology: a review of the second international meeting. *IEEE Transactions on Neural Systems and Rehabilitation Engineering* (2003).
- [21] VON AHN, L., BLUM, M., HOPPER, N., AND LANGFORD, J. Captcha: Using hard ai problems for security. In *Eurocrypt* (2003).
- [22] YAN, J., BLACKWELL, A., ANDERSON, R., AND GRANT, A. Password memorability and security: Empirical results. *IEEE Security and Privacy* (2004).