INFORMATION SECURITY APPLICATIONS OF NATURAL LANGUAGE PROCESSING TECHNIQUES

A Dissertation

Submitted to the Faculty

of

Purdue University

by

Umut Topkara

In Partial Fulfillment of the

Requirements for the Degree

of

Doctor of Philosophy

August 2007

Purdue University

West Lafayette, Indiana

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABBREVIATIONS

NL	Natural Language
NLP	Natural Language Processing
NLG	Natural Language Generation
RCV1	Reuters Corpus Volume 1
AMG	Automatic Mnemonic Generator
CR	Coverage Ratio

GLOSSARY

mnemonic	Something used to aid the memory recall, as an easily remembered sentence or acronym
hypernym	A word that is more generic than a given word
hyponym	A word that is more specific than a given word

ABSTRACT

Topkara, Umut Ph.D., Purdue University, August, 2007. Information Security Applications of Natural Language Processing Techniques . Major Professor: Mikhail J. Atallah.

In this thesis we investigate applications of natural language processing (NLP) techniques to information security problems. We present our results in this direction for two important areas: password authentication, and information hiding in natural language text. We have limited this thesis to the realm of language engineering, i.e., our emphasis is on adapting the existing NLP techniques for our purposes, rather than in developing new NLP techniques. Our password mnemonics system helps users to remember random passwords, hence making it possible to implement organizational policies that mandate strong password choices by users. Moreover, in our system password changes do not necessitate a new mnemonic, thereby further easing the users' task of memorizing their respective mnemonics. Our robust natural language text watermarking system can avoid the removal of the watermark text by an automated adversary, in the same way used by authentication systems to avoid an automated adversary's compromise of the password string hidden within the password mnemonic. We have also laid the groundwork for followup research in this area.

1. INTRODUCTION

Language is the most natural mode of information exchange for humans. It is only natural for fields of Computer Science to adopt natural language technologies to facilitate the smooth transition of their products into usage in daily life. In this thesis, we have demonstrated that Natural Language Processing (NLP) technology provides methods that can be used to significantly enhance the security of a variety of applications by providing improved authentication and information hiding mechanisms. Our work is among the first ones to apply NLP to these important information security problems and it introduces significant improvement over the previous approaches.

Natural Language Processing (NLP) studies problems in automated understanding and generation natural human languages. Traditional applications of NLP include the broad areas of (i) computers interfaces that can interact with users in a spoken or written natural language medium, (ii) knowledge acquisition from existing information sources such as books and tv broadcasts, (iii) information retrieval from repositories such as the World Wide Web, and (iv) translation of speech or text among different natural languages. Most of the systems that implement these applications have to perform a common set of analysis tasks which vary in their depth from word forms, and syntax to semantics. In this thesis we have limited our focus to adapting the existing NLP techniques for our purposes, rather than in developing new NLP techniques. We will introduce these NLP techniques throughout the dissertation as we use them.

1.1 Applications of NLP to Authentication

Human aspects of information security have been under the spotlight of recent research in various fields including human-computer interaction, information security and psychology [1]. This interest is well justified by the recent rise of threats that exploit security vulnerabilities involving human factors.

Passwords have a critical place among the security vulnerabilities that involve human aspects, as they are still the first line of defense in many computer systems. Usability of password authentication has been an issue ever since their initial usage. The conflicting requirements of more entropy for increased security, and less entropy for increased memorability have created the password usability problem [2]. Passwords are usually seen by employees as burdens before doing real work. They are almost always disabled if company policy allows or otherwise weakened against the policy. It has been repeatedly shown that users' preference for easily memorizable passwords creates major vulnerabilities in systems [3–5] in the form of the two step attack process of (i) gaining access as the user with a weak password, followed by (ii) escalation of privilege (e.g., through buffer overflow [6]) or other forms of system subversion.

Cost effective alternatives to current text-based password authentication have yet to find widespread acceptance. Alternatives to current password authentication include single sign-in distributed authentication and smartcard-based authentication as well as alternative password methods such as graphical passwords. Even if an alternative to it arises among these candidates, the widespread usage of text password authentication system hints that it will continue to be used quite a while.

Instead of replacing text password authentication with alternatives, we have augmented existing authentication systems with Natural Language Processing technologies to improve password security without sacrificing usability and memorizability (in fact doing quite the opposite). Usability research suggests that a considerable portion of vulnerabilities are due to user behaviors that are related to poor user experience. The integration of natural language into security applications provides a better experience for users, who are more comfortable in dealing with plain natural language text than random strings, thereby making the underlying security stronger.

It is not an uncommon practice among experienced users to create passwords by picking letters from a "mnemonic" phrase; thereby compromising security, because their choices are not random [5]. Inspired by this practice, we have devised methods to automatically generate password mnemonics in order to help people remember their truly random passwords, hence increase the security of existing password authentication systems.

Password mnemonic generation is the task of creating memorable sentences that can encode random passwords; given a random password a password mnemonic generator will output a sentence that encodes the password and is easy to remember. Note that this does not decrease the password's security because is is truly randomly selected *before* the mnemonic is generated.

We have identified the following specific requirements for a password mnemonic system:

- The security of the system with the mnemonic should be at least as good as the security of the system it replaces.
- The attacker should not gain any advantage by better knowledge of the underlying natural language.
- The password space that can be addressed by mnemonics should be as large as without it.
- Mnemonic sentences should be easily memorizable by users.
- The user should be able to decode the password from the mnemonic trivially or easily, without the need for a portable computational device.

User studies suggest that random passwords are an even bigger problem for users that have to remember more than one password for several systems [1, 7, 8]. In a mnemonic system that uses one mnemonic per password, these users would have to remember a different mnemonic for different accounts, unless they reused the same password across these systems. Therefore, there is a need for a mnemonic system in which the users are required to remember only one mnemonic sentence for many passwords ("one mnemonic to rule them all"). The requirements for single password mnemonics are augmented with the following three requirements so that the security guarantees apply to all the passwords encoded with the multiple password mnemonic:

- The scheme produces a single mnemonic for multiple passwords
- Even if the adversary obtains i out of k passwords (by some means), it should not aid him in cracking the $(i+1)^{st}$ password, for all $i < t$. Here t is a security guarantee, such that $0 < t \leq k$
- Changing one of the passwords should not require the change of the mnemonic

Our efforts have led to the novel notion of mnemonics for multiple truly random passwords.

1.2 Applications of NLP to Information Hiding

As we shall discuss next, the Natural Language Processing techniques we developed for improving password security have led us to better results in the area of information hiding. Watermarking is the practice of embedding a message into a cover document such that the message becomes part of the cover document [9] and this embedded message cannot be removed without destroying the value of the document. Watermarking is a valuable tool for systems that manage rights for digital content as well as metadata associated with digital documents. Watermarking has traditionally been studied in the context of video and images, and it has only recently been studied for natural language text documents. Watermarking in natural language text has, in the past, consisted mainly of carrying out approximately meaning-preserving modifications on the given cover text until it encodes the intended mark. A major technique for doing so has been synonym-substitution. In these previous schemes, synonym substitutions were done until the text "confessed", i.e., carried the intended mark message. We propose here a better way to use synonym substitution, one that is no longer entirely guided by the mark-insertion process: It is also guided by a resilience requirement, subject to a maximum allowed distortion constraint. we favor the more ambiguous alternatives. In fact not only do we attempt to achieve the maximum ambiguity, but we want to simultaneously be as close as possible to the above-mentioned distortion limit, as that prevents the adversary from doing further transformations without exceeding the damage threshold. The quantification we use makes possible an application of the existing information-theoretic framework, to the natural language domain, which has unique challenges not present in the image or audio domains. The resilience stems from both (i) the fact that the adversary does not know *where* the changes were made, and (ii) the fact that automated disambiguation is a major difficulty faced by any natural language processing system (what is bad news for the natural language processing area, is good news for our scheme's resilience).

Our approaches to information hiding and mnemonic generation are similar in the steps they take as they both use a very simple but powerful text generation technique. Mnemonic sentence generation is markedly more flexible, since we do not have an attacker who is trying to remove our modifications to the original sentence, nor there is a requirement that the generated sentence carries the same meaning as the original sentence. On the other hand, natural language watermarking does not have the requirement that the decoding be possible through a simple algorithm that can be carried out in a short time by a human.

1.3 Organization of the Dissertation

In **Chapter 2** we propose, develop and evaluate a system that automatically generates memorable mnemonics for a given password based on a text-corpus.

In **Chapter 3** we propose a password mnemonic scheme that can handle multiple passwords with a single mnemonic, and is applicable to any existing system without any modification, as it does not require any form of involvement from the service provider (e.g., bank, brokerage).

Chapter 4 deals with the question of authentication in environments where the inputs are constrained to be yes/no responses to statements displayed on the user's screen (e.g., authentication in hands-free environments, authentication to tiny mobile devices and authentication of people with disabilities, etc.) We present 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).

In **Chapter 5** we explore another application of the techniques we presented in the previous chapters: The resilient watermarking of natural language text through a mechanism that was previously thought to be inherently non-resilient, namely synonym substitution (we used synonym substitution in the first three chapters to encode passwords into mnemonic sentences).

Chapter 6 summarizes the contributions our work and concludes this dissertation.

2. TEMPLATE-BASED MNEMONIC GENERATION

Text-password based authentication schemes are a popular means of authenticating users in computer systems. Standard security practices that were intended to make passwords more difficult to crack, such as requiring users to have passwords that “look random” (high entropy), have made password systems less usable and paradoxically, less secure. We have addressed the need for enhancing the usability of existing text-password systems without necessitating any modifications to the existing password authentication infrastructure. We propose, develop and evaluate a system that automatically generates memorable mnemonics for a given password based on a text-corpus. In this chapter we describe our initial experimental results, which suggest that automatic mnemonic generation is a promising technique for making text-password systems more usable. The initial system that we have built was able to generate mnemonics for 80.5% of six-character passwords and 62.7% of seven-character passwords containing lower-case characters (a-z), even when the text-corpus size is extremely small (1000 sentences). In the next chapter, we will use an improved implementation of the mnemonic generation system which generates longer passwords with improved security.

2.1 Introduction

Text-password based authentication schemes are a popular means of authenticating users in computer systems. The security of password based authentication systems is directly proportional to the difficulty with which an adversary can crack the passwords. A password that is difficult to crack could be intuitively thought of as a string that is not based on a dictionary word and has maximum entropy (truly random) [4, 10–12]. However, the ability to remember completely unrelated sequence of items is very limited in human beings. Hence, the more secure the password is (the greater the randomness), the more difficult it is for users to remember it. This limited ability is further taxed by the fact that a typical user has access to multiple computer systems and is advised to use a unique password for each system. The very requirements that make text-password systems secure seem to make them less user-friendly and paradoxically, less secure. Addams et al. [2] in their survey of users from various organizations, found that standard practices adopted to ensure the security of password authentication systems e.g. expiry mechanisms that force users to periodically change passwords, actually resulted in the lowering of security. For example, 50% of the interviewed users wrote down their passwords in some form – making the passwords susceptible to social engineering attacks. Also, it was found that password expiry mechanisms resulted in the users frequently making "poor" password choices.

Many alternate schemes have been proposed to achieve both security and usability in authentication systems. For example, Graphical Password schemes [13–16], leverage the fact that it is easier for the users to remember pictures. However, we believe that text-password based systems will still remain prevalent in the near future for reasons such as user resistance to change and cost of modifying the existing systems. In this work, we address the need for enhancing the usability of text-password systems without necessitating any modifications. We have developed a system which could help users memorize several random passwords with ease using mnemonics. Our system is complementary to the existing text-password systems and could be used as an add-on or a helper utility. Our work is a promising first step towards reconciling, the seemingly conflicting requirements of text-password systems: security and usability.

Our system increases the memorability of random looking passwords by supplying the users with mnemonics that serve as "reminders" of the passwords. Mnemonics have been widely-used as effective memory enhancement tools to help people remember lists of words [17]. In fact, many security-conscious users generate their passwords based on phrases that they can remember easily. For example, the sentence "The quick brown fox jumped over the lazy dog" could be used as a mnemonic to remember the password “Tqbfj01d”. A recent study by Yan et al. [12] confirms the intuition that mnemonic-based passwords are memorable. The security of passwords generated using such an approach depends heavily on the ability of an user to come up with memorable mnemonics as well as the randomness of the password that these mnemonics encode. It is conceivable that, as the number of passwords each user has to remember increases, the user’s capability to generate memorable mnemonics becomes increasingly taxed. This could lead to insecure practices such as reusing passwords (or their minor variants) across different systems. Another vulnerability of these passwords is that they inherit some regularity from the natural language (e.g., English) they belong. Unless they are carefully constructed to remove this regularity, dictionary attacks can be successfully launched against such passwords [5]. Our system removes relieves the user from the task of generating memorable mnemonics, and the mnemonics that it generates encode truly random passwords.

The system that we describe in this chapter takes an existing text corpus and precomputes a database of syntactic and semantic variations of its sentences, where each variation encodes a different password. Later, for any given random text-password, our system searches through the pre-computed sentence space and automatically generates a list of easier-to-remember natural language sentences that could serve as memory-aids for remembering the password.

The rest of the chapter is organized as follows. In Section 2.2, we describe the architecture and implementation of the automatic mnemonic generation system. In Section 2.3, we discuss some of the issues that arose while developing our mnemonic generation system. We evaluate our system and report our results in Section 2.4. In Section 2.5, we discuss related work that address the challenge of making authentication systems more user-friendly. Finally, we conclude in Section 2.6.

2.2 Automatic Mnemonic Generation

The ability of human beings to remember a sequence of unrelated items is very limited [18, 19]. Hence, remembering a randomly generated password that consists of a sequence of unrelated characters is an onerous task that frequently results in users resorting to unsafe practices such as writing down their passwords [12]. However, research in cognitive psychology [20, 21] has shown that the ability to memorize and then recollect information is positively influenced by associating additional semantic content with that information. The key idea is to automatically generate and associate additional semantic content for any given password so that the additional semantic content then acts as a mnemonic device that assists the user in recalling the password. We use natural language sentences that convey some news or a story to provide the semantic content. We refer to the association between a password string and a mnemonic as *encoding* the password into the mnemonic. One possible way to encode a password in a mnemonic could be to represent every character in the password string by the first letter of a word in the mnemonic. For example, "The **q**uick **b**rown **f**ox **j**umped" can be used to encode the password "qbfj". We do not use stop-words to encode characters (in the previous example, the word 'The' was not used to encode any character). For expository purposes, in the rest of this section, we assume that the passwords consist entirely of lower-case Latin characters (a-z). Later in Section 2.3.1, we explain schemes to encode upper-case, digits and special characters.

The goal of automatic mnemonic generation is to automatically generate natural language sentences containing some news or a story to encode a given password. Instead of generating the sentences from scratch, we use a manually created corpus of natural language sentences each of which contain some news or story. For a password of a given length, if we have a large enough corpus, then it should be trivially possible to encode all possible passwords using the sentences in that corpus. However, it is difficult to obtain a large manually created corpus of sentences. Though there is a lot of text available electronically (e.g., web pages, project Gutenberg [22], etc.), most of the sentences do not contain memorable information. For example, the sentences found in literary works obtained from project Gutenberg are mostly descriptive in nature, with very little "interesting" information to make them memorable. Moreover, even if such a corpus is available, the space requirements of such a corpus is beyond the reach of most modern day computers. For example, to support a password space of 26^8 passwords (8 character passwords with characters from a-z), we need a corpus of the size in the order of 26^8 sentences. Even if each sentence were to take only 10 bytes, such a corpus would occupy more than a *terabyte* of space.

We address these concerns by using a small core-corpus of highly memorable natural language sentences. To offset the lack of variability in a small corpus, we generate and store the semantic variants of each sentence in a compact fashion. Given a password, we check if the sentences in the core-corpus are sufficient to encode it. If not, we perform a dynamic search through the space of semantic variants to identify a variant that can encode the password. Figure 2.2 provides a schematic representation of the automatic mnemonic generation process. In the following sections, we describe the major components involved in the automatic generation of mnemonics.

2.2.1 Reuters Corpus

We use the Reuters Corpus Volume 1 (RCV1) to obtain our core-corpus. The RCV1 has over 800,000 news stories — typical of the annual English language news output of Reuters. Specifically, we use the headlines from each news story to form our core-corpus. The headlines are particularly attractive candidates as mnemonics because:

1. They are simple in structure and hence easy to understand by even the average user.

2. They provide summaries of events and hence contain more semantic information than sentences found elsewhere.

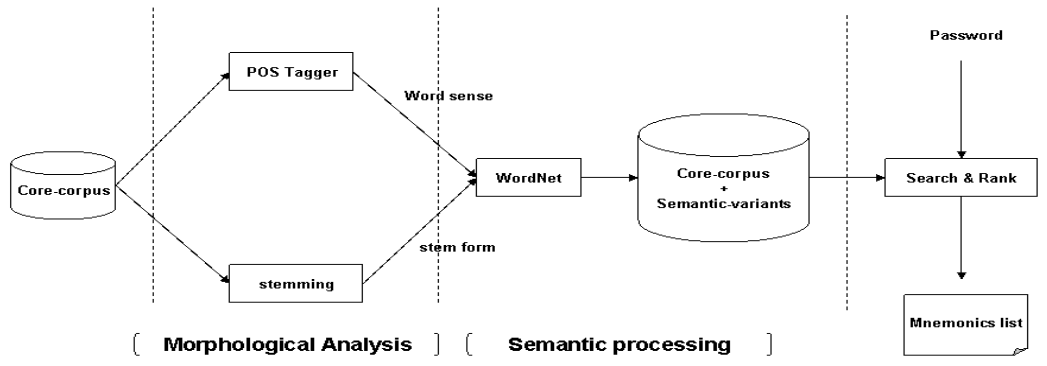


Fig. 2.1. Schematic diagram of the automatic mnemonic generation process. The part-of-speech (POS) and stem information of each word in the core-corpus sentences is input to WordNet to produce semantic variants. For any given password, the semantic variants can then be searched for a match.

3. They are written with the intention of catching the attention of the reader and hence are bound to be very memorable.

An example of a headline that contains memorable information is: “Egyptian plane, overshoots runway, hits Turkish taxi.”

2.2.2 Generating Semantic Variants

We generate semantic variants of each sentence in the core-corpus as follows:

1. We use a sophisticated part-of-speech (POS) tagger to morphologically analyze every word in each sentence and tag it with the appropriate part-of-speech. We use the part-of-speech as a first-degree approximation of the real *sense* of the word.

2. We then use WordNet [23] and the part-of-speech information to generate semantic variants of each word.

Part of speech tagging: A vital piece of information that is necessary for generating semantic variants of a sentence is the *sense* information about each word in the sentence. Many words have different meanings or senses in different contexts. For example, the word *bank* can have the following senses: ‘the river bank’ or ‘the Chase-Manhattan bank’. The correct sense of words is necessary to maintain semantic coherence of the generated variants. Without the knowledge of the appropriate sense, semantic variation techniques such as *synonym substitution* do not work very well. Currently, we do not distinguish between all the different senses of a word. We detect only those senses that manifest themselves as differences in parts of speech. For example, the word *butter* can be used both as a noun and a verb. We use the Stanford Log Linear Part-of-speech Tagger [24] to obtain the part-of-speech information of the words in each sentence. The part-of-speech information serves as a first-degree approximation of word sense. In addition to the part-of-speech information, we perform

Egyptian	{Algerian, Angolan, Basotho, Bantu, Zairese, Zimbabwean, Zulu}
plane	{airplane, autogiro, drone, glider, helicopter, orthopter, warplane}
overshoots	{miss, shoot, overshoot, undershoot}
runway	{platform, auction_block, bandstand, catwalk, dais, dock}
hit	{play, foul, ground_out, toe, snap, kill, drive, hit, launch, loft}
Turkish	{Azerbaijani, Kazak, Tatar, Uzbek, Uighur, Yakut, Kirghiz}
taxi	{cab, hack, taxi, taxicab, minicab, car, automobile, machine}

Fig. 2.2. Examples of semantic relatives of the words in the sentence "Egyptian plane, overshoots runway, hits Turkish taxi" obtained from WordNet.

morphological analysis using PC-KIMMO [25] to obtain the "root" forms, tense and number information about verbs and plural nouns.

WordNet and Semantic variant space: WordNet is a lexical reference system developed by researchers at Princeton University [23]. It organizes English nouns, verbs, adjectives and adverbs into synonym sets, each representing one underlying lexical concept. Different syntactic and semantic relations link the synonym sets. WordNet can be logically thought of as a labeled graph where synonym sets are the nodes and the edges represent semantic relationships. For a given word and its sense, it is possible to obtain semantically related words such as synonyms ($taxi \rightarrow cab$) , antonyms ($hit \rightarrow miss$), hyponyms ($taxi \rightarrow minicab$) and hypernyms ($taxi \rightarrow car$). It is possible to traverse further along the links to obtain a "transitive closure" of hypernyms and hyponyms (hypernyms of hypernyms, hyponyms of hyponyms, synonyms of hypernyms and so on). Let sentence S_i be a sequence of words $w_{i,1}, w_{i,2}, \cdots, w_{i,j}$. Then the semantic variant space of S_i is represented in the corpus as $W_{i,1}, W_{i,2}, \cdots, W_{i,j}$ where $W_{i,k}$ is the set of words that are semantically related to $w_{i,k}$. Figure 2.2 contains an example of semantically related words obtained through WordNet. By traversing through the semantic variant space, specific variants can be obtained. For example, "Zulu balloon, misses dock, kills Kirghiz ambulance" is a semantic variant of "Egyptian plane, overshoots runway, hits Turkish taxi".

2.2.3 Dynamic Search of Semantic Variants

For a given password, if none of the sentences in the corpus can be used as a mnemonic, then the space of semantic variants of each sentence is searched for a possible match. Informally, a match is found if there exists at least one semantic variant that is capable of encoding the password. Formally, a match is found for a password with k characters $c_1, c_2, \cdots, c_k$, if and only if $\exists i$ such that there is a mapping $\{w_1, w_2, \cdots, w_k\} \longrightarrow \{W_{i,x} \times W_{i,x+1} \times \cdots \times W_{i,x+k-1}\}$, where $1 \leq x, x+k-1 \leq |S_i|$ and w_y encodes c_y for $1 \leq y \leq k$. For each match found, we use the tense and number information obtained using morphological analysis to regenerate the surface form of the words in the mnemonic (e.g. $\{flip, presenttense, thirdperson, singular\} \rightarrow flips$). Finally, the user chooses one mnemonic from all the listed matches.

2.3 Discussion

2.3.1 Encoding Passwords in Mnemonics

Apart from the first letter encoding technique used to describe the automatic mnemonic generation process, the other possible encoding schemes are:

1. nth letter encoding: This is a generalized version of the first letter technique where every character in the password string is represented by the nth letter of a word in the mnemonic. For example, the same mnemonic "T**h**e q**u**ick b**r**own f**o**x" can be used to encode the password "huro" using a *second* letter encoding. The obvious limitation for this technique is the fact that words are of limited size. For example, we cannot use a 10^{th} letter encoding in the previously mentioned mnemonic since none of the words are that long. Another limitation might be the fact that the nth letter encoding might not be as user-friendly as the first letter encoding.

2. Last letter encoding: Similar to the first letter encoding, every character in the password string is represented by the last letter of a word in the mnemonic.

Using this encoding, the mnemonic "Th**e** quic**k** brow**n** fo**x**" can be used to represent the password "eknx".

Any of the aforementioned encodings can be used to encode the passwords in mnemonics and for a given password, the generated mnemonics are listed along with the encoding technique used.

2.3.2 Handling Non Lower-case Characters

A challenge that arises due to the use of natural language text as mnemonics is the ability to handle characters that appear infrequently, if at all, in natural language text. e.g. Upper-case characters (A-Z), digits (0-9), special characters ($, % etc.). In the following subsections, we propose and discuss various possible methods to encode such characters in natural language text.

Handling Digits

Numeric digits (0-9) can be encoded in mnemonics using any combination of the following ways:

1. Digits can be treated in exactly the same fashion as the letters. For example, the password "1ioi" can be encoded using the mnemonic "In 1947 India obtained independence", where 1947 is used to encode the digit '1'. This process can be generalized to enable the same mnemonic to be able to encode multiple passwords. If we replace the example sentence with "In NUMBER India obtained independence", then the same sentence can be used to encode any password conforming to the regular expression $[0-9]ioi$. If the password is "2ioi", the mnemonic could be "In 2000 India obtained independence". This is an example of a semantic variation. However, WordNet does not handle numbers. Hence, we perform the tagging and variant generation for numbers on our own.

2. Lower-case characters can be overloaded to represent digits. A scheme similar to what is popularly referred to as "leet speak" can be used. For example, '0' is 'o', '1' is 'l', '3' is 'e' and so on. We discuss the limitations of overloading in Section 2.3.2.

3. Instead of using a single letter to represent a digit, a whole word can be used instead. This scheme is similar to the classical "pegword" mnemonic device which uses words that rhyme with digits. For example, '1' is 'bun', '2' is 'shoe', '3' is 'tree' and so on.

4. Digits can be encoded as length of the word. A well known mnemonic of this type is "May I have a large container of coffee" which encodes the first 8 digits of the number π. However, alternating numbers and alphabet letters may create problems for users, as well as the task of counting letters in words from memory.

Handling Upper-case Letters

We use the first letters of proper nouns found in the natural language text to encode upper-case letters. For example, "Paul chased the dog" is a mnemonic for the password "Pctd". To be able to accommodate occurrences of upper-case letters in different positions, we use semantic tagging and variant generation similar to the tagging used in handling digits. For example, the previous mnemonic is replaced in the corpus with "NAME chased the dog" and a list of all names is maintained separately. So now, the mnemonic could be used to encode any password satisfying the regular expression $[A - Z]ctd$.

Handling Special Characters

There are 32 special characters that are printable in the ASCII characters set, almost none of which appear in natural language text (except those used for punctuation). So, we handle the special characters by overloading the lower-case characters.

If we restrict the number of special characters that appear in the password to less than 26, then for each special character, a unique mapping to a lower-case character can be found. On the other hand, if we want to allow all 32 special characters, then we need to use a sequence of two lower-case characters to represent a single special character e.g., '%' is 'aa', '#' is 'bb', etc.

Overloading Lower-case Characters

A consistent theme that arises during attempts to encode characters that do not frequently appear in natural language text such as digits and special characters, is that of overloading. Overloading a lower-case character to represent digits and special characters introduces an additional level of encoding that destroys the one-to-one mapping between a mnemonic and the actual password. For example, the mnemonic "The asinine brown fox" could encode both "abf" or "4bf" (if '4' is encoded as 'a' as discussed in section 2.3.2) or "]f" (if ']' is encoded as 'ab'). In the worst case, a user who just remembers the mnemonic and the encoding scheme might have to try all the different possible password candidates. We are of the opinion that after a few initial trials during which the user might have to try all possible candidates, the user will be able to recollect the overloaded information precisely. We also believe that the cost of having to remember which letter has been overloaded is very minimal when compared to actually remembering the password itself without the aid of the mnemonics.

2.3.3 Personalization of the Corpus

Though the core-corpus contains highly memorable sentences, it could be made more memorable by personalizing the mnemonics generated according to the user. For example, an user who is a soccer fan, might find a soccer related headline to be more memorable than a headline about the latest stock exchange news. The RCV1 corpus was designed with text-classification applications in mind. Every headline and news story comes tagged with information about one or more of the categories that it belongs to. This makes personalization extremely simple.

2.3.4 Ranking the Mnemonics

For many passwords, there are multiple candidates as mnemonics. There is no general way to rank them. One possible way to rank candidates that are among the semantic variants, is to use a *semantic distance metric* that measures how close they are semantically related to the original sentences they are derived from. The semantically closer relatives are ranked higher. The intuition behind using the semantic distance metric is as follows: The original sentences are the most desirable due to their memorability. The further removed the relation between a semantic variant and the original sentence is, the greater the chance that the variant does not retain the memorability of the original sentence.

2.4 Evaluation

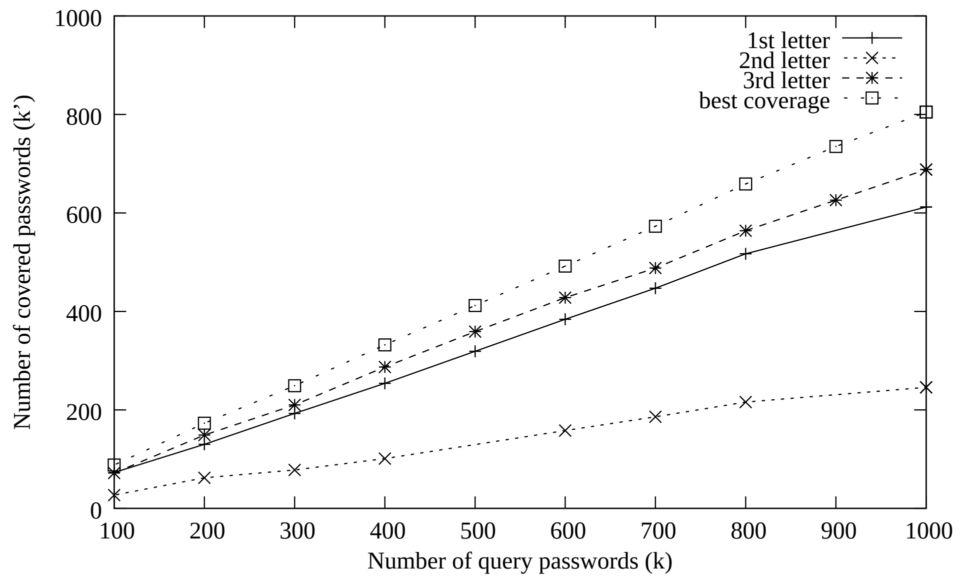
The two properties that we desire in our automatic mnemonic generator(AMG) are *memorability* and *coverage*. Memorability refers to the ease with which the generated mnemonics can be remembered and recollected. We depend on the appealing nature of the sentences in our core-corpus for the memorability of the generated mnemonics. Coverage refers to the number of passwords for which our system can generate mnemonics. The coverage of AMG depends on a variety of factors such as the length of the password, the size of the corpus, and the encoding technique. To measure coverage, we define a ratio called Coverage Ratio (CR). Let n be the maximum number of characters in the password and let S be the alphabet from which the characters of the password are obtained. For a given n, the maximum number of passwords that can be generated using S, $N = \sum_{i=1}^{n} |S|^i$. For a given n and S, the ratio CR is defined as m/N where m is the maximum number of passwords for which AMG can generate mnemonics.

To measure CR, for a given n and S, we need to be able to measure m. m is dependent on factors such as the size of the corpus, the nature of the corpus, the size of the WordNet. This makes it difficult to measure m accurately without having to exhaustively enumerate all the passwords that our system can support. Exhaustive enumeration is computationally expensive for large values of m and n (it is equivalent to a cracker using brute-force to crack a password). Hence, we resort to *sampling* to obtain an estimate of CR. We randomly generate k passwords and test whether AMG can generate mnemonics for the generated passwords. The ratio k'/k is then a probabilistic estimate of CR, where k' is the number of passwords for which AMG is successful in generating mnemonics.

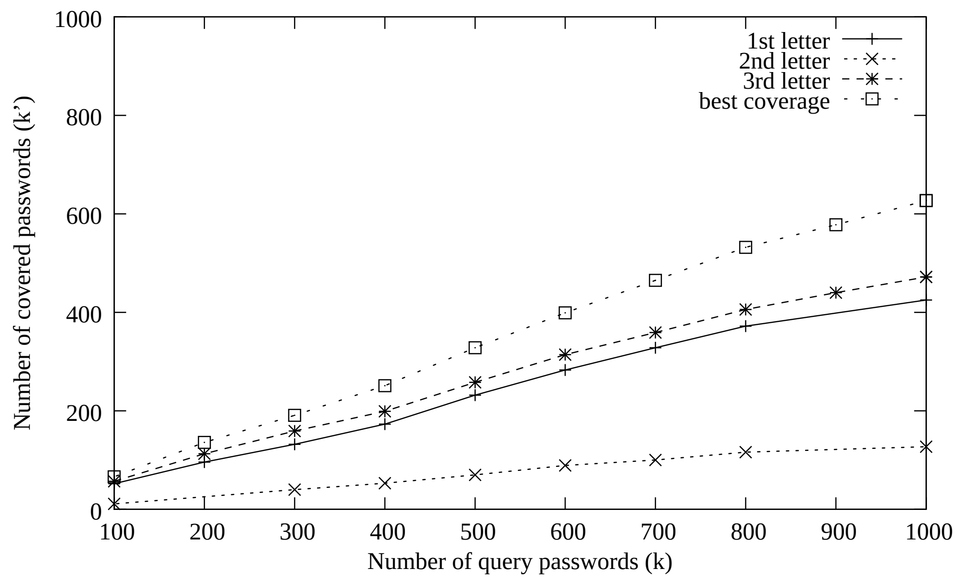
We perform two sets of experiments, one for $n = 6$ and another for $n = 7$. For both experiments, we generate passwords that contain only lower-case characters. In each experiment, we randomly generate a sample of $k = 1000$ passwords and obtain k' using three different encodings (1st letter, 2nd letter and 3rd letter) to obtain mnemonics. The Best-coverage encoding is the union of the results of the three encoding schemes. The size of the core-corpus used was extremely small – 1000 sentences. Figure 2.3 plots the measure of success of AMG in generating mnemonics for both the experiments. Even with such an extremely small core-corpus, our system is able to achieve a coverage ratio of 80.5% for six-character passwords and 62.7% for seven-character passwords.

2.5 Related Work

In this is section, we discuss related work that either share or could be potentially used towards realizing the same broad goal as ours i.e. making password system more usable.



(a)



(b)

Fig. 2.3. Coverage plots for (a) six-character, and (b) seven-character passwords.

2.5.1 Graphical Passwords

The lack of usability in text password systems and the resultant reduction in security has led researchers to propose alternate authentication schemes such as graphical passwords. Graphical passwords leverage the fact that pictures are easier to remember than text [26–28]. In general, graphical password schemes can be divided into two categories: recognition-based and recall-based schemes.

Recognition-based schemes such as $Passfaces^{TM}$ [29, 30] and Deja-Vu [16, 31] leverage the proficiency of human beings in recognizing previously seen images [16,32]. However, as Davis et. al. [13] found out, user choice in such systems could lead to passwords that have much lower entropy than the theoretical optimum, and in some cases, are highly correlated with the race and gender of the users.

Recall-based schemes require users to precisely recall a sequence of strokes or points on pictures. For example, Jermyn et al. [14] describe a scheme where a password consists of a series of lines drawn by the user using an input device (for example, stylus). By decoupling the temporal order in which the lines are drawn from their positions, they observe that they can produce interesting password schemes that are at least as secure as the text passwords while being easier to remember.

Graphical passwords are a promising alternative for text-based password systems in the future. However, as studies by Davis et al. and Thorpe et al. [33, 34] suggest, graphical password technologies are not mature enough to be deployed immediately. Text-password systems with enhanced usability (using systems such as ours), could serve as a viable alternative until such technologies become ready for widespread deployment.

2.5.2 Password Generators

Commercially available tools such as $PasswordGenerator^{TM}$ [35], [36] use a series of ad-hoc techniques to help users generate secure and memorable passwords. As far as we know, there has been no study done on the security and the memorability of the passwords generated by these tools. Also, there has been no study to test the ability of such tools to generate a large number of passwords. The ability to cover a large password space is important because otherwise a cracker could brute-force through all the possible passwords generated by such a tool.

Mnemonic Generators

Research by Raskin et. al. [37] and the mnemonic-password generator from [38] are the closest to our work. As far as we are aware, those are the only two attempts, other than our work, at generating mnemonics by leveraging Natural Language Processing algorithms. Raskin et. al. [37] try to automatically generate "humorous" verse to help remember passwords. While humorous verse is a potentially effective way to remember passwords, automatic generation of humorous verse is a daunting challenge. Existing tools for automatic humor generation do not scale well enough to encode a large password space. For example, Raskin's system can generate mnemonics for passwords that consist of eight characters chosen from a Latin character set (a-z). The main reason for this limitation is the dependence of this system on a hand-written rule for generating new verses (which allows it to run without taxing a heavy computational burden on the system that runs it). Our system can be easily extended to handle arbitrary password length and is capable of handling passwords with characters from an arbitrarily large character-set.

Very little public documentation is available for [38]. Their system automatically generates a series of <password, mnemonic> tuples some of which could be appealing to an user. Similar to our system, they maintain a set of sentence templates internally. However these templates are more generic as they are strings of part-of-speech tags, e.g., "<noun>'s <verb> <adv>, <conj> <noun>'s <verb> <adv>". For each token in the set of templates, a set of words are maintained internally as possible mnemonics. For example, <noun> could be associated with Mark, Dan, Gene, Snake etc., Every time a user requests a password to be generated, the system randomly picks a sentence template and for each token in the template, randomly chooses a word associated with the token. Each word acts as a mnemonic for a character in the generated password. For example, the template "<person>'s <noun> <verb>s <person>'s <adj> <noun>" could be used to generate the sentence "Mark's canary illustrates Dylan's ninth haddock" which encodes a password "MciD9h" [39]. The following differences highlight the advantages of our system over [38]:

- **Sentence templates:** Our system automatically generates the sentence templates from an existing text corpus whereas [38] depends on a manually created template corpus. Using an existing corpus has the following advantage: The generated mnemonics look more "real" and have a greater variety as opposed to the "manufactured" feel of the sentences generated by [38].
- **Semantics of the generated mnemonics:** By virtue of using a text-corpus, the mnemonics generated by our system have richer semantic content than those generated by [38].
- **Extensibility:** Because of the ability to generate syntactic and semantic variants of the sentences from the text-corpus, our system can be easily extended to generate memorable mnemonics for a larger set of passwords and for passwords of arbitrary length.

2.5.3 Automatic Story Generation Systems

Automatic story and prose generation are classical problems studied in the fields of Artificial Intelligence and natural language generation respectively. Stories generated by automatic story telling machines could be potentially used as mnemonics. However, existing automatic story generation tools [40–42], were not designed with our needs in mind and hence are not directly applicable in our context because of the following reasons:

- **Interactivity:** Most of the story generation systems are interactive and are not completely automatic. We need to be able to automatically generate stories.

- **Lack of semantic variability:** The story generation systems do not generate plot-lines for the stories by themselves from scratch. The plot-lines and most of the information needed for narrating a story is available to the generation systems in the form of a knowledge base. Unfortunately, all the systems we know of are research prototypes that were built to demonstrate the effectiveness of the story generation procedure itself. Hence their knowledge bases are often extremely limited. Because of this limitation, the stories generated are often repetitive and "mechanical" in nature. This not only severely affects the memorability of the generated stories, but also limits the number of unique stories generated and hence the number of passwords covered.

2.6 Summary

We propose an automatic mnemonic generating system to make text-password systems more usable. Our system helps users remember crack-resistant passwords by automatically generating mnemonics. We discuss the issues that arise while trying to encode passwords using mnemonics. We evaluate our system and initial results indicate that automatic mnemonic generation is a very promising approach for infusing usability into text-password systems.

The most appealing feature of our core-corpus is that the sentences contain appealing semantic information that make them highly memorable. Semantic variants of a sentence might be less memorable than the original sentence. However, syntactic variants are very attractive since they retain the semantic content and hence the same level of memorability. Similar to the generation of semantic variants, it is possible to generate syntactic variants of every sentence using Natural Language Generation (NLG) techniques. Once a sentence is parsed and is represented using an abstract text representation, the flexibility of the abstract representation allows us to generate many syntactic variants of the same semantic content using grammar rule engines like RealPro.

While generating semantic variants of sentences in our core-corpus, we rely on part of speech information as indicators of senses of the words. It is also possible to incorporate the ability to perform word sense disambiguation at a finer granularity than past of speech for improving the mnemonic generation quality. Existing tools such as [43] can be used to leverage the performance of the current system.

The headlines used in the core-corpus are short in nature. Hence, the length of the passwords that can be encoded using the headlines is limited. There is a need for exploring ways to generate memorable natural language text that can encode longer passwords.

Unlike coverage, there are no objective measures for memorability. User trials may be used to prioritize the mnemonics generated by the system with respect to their memorability.

3. SECURE AND RE-USABLE MULTIPLE PASSWORD MNEMONICS

Research on password authentication systems has repeatedly shown that people choose weak passwords because of the difficulty of remembering random passwords. Moreover, users with multiple passwords for unrelated activities tend to choose almost similar passwords for all of them. Many password schemes have been proposed to alleviate this problem, but they either require modification to the password entry and processing infrastructure (e.g., graphical passwords) or they require the user to have some trusted computing power (e.g., smartcard-like portable devices, browser plug-ins, etc). We propose a scheme that is applicable to any existing system without any modification, as it does not require any form of involvement from the service provider (e.g., bank, brokerage). Nor does it require the user to have any computing device at hand (not even a calculator). Our approach consists of generating a mnemonic sentence that helps the users remember a multiplicity of truly random passwords, which are independently selected. The scheme is such that changes to passwords do not necessitate a change in the mnemonic sentence that the user memorizes. Hence, passwords can be changed without any additional burden on the memory of the user, thereby increasing the system's security. 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 in 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. Passwords' renewal requires only the re-generation of the helper card. Our scheme resolves the apparent contradictory requirements from most password policies: That the password should be random, and that it should be memorized and never written down. This makes possible passwords that are more secure against an adversary who illicitly gains access to the password file, as a dictionary attack is now unlikely to succeed (the attacker now needs to carry out a more daunting brute force enumerative attack). Even if the adversary somehow obtains the helper card, it gets quantifiably limited information about the passwords of the user (so the helper card may be lost or stolen without disaster immediately striking the user). We quantify the time period required for this adversary to successfully crack the password.

3.1 Introduction

Speaking at the opening of the 2005 AusCERT conference, Microsoft's Jesper Johansson said that "companies should not ban employees from writing down their passwords, because such bans force people to use the same weak password on many systems." In [44] Schneier provocatively suggests to users, "write down your passwords", in agreement with Johansson, and adds that people "are much more secure if they choose a password too complicated to remember and then write it down" , since "we are all good at securing a small piece of paper". This is because the probability of the above mentioned sheet of paper compromising the password is much lower than the probability that a weak password will be exploited by an adversary.

In this thesis we show that one can actually write something that is (to the user) a complete description of the password, yet is of little value if the sheet of paper is lost or stolen or otherwise compromised. Our approach is to complement the written sheet of paper (henceforth referred to as the helper card) with an unwritten mnemonic sentence, thereby making what is written on the helper card more cryptic in case it is lost or stolen. In other words, both card and mnemonic sentence are needed for reproducing the passwords; as the mnemonic lies between the user's ears, it provides the security commonly categorized as "something you know", whereas the helper card provides the security categorized as "something you have". We also describe a mnemonic sentence generation mechanism such that an adversary who

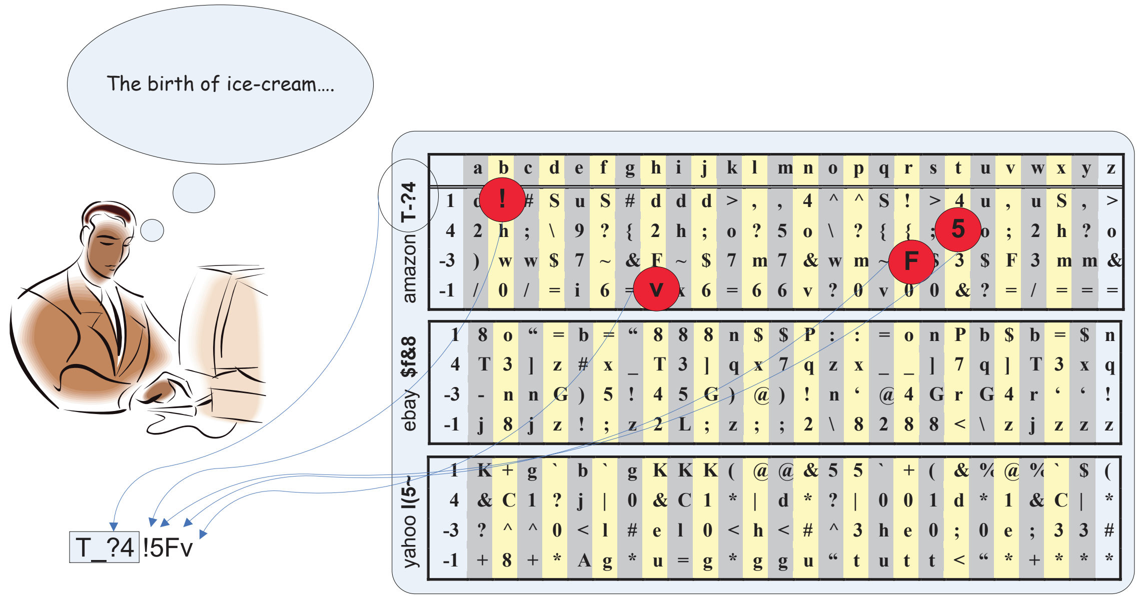


Fig. 3.1. Helper card and a password lookup: The user reconstructs the desired password (e.g., for "amazon") in two steps: i) copying part of the password written in clear text (e.g., "T-?4") ii) looking up the rest of the password from the table by using the letters of the mnemonic word (e.g., "birth" in this case) which are indexed by the letter positions given in the first column (e.g., the first, the fourth, the third from last and the last letters of the mnemonic word). Note that the helper card does not have any markings, the highlights in the figure are inserted for the sake of narrative clarity. Every word that is longer than 4 letters are mnemonic words, and are used in the same order to decode passwords (e.g. "birth" encodes the first password, "ice-cream" encodes the second password on the card)

breaks one of the passwords does not gain additional information about the other passwords even with access to the helper card. Overall our approach does not reduce the security of the existing password authentication infrastructure, it provides a more secure alternative for the users who write down their random passwords.

Before plunging into the details, we briefly mention the challenges we had to overcome, and give a glimpse of our approach to them.

- Using the same mnemonic for multiple passwords, in such a way that one password's compromise does not translate into another password's compromise. The highly structured nature of natural language text, and its known statistical properties, stood in the way of achieving this goal. In a nutshell, we "decoupled" passwords from each other by using only a subset of English: Rich enough to prevent an adversary's enumeration of all possible mnemonics, yet restricted enough that it does not have the above-mentioned drawbacks. For example, the existence of a password that can be encoded with the word "ink" does not imply a more likely existence of another password that can be encoded with the words "paper" or "pen". This requirement turns out to be surprisingly easy to implement, by independently selecting the mnemonic words from each other and later using these words to construct the mnemonic sentence.

- Achieving easy user reconstruction of a password from the helper card and the mnemonic, without any access to a portable computing device (not even a simple calculator). In our system the user makes one simple table lookup per encoded password letter while reconstructing the password. Refer to Figure 3.1 for an example of the helper card table used in this kind of lookup.

- Generating mnemonic sentences that have both desired properties (of being memorable and having the required encoding capacity). Here is an example: "The birth of ice-cream: Why and how we sneeze at midnight." We achieved this by carrying out judicious word-substitutions on a sentence drawn from the news headlines. We used newspaper headlines since they summarize a story, thus form a connected discourse, which was experimentally shown [18] to be much easier to learn than same amount of nonsense. We will explain how later, for now we merely mention that we do this, guided by the distance between words as reported in WordNet [45].

We will first give a usage scenario in Section 3.2 leaving out the discussion of the details that are transparent to the user. Then in Section 3.3 we will discuss the details

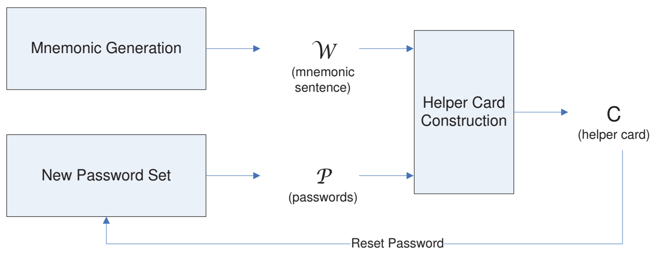


Fig. 3.2. Mnemonic System: The computational parts.

that pertain to the construction of the helper card, the encoding of random strings using a limited vocabulary of words and the generation of natural language sentences. Finally, in Section 3.5, we include a survey of related work in password authentication and with analysis of their impact on our work as well as their similarities, and differences from our work.

3.2 Mnemonic Usage

In this section we describe a typical scenario of mnemonic password usage.

The life cycle of a mnemonic password (Figure 3.2) starts with selection of a mnemonic sentence by the user. The system generates a set of mnemonic sentences, from which the user picks one sentence that is easier to remember and memorizes this sentence. A mnemonic sentence $\mathcal{W}$ contains k mnemonic words $W_1, \ldots, W_k$, each of which are of length $\geq m$.

Later, the user is assigned a set of l random passwords, $\mathcal{P} = \{P_1, \ldots, P_l\}$, by third parties (e.g., banks, brokerages, online shops, school or work accounts, etc.) or the user picks some or all of the passwords. Each of these passwords are composed of $\leq n$ password symbols, $P_i = (p_{i,1}, \ldots, p_{i,n})$.

The system then constructs a lookup table by using $\mathcal{P}$ and $\mathcal{W}$, and this table is printed on a card (the size of a credit card for convenience) as in Figure 3.1. We refer to this lookup table as *helper card*. The user keeps the helper card secure, preferably along with credit cards in the user's wallet.

When the user wants to remember $P_i = (p_{i,1}, \ldots, p_{i,n})$, the user does the following:

1. finds the i^{th} table in the helper card. This table has two parts, a string G_i and a table T_i (e.g., G_1 is `T-?4` in Figure 3.1).

2. types G_i, which is the first $n - m$ symbols of the password $(p_{i,1}, \ldots, p_{i,n-m})$ printed in clear on the helper (possibly $G_i = \emptyset$, if desired).

3. retrieves W_i from memory (e.g., W_1 is the word "birth" in Figure 3.1).

4. derives the remaining symbols of the password $p_{i,n-m+j}$ by i) finds $w_{i,j}$ by retrieving the mnemonic letter pointed by the index value stored in the first column of j^{th} row ii)looking up $w_{i,j}$ in the j^{th} row of T_i (e.g., the first encoded symbol of the "amazon" password is decoded using "b", the 1^{st} letter of the word "birth").

Password Changes: The passwords that are assigned to the user can change, and the user need not change the mnemonic; only the table of the helper card that corresponds to the changed password needs to be reconstructed.

Lost Helper: As we noted earlier, the user needs to keep the helper card secure. However, the helper card may be lost or stolen - without disaster immediately striking the user: the user has a limited but known amount of time before a password in the helper card can be compromised. This secure time period is ensured by the encoding of the password on the helper card; the adversary will likely have to perform a known, large number of brute force login trials before successfully compromising the system. As soon as it is known that the helper card is lost or has been copied by an adversary, the user needs to change all the passwords and generate a new helper card to remember them, however the user does not need to change the mnemonic.

Compromised Password: A very important property of password mnemonic sentences are their resilience against an adversary that has compromised one of the passwords. It is possible that one of the passwords is compromised by an adversary without the information of the helper, e.g., by catching user's keystrokes. Since the passwords are independently generated from each other the rest of the passwords cannot be compromised. However, in the case which the adversary also has access to the helper, the compromise of P_i will leak information about W_i. In a normal English sentence, such knowledge would reveal other words, e.g., the word “ink” is usually used in the same sentence with words “paper” or “pen”. This would have disastrous consequences since the adversary would learn the passwords encoded by those words for free. In order to rule out such situations, the mnemonic is constructed in a way that W_i will not leak information about any other part of the mnemonic; hence the adversary gains no additional information about the rest of the mnemonic or the rest of the passwords by compromising P_i (more on this in Section 3.3).

3.3 Behind the Scenes

This section discusses details of the system that pertain to the parts that do not involve user interaction: The computation of helper cards' content, the generation of candidate mnemonics.

3.3.1 Passwords and Helpers

We first discuss the relationship between the input (i.e., the password) and the outputs (i.e., the mnemonic and the helper card) in the computational parts of the mnemonic system. As noted earlier, the requirements of our multi-password mnemonic system for helper card and mnemonic sentence are: i) both the mnemonic sentence and the helper card are needed to reconstruct the password, ii) this reconstruction of the password can be performed by the user without computational aids, using table lookups.

We now make a brief digression to discuss why the simple “secret splitting” idea from cryptography is not suitable for our purpose. Recall from [46] that to split a secret password $\mathcal{P}$, into two you (i)pick a random R_s , (ii)create the two parts as R_s

(a random seed), and $E(\mathcal{P}, R_s)$. There are several problems with this approach in our application:

1. The R_s (the mnemonic sentence) can not be a random string as it has to be memorable to the user.
2. The decryption of $\mathcal{P}$ is not possible for a human to perform without a computational aid.

Replacing in the above $E(\mathcal{P}, R_s)$ by $\mathcal{P} \oplus R_s$, where $\oplus$ is the bitwise exclusive-or operator, makes it moderately doable by a computer-less human, but has the severe drawback that compromise of one password automatically reveals R_s as well as all the other passwords. Our system uses a different R_s for each password to alleviate this problem.

Note that, $\mathcal{P} \oplus R_s$ would not leak any information about either of R_s or P without the knowledge of the other, hence $\mathcal{P} \oplus R_s$ can be made public. However, R_s, which needs to be kept secret, is still a random string, and is not easier to memorize than the password $\mathcal{P}$ itself. We achieve memorability of R_s by a process called "mnemonic generation", which is the conversion of R_s into an easy to remember human language sentence (e.g., English). The details of mnemonic generation is given in Section 3.3.3; in summary, our system first finds a set of words that encode each R_s (every P will have a different one), then generates sentences by picking one word from each of these sets with the help of natural language generation techniques.

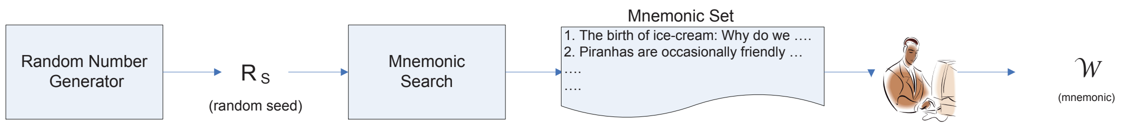


Fig. 3.3. Mnemonic Generation and Selection: Initial selection of the mnemonic sentence involves the users, since memorability of a sentence depends on individual experiences and tastes.

Mnemonic generation (see Figure 3.3) is the task of creating a candidate set of easy to remember sentences that can encode a given random seed string. Mnemonic generation concludes with the user's selection of one of the candidate mnemonic sentences for remembering as a mnemonic.

Let R_s be the random seed that generated the candidate mnemonic set and let the mnemonic chosen by the user be $\mathcal{W}$.

The computation of the mnemonic sentence from the R_s uses an encoding of the R_s as words in a vocabulary; the details of this encoding will be discussed in Section 3.3.3. Let us refer to $d()$ as the corresponding decoding function that can derive R_s from $\mathcal{W}$, such that $d(\mathcal{W}) = R_s$, and refer to $e() = d^{-1}()$ as the encoding function. Note that R_s is different from a password, it is a seed that is used internally by the mnemonic system to produce the candidate set of mnemonic sentences that encode R_s, and is also transparent to the user.

After the mnemonic generation and selection of $\mathcal{W}$ as the mnemonic, a password P is needed to proceed to the helper construction task. Since the mnemonic is selected without the knowledge of password, an adversary who learns the password at this step will not have any information about the mnemonic. Also if an adversary learns the mnemonic, this information will not reveal the password.

Let R_h be the result of $R_s \oplus P$. Since both R_s and P are random strings R_h will also be random. One can compute P using R_s and R_h from $P = R_s \oplus R_h$, or $P = d(\mathcal{W}) \oplus R_h$. In order to be able to perform this operation by hand we choose our decoding function such that $p_i = d(w_i) \oplus r_{h,i}$ as well as $r_{s,i} = d(w_i)$ hold: P can be reconstructed by decoding it symbol by symbol.

Helper construction is the task of building a lookup table that can perform $lookup(i, w_i) = d(w_i) \oplus r_{h,i}$ by looking the i^{th} mnemonic letter, w_i, in the i^{th} row of the table. The password is the concatenation of the successive table lookups for the mnemonic letters in the mnemonic, $P = (lookup(1, w_1), \ldots, lookup(n, w_n))$. An example of helper card lookup was given in Section 3.2. We will discuss the details of the encoding function in Section 3.3.2.

3.3.2 Encoding with Word Sets

Mnemonic generation (see Figure 3.3) is the task of creating sentences that encode a set of random seed strings. In this subsection we will discuss how to find an encoding function and in the next subsection we will describe the generation of mnemonic sentences from word sets.

Let V be the vocabulary of words that our mnemonic sentences will use, $W_i \in V$, and let $Q = \{0, 1, \dots, |Q| - 1\}$ be an alphabet, let S be the i.i.d. set of all strings of length m defined on Q, $S = \{x : x \in Q^m\}$, and $R_s \in S$.

In order to efficiently use the memory of users, we should find an *onto* decoding function that maps V to the largest possible random string set (i.e., maximize the size of S, $|S|$) in a given number of lookup operations, m.

Let N be a sequence of letter indices, such that $length(N) = m$. Let $d_N : V \rightarrow S$ be a desired decoding function. d_N can be computed from individual letters of its input as indexed by N using m mapping functions $d_{N,i} : \{a, \dots, z\} \rightarrow Q$, such that : $\forall v \in V, d_N(v) = (d_{N,1}(v_{N[1]}), \dots, d_{N,m}(v_{N[m]}))$.

In our implementation we performed a heuristic search to obtain a decoding function that maximizes $|S|$ for a given $|Q|$. See Table 3.3.2 for statistics of the results of our implementation for different $|Q|$ values when N is a subsequence of $(1, 2, 3, 4, -4, -3, -2, -1)$. Note that negative values in elements of N refer to positions counting from the last letter backwards. The resulting $|S|$ is very small compared to the size of the `dict` file, this is because we have limited our R_s to be i.i.d. in Q^m.

It is possible to achieve $|S|$ relatively closer to 26^m, which is the maximum number of strings of length m that can be constructed using English alphabet, if we allowed $Prob(R_s) = 0$ for some $R_s \in Q^m$. We have excluded this case from the scope of this chapter, for the sake of simplicity. The number of passwords that can be covered in this case can be trivially computed by finding out the number of unique substrings that the vocabulary yields using N. If we use $N = \{1, 4, -3, -1\}$, thes same word list used to compute Table 3.3.2 yields $67,717$ unique substrings.

The recommended mode of use for our system is not to use more than one letter from a given word to encode a password: we recommend the i^{th} password be encoded in the $N[i]^{th}$ letters of the mnemonic words. This is different from what we are describing in the rest of the chapter the sake of simplicity of description, as well as of practicality (as it may be more cumbersome to go through many words from memory, than to remember only one). The algorithm that computes N ensures that each password symbol encoded by a word is independent of the next password symbol encoded by the same word. Therefore, when the recommended mode of use is adopted, if a password is compromised, the adversary will not be able to infer any information about the rest of the password symbols.

We then used $lookup(i, v) = (d_{N,i}(v_{N[i]}) + z_{N,i}) \mod |Q'|$, where $(z_{N,i} + w_i) \mod |Q'| = p_i$ and Q' is the alphabet from which P is derived (e.g., Q' is the set of printable ASCII characters in most UNIX systems). A helper card constructed this way decodes the i^{th} mnemonic letter, w_i (which is the $N[i]^{th}$ letter of the mnemonic word), into the i^{th} password letter, p_i. Note that, the same $d_{N,i}$ is used for every mnemonic word in Figure 3.1, only $z_{N,i}$ values are different, therefore corresponding rows of different helper tables share the same mapping pattern. For instance the mnemonic letters "h,i,j" always map to the same password letter in the 1^{st} row.

The time required to break a password when the helper card is lost, depends on the size of domain of the encoding function we use, $|S| = |Q^m|$. If t is the time that it will take the adversary to try 1 password for login, on the average it will take $\frac{|Q^m| \times t}{2}$ units of time to break one password using the helper card.

3.3.3 Mnemonic Generation

Recall that our system starts with generating a set of random seeds which will be used to conceal passwords in helper cards. Let $\mathcal{R}_s = (R_{s,1}, \ldots, R_{s,k})$ be the list of k random seeds that were generated for the user, which can be encoded in a mnemonic sentence that has the capacity to remind up to k passwords. In Section 3.3.2 we have

Table 3.1
Decoding function: The results of our heuristic search algorithm to find decoding functions. The input vocabulary was 353056 alphabetic strings from the Linux `dict` file that are longer than 3 letters. The index sequence from which N is computed was $(1, 2, 3, 4, -4, -3, -2, -1)$, and the program was run for alphabet sizes ranging from 3 to 10. The program found an encoding that can map elements from a domain set of 6561 random strings to strings in `dict` file, and has a decoding table that requires 4 lookups.

Size of Alphabet $\|Q\|$	Number of Lookups m	Number of Passwords $\|Q^m\|$	Mnemonic Letter Positions N
3	7	2187	1,2,3,4,-4,-3,-1
4	6	4096	1,3,4,-4,-3,-1
5	5	3125	1,3,-4,-3,-1
6	4	1296	2,4,-3,-1
7	4	2401	2,4,-3,-1
8	4	4096	1,4,-3,-1
9	4	6561	1,4,-3,-1
10	3	1000	1,3,-3

described an encoding function that maps random strings into sets of vocabulary words, $\mathbf{W}_i = e(R_{s,i})$.

In this section we will describe how our system generates sentences by picking one word from each word set $\mathbf{W}_i$ with the help of natural language generation techniques.

In order to achieve memorability of the generated sentences we use a large corpus of newspaper headlines as template sentences: while doing the best effort to maintain the original meaning of them, we modify them through word substitutions to contain a subsequence $(W_1, \ldots, W_k)$ where $W_i \in \mathbf{W}_i$. Sentences that our system generates are usually easy to memorize, since the templates that we use have been curated by newspaper editors to summarize stories thus form a connected discourse, which was experimentally shown [18] to be much easier to learn than same amount of nonsense.

Let L be a template sentence from our corpus, where $L_s = (l_1, \ldots, l_k)$ are the words that our system uses for word substitutions. In order to preserve the attractiveness of the original sentences, we employ a heuristic search for substitutions which minimizes the maximum value of $\delta(l_i, W_i)$, where δ is a function that quantifies the similarity of the input words.

In our implementation we used the `path` function that is part of the `WordNet::Similarity` [47] library as our similarity function δ. `path` returns the length of the shortest path of "is-a" relationships between two concepts in WordNet [45].

Let us use an example to demonstrate the search for a good substitution word, where l_1 is the template word "newspaper" and $\mathbf{W}_1 = \{mirror, letter\}$ are the words that encode our desired random string. The word "letter" would be preferred by our system to substitute the word "newspaper", because $\delta(newspaper, mirror)$ is 8 and $\delta(newspaper, letter)$ is 6 ("newspaper#n#1" denotes the 1st noun sense of the word "newspaper"):

- Shortest path between "newspaper" and "mirror":
  "newspaper#n#1 press#n#3 print_media#n#1

medium#n#1 instrumentality#n#3 device#n#1
reflector#n#1 mirror#n#1"

- Shortest path between "newspaper" and "letter":
  "newspaper#n#3 product#n#2 creation#n#2
  representation#n#2 document#n#2 letter#n#1"

As we have noted before, if the adversary compromises the helper card as well as the password P_i, the set of mnemonic words from which W_i is picked can be easily revealed. This would have disastrous consequences about the security of the rest of the words in the mnemonic sentence, since natural languages have a regular structure. In our approach, $\mathcal{W}$ is derived from $\mathcal{R}_s$ which is randomly generated. Even if the adversary learns one of the sets $\mathbf{W}_i$ used in the mnemonic sentence, this will not leak information about the rest of the mnemonic words.

Our approach resembles synonym substitution based sentence generation used by several security applications [48–51]; we include a review of these applications in Section 3.5. In these applications a given sentence is converted to a new sentence by changing its words with one of their synonyms.

3.4 Security and Usability of Mnemonics

In this work, we have used mnemonics to encode a secret string which is then used to conceal the real password through a secret-sharing mechanism. The security of the password is dependent on the strength of this string against an adversary who performs brute-force cracking. The length of the mnemonic string is a direct contributor to the security of the mnemonic in this respect: as it dictates the length of the secret string. On the other hand, the user may find it hard to remember a long mnemonic sentence, and resort to practices that may compromise the passwords, such as writing down the mnemonic sentence.

A desired mnemonic system should allow the use of shorter mnemonic sentences even though it means lowered security. The challenge for multiple-password mnemonics is that, there are less letters in the mnemonic to distribute to different passwords.

Our system allows the use of shorter mnemonic sentences for encoding multiple passwords. Recall from the example in Table 3.3.2 that the encoding function allowed us to encode up to $\log_2 9$ bits per letter when we used 4 letters per word in the mnemonic sentence to encode passwords. The system will need to distribute these letters to each password of the user, when the passwords are selected. If the user chooses to use a mnemonic of 4 words, which has 16 mnemonic letters, these letters can be used to encode all characters of 2 passwords of length 8. Alternatively, the same mnemonic can be used to encode 4 characters of 4 passwords, where the other 4 characters are written in clear in the helper card. One could concievably choose the encoding function, from the second row of Table 3.3.2, that uses 6 letters from each mnemonic word. However, this encoding function is inferior to the one in our discussion, because it can encode only $\log 26$ bits per letter; hence it provides less security, but still requires more steps to decode the password from a mnemonic of the same length. Therefore, we use the encoding function that packs the largest number of bits for each mnemonic letter.

The length of the mnemonic sentence can be adjusted in a trade-off between the usability and security. The security that the mnemonic provides increases by a constant number of bits with the addition of another mnemonic word. It is not clear how the memorability of the mnemonic reduces with this change. However, one should expect that the decline is usability should depend on the individual user, and beyond a certain mnemonic length, the usability diminishes for all the users.

We envision our system to be used for passwords that the users use infrequently and still require a high security; both memorability of the mnemonic and the security of the passwords are important. As we have qualitatively demonsrated above, these two are competing goals and they balance on a trade-off. If the security emphasis is paramount the mnemonic should be chosen long enough such that all the password letters are encoded in the mnemonic sentence, i.e. each password letter is derived using a table lookup from the helper card. If the implementation requires a higher usability, our system mitigates the security implications for a brute-force adversary by choosing a longer password, but writing some part of this password in clear text on the helper card. However, the portion of the passwords encoded in the mnemonic sentence cannot be too small, for the reasons discussed in the previous sections.

3.5 Related Work

Morris and Thompson have studied the vulnerabilities of the initial UNIX password scheme in 1979, and found out that 86% of the passwords they have collected were easily cracked in a day during an exhaustive but intelligent search of the password space [52]. They identified several improvements to UNIX password system, including the use of DES encryption and the enforcement of longer password, so that an automated attack would suffer from a longer running time.

In 1989, Feldmeier and Karn revisit the password authentication problem in UNIX [3]. This time they are armed with faster *crypt* function implementation that is used is encrypting passwords in password files, as well as faster and cheaper computers. They suggest facilitating of shadow password file as opposed to public password file, as a first measure to defend the passwords. However they point out that the ultimate password protection is picking passwords that are hard to guess by an automated brute-force attacker. They recognize that the usability is a barrier against forcing users to remember long random passwords. Giving Shannon's results for per-character entropy of English, they suggest that a 5-10 word English sentence will embody an entropy required in an 8 character password. The users of these *passphrase*s will *XOR* each 8 character block of their passphrase and use the result as their random password.

The passphrase system suggested by Feldmeier and Karn could be implemented easily. However the number of key strokes required at every authentication would render this passphrase hard to use for frequent authentication. For this reason they suggest distributed authentication systems such as Kerberos tickets to be used in conjunction with the passphrase system for decreasing the overall number of keystrokes and the additional burden of remembering multiple password phrases as a usability fix. Moreover the authors suggest frequent changes of passwords to increase the difficulty of cracking passwords. They identify the complexity of the users' password as the major security requirement.

In 2004, Yan et al. conducted a controlled experiment to compare the effects of giving three alternative forms of advice about password selection [12]. 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 (e.g., keeping the helper card secure). Some of the user incentives we provide are as follows:

- providing the users with a facility that they can also use secure passwords for their personal (non-work) online accounts, since one mnemonic can encode multiple passwords. This way the users will have a good reason to keep the helper cards secure.

- 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 a password reset mechanism that does not require the reseting of the mnemonic.

The reward is high in a system that is usable for multiple passwords. A uniform interface (e.g. helper card) will attract users, because repetition across different systems will bring ease of use, and familiarity.

Kuo et al. performed another user study to check whether mnemonic passwords are vulnerable to dictionary attacks when the dictionary is specifically generated using the popular phrases available on the Internet such as advertising slogans, children's nursery rhymes and songs, movie quotes, song lyrics, etc. [53]. Even though 65% of the mnemonic phrases picked by the users can be found by a Google search, only 4% of the mnemonic passwords, that the users generated, was cracked by a brute force attack using a special dictionary generated by deriving mnemonic passwords from phrases grabbed from popular Internet sites.

A user study by Gaw and Felten showed that users have a tendency to "re-use" their passwords across multiple accounts [54]. They have interviewed with 49 users. The majority of these users had three or fewer passwords and passwords were re-used twice. The top reason mentioned for reusing a password was "easier to remember". Results of this study agrees with our motivation to design a system that encodes multiple passwords with one mnemonic phrase.

In 2005, Jeyaraman and Topkara proposed a system that automatically generates memorable mnemonics for a given random password [50]. 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). This system was able to generate mnemonics for 80.5% of 6-character passwords and 62.7% of 7-character passwords containing lower-case characters (a-z), even when the text-corpus size is extremely small (1000 sentences). Even though this is a very important first step in developing secure mnemonic password systems, it was limited in providing following items:

- Generating mnemonics that can encode symbols that are not lower-case characters
- Encoding multiple passwords with one mnemonic sentence
- Encoding long passwords

There are several other studies that use synonym substitution as a technique for information hiding into natural language text. Natural language information hiding systems, that are based on modifying a cover document, mainly re-write the document using linguistic transformations. T-Lex is one of the first implemented systems that embed hidden information by synonym substitution on a cover document [48]. A given message is embedded into the cover text using a pre-generated synonym set database as follows. First, the letters of the message text are Huffman coded according to English letter frequencies. The Huffman coded message is embedded into message carrying words in the cover text by replacing them with their synonyms in the synonym database of T-Lex. The synonym sets in this database are interpreted as a mixed-radix digit encoding according to the set elements' alphabetical order.

In [51], Topkara et al. introduced a natural language watermarking system that embeds the watermark message (e.g, copyright notice) into a given document using robust synonym substitution. Compared to naive synonym substitution, robust synonym substitution introduces ambiguities in order to make it harder for the watermark embedding modifications to be undone. This scheme first selects a subset of words from a given dictionary, and later assigns colors to these words, where the colors are used to represent the role of the word in embedding the watermark, such as "carries 0", "carries 1" or "non-encoding". A secret key, shared between watermark embedder and watermark reader, is used for both the subset selection and the color assignment of the words. When there are many alternatives to carry out a substitution on a word (i.e. more than one synonym carries the required embedding bit), they prioritize these alternatives according to a quantitative resilience criterion and favor more ambiguous alternatives. For example, if the original sentence is "he survived without water and food for 3 days", this watermarking system rewrites it as "he went without water and food for 3 days", since the word "go" is more ambiguous (i.e. has many different meanings). The approximately meaning-preserving changes, that are done in the direction of more ambiguity, are harder for the adversary to resolve with automated disambiguation, however, a human reader can quickly disambiguate the meaning when reading the marked text. This system exploits the well-established fact in the natural language processing community, that humans are much better than computers at disambiguation [55].

The study of Reverse Turing Tests in [56] 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 [57]). Our system 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 in the helper card. Moreover, if such a system is in use, the number of table lookups can be adjusted to fit the time that the users need to before the adversary can break the password.

In another work, Bergmair et al. proposes a Human Interactive Proof system which exploits the fact that even though machines can not disambiguate senses (i.e. meanings) of words, humans can do disambiguation highly accurately [49]. In this system, the users are shown several "challenges", where each challenge is composed of several sentences generated by synonym substitution of a word in the template sentence. The word that is substituted is replaced with its synonyms from the same sense in some of the challenge sentences and its synonyms from other senses in the remaining sentences. The user is asked to mark the sentences that carry the same meaning. It is expected that they will not pick the sentences that have the substitutions from mis-matching senses. The system keeps the dictionary, that is used to find the synonyms, secret. This dictionary is augmented with new synonyms by asking the user about randomly replaced words: if the users frequently mark the sentence that has the random replacement as an equal meaning sentence, then this random word is added to the synonym set of the word that it replaced in the original sentence. An example of a challenge is as follows, the user is asked to mark the sentences that are meaningful replacement of others in the following set:

- The speech has to move through several more drafts.
- The speech has to run through several more drafts.
- The speech has to go through several more drafts.
- The speech has to impress through several more drafts.
- The speech has to strike through several more drafts.

To the best of authors' knowledge the only password scheme that uses a table lookup technique was employed by a bank in 1992. The customers of this bank were suggested to conceal their PIN by writing it down on a special piece of square cardboard that is designed to be kept along with the ATM card. The cardboard presented a table of 4 rows and 10 columns, each column corresponding to a digit (0-9). The customer was asked to pick a four-letter word and write the letters of the word, in consecutive order, to the columns that correspond to digits of their PIN (i.e. first letter goes to the first row and the column of first digit of the PIN). After this step, the empty table cells should be filled with random letters. Anderson described this scheme in [58], where he also mentioned that this cardboard was increasing vulnerability of the system, since the adversary's job is now reduced to finding the four-letter English words (which are not many) in the consecutive rows of the table and trying the corresponding PIN numbers.

3.6 Summary

We believe that natural language processing is a good technology to use in password mnemonics, as it offers the multiple advantages that were mentioned over other approaches. This work is an important first step in the direction of a single-mnemonic for multiple passwords with both good security and good usability properties.

The “best use” we recommend for a deployment of our scheme, would include a policy that sets the passwords’ expiration period (i.e., requiring renewal) to roughly coincide with the time period required by an adversary, who has misappropriated the helper card, to carry out an attack using that card: This will ensure that even users who do not realize that their helper card was stolen, are not unduly exposed to password compromise. We also note that, in addition to the password renewal not requiring a change of mnemonic, the renewal process does not burden the users with the task of choosing a random new password (users are notoriously poor judges of the quality of randomness): A quality random-password generator can be used instead. The users do have a say, but where it really matters to them: At the initial mnemonic-creation time, for the choice of which candidate mnemonic they will have to memorize (this has to involve the user, as what is memorable for a person is highly subjective).

While it is clear that a password that is compromised has to be changed, and that it requires no change of mnemonic, we stress that it is imperative that the mnemonic be changed if *both* helper card and password(s) are compromised (the latter possibly through shoulder-surfing, spyware, phishing, etc).

In order to transfer our system into practice, a user study is needed to adjust the parameters of our system to fit the comfort of users. The current mnemonic generation system involves the user only after a candidate set of mnemonics are found. It is also possible to improve the user experience by incorporating the user’s preferences and interests to the topic of mnemonic sentences that are generated.

4. ACCESSIBLE AUTHENTICATION THROUGH SECURE MNEMONIC-BASED PASSWORDS

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 chapter 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 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.

4.1 Introduction

In 1998, Congress amended the Rehabilitation Act to require Federal agencies to make their electronic and information technology accessible to people with disabilities [59]. 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 [60, 61]. There have not been many studies on electronic accessibility issues for users with motor disabilities [62].

The ability to securely use computer resources and the web provides more freedom for users with disabilities: It enhances their educational and entrepreneurial opportunities [62], 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 rheumatoid 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 that will allow users to own 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. Our system 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 [57] [2] format if the environment allows graphical display. We also achieve duress resistance, which we will discuss more in the following sections.

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

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

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 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 [63–65]. 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. Our system 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 [18] 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. Our system 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 our system 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 the proposed system, 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 [66] 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 chapter presents an authentication method that combines good usability with high security. The crucial 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). 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 4.2 includes an overview of the system and how a user will interact with the system. Section 4.3 covers our adversary model. We will discuss the details of the system and our implementation in Section 4.4. The work on related literature will be summarized in Section 4.5.

4.2 System Overview

We propose a system that is based on mnemonic passwords [66], whose details will be described in the following subsections but whose essential elements are as follows, 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).

4.2.1 Password Initialization Step

1. The system generates a number of random sentences $s_1, \ldots, s_\lambda$ and displays them to the user. See Figure 3.3. 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 4.2.2, using some of the password bits to select one word from each column (in this case, 4 password bits are used up per column). For example, if $\mathcal{P}$ = 0101100101010011111010010001010100011 01 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 4.2.2 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 $m_1, m_2, \ldots, m_\mu$.

3. The column from 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 case we used $r = 16$. In the above-mentioned example, the equivalence class C_2 of the word m_2 is {Dutch, British, ...Romanian}. The user does not need to memorize any such C_j (only m_j needs to be remembered).

4.2.2 Authentication Step

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

1. The system randomly permutes the entries of column C_j. For the ith 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 4.2.2 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 ith word of the permuted C_j is included in Q_k if and only if $b_{i,k} = 1$. For example, for Table 4.2.2, 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}. See Figure 4.1 for the CAPTCHA-displayed version of these questions.

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.) CAPTCHAs can be used to foil an adversary at this step as shown in Figure 4.2. CAPTCHA is essentially a side-channel between the user and the system. The side channel can be used to hide the challenges from the adversary, as in the previous step, or the responses as well. We can use a sequence of binary CAPTCHA challenges to hide the user responses in the bit-wise exclusive-OR of the CAPTCHA answer and the answer to the password challenge.

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,

Table 4.1
The mnemonic generation table for the sentence "Leading U.S. fashion designers are strongly resisting pressure to regulate the thinness of the popular models." Note that the encoding order is random for every set of candidate words.

	leading	U.S.	fashion designers	strongly	resist	pressure	regulate	thinness	popular	models
0000	peaceful	viking	tailor	alarmingly	welcome	attempt	modify	rent	passive	queen
0001	thoughtful	metropolitan	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	accomplished	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	choreographer	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

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 relate the CAPTCHA-displayed words to the user's mnemonics unless it has sophisticated image-processing capabilities (in which case we would have raised the bar considerably compared to current keyboard-entry based techniques which easily fall prey to keystroke-loggers); (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 phact 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).

4.3 Adversary Model

We assume that all of 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 [57]. 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 inputing 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 system 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"). Our system does not weaken the security of the existing authentication system and is based on mnemonic passwords, which lets the users to have random passwords that are secure against dictionary attacks [66].

- **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, our system 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. Our system is designed to have duress codes, that the user can enter to login under duress. See Section 4.4 for details on how duress codes are implemented.

4.4 Implementation Details

We assume that the environment enables the user to read (or hear, in the case of vision impairment) the statements 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 4.2.2, are used in 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. See Table 4.2.2 generated for the example source sentence "Leading U.S. fashion designers are strongly resisting pressure to regulate the thinness of the popular models."

In Table 4.2.2, 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 set for replacing the word in the first row.

4.4.1 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.

The system generates a small set of random salt values, and extract the source table IDs from these salt values (e.g. $\log_2 |S|$ least significant bits). If the size of the table set, S, is 1024, least significant 10 bits of the salt will be used to pick the source sentences that are stored in the first row of the table.

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 4.2, 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 3.3)

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 salt 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 [13] to such a flexible system; hence we advise only a small random portion of these possible mnemonics be given as choice to the users.

4.4.2 Mnemonic Usage

The mnemonic sentence is not stored in the system, instead the source table ID is stored in the salt. 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. Note that, we want each question to narrow the search space for the mnemonic word by 1. Recall the discussion in Section 4.2.

[3]This 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/

The key idea behind generating each challenge is very similar to the idea behind *non-adaptive blood testing* technique [67]. 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 [67]. 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) for the password's bit string associated with m_j.

Does the adversary learn anything from the questions? To prevent that, even though the server asks the questions in succession, 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: The adversary would easily pick out m_j from all the words in its equivalence class.

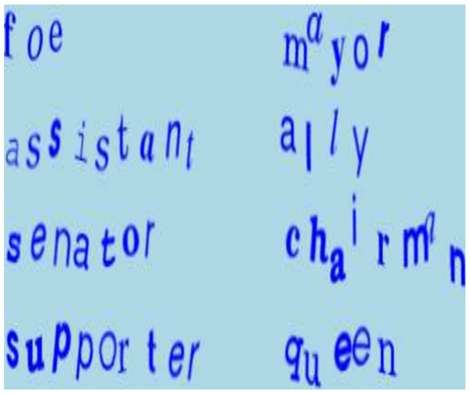
Our scheme will use $d = 1$, for which an $\ell = \lceil \log_2 r \rceil$ test non-adaptive solution is well known and in fact quite straightforward. We briefly sketch it, for the sake of making this chapter self-contained. In what follows C_j is the equivalence class of word m_j, where $|C_j| = r$. (Recall the authentication step described in Section 4.2)

1. Let the words of C_j be listed in an order (which will be randomly changed at every authentication session) as $w_1, \ldots, w_\mu$.
2. For each word w_i, let the ℓ bit binary representation of i be denoted as the bit string $b_{i,\ell-1}, \ldots, b_{i,0}$.
3. For $k = 0, \ldots, \ell - 1$ 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.

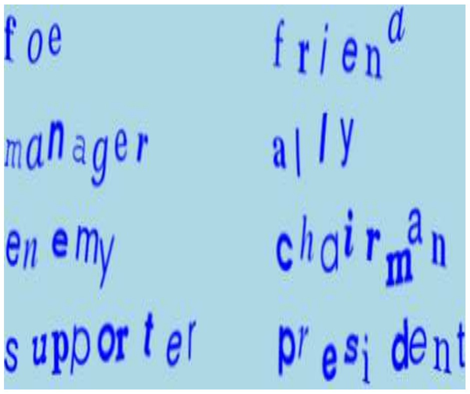
The number of server questions is ℓ, and each question is constructed without any dependency on which element of C_j is the "contaminated" one, m_j. The server can easily determine m_j: It is the only word of C_j such that all of the Q_k's that contained it were answered with a "yes" by the user.

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 set of questions for finding which word is the mnemonic word in the last column of Table 4.2.2 would be as follows:

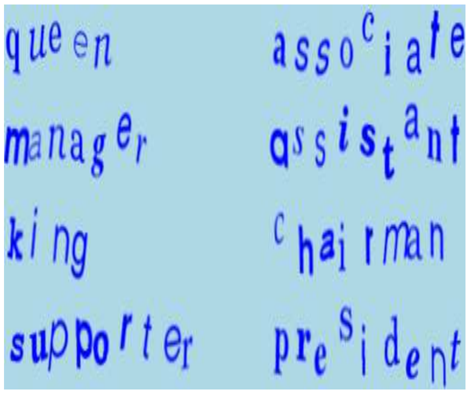
- "Does your mnemonic contain one of the following words?:"
    - "senator, foe, queen, chairman, mayor, ally, assistant, supporter"



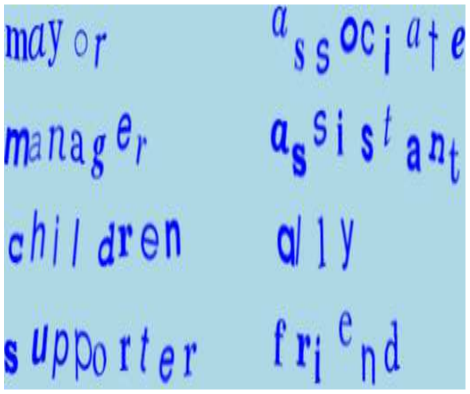
(a)



(b)



(c)



(d)

Fig. 4.1. One of the possible set of challenges Q_0 to Q_3 that could be created for the last column of 4.2.2. Note that, even though we have implemented our system to display the challenges in CAPTCHA form, it is also possible display them in plain text format on text-only consoles or LED displays.

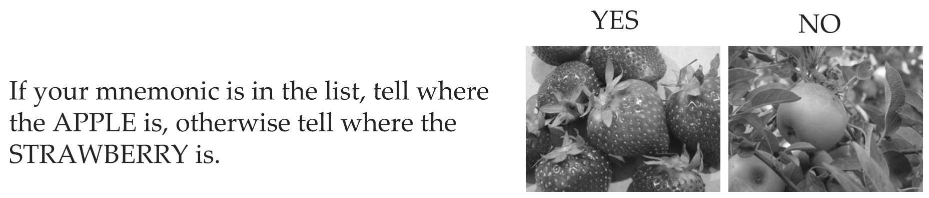

If your mnemonic is in the list, tell where the APPLE is, otherwise tell where the STRAWBERRY is.
YES
NO

Fig. 4.2. CAPTCHAs are a side-channel between the user and the system. It is possible to hide the real user responses from an adversary by using a bit-wise exclusive-OR operation with these responses and a random bit-string that is shared in this channel. The advesary has to solve the CAPTCHA to get the real responses.

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 substring encoded by the mnemonic words. These substrings are concatenated in the order of corresponding words in the source sentence to form the 40 bit length bit string. The hash of the 40 bit string with the salt is compared to the password field in the password file where the hash of the password is stored as in the regular UNIX password file systems. 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 design 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 [68] – 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.

4.4.3 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 salt of the user. Our system provides 2 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

$\mathcal{P}$ = 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 the system displays to the user that "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.

4.5 Related Work

There are many studies in the literature about the requirements for increasing the accessibility of electronic resources for disabled [59–62, 69, 70] and possible techniques (including both hardware and software solutions) to increase the bandwidth of input from these users [63, 64, 71]. Similarly there are many scientific and commercial work on providing access to web via smaller devices where the input capabilities are restricted [72, 73]. There is a big overlap in the design requirements for the two research areas such as assuming low input bandwidth; giving high emphasis on usability requirements (such as providing easy access to all content); or the need for being device and system independence. Trewin discusses the overlap between the user agent accessibility requirements for desktop browsers for Web, and the requirements for a usable Mobile Web in [73].

Mankoff et al.discuss the needs of motor disabled users for accessing the web in [62] . 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 [63], or Brain Computer Interface (BCI) [74]. 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. [65]. Pass-thoughts system is based on recognition of unique brain signals send by the users. This system benefits from the BCI technology [74] 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 [12]. 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. Our system provides the following incentives for the users, in order to encourage them to comply with the instructions for effective use:

- for users with motor disability 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 [50]. 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. [66] allows the users to maintain a multiplicity of truly random passwords, which are independently selected, by just 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.

The study of Reverse Turing Tests in [56] 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 [57]). Our system 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.

4.6 Summary

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 stickies 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.

5. ANOTHER APPLICATION: NL WATERMARKING

This chapter explores another application of the techniques we presented in the previous sections: The resilient watermarking of natural language text through a mechanism that was previously thought to be inherently non-resilient, namely synonym substitution. The reason it was thought to be non-resilient is the belief that whatever synonym substitutions are done for marking can be reversed or masked by the massive use of such substitutions by the adversary (after all, the list of synonyms is widely available online). Before presenting our scheme, we provide a brief review of information hiding in natural language text.

Information-hiding in natural language text has, in the past, consisted mainly of carrying out approximately meaning-preserving modifications on the given cover text until it encodes the intended mark. A major technique for doing so has been synonym-substitution. In these previous schemes, synonym substitutions were done until the text "confessed", i.e., carried the intended mark message. We propose here a better way to use synonym substitution, one that is no longer entirely guided by the mark-insertion process: It is also guided by a resilience requirement, subject to a maximum allowed distortion constraint. Previous schemes for information hiding in natural language text did not use numeric quantification of the distortions introduced by transformations, they mainly used heuristic measures of quality based on conformity to a language model (and not in reference to the original cover text). When there are many alternatives to carry out a substitution on a word, we prioritize these alternatives according to a quantitative resilience criterion and use them in that order. In a nutshell, we favor the more ambiguous alternatives. In fact not only do we attempt to achieve the maximum ambiguity, but we want to simultaneously be as close as possible to the above-mentioned distortion limit, as that prevents the adversary from doing further transformations without exceeding the damage threshold; that is, we continue to modify the document even after the text has “confessed” to the mark, for the dual purpose of maximizing ambiguity while deliberately getting as close as possible to the distortion limit. The quantification we use makes possible an application of the existing information-theoretic framework, to the natural language domain, which has unique challenges not present in the image or audio domains. The resilience stems from both (i) the fact that the adversary does not know *where* the changes were made, and (ii) the fact that automated disambiguation is a major difficulty faced by any natural language processing system (what is bad news for the natural language processing area, is good news for our scheme’s resilience). In addition to the above-mentioned design and analysis, another contribution of this thesis is the description of the implementation of the scheme and of the experimental data obtained.

5.1 Introduction

In recent years, there has been an increased interest in using linguistic techniques for designing information hiding systems for natural language text. These techniques are based on using the knowledge of language to generate or re-write a document in order to encode hidden information [75].

Even though there is a growing interest in information hiding into natural language, there has not been much movement in the direction of quantification that makes possible using the considerable theoretical work on the analysis of the communication channel established by information hiding. To avail oneself of the information hiding model proposed by Moulin et al in [76] requires quantification of the distortion effect of each linguistic transformation. In this thesis we carry out such an analysis, using a natural language watermarking system based on a novel twist on the old idea of synonym substitution. Section 5.2.1 will discuss how we use the existing information hiding model for the natural language domain.

Publicly available methods for information hiding into natural language text can be grouped under two branches. The first group of methods are based on generating a new text document for a given message. Spammimic [77] is an example of this first group. The second group of methods are based on linguistically modifying a given cover document in order to encode the message in it. Natural language watermarking systems (and our system's framework) fall under the second type of systems, where there is also a need for robustness against an adversary who is attempting to destroy the mark without destroying the *value* of the watermarked document. For a review of closely related work in information hiding into natural language, refer to Section 5.5.

The watermarking system proposed in this chapter is based on improving resilience of synonym substitution based embedding by ranking the alternatives for substitution according to their ambiguity and picking the one that has maximum ambiguity within the synonyms (subject to not exceeding the maximum cumulative distortion limit). The encoding is designed in a way that the decoding process does not require the original text or any word sense disambiguation in order to recover the hidden message. This system follows the Kerckhoff's rule, namely, that the decoding process depends only on the knowledge of the secret key and public domain information (no "security through obscurity").

It is possible to determine infringement of copyright using simple string matching if the infringement is in the form of verbatim copying of the text. However the adversary can foil the string matching based infringement detection, through automated meaning preserving changes to the text [75]. A desired natural language watermark should be resilient against these modifications. Refer to Section 5.2.2 for more discussion on the model of adversary.

The detection of copyright infringements on web publishing could be one of the major applications of natural language watermarking. In this application the copyright holder will be able to find out infringements by running a web crawler that detects the copyright holder's watermark; or by subscribing to a web crawler service that searches for watermarked text on the web. In order to realize this, it is crucial that the watermark detection can be performed automatically without human intervention. This requirement is satisfied by the system introduced in this thesis.

In case a news agency *does not watermark* its news articles, but uses a web crawler to search for the illegal copies of the articles on the internet. An adversary, who wants to re-publish an article from this agency, can perform synonym substitution to deceive a string matching based web crawler. The infringement detection can be performed by checking whether the words in the suspicious copy and the original document are synonyms.

The details of the proposed watermarking system are explained in Section 5.3, followed by experimental results in Section 5.4.

Even though we have focused our attention directly on synonym substitution based watermarking, the analysis and discussions made in this thesis shed light on the information theoretic analysis of other systems that achieve information hiding through approximately meaning-preserving modifications on a given cover text.

5.2 Framework

This section discusses the general framework we use, including our model of the adversary. Where appropriate, we explain how the peculiarities of the natural language application domain pertain to the framework.

5.2.1 Review of Distortion Quantification

Here we briefly review the general model proposed by Moulin et al in [76] and use the same notation, as applicable. The later section on experimental results (Section 5.4) will explain how we computed the values for the below equations. In this notation, random variables are denoted by capital letters (e.g. S), and their individual values are donated by lower case letters (e.g. s). The domains over which random variables are defined are denoted by script letter (e.g. $\mathcal{S}$). Sequences of N random variables are denoted with a superscript N (e.g. $S^N = (S_1, S_2, ..., S_N)$).

Natural language watermarking systems aim to encode a watermark message, M, into a given source document, S^N, using a shared secret, K^N, where K^N is the only side information shared between the encoding and decoding processes. The goal of the encoding process is to maximize the robustness of watermark against possible attacks while keeping the distortion inflicted on the S^N during watermarking within allowable limits. There are two distortion constraints on a given natural language watermarking system.

The first distortion constraint is introduced to capture the fact that the watermark encoding process, $f_N : \mathcal{S}^N \times \mathcal{M} \times \mathcal{K}^N \rightarrow \mathcal{X}^N$, has to preserve the "value" of the source document, while creating the watermarked document X^N. Moulin et al formalizes this constraint as below:

$$\sum_{s^N \in \mathcal{S}^N} \sum_{k^N \in \mathcal{K}^N} \sum_{m \in \mathcal{M}} \frac{1}{|\mathcal{M}|} p(s^N, k^N) d_1^N(s^N, f_N(s^N, m, k^N)) \leq D_1 \tag{5.1}$$

where p is the joint probability mass function and d_1 is a nonnegative distortion function defined as $d_1 : \mathcal{S} \times \mathcal{X} \rightarrow \mathbb{R}_+$. The distortion functions d_i [1] are extended to per-symbol distortion on N-tuples by $d_i^N(s^N, x^N) = \frac{1}{N} \sum_{k=1}^{N} d_i(s_k, x_k)$.

The second constraint denotes the maximum distortion an adversary can introduce on the modified document, Y^N, without damaging the document's "value" for the adversary. The constraint on the attack channel for all $N \geq 1$ is formalized as below:

$$\sum_{x^N \in \mathcal{X}^N} \sum_{y^N \in \mathcal{Y}} d_2^N(x^N, y^N) A^N(y^N | x^N) p(x^N) \leq D_2 \tag{5.2}$$

where $A^N(y^N|x^N)$ is a conditional probability mass function that models an adversary who maps $\mathcal{X}^\mathcal{N}$ to $\mathcal{Y}^\mathcal{N}$, and d_2 is the adversary's distortion function (similar to d_1). The decoder process receives Y^N.

For image, video or numeric databases, the space can be modeled as a Euclidean space and the effect of changes on the objects can be quantified as a continuous function [76,78]. However, it is rather hard to model the natural language text input.

[1] $i \in 1, 2$

The value of a natural language document is based on several properties such as meaning, grammaticality and style. Thus, the distortion function should be designed to measure the distortion in these properties.

In fact we cannot even talk of a distance in natural language processing, as the triangle inequality need not be satisfied. For example, both "lead" and "blend" are synonyms of different senses of the word "go", as the following entries (obtained from WordNet) indicate:

- blend, go, blend in – (blend or harmonize; "This flavor will blend with those in your dish"; "This sofa won' t go with the chairs")
- go, lead – (lead, extend, or afford access; "This door goes to the basement"; "The road runs South")

The difference between the word "lead" and the word "go", and the difference between the word "blend" and the word "go", are rather low, whereas the difference between "blend" and "lead" is high. Figure 5.1 uses *pathlen* measure to illustrate the difference between word senses.

We cannot use that part of [76] that assumes a Euclidean distance, since the triangle inequality does not hold in the natural language framework of our application. However, the other requirements that the difference function must obey, are satisfied, namely

Boundedness This is the requirement that the distortion is finite. This holds in our case, because no matter how different two sentences are, our difference function between them will produce a finite outcome.

Symmetry This is the requirement that $d(a, b) = d(b, a)$. That we satisfy this follows from the fact that the numbers we use for differences are weights of edges in an *undirected* graph (as will become apparent in section 5.3).

Equality This is the requirement that $d(a, b) = 0$ if and only if $a = b$. This holds in our case.

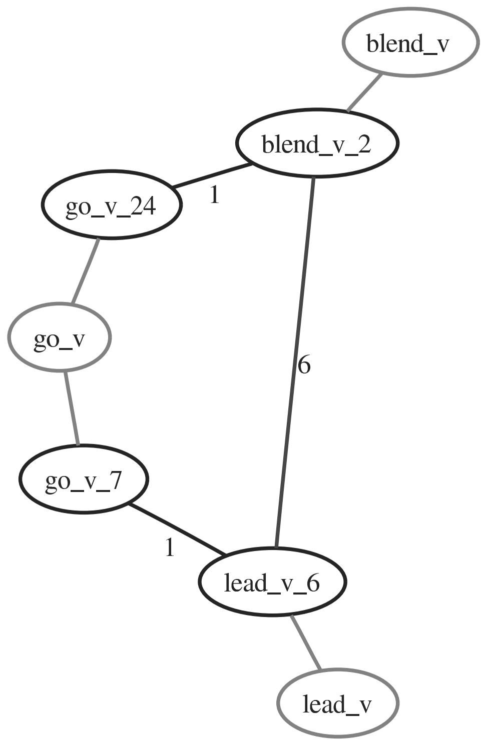


Fig. 5.1. An example illustrating how the differences in natural language need not satisfy the triangle inequality. The presented differences are calculated by *pathlen* measure of WordNet::Similarity library

5.2.2 Model of the Adversary

The Achille's heel of traditional synonym-substitution based watermarking is an adversary who, randomly and in a wholesale fashion, carries out synonym substitutions. While such an adversary may be effective against these previous uses of the synonym substitution technique, our scheme thwarts such an adversary (as will be discussed later in the chapter). However, thwarting such a random adversary is not a fair criterion, as it is a rather naive form of attack. Therefore our model of the adversary is one who fully knows our scheme (except the key) and has the same knowledge and computational capabilities (including automated natural language processing tools, and access to all the databases used by the encoding and decoding processes). The security of our scheme therefore does not depend on an assumption of naive ignorance on the part of the adversary, rather, it depends on the following facts.

First, the approximately meaning-preserving changes that we make are in the direction of more ambiguity, and automated disambiguation is harder for the adversary than it is for us because we start with a less ambiguous (original document) document than the one in the hands of the adversary (watermarked document). A human, however, is able to quickly disambiguate when reading the marked text: We are exploiting the well-established fact in the natural language processing community, that humans are much better than computers at disambiguation [55].

Second, we carry out substitutions not only for the purpose of encoding the mark in the text, but also for the purpose of getting as close as possible to the allowable cumulative distortion limit, an idea that was previously suggested in a broader framework (see [79]). That is, we keep doing transformations even after the mark is embedded, for the sole purpose of accumulating enough distortion to get close to the allowable limit. This is crucial: The adversary, not knowing the key, does not know *where* we carried out the modifications (as that choice is key-based), and trying to "un-do" them by wholesale application of transformations will cause the adversary to exceed the allowable distortion limit (because s/he started out close to it in the first place).

In practice the adversary is not limited to synonym substitutions; s/he can also make meaning-preserving syntactic changes which effect the ordering of words without altering them [75]. This sort of attacks are more problematic in languages with free word order (e.g. Finnish, Hindi, Turkish); since the adversary can put the words in any permutation without damaging the text too much. If the adversary performs a more sophisticated attack using syntactic modifications, even though the root words will be preserved, their order may change in the copied text. In this case, the watermarking mechanism should take into account the possible syntactic modifications. The watermarking mechanism can use an auxiliary fixed syntax with a fixed word order for watermark embedding and detection purposes (e.g. subject, object, verb). In addition to this, ambiguity may be used to prevent from syntactic modifications as a pre-emptive defense at the watermark embedding time (e.g. using pronouns as ambiguous references).

Note that the adversary in our scheme uses an automated process to attack the watermark. Our aim is to raise the bar for the cost of removing the watermark message. In this sense, our scheme can be considered successful if it forces the adversary to manually process the document for removing the watermark.

5.3 Synonym Substitution Based Watermarking System

Most of the previous work on information hiding in natural language text was designed to get better as the accuracy of natural language processing tools improves. In [80], Bergmair discusses the need for an accurate word sense disambiguator for fully automating a synonym substitution based steganography system that requires sense disambiguation both at encoding and decoding time. Topkara et al [81] give examples of how accuracy of the text processing tools affects the quality of the watermarked sentences.

Whereas previous work in this area typically benefits from progress in natural language processing, we propose a watermarking system that benefits from the difficulty of automated word sense disambiguation, as it increases the adversary's complexity of removing the hidden message.

We propose a lexical watermarking system that is based on substituting certain words with more ambiguous words from their synonym set. Here by ambiguous word, we mean a word that is a member of several synonym sets and/or has many senses. For example, if the geographic context is North Carolina, then "the Raleigh-Durham area" can equivalently be re-stated as "the triangle" where "triangle" refers to the "research triangle area". The adversary now has to figure out that this particular "triangle" is neither a three-sided polygon, nor a musical instrument, nor a situation in which two people are competing for the love of the same third person. The difficulty of the adversary's task of automated disambiguation is widely accepted in the natural language processing community. Although our implemented system cannot yet carry out the above specific transformation (because its public knowledge base does not yet contain the specific equivalence it uses), we mentioned it because it perfectly exemplifies the kinds of substitutions we seek (and that will undoubtedly be present in a more refined version of our prototype).

Homograph is a more specific linguistic term used for the "ambiguous" words. Two or more words are homographs if they are spelled the same way but differ in meaning and origin, and sometimes in pronunciation. For example the word "bank" is a homograph, and means either a financial institution, the edge of a stream, or a slope in the turn of a road. We have implemented our system to consider the words with more than one sense as homographs, and only homographs within a synonym set are considered as the target words for synonym substitution.

An example of what our system does carry out today is when we encounter the word "impact" as a verb in our cover text: We will find that it is a member of {affect, impact, bear upon, bear on, touch on, touch} synonym set. The verbs "affect" and "touch" are possible alternatives for replacing the verb "impact". Our system favors replacing the word “impact” with the word “touch” over the word “affect”, because the expected distortion that will be imposed by the verb “touch” on the adversary, $E(d_2(touch; impact, s2))$, is higher than the expected distortion, $E(d_2(affect; impact, s2))$, that will be imposed by the verb “affect”. $E(d_2(w_c; w_o, s_o))$ is the average difference of every sense of watermark carrying word, w_c, to the original (word,sense) pair, (w_o, s_o). Refer to Section 5.3.1 for the details of how this expected distortion is calculated in our system. See Figure 5.2, for a simplified illustration of this embedding. For simplicity of the graph, word sense nodes are collapsed into one node for each word except the verb “impact”, whose sense is learned from the cover text. Only relevant nodes are colored and labeled. Edge weights are again omitted for simplicity. “affect” has five senses, “touch” has fifteen senses, “impact” has two senses, “bear on” has four senses, “touch on” has four senses, and “bear upon” has only one sense.

In our scheme, more information is available about the sense (meaning) of the words at the watermark embedding time, since the original document is available. The watermarking process we describe, replaces as many as possible words with one of the homographs in their synonym set. Hence the watermarked text has “blurred” meaning and it becomes harder for an adversary to perform word sense disambiguation on it (i.e., the ambiguity has increased in such a way that it is harder to find the correct synonym of the words without human intervention). In such a setting, the adversary will not be willing to replace every homograph word with a non-homograph automatically and the watermark will be successfully retained. Note that, it may also be possible to magnify this asymmetry of information further by consulting to the actual author of the text during watermark embedding, for the correct sense of a word at watermarking time.

As an example, consider the sentence “he went without water and food for 3 days” coming from a watermarked text. If the adversary had replaced the word “went” with the word “survived” then the change in the meaning is minimal. However, if he had replaced “went” with “died”, the meaning of the sentence would be taken very far

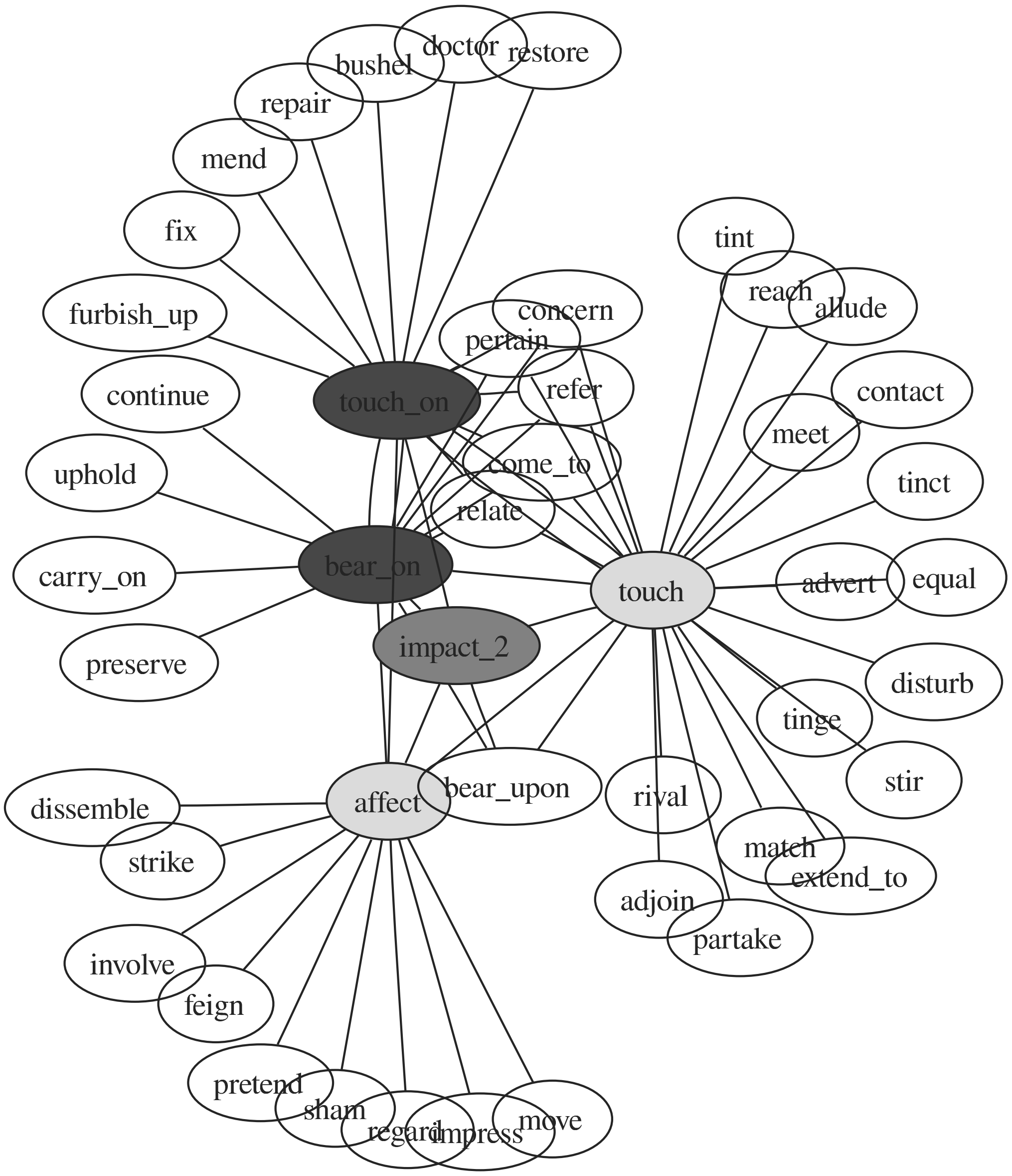


Fig. 5.2. A sample colored graph that shows the connections of the verb "impact". For simplicity of the graph, word sense nodes are collapsed into one node for each word and edge weights are omitted. Only relevant nodes are colored and labeled.

from its original meaning. Yet both "survive" and "die" are synonyms of different senses of the word "go".

Loosely speaking, we are using ambiguity in natural language to imitate one-way hash functions. For example, when given as original sentence, "the gas station is after the cant on highway 52." we can replace the word "cant" with its synonym, "bank". The transformed sentence will now say "the gas station is after the bank on highway 52", where it is not obvious whether the gas station is after the financial institution, after the inclined slope on the turn of the road, or after the stretch of road that briefly adjoins the river's bank. Our system uses this deliberate injection of ambiguity whenever possible, and replaces words with more ambiguous words from their respective synonym set.

The decoding process is not dependent on the original text and there is no need to know the sense of a word in order to decode the message. This simplicity of decoding process makes it computationally light, and it enables the copyright infringement detection to be performed by a web crawler on a large number of online documents.

The details of the encoding and decoding processes are explained in the next subsection.

5.3.1 The Encoding and Decoding Algorithms

Our system is based on building a weighted undirected graph, G, of (word,sense) pairs, where an edge between two nodes represents that they are synonyms. In our experimental implementation, the synonym sets of words are taken from WordNet [45]. Each weight on a graph's edge is a measure of the similarity between its two endpoints.

Several different techniques and similarity functions have been proposed in the natural language processing literature to quantify the similarity of two words. A large number of these techniques are based on WordNet, which is an electronic dictionary that organizes English nouns, verbs, adjectives and adverbs into synonym sets, each representing one underlying lexical concept [45]. See Table 5.1 for statistics about the

Table 5.1
Wordnet2.1 Database Statistics

Category	Unique Strings	Synsets	Word-Sense Pairs	Monosemous Words	Polysemous Words	Polysemous Senses
Noun	117097	81426	145104	101321	15776	43783
Verb	11488	13650	24890	6261	5227	18629
Adjective	22141	18877	31302	16889	5252	14413
Adverb	4601	3644	5720	3850	751	1870
Total	155327	117597	207016	128321	27006	78695

content of WordNet [2]. There are several semantic relations that link the synonym sets in WordNet such as "is-a-kind-of", or "is-a-part-of" relations. Some of the word similarity functions are available as a Perl Library called WordNet::Similarity [47,82]. WordNet::Similarity package implements six different similarity measures that are in some way based on the structure or the content of WordNet.

Three of the WordNet::Similarity measures are based on the information content of common subsumers of two concepts. The Resnik measure is based on using the information content of most specific common subsumer, Least Common Subsumer (LCS), of two concepts as a similarity value. The Lin measure scales the information content of the LCS by the sum of the information contents of the two concepts, while the Jiang and Conrath measure takes the difference of this sum and the information content of the LCS. The information content of a concept is learned from a sense-tagged corpus, Semcor [83].

The other three similarity measures are based on path lengths between pairs of concepts. The Leacock and Chodorow measure is based on scaling the shortest path length between two concepts by the maximum path length found in the subtree of

[2]`http://wordnet.princeton.edu/man/wnstats.7WN`, last visited on May 9th 2006

the “is-a” hierarchy that includes these two concepts. The Wu and Palmer measure is calculated by finding the depth of the LCS of the two concepts and scaling it by the sum of the depths of two concepts. The *path* measure is a baseline metric and is equal to the inverse of the length of the shortest path between the two concepts.

Another method for measuring the similarity between the words is statistically analyzing a large and balanced text corpus in order to learn the mutual information of the two words, and resemblance of their context. Several mutual information measures have been proposed for calculating word similarity, see [84] for a good survey of these measures.

Language models can be used to learn more information about the similarity of the context of two concepts. A Language Model (LM) is a statistical model that estimates the prior probabilities of n-gram word strings [85]. An n-gram LM models the probability of the current word in a text based on the $n-1$ words preceding it; hence, an n-gram model is a $n-1^{th}$ order Markov model, where, given the probability of a set of n consecutive words, $W = \{w_1, \ldots, w_n\}$, the LM probability is calculated using

$$P(w_1, \ldots, w_n) = \prod_{i=1}^{n} P(w_i | w_0, \ldots, w_{i-1}), \tag{5.3}$$

where the initial condition $P(w_1|w_0)$ is chosen suitably. A set of words $C = \{c_1, \ldots, c_k\}$ may be considered “similar” if $P(c_j|w_1, \ldots, w_n) \simeq P(C|w_1, \ldots, w_n)P(c_i|C)$ holds for all $c_j \in C$. Readers are referred to [86] for a heuristic algorithm that can be used to compute C using language models.

In our experiments we have used the Wordnet::Similarity measures for simplicity. Alternatively, a corpus-based measure can be used for the same purpose. More details about our experiments can be found in Section 5.4.

After graph G is formed, we select a subgraph, G^W of G using the secret key k. This subgraph selection is performed over the words that have homographs in their synonym sets. After this, we use k once more to color the graph in such a way that approximately half of the homograph neighbors of a non-homograph word are colored with blue to represent the encoding of “1”, and the other half are colored with green to represent the encoding of "0", while non-homographs are colored with black to represent "no-encoding". See Figure 5.2 for a simplified example where the word "impact" is colored "red" only to show that it is the word that will be replaced during embedding.

At encoding time, we calculate the expected distortion value for the adversary, which in some sense measures how hard it would be for the adversary to find the original word, w_o, given the mark carrying word, w_c. Note that, if the adversary can replace w_c with w_o, then not only the mark bit encoded by that word will be removed, the distortion introduced by the watermarking process will also be undone. In our implementation, $E(d_2(w_c; w_o, s_o))$ is calculated by summing up the differences of every sense of w_c to the original (word,sense) pair, (w_o, s_o) normalized over the number of senses of w_c, which is denoted with $|S(w_c)|$. This is formalized as below:

$$E(d_2(w_c; w_o, s_o)) = \frac{\sum_{s_i \in S(w_c)} sim(w_c, s_i; w_o, s_o)}{|S(w_c)|} \tag{5.4}$$

where s_o is the sense of the original word, w_o, in the original document, and $sim(w_c, s_i; w_o, s_o)$ is the similarity based difference between (word,sense) pairs, it increases as the words get more dissimilar.

If there are more than one candidate homograph with the same color (the color that is required to encode the current bit of the message, m) then the one with the maximum $E(d_2())$ value is picked. Since we are using the WordNet-based similarity measures, the difference between pairs from the same synonym set is the same for all the words in that set, which makes the encoding distortion identical for all alternative pairs from the same synonym set. However, this need not be the case if alternative similarity measures are used. The following are summaries of the encoding and decoding algorithms, based on the above discussion.

Steps of the encoding algorithm:

- Build graph G of (word,sense) pairs. Use WordNet to find synonym sets of (word, sense) pairs. In addition, connect different senses of the same word with a special edge in order to follow the links to every neighbor of a word independent from its senses.

- Calculate differences between the (word,sense) pairs, $d(wi_{sense_k}, wj_{sense_l})$, using a similarity measure. Assign these values as edge weights in G.
- Select a subgraph G^W of G using the secret key k.
- Color the graph G^W. Detect the pairs of words (w_i, w_j), where w_i and w_j are in the same synonym set with one of their senses, and have more than one sense. In other words, these words act as homographs. Color w_i and w_j with opposite colors in graph G^W, using k to decide which one gets to be colored in blue (i.e, encodes a "1") and which one gets to be colored in green (i.e., encodes a "0"). Color non-homographs as black.
- $c = 1$
- For each word w_i in the cover document S
    - $bit_c = M[c]$
    - if $w_i \in G^W$ then replace w_i with the neighbor that carries the color that encodes bit_c
      if there are more than one neighbor that encodes bit_c
      for each, w_j, of these neighbors calculate
      $E(d_2(w_j; w_i, s_k)) = \frac{\sum_{s_l \in S(w_j)} sim(w_j, s_l; w_i, s_k)}{|S(w_j)|}$
      pick the neighbor with the maximum $E(d_2(w_j; w_i, s_k))$ value
      Increment c (if $c = |M| + 1$ then set $c = 1$)

If the cover document's size is long enough the message, M is embedded multiple times. We assume that the message M, that is input to the watermarking system, has already been encrypted and encoded in a way that it is possible to find the message termination point when reading it sequentially from an infinite tape. The encrypted M could have an agreed-upon fixed length (symmetric encryption preserves length, so we would know how to chop the decoded message for decryption). Or, alternatively, if the length of M is unpredictable and cannot be agreed upon ahead of time, the encrypted M could be padded at its end with a special symbol, i.e. #, that would act as a separator between two consecutive copies of the encryption.

Steps of the decoding algorithm:

- Build the same graph G of (word,sense) pairs using the same difference function as the one used for the encoding process
- Select a subgraph G^W of G using the secret key k
- Color the graph G^W using k (as for the encoding process)
- $c = 1$
- For each word w_i in the cover document S
  - if $w_i \in G^W$ then check the color of the node that represents w_i.
    if it is black, move to the next word
    if it is blue, assign 1 to $M[c]$ and increment c
    if it is green, assign 0 to $M[c]$ and increment c

The decoding algorithm is simply a series of dictionary lookups. We envision that this simplicity will enable our system to be used for watermaking online text documents. Then, web crawlers that are indexing web pages can also check for watermarks or metadata embedded using our system in the pages they visit.

5.3.2 Context-Dependent Synonyms

The notion of a synonym that WordNet subtends is, as in any fixed dictionary, rigid in the sense that it fails to capture the notion of a *context-dependent* synonym: A synonym relationship that holds only within a particular text document. Such a relationship holds between words that in general are not synonyms, but that in a particular text's context are de facto synonyms, pretty much the way "the sleuth" is synonym of "Sherlock Holmes" in most of Arthur Conan Doyle's books, but is a synonym of "Hercule Poirot" or "Miss Marple" in some of Agatha Christie's books.

Fully automating the encoding of such a scheme is highly nontrivial and error-prone, and we therefore envision a semi-automatic interactive encoding mechanism where the text's author decides on the acceptability (or lack thereof) of substitutions proposed by the system, or even suggests substitutions from scratch. This is especially appropriate in texts where the quality of the prose is as important as the meaning it carries, or when a fully automatic process is likely to introduce unacceptable errors. For example, Daniel Defoe would not approve a proposed substitution of "Friday" by "sixth day of the week" in the context of his Robinson Crusoe book.

There are two benefits to this more general notion of a synonym: (i) it increases the repertoire of effective synonyms available, thereby increasing encoding capacity; and (ii) it is harder for the adversary to un-do, both because of the context-dependency and because an outside (possibly human) adversary incurs a larger time penalty to analyze and comprehend the text than the author.

5.3.3 Generalization Substitutions

Further resilience can be provided by the use of meaning-preserving generalizing substitutions (replacing the specific by the general, e.g., "lion" by the less specific "big cat" or the even more general "carnivore"). As stated earlier, WordNet includes a "`is_a`" types of hierarchies that we could use to achieve this – we could advantageously "move up" one of these `is_a` hierarchies of WordNet in a manner that does not destroy meaning: For example, it may be perfectly acceptable to replace "lion" with "big cat" or even with "carnivore" even though the latter two are not synonyms of the former (or of each other), as lion-big_cat-carnivore form a chain of ancestors in the `is_a` hierarchy. But the substitution of "lion" with "carnivore" would not be acceptable if the text also contains much about a wolf (which is also a carnivore), although the substitution of "lion" with "big cat" is still acceptable (as would be the substitution of "wolf" with "canine"). More formally, when making a substitution we are allowed to move up a particular link in the `is_a` hierarchy as long as doing so does not gobble up an extra descendent node that also appears in the text (as that would be meaning-damaging). Because in a WordNet hierarchy there can be more than one parent, it is possible that we are unable to move up one link, but able to move up another link: For example, in a text with a camel and a kangaroo, we cannot generalize "kangaroo" to "herbivore" but we can generalize it to "marsupial".

The above process makes it harder for the adversary to using such transformations to attack the watermark, for two reasons: (i) Many of these substitutions have already been applied (by the encoding process) to the maximum extent possible ("as far up the hierarchy as possible"); and (ii) most hierarchies have a higher branching factor going down than going up (many children, fewer parents), and therefore replacing the general by the specific in an attack is problematic (the adversary has more choices).

5.4 Experimental Results :EQUMARK

We have implemented a natural language watermarking tool, EQUMARK, according to the system design proposed in Section 5.3. Equmark is implemented in Perl and uses WordNet::Similarity and WordNet::QueryData libraries [47,87] [3]. WordNet 2.1 is used during the experiments presented in this section.

We used *pathlen*() function of WordNet::Similarity library in order to learn the difference between (word,sense) pairs, in other words:

$$sim(w_j, s_l; w_i, s_k) = pathlen(w_j, s_l; w_i, s_k)$$

pathlen() outputs the length of shortest path between the (word,sense) pairs in the "is-a" hierarchy of WordNet. As we have mentioned in Section 5.3, *pathlen* measure is

[3]These libraries are implemented in Perl language and they are freely available from CPAN website at `http://cpan.org/`. Last visited on May 9th 2006.

a baseline metric and is equal to the inverse of the length of shortest path between the two concepts. Alternatively $pathlen()$ can be replaced with other similarity measures.

Equmark builds the graph, G_W, as follows. $listAllWords()$, a function from the WordNet::QueryData library, is used to generate a list of words, L. Later the content of L is increased by exploring the synonyms of the words in the initially generated list. After this step, Equmark assigns random colors to all words in L, using the secret key k. Later, words from L are processed by taking one word, w_i, at a time from the beginning of L and starting a breadth first exploration of WordNet, where w_i is the root. If a node is assigned blue or green initially, during the breadth first traversal, we try to color the neighbors (synonyms) of each word with the opposite of the word's color. Whenever there is a conflict, Equmark colors the newly explored node with black, marking it as a "no-encoding" word.

A node gets an encoding color, i.e. blue or green, if it has more than one sense and there is at least one synonym for each one of its senses. All single sense words (monosemous words) are colored with black, since they are not "ambiguous", they do not increase the resilience if they are used for marking.

We color a word with black if it does not have any synonyms for one of its senses, even though it has more than one sense. For example, consider the word "jury":

- jury – (a body of citizens sworn to give a true verdict according to the evidence presented in a court of law)
- jury, panel – (a committee appointed to judge a competition)

We can not include the word "jury" in our set of encoding words in the current system, because if we color it with blue or green, and if it appears with its first sense in the cover document we will not be able to undo the effect of its color to the encoding. Wet Paper Codes [88] can be used in order to be able to increase the capacity of watermarking with Equmark. When the wet paper codes are used, we can mark the word "jury" as a stuck cell if it is used in its first sense in the cover document. But, when it is used in its second sense, we can mark it as a changeable cell and use it for encoding. Note that, if word sense disambiguation was possible at watermark decoding time, this limitation would not be an issue, since we could discard those word senses without synonyms (our decoding algorithm is a series of dictionary lookups).

An adversary who is aware of the fact that some of the words in G_W have to be colored "black" due to the phenomenon explained in the previous paragraph, can find instances of those words in the watermarked text (by checking if a word does not have synonyms for at least one of its senses). Later, the adversary can randomly alter those words, since there is a non-zero probability of substituting them with an encoding word. This attack will add noise into the decoded message. In the scope of this work, the effects of such attacks are not quantified. But as mentioned above, Equmark can be enhanced to be able to prevent the effect of noise addition to the watermark message either by the use of wet paper codes or error correction codes; besides increasing the bandwidth, this enhancement will also increase the resiliency of the system.

We still included a "relaxation" on the above coloring restriction in our implementation by checking the WordNet frequency value for the words that have more than one sense and some of them have at least one synonym, we check the senses that do not have any synonyms: if the frequency of this (word,sense) pair is below a threshold, we interpret it as "this pair is very rarely used in the language", and it is unlikely that we will encounter this particular sense of w_i in the cover document. Thus, we color w_i with an encoding color. See Table 5.2 for a sample distribution of colors for different word categories.

Equmark takes in four inputs: a cover document, a message M, a colored graph G_W and an embedding distortion threshold D_1.

In our experiments, we took our cover documents from a sense-tagged corpus, Semantic Concordance (SemCor) [83]. We used SemCor2.1 [4] since its sense-tags are

[4]Downloadable from Rada Mihalcea's homepage at `http://www.cs.unt.edu/~rada/downloads.html`. Last visited on May 9th 2006.

Table 5.2
A sample coloring performed by Equmark

Category	Black	Green	Blue
Noun	141408	7873	7998
Verb	7716	1919	1885
Adverb	4107	261	257
Adjective	19058	1940	1976
Total	172289	11993	12116

mapped to WordNet 2.1 instead of the original SemCor that is tagged with WordNet 1.6 senses. SemCor has three parts: first part, *brown1*, consists of 103 semantically tagged Brown Corpus [89] files, in which all content words are tagged; second part, *brown2*, consists of 83 semantically tagged Brown Corpus files, in which all content words are tagged; third part, *brownv*, consists of 166 semantically tagged Brown Corpus files, in which only verbs are tagged. We have used nouns, and verbs as watermark carrying words.

Using already sense-tagged input text was a natural choice for our experiments since it let us focus on analyzing the amount of distortion the embedding incurs on the cover document and the resilience we can achieve. In another setting, at the encoding time, the author of the document might be prompted for help on disambiguating the sense of ambiguous words. As we have mentioned before, there is no need for sense disambiguation at decoding time.

Equmark restricts the cumulative embedding threshold to be below D_1. We used *pathlen*() as the difference function. Refer to Section 5.2.1 for a discussion of the model proposed by Moulin et al.

In our experiments, $d_1(w_c, s_c; w_o, s_o)$ was always equal to 1 as we consider substitution only between the words from the same synonym set. $d_1(w_i, s_l; w_j, s_k) = 0$ only when $w_j = w_i$, since $pathlen(w_i, s_l; w_j, s_k)$ is length of the shortest path between the two concepts. However, usage of other similarity measures as the difference function might change this.

Our embedding algorithm picks synonyms that have maximum expected d_2 as described in Section 5.3 and Equation 5.4.

Refer to Figure 5.3 for an analysis of the relationship between the embedding distortion (x-axis), D_1, and the resilience (y-axis), D_2.

The graph in the Figure 5.3 corresponds to an experiment run over one of the *brown1* files, which has 1815 tagged tokens. Here, the watermark was a random string of 10 bits, for the sake of readability of the graph. Larger number of watermark

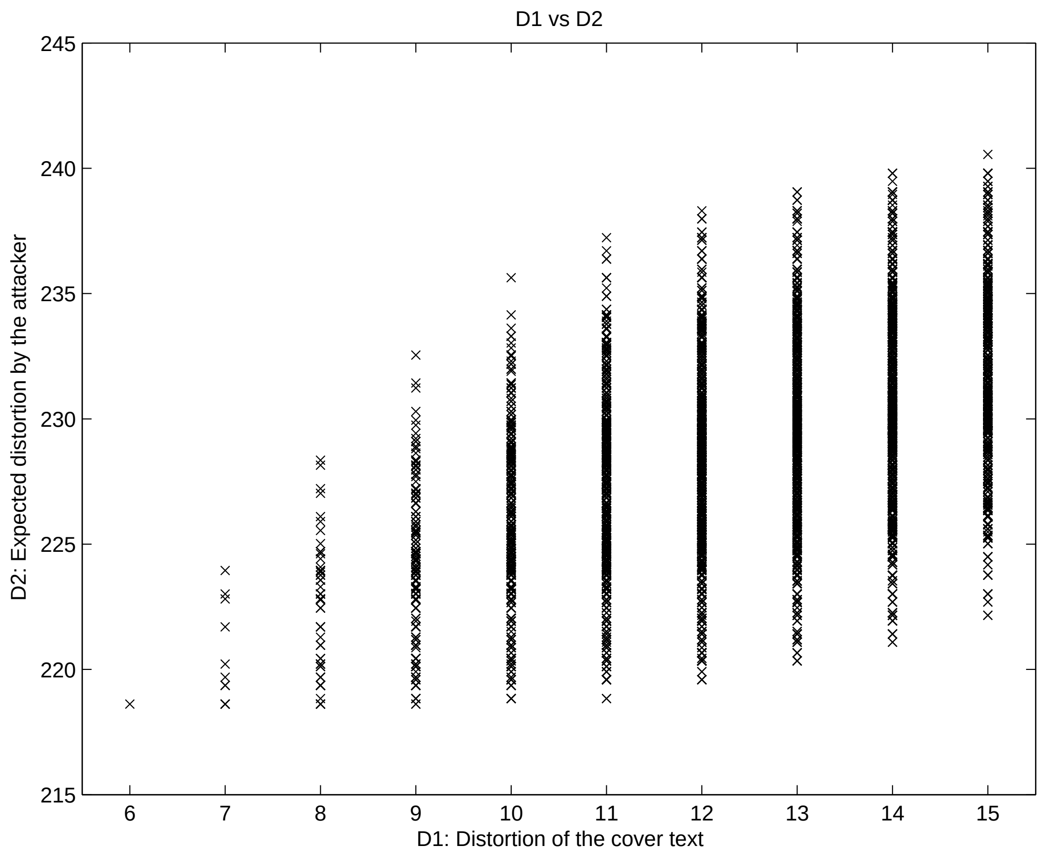


Fig. 5.3. Each point indicates a successful watermark insertion. The X-axis is the incurred distortion to the cover document and Y-axis is the expected distortion that will be incurred by adversary to undo the watermark.

bits create more branches in the search tree and results in very dense graphs. The maximum distortion, D_1 in this experiment was limited to 15 substitutions.

5.5 Related Work

Natural language information hiding systems, that are based on modifying a cover document, mainly re-write the document using linguistic transformations such as synonym substitution [48, 90], paraphrasing [81, 91] or translation to another language [92]. Most of the proposed information hiding systems are designed for steganography. Even though the steganography systems do not need to take into consideration an active warden, they still need to obey to stealthiness constraints. This brings both watermarking and steganography to the same point: imposing minimum embedding distortion to the cover document. This requirement is also used as a justification for performing isolated changes on the cover document by doing in place replacement of mark carrying units i.e. words or sentences, with a synonym instead of text-wide-transformations. To the best of authors' knowledge this is the first work that quantifies distortion done on the cover document and that is guided by both the message insertion and the resilience requirements.

T-Lex is one of the first implemented systems that embed hidden information by synonym substitution on a cover document [48, 80]. T-Lex first generates a database of synonyms by picking the words that appear only in the same set of synonym sets from WordNet. For example, if the synonym sets are given as $S_1 : \{w_1, w_2\}$, $S_2 : \{w_1, w_2, w_3\}$ the words w_1 and w_2 are inserted into this database as synonyms and w_3 is filtered out. The intersections between distinct synonym sets are eliminated to avoid usage of ambiguous words for encoding. This filtering causes the use of uncommon words while performing the substitutions (e.g. replacing "nothing" with "nada") due to the fact that common words tend to span through several unrelated synonym sets [93]. A given message is embedded into the cover text using the synonym set database as follows. First, the letters of the message text are Huffman coded according to English letter frequencies. The Huffman coded message is embedded into message carrying words in the cover text by replacing them with their synonyms in the synonym database of T-Lex . The synonym sets in this database are interpreted as a mixed-radix digit encoding according to the set elements' alphabetical order.

In [80], Bergmair provides a survey of linguistic steganography. He also discusses the need for an accurate word sense disambiguator for a fully automated synonym substitution based steganography, where sense disambiguation is required both at decoding and encoding time. The lack of accurate disambiguation forces the synonym substitution based information hiding systems to restrict their dictionaries to a subset of words with certain features. Besides decreasing the communication bandwidth, such restrictions cause the systems to favor use of rare words for encoding information [93]. In another work, Bergmair et al. proposes a Human Interactive Proof system which exploits the fact that even though machines can not disambiguate senses of words, humans can do disambiguation highly accurately [49].

Grothoff et al presented so far the only steganography system that advantageously exploits the weaknesses of current natural language processing tools [92]. They use the low quality of automatically translated text to conceal the existence of an embedded stego message. In their system, several machine translation systems are used to have several alternative translations for a given sentence, if the machine translation systems had been perfect they would have produced very similar or the same translation sentence. A similar approach has been introduced for using different MP3 encoders for steganography in [94]. In the system proposed by Grothoff et al., the quality of the output document is less important than being able to deliver the message and the stealthiness of the communication. Thus, they do not need to limit the distortion done on a cover document as long as it carries the message and shows the statistical and stylistic characteristics of machine translation systems' output.

Privacy-preserving data mining techniques aim to ensure privacy of the raw local data while supporting accurate reconstruction of the global data mining models. *Data perturbation* is one of the approaches used in this area. This approach is based on distortion of the user data by user-set parameters in a probabilistic manner such that accurate models for joint data of several users can be generated by a central data mining process. The data miner is given the perturbed database, V, and the perturbation matrix, A, where $A_{vu} = p(u \rightarrow v)$ and $p(u \rightarrow v)$ is the probability of an original user record, $u \in U$, being perturbed into a record $v \in V$. U is the user's local copy of the database. After receiving V and A, the data miner attempts to reconstruct the original distribution of database U and generate data mining models for U.

Agrawal et al. in [95] proposes further randomization of perturbation parameters in, A, for each user separately in order to provide extra privacy for the users. Randomization of the perturbation parameters make it harder for the data miner to guess the original values of the records in U. Previous data perturbation systems were using deterministic perturbation matrices. Agrawal et al. quantify this extra privacy by enforcing an upper limit on the ratio of the probability of a record, $v \in V$ being perturbed from either of the two records, $u_1 \in U$ or $u_2 \in U$. This quantification is formalized below, where S_X is the domain set for the values of the records in database X:

- A randomization $R(U)$ is at most γ-amplifying for $v \in S_V$ if

  $$\forall u_1, u_2 \in S_U : \frac{p(u_1 \to v)}{p(u_2 \to v)} \leq \gamma \tag{5.5}$$

  where $\gamma \geq 1$ and $\exists u : p(u \to v) > 0$.

Above assertion also means that the ratio of any two matrix entries in A should not be more than γ.

This idea has a similar intuitive motivation as what we are proposing. While Agrawal et al. propose randomization of perturbation parameters for improving privacy of data mining, we propose increasing ambiguity of a watermark carrying document for improving the resiliency of watermarking.

5.6 Summary

We presented and discussed a synonym-based natural language watermarking system that we designed and built. This is the first instance of the use of quantified notions of differences between sentences in natural language information hiding. The use we make of such differences is twofold. First, we use them to maximize capacity without exceeding the maximum allowable cumulative distortion, and achieve resilience by giving preference to ambiguity-increasing transformations that are harder for the adversary to un-do. Second, we achieve additional resilience by getting close to the maximum allowable cumulative distortion ourselves, as a way of preventing the adversary from carrying out attacking transformations (as these are likely to push the text beyond the allowable distortion limit).

The current system can be enhanced in the following ways.

- Move from a solely WordNet-based difference function, to a more domain-specific difference function that is also corpus-based. For example, using the Reuters corpus in addition to WordNet will result in a better watermarking scheme for Reuters articles.
- Increasing the information-carrying capacity through the use of wet-paper codes [88]. The specific way this increases capacity was discussed in Section 5.4.
- Increasing the resiliency through the use of wet-paper codes or error correction codes. The specific way this increases resiliency was again discussed in Section 5.4.
- Making experiments to evaluate the performance of copyright infringement detection systems for two cases; where the original document is either watermarked or not-watermarked.
- Using a more powerful ontology to increase both capacity and resilience. This kind of knowledge base would allow such substitutions as replacing “Washington D.C.” by “the capital”. Such a powerful ontology can provide us the ability to re-write a sentence like “Bush returned to Washington D.C” as “The President came back to the capital”.

6. SUMMARY AND CONCLUSIONS

We have demonstrated that Natural Language Processing (NLP) technology provides methods that can be used to significantly enhance the security of some important applications.

In Chapter 2, we propose, develop and evaluate a system that automatically generates memorable mnemonics for a given password based on a text-corpus. Our system helps users remember crack-resistant passwords by automatically generating mnemonics.

In Chapter 3, we propose a password mnemonic scheme that can handle multiple passwords with a single mnemonic, and is applicable to any existing system without any modification, as it does not require any form of involvement from the service provider (e.g., bank, brokerage). Our approach consists of generating a mnemonic sentence that helps the users remember a multiplicity of truly random passwords, which are independently selected. The scheme is such that changes to passwords do not necessitate a change in the mnemonic sentence that the user memorizes. Hence, passwords can be changed without any additional burden on the memory of the user, thereby increasing the system's security. An adversary who breaks one of the passwords encoded in the mnemonic sentence does not gain information about the other passwords.

In Chapter 4, we deal with the question of authentication in environments where the inputs are constrained to be yes/no responses to statements displayed on the user's screen (e.g., authentication in hands-free environments, authentication to tiny mobile devices and authentication of people with disabilities, etc.) We present 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.

In Chapter 5, we explore another application of the techniques we presented in the previous chapters: The resilient watermarking of natural language text through a mechanism that was previously thought to be inherently non-resilient, namely synonym substitution (we used synonym substitution in the first three chapters to encode passwords into mnemonic sentences). We proposed a better way to use synonym substitution, one that is no longer entirely guided by the mark-insertion process: It is also guided by a resilience requirement, subject to a maximum allowed distortion constraint (above which the value of the document becomes unacceptable for the purpose of the particular application context). When there are many alternatives to carry out a substitution on a word, we prioritize these alternatives according to a quantitative resilience criterion and use them in that order. In a nutshell, we favor the more ambiguous alternatives. In fact not only do we attempt to achieve the maximum ambiguity, but we want to simultaneously be as close as possible to the above-mentioned distortion limit, as that prevents the adversary from doing further transformations without exceeding the damage threshold; that is, we continue to modify the document even after the text has "confessed" to the mark, for the dual purpose of maximizing ambiguity while deliberately getting as close as possible to the distortion limit. The quantification we use makes possible an application of the existing information-theoretic framework, to the natural language domain, which has unique challenges not present in the image or audio domains. The resilience stems from both (i) the fact that the adversary does not know *where* the changes were made, and (ii) the fact that automated disambiguation is a major difficulty faced by any natural language processing system (what is bad news for the natural language processing area, is good news for our scheme's resilience).

LIST OF REFERENCES

LIST OF REFERENCES

[1] M. Sasse, S. Brostoff, and D. Weirich, “Transforming the ‘Weakest Link’ a Human/Computer Interaction Approach to Usable and Effective Security,” *BT Technology Journal*, vol. 19, no. 3, pp. 122–131, 2001.

[2] A. Adams and M. Sasse, “Users are not the enemy,” *Communications of the ACM*, vol. 42, no. 12, pp. 40–46, 1999.

[3] D. C. Feldmeier and P. R. Karp, “Unix password security – Ten years later,” in *CRYPTO*, pp. 44–63, 1989.

[4] E. Spafford, “Observing reusable password choices,” *Proceedings of the 3rd UNIX Security Symposium*, pp. 299–312, 1992.

[5] A. Narayanan and V. Shmatikov, “Fast dictionary attacks on passwords using time-space tradeoff,” *Proceedings of the 12th ACM conference on Computer and communications security*, pp. 364–372, 2005.

[6] A. One, “Smashing the stack for fun and profit,” *Phrack Magazine*, vol. 49, no. 7, 1996.

[7] A. Volchkov, “Revisiting single sign-on: A pragmatic approach in a new context,” *IT Professional*, vol. 3, no. 1, pp. 39–45, 2001.

[8] B. Ives, K. Walsh, and H. Schneider, “The domino effect of password reuse,” *Communications of the ACM*, vol. 47, no. 4, pp. 75–78, 2004.

[9] S. Katzenbeisser and F. A. P. Petitcolas, *Information hiding techniques for steganography and digital watermarking.* Artech House Publishers, 2000.

[10] D. Klein, “Foiling the cracker: A survey of, and improvements to, password security,” *Proceedings of the USENIX UNIX Security Workshop,(Portland)*, pp. 5–14, 1990.

[11] R. Morris and K. Thompson, “Password security: A case history,” *Communications*, 1979.

[12] J. Yan, A. Blackwell, R. Anderson, and A. Grant, “Password memorability and security: Empirical results,” *IEEE Security and Privacy*, vol. 2, pp. 25–31, 2004.

[13] D. Davis, F. Monrose, and M. K. Reiter, “On user choice in graphical password schemes,” in *Proceedings of the 13th USENIX Security Symposium*, pp. 151–164, August, 2004.

[14] I. Jermyn, A. Mayer, F. Monrose, M. Reiter, and A. Rubin, “The design and analysis of graphical passwords,” *Proceedings of the 8th USENIX Security Symposium*, pp. 1–14, 1999.

[15] G. Blonder, "Graphical password," 1996. US Patent 5,559,961.

[16] R. Dhamija and A. Perrig, "Deja Vu: A user study using images for authentication," *Proceedings of the 9th USENIX Security Symposium*, pp. 45–48, 2000.

[17] "Wikipedia entry for mnemonic." `http://en.wikipedia.org/wiki/Mnemonic`.

[18] G. Miller, "Human Memory and the Storage of Information," *Information Theory, IEEE Transactions on*, vol. 2, no. 3, pp. 129–137, 1956.

[19] J. Anderson and C. Lebiere, *The Atomic Components of Thought.* Lawrence Erlbaum Associates, 1998.

[20] G. Bower and J. Reitman, "Mnemonic elaboration in multilist learning," *Journal of Verbal Learning and Verbal Behavior*, vol. 11, pp. 478–485, 1972.

[21] G. Bower, "Analysis of a mnemonic device," *American Scientist*, vol. 58, no. 5, pp. 496–510, 1970.

[22] "Project gutenberg." Available at: `http://www.gutenberg.org/`.

[23] "Wordnet." Available at: `http://wordnet.princeton.edu/`.

[24] "Stanford log-linear tagger." Available at: `http://www-nlp.stanford.edu/software/tagger.shtml`.

[25] "PC KIMMO – A morphological parser." Available at: `http://www.sil.org/pckimmo/`.

[26] D. Nelson, U. Reed, and J. Walling, "Picture superiority effect," *Journal of Experimental Psychology: Human Learning and Memory*, vol. 3, pp. 485–497, 1977.

[27] G. Bower, M. Karlin, and A. Dueck, "Comprehension and memory for pictures," *Memory and Cognition*, vol. 3, no. 2, pp. 216–220, 1975.

[28] A. Paivio, T. Rogers, and P. Smythe, "Why are pictures easier to recall than words," *Psychonomic Science*, vol. 11, no. 4, pp. 137–138, 1968.

[29] H. Davies, "Physiognomic access control," *Information Security Monitor*, vol. 10, no. 3, pp. 5–8, 1995.

[30] "The science behind passfaces.," September 2001. `http://www.realuser.com/published/ScienceBehindPassfaces.pdf`.

[31] A. Perrig and D. Song, "Hash visualization: A new technique to improve real-world security," *International Workshop on Cryptographic Techniques and E-Commerce*, pp. 131–138, 1999.

[32] J. Nielsen, *Usability Engineering.* Morgan Kaufmann, 1994.

[33] J. Thorpe and P. van Oorschot, "Graphical dictionaries and the memorable space of graphical passwords," *13th USENIX Security Symposium*, 2004.

[34] J. Thorpe and P. van Oorschot, "Towards secure design choices for implementing graphical passwords," *20th Annual Computer Security Applications Conference*, pp. 6–10, 2004.

[35] "Password generator," 2005. Available at: `http://www.diplodock.com/Products/PasswordGenera\\tor/default.aspx`.

[36] "Password generator." Available at: `http://www.alphalink.com.au/~sergeb/easy-to-remember-passwords.html`.

[37] M. Atallah, C. McDonough, V. Raskin, and S. Nirenburg, "Natural language processing for information assurance and security: An overview and implementations," *Proceedings of the 2000 workshop on New security paradigms*, pp. 51–65, 2001.

[38] "The password mnemonic generator.." Available at: `http://www.aber.ac.uk/cgi-bin/user/syswww/gw/mnemonic`.

[39] Personal Communication.

[40] J. Meehan, "Tale-spin and micro tale-spin," *Inside computer understanding. Erlbaum Lawrence Erlbaum Associates, Hillsdale, NJ*, 1981.

[41] H. Liu and P. Singh, "Makebelieve: Using commonsense knowledge to generate stories," *Eighteenth national conference on Artificial intelligence table of contents*, pp. 957–958, 2002.

[42] S. Bringsjord and D. Ferrucci, *Artificial Intelligence and Literary Creativity: Inside the Mind of Brutus a Storytelling Machine*. Lawrence Erlbaum Associates, 1999.

[43] "Senseclusters." `http://www.d.umn.edu/~tpederse/code.html`.

[44] B. Schneier, "Write down your password," June 2005. `http://www.schneier.com/blog/archives/2005/06/write\_down\_your.html`.

[45] C. Fellbaum, *WordNet: An Electronic Lexical Database*. MIT Press, 1998.

[46] B. Schneier, *Applied Cryptography: Protocols, Algorithms, and Source Code in C*. New York, NY: John Wiley & Sons, Inc., 1993.

[47] T. Pedersen, S. Patwardhan, and J. Michelizzi, "Wordnet::Similarity - Measuring the Relatedness of Concepts," in *Proceedings of Fifth Annual Meeting of the NAACL*, (Boston, MA), May 2004.

[48] K. Winstein, "Lexical steganography through adaptive modulation of the word choice hash," 1998. `http://www.imsa.edu/~keithw/tlex/`.

[49] R. Bergmair and S. Katzenbeisser, "Towards human interactive proofs in the text-domain," in *Proceedings of the 7th Information Security Conference*, vol. 3225, pp. 257–267, Springer Verlag, September, 2004.

[50] S. Jeyaraman and U. Topkara, "Have the cake and eat it too – infusing usability into text-password based authentication systems," in *ACSAC '05: Proceedings of the 21st Annual Computer Security Applications Conference*, (Washington, DC), pp. 473–482, IEEE Computer Society, 2005.

[51] U. Topkara, M. Topkara, and M. J. Atallah, "The hiding virtues of ambiguity: Quantifiably resilient watermarking of natural language text through synonym substitutions," in *Proceedings of ACM Multimedia and Security Workshop*, (Geneva, Switzerland), September 26-27, 2006.

[52] R. Morris and K. Thompson, "Password security: A case history," *ACM Communcations*, vol. 22, no. 11, pp. 594–597, 1979.

[53] C. Kuo, S. Romanosky, and L. F. Cranor, "Human selection of mnemonic phrase-based passwords," in *SOUPS '06: Proceedings of the Second Symposium on Usable Privacy and Security*, (New York, NY), pp. 67–78, ACM Press, 2006.

[54] S. Gaw and E. W. Felten, "Password management strategies for online accounts," in *SOUPS '06: Proceedings of the Second Symposium on Usable Privacy and Security*, (New York, NY), pp. 44–55, ACM Press, 2006.

[55] P. Resnik., "Selectional preference and sense disambiguation," in *the ACL SIGLEX Workshop on Tagging Text with Lexical Semantics: Why, What, and How?*, (Washington D.C.), April 1997.

[56] B. Pinkas and T. Sander, "Securing passwords against dictionary attacks," in *Proceedings of the ACM Computer and Security Conference*, pp. 161–170, November, 2002.

[57] L. von Ahn, M. Blum, N. Hopper, and J. Langford, "CAPTCHA: Using hard AI problems for security," in *Proceedings of Eurocrypt*, pp. 294–311, 2003.

[58] R. Anderson, "Why cryptosystems fail," in *CCS '93: Proceedings of the 1st ACM Conference on Computer and communications Security*, (New York, NY), pp. 215–227, ACM Press, 1993.

[59] I. T. Accessibility and Workforce-Division, "Section 508: The road to accessibility," 1998.

[60] C. Brown, "Assistive technology computers and persons with disabilities," *ACM Communications*, vol. 35, no. 5, pp. 36–45, 1992.

[61] BBC-News, "Most websites failing disabled," Published on 2006/12/05.

[62] J. Mankoff, A. Dey, U. Batra, and M. Moore, "Web accessibility for low bandwidth input," in *Proceedings of the 5th International ACM Conference on Assistive Technologies*, (New York, NY), pp. 17–24, ACM Press, 2002.

[63] K. Grauman, M. Betke, J. Lombardi, J. Gips, and G. Bradski, "Communication via eye blinks and eyebrow raises: Video-based human-computer interfaces," *Universal Access in the Information Society*, vol. 2, no. 4, pp. 359–373, November, 2003.

[64] L. Adams, L. Hunt, and M. Moore, "The 'aware-system' – Prototyping an augmentative communication interface," in *RESNA*, 2003.

[65] J. Thorpe, P. C. van Oorschot, and A. Somayaji, "Pass-thoughts: Authenticating with our minds," in *NSPW '05: Proceedings of the 2005 workshop on New Security Paradigms*, (New York, NY), pp. 45–56, ACM Press, 2005.

[66] U. Topkara, M. J. Atallah, and M. Topkara, "Passwords decay, words endure: Secure and re-usable multiple password mnemonics," in *Proceedings of 22d Annual ACM Symposium on Applied Computing*, (Seoul, Korea), March 2007.

[67] R. Dorfman, "The detection of defective members of large populations," *Annals of Mathematical Statistics*, vol. 14, pp. 436–440, 1943.

[68] U. Manber, "A simple scheme to make passwords based on one-way functions much harder to crack," *Computers and Security*, vol. 15, pp. 171– 176, 1996.

[69] "Australian banker's association inc., guiding principles for accessible authentication," 4 December 2006.

[70] "W3C web accessibilty initiative," Accessed on January 10, 2006.

[71] A. Copestake, "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.

[72] "W3C mobile web initiative," Accessed on January 10, 2006.

[73] S. Trewin, "Physical usability and the mobile web," in *W4A: Proceedings of the 2006 international cross-disciplinary workshop on Web accessibility (W4A)*, (New York, NY), pp. 109–112, ACM Press, 2006.

[74] T. Vaughan, W. Heetderks, L. Trejo, W. Rymer, M. Weinrich, M. Moore, A. Kubler, B. Dobkin, N. Birbaumer, E. Donchin, E. Wolpaw, and J. Wolpaw, "Brain-computer interface technology: A review of the second international meeting," *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, vol. 11, pp. 94–109, June 2003.

[75] M. Topkara, C. M. Taskiran, and E. Delp, "Natural language watermarking," in *Proceedings of the SPIE International Conference on Security, Steganography, and Watermarking of Multimedia Contents*, 2005.

[76] P. Moulin and J. A. O'Sullivan, "Information-theoretic analysis of information hiding," *IEEE Transactions on Information Theory*, vol. 49, pp. 563–593, 2003.

[77] M. Chapman and G. Davida, "Plausible deniability using automated linguistic stegonagraphy," in *roceedings of the International Conference on Infrastructure Security*, (Bristol, UK), pp. 276–287, October 1-3 2002.

[78] R. Sion, M. Atallah, and S. Prabhakar, "Rights protection for discrete numeric streams," *IEEE Transactions on Knowledge and Data Engineering*, vol. 18, no. 5, May, 2006.

[79] R. Sion, M. Atallah, and S. Prabhakar, "Power: A metric for evaluating watermarking algorithms," in *IEEE ITCC*, (Las Vegas, Nevada), pp. 95–99, 2002.

[80] R. Bergmair, "Towards linguistic steganography: A systematic investigation of approaches, systems, and issues.," tech. rep., University of Derby, November, 2004.

[81] M. Topkara, G. Riccardi, D. Hakkani-Tur, and M. J. Atallah, "Natural language watermarking: Challenges in building a practical system," in *Proceedings of the SPIE International Conference on Security, Steganography, and Watermarking of Multimedia Contents*, 2006.

[82] P. Resnik., "Using information content to evaluate semantic similarity in a taxonomy.," in *Proceedings of the 14th International Joint Conference on Artificial Intelligence*, pp. 448–453, 1995.

[83] S. Landes, C. Leacock, and R. I. Tengi, "Building semantic concordances," in *In Fellbaum, C. (ed.) (1998) WordNet: An Electronic Lexical Database.*, (Cambridge, Mass.), 1998.

[84] E. Terra and C. Clarke, "Frequency estimates for statistical word similarity measures," in *Proceedings of Human Language Technology Conference*, (Edmonton, Alberta), pp. 244–251, 2003.

[85] A. Stolcke, "SRILM – An extensible language modeling toolkit," in *Proceedings of International Conferrence on Spoken Language Processing*, 2002.

[86] P. F. Brown, P. V. deSouza, R. L. Mercer, V. J. D. Pietra, and J. C. Lai, "Class-based n-gram models of natural language," *Compututational Linguistics*, vol. 18, no. 4, pp. 467–479, 1992.

[87] J. Rennie, "Wordnet::Querydata: A Perl module for accessing the WordNet database," 2000. `http://people.csail.mit.edu/~jrennie/WordNet`.

[88] J. Fridrich, M. Goljan, and D. Soukal, "Efficient wet paper codes," in *Proceedings of 7th Information Hiding Workshop*, vol. LNCS 3727, pp. 204–218, Springer-Verlag, 2005.

[89] H. Kucera and W. N. Francis, *Computational Analysis of Present-Day American English.* Providence, Rhode Island: Brown University Press, 1967.

[90] M. Atallah, C. McDonough, S. Nirenburg, and V. Raskin, "Natural Language Processing for Information Assurance and Security: An Overview and Implementations," in *Proceedings 9th ACM/SIGSAC New Security Paradigms Workshop*, (Cork, Ireland), pp. 51–65, September, 2000.

[91] M. Atallah, V. Raskin, C. F. Hempelmann, M. Karahan, R. Sion, U. Topkara, and K. E. Triezenberg, "Natural language watermarking and tamperproofing," in *Proceedings of the Fifth Information Hiding Workshop*, vol. LNCS 2578, (Noordwijkerhout, The Netherlands), 7-9 October 2002.

[92] C. Grothoff, K. Grothoff, L. Alkhutova, R. Stutsman, and M. Atallah, "Translation-based steganography," in *Proceedings of Information Hiding Workshop (IH 2005)*, p. 15, Springer-Verlag, 2005.

[93] C. M. Taskiran, U. Topkara, M. Topkara, and E. Delp, "Attacks on lexical natural language steganography systems," in *Proceedings of the SPIE International Conference on Security, Steganography, and Watermarking of Multimedia Contents*, 2006.

[94] R. Böhme and A. Westfeld, "Statistical characterisation of mp3 encoders for steganalysis," in *Proceedings of the ACM Multimedia and Security Workshop*, (Magdeburg, Germany), pp. 25–34, September 2004.

[95] S. Agrawal and J. R. Haritsa, “A framework for high-accuracy privacy-preserving mining,” in *Proceedings of the 21st International Conference on Data Engineering*, (Tokyo, Japan), April 5-8 , 2005.

VITA

VITA

Umut Topkara received his B.Sc. and M.Sc. degrees from the Computer Engineering Department of Bilkent University in 1999 and 2001. He started his graduate studies at Purdue University in 2002. He received his Ph.D. degree from the Computer Science Department of Purdue University in 2007. His research interests lie at the confluence of information security, machine learning, natural language processing and computer-human interaction, specifically their intersection in the field of usable security. He has also been involved in grid middle-ware engineering research during his term as a research assistant at the Rosen Computer for Advanced Computing.