Rethinking Passwords to Adapt to Constrained Keyboards

Markus Jakobsson
Extricatus LLC
Mountain View, CA 94041
markus@extricatus.org

Ruj Akavipat
Computer Engineering Department
Faculty of Engineering, Mahidol University
Nakhon Pathom, Thailand

ABSTRACT

We describe and analyze a variant of the traditional password scheme. This is designed to take advantage of standard error-correcting methods of the types used to facilitate text entry on handsets. We call the new approach *fastwords* to emphasize their primary feature compared to regular passwords. Compared with passwords, fastwords are approximately twice as fast to enter on mobile keyboards, and approximately three times as fast on full-size keyboards. This is supported by user studies reported on herein. Furthermore, these user studies show that fastwords also have considerably greater entropy than passwords, and that their recall rates are dramatically higher than that of passwords and PINs.

The new structure permits a memory jogging technique in which a portion of the fastword is revealed to a user who has forgotten it. We show that this results in boosted recall rates, while maintaining a security above that of traditional passwords. We also introduce the notion of equivalence classes – whether based on semantics or pronunciation – and describe uses, including voice-based authentication. The new technology does not need any client-side modification.

Categories and Subject Descriptors

D.4.6 [**Security and Protection**]: Authentication; K.4.4 [**Electronic Commerce**]: Security

General Terms

Authentication, Security

Keywords

Error correction, fastword, handset, mobile, password

1. INTRODUCTION

Security protocols have developed at a pace largely matching the development of online threats, but passwords remain the same – in spite of increasing pressure on authentication mechanisms [4]. Mobile authentication, in particular, poses new problems due to the limitations of handset keyboards [14].

Text entry on handsets is time-consuming and error prone, and, as a result, auto-correction and auto-completion methods are ubiquitous. However, auto-correction and auto-completion only work for *text* – not for password entry. This is due to the fact that good passwords are much like poorly spelled words, and that error-correction techniques help *remove* poor spelling. While the dictionaries used by error-correction techniques can be augmented with words they should recognize, it is naturally not a good idea to augment them with passwords – even though it *would* help entering these. Therefore, better error correction techniques help maintain usability of text entry as we move towards smaller, feedback-free on-screen keyboards. However, they do *not* help us enter traditional passwords, which therefore are becoming *harder* to input, relatively speaking. This is likely to give rise to increased reliance on password managers and short passwords – neither of which bode well for bottom-line user security.

A recent study by Jakobsson et al. [7] reports that consumers are only *slightly* less frustrated by entry of text and passwords on handsets than they are of slow web connections on such devices, and much more annoyed with all of these than lack of coverage and poor voice quality. In a survey we performed, two of five users expressed annoyance with entering passwords on handsets, and one of five stated that they avoid situations that require them to enter passwords on handsets.

This paper addresses the question of how to facilitate human-machine authentication on input-constrained devices. To benefit from error correction techniques, we need to permit dictionary words. At the heart of our solution is the insight that dictionary words are easy to enter if error-correction is enabled – and that a *sequence* of dictionary words becomes a secure credential. We refer to this as a *fastword*.

To make our proposal concrete, let's consider an example. As a user sets up an account or changes access credentials, he is offered the possibility of selecting a "mobile friendly" credential. A particular user who opts to do this may choose the fastword "frog work flat", which might correspond to a mnemonic of "I ran over a frog on my way to work, and now I have a flat frog under the tire."[1]

[1]Research into human memory suggests that colorful phrases are easier to recall than more mundane ones, but this is orthogonal to the work described herein.

Our example system would accept the fastword "frog word flat" as a strong credential, given that the frequencies[2] of the three words in English language are $2^{-17.0}$, $2^{-10.6}$ and $2^{-14.5}$ (resulting in a *product* of frequencies of $2^{-42.3}$); and that the 3-gram frequency is $2^{-49.5}$. The latter is the frequency in English language of the three words *together*. In contrast, the four-word fastword "I love you honey" might be rejected in spite of the fact that the product of word frequencies ($2^{-7.8}$, $2^{-11.8}$, $2^{-7.8}$ and $2^{-16.3}$) is $2^{-43.7}$, since the frequency of the 4-gram is only $2^{-25.8}$.

The frequency measures described above do not reflect how secure a credential is against an adversary who tries the k most common credentials, but how secure it is *on average*. However, by using a database of frequencies of keywords in previously registered fastwords – and not only considering their frequencies in English language – the system can turn down fastwords that are starting to become too common, thereby avoiding the "most popular credential" attack. This is analogous to the work by Schechter et al. [16] in the context of traditional passwords.

We do not permit keywords to be selected as names[3], or many users may be tempted to select names of friends and family members – which are often possible to gather from social networks. It is possible to implement any such policies by simple changes on the backend; it is also much easier to enunciate the policies in the context of our proposed solution than it is for traditional passwords since we can easily parse fastwords on a component level. Any fastword that is considered insufficiently strong will be refused; the user can either simply be told to enter a new one, or be told of the rule that caused the fastword to be rejected.

The proposed solution has three main benefits: (1) The *increased speed and convenience* it provides – measured in terms of the time it takes to enter a credential; (2) the *improved security* – both in terms of the average and minimum security; and (3) substantially *higher recall rates* than passwords and PINs. These rates are further boosted by the use of hints given to a user who has forgotten his or her fastword – for the fastword "frog work flat," the hint[4] might be the word "frog". This maintains sufficient security as long as the frequency measures of the *remaining* words are sufficiently low.

Our new structure allows for a class of new features that are not supported by the traditional password paradigm – such as voice-based entry and the use of equivalence classes. Equivalence classes – such as normalizing different tenses of verbs – permits the adaptation of the authentication mechanism to how people remember and enter credentials. This includes order invariance – "flat frog work' is considered equivalent to "frog work flat". It also includes synonyms (making "fat cat bite" equivalent to "chumpy kitty bite") and homophones and their approximations (making "red flower fly" equivalent to "read flour fry"). The latter helps with voice entry of credentials.

Yet another benefit of our proposed technique is that it allows for a crude determination (on the backend) of the *degree* of correctness of a given login attempt, in contrast to what can be done in traditional password systems. While this type of data should never be fed back to the user (or it can be used as an oracle to attack the system), it can aid in the collection on analytics on the backend. The fuzziness is achieved at the cost of a slight expansion of the records used to store salted and hashed credentials, but without any associated reduction of security.

Outline: We begin by describing the related work (section 2). We detail the basic structure of our proposal in section 3 – both from the point of view of the user experience and in terms of the backend solution. We then describe how to achieve an extended feature set in section 4. As examples of such extended features are voice-entry of fastwords and hints given to users who fail to log in. We report on a usability study in section 5, wherein we compare recall rates for different types of credentials. We then describe our adversarial model and provide security analysis of our proposal in section 6. In section 7, we report on a second usability experiment that lets us establish speed of entry for passwords and fastwords – for both handsets and traditional keyboards.

2. RELATED WORK

Typical typists enter on average 33 words per minute on a regular keyboard, according to a study by Karat et al. [8]. MacKenzie and Soukoreff [12] estimate that the mean entry rate is typically in the range of 15 to 30 words per minute for on-screen keyboards without error correction, while Kristensson and Zhai [9] show that it is common for users to reach 45 words per minute on on-screen keyboards if error correction *is* used. It is clear that the entry rates go down for any of these cases if the user has to make extra key presses to change case or to shift between letters, numerals and "special" characters, but we are not aware of any previous study that measures the effect of this. It could be estimated by the approximate increase of the number of key presses. On an iPhone, for example, each capitalization costs one extra click, as does each shift to/from numerals and special characters. This means that an example password, "flY2theM0On!", *costs* 21 clicks – in spite of having only twelve characters. We performed timing experiments on various platforms, and found that the amount of time taken to type simple credentials – such as fastwords or – almost doubles when using a mobile device instead of a traditional keyboard.

The handset market is increasingly moving towards soft buttons, as opposed to hardware keyboards as is standard for traditional computers. This leads to a higher rate of errors. Lee and Zhai [11] report that when data is entered using fingers (as opposed to a stylus), then the error rate is 8% higher for soft buttons than for hard buttons. (This result is for a situation without tactile or audio feedback, and where the sizes of the soft and hard buttons were approximately the same.)

Traditional password strength checking is a heuristic approach that provides some estimates on the strength of the credential. Different service providers implement vastly different approaches, which explains why one password may

[2]We use the Microsoft N-Gram Service to assess word frequencies; alternative services may result in slightly different estimates.

[3]Many names are not found in dictionaries; those who are can easily be excluded in an automated manner, given the labeling of words in common dictionaries. In Webster, for example, all names are labelled *biographical name*.

[4]For a particular fastword, one and the same word would always be the hint. This could be the first word or the least common word, for example.

be considered strong by one provider and weak by another. Our proposed credential strength checker will still rely on heuristics, but with an underpinning of quantifiable metrics, such as the frequencies of single words, pairs of words, and more generally, any n-tuple of words. This is referred to as an n-gram. In this study, we use Microsoft's Web N-Gram Service [18]. However, the word frequencies can be relative to any preferred source, such as standard spoken English; tweets; or already registered credentials – although the latter must be determined in a way that does not compromise the integrity of individual credentials. Therefore, while our proposed strength checker is not perfect, it *is* based on more clearly enunciated metrics than password strength checkers are, given that our proposed credentials have a simpler structure than good passwords do.

It is known that the more concrete and meaningful information is, the easier it is to remember [3]. We use sequences of dictionary words to enable easy error correction and simplify recall. We measure both the speed of entering credentials and the recall of these, relative to other common types of credentials.

The use of dictionary words as credentials is not new. This approach was used both by Compuserve and AOL in the mid-eighties. As a user would sign up, he or she would be assigned a password that consisted of two dictionary words with some connector. Compuserve, for example, used the format expressed by the sample password "evening_crucial". Neither Compuserve's nor AOL's scheme can take advantage of error-correction techniques, nor were they designed to.

S/KEY, a one-time password system for Unix systems, translates 64-bit one-time passwords to sequences of words by mapping the bit strings to six words drawn from a public 2048-word dictionary. The use of words in S/KEY was to improve usability of entering keys, and not to take advantage of error-correction or mnemonics.

Similarly, Bard [1] proposed a technique in which users are assigned a collection of words as their credential. In that scheme, the collection of words is drawn uniformly at random from a large set of words that exhibit optimal distance characteristics from other selectable words. This permits error correction of words within this dictionary. However, with words like "abarticular" and "galaxidae"[5] being equally likely to be assigned to a user as "love" and "foot" are, his system decidedly is not very practical. In contrast, our goals are pragmatic: To maximize authentication success and speed. We achieve this by allowing the user to select his or her own credential. While our error correction does not have theoretically optimal properties, it is practical, and our system can use *standard* auto correction and completion algorithms[6].

Turning to the security of a credential, it is worth noting that there are two appropriate but very distinct security measures worth considering. One aims at assessing the complexity of passwords, then equating security with complexity. (This is what traditional password strength checkers do.) The other one focuses solely on avoiding the weakest (i.e., most common) credentials, since these are what most attackers try. In a sense, traditional password strength checkers do this using a manually entered list passwords that are believed to be weak. Another more elegant approach was proposed by Schechter et al. [16]. Their technique automatically avoids common passwords, without any explicit identification of what these are. Our structure supports both of these approaches. The *observed* commonality of a fastword can be determined using methods analogous to those described in [16]; moreover, the *estimated* likelihood can be computed using the frequencies of words and their combinations – to produce a more fine-grained strength estimate than is possible for passwords. The ability to break down a credential into its components and determine the likelihood of the combination makes it possible to detect and avoid common phrases – whether by relying on search engines, a corpus of common phrases, or simply N-gram services. This makes it possible to avoid weak credentials, which otherwise is a security risk associated with mnemonic passwords [10].

We describe voice-based entry of fastwords. Voice-based authentication was studied by Monrose et al. [13]; however, they focused on *how* a phrase was spoken – not just what the phrase was. The voice-based entry in our proposal is not about biometrics, but simply a matter of what user interface we rely on for the entry of the fastword. As a result, we can use standard dictation tools to interpret and perform error correction of the audio data. Using equivalence classes, we can avoid problems associated with lack of precision without having to train the system on the level of individuals.

3. BASIC FEATURE SET

User experience. The user experience of entering fastwords will be very similar to that of entering passwords – except with the added benefits endowed by error-correction and auto-completion features. Instead of entering a password, the user would simply enter a sequence of words, separated by spaces. As a user completes a word, the word can be shown for an instance before each letter of the word is replaced by a star. Like for traditional passwords, the user would press enter at the end of the sequence. This user experience is the same when fastwords are *registered* (enrollment) and when they are *used* (authentication.) A credential strength meter can be used to indicate the quality of what has been entered during enrollment. A sample user interface is shown in Figure 1.

Client-side process. In contrast to text entry, traditional password entry does not rely on error-correction techniques. The incorrect password submission simply triggers the event which asks the user to try again. Fastword entry is instead implemented like regular text entry, which means that auto-correction and auto-completion are not *disabled*, and are therefore *automatically* performed in the selected language.

Analogous to how characters are often replaced by stars or other characters during password entry (whether immediately or as the next character is entered), completed words can be replaced by stars during fastword entry. This can be achieved using Javascript or an embedded program such as Flash or Java applet.

The credential is transmitted over an encrypted channel to a backend server in charge of enrollment or authentication;

[5]These are actual examples from [1]. However, while words known only by linguists could surely be filtered out, it is indisputable that user-selected credentials have better recall rates than schemes based on assigned credentials.

[6]On a handset, this is trivial: We simply do not disable the auto correction/completion features, as is done for traditional passwords. It is also possible to have auto correction/completion on traditional computers, where it can be added either as a client-side plugin, a javascript snippet, or a server-side feature.

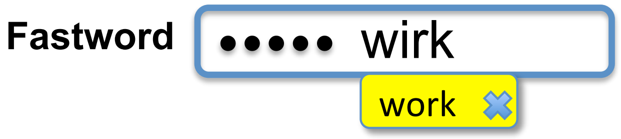

Figure 1: What the user may see when entering a fastword. The first word has been replaced by stars, and the second word is shown with an auto-correct suggestion. The use of auto correct and auto complete allows users to type faster and with less precision. To accept a suggestion, the user simply presses space and continues writing the next word – or presses enter or submit to conclude. To turn down a suggestion – which should typically not happen in the context of fastword entry – the user taps the X next to the suggestion.

this can be done in installments (e.g., after each keyword) or after the entire fastword has been entered. The backend server then signals back whether the credential is accepted or not. For enrollment, this corresponds to communicating the result of a credential strength check (described below). For authentication it is simply a matter of signaling success or failure.

Backend process.

- **Credential strength checker.** As a new credential is submitted, whether as an account is set up or to replace another credential, the credential strength checker is used to verify that the credential is sufficiently strong.

  The credential strength checker determines the product of single-word frequencies of the words in a credential, and uses that as one strength estimate. The strength checker also determines the N-gram frequency of the sequence, and uses that as a second estimate. These two security assessments are performed relative to frequencies in English language (e.g., using the Microsoft Web N-Gram Service [18]) and relative to already registered fastwords. If any of these measures indicates that the new credential is more likely than a system security threshold (such as 2^{-30}) then the credential is rejected.

  The output of the credential strength checker is the inverse of the maximum of the the result of the different checks, which is the estimated probability with which the adversary is expected to be able to guess the credential. Alternatively, it can be represented as the minimum of the bits of security of the two tests, i.e., the negative second logarithm of the associated frequency.

- **Dictionary words.** In contrast to typical passwords, it is not desirable for the user to include non-dictionary words in a fastword. This is because the auto-completion feature on the client device would *learn* these new terms eventually – which inevitably means to *store* them. This is undesirable from a security stance. To avoid this, the server-side will verify that all words are dictionary words when the user registers a fastword. (It would either have to ask the user what language is used, or infer it from the words used.)

- **Enrollment.** After a credential has been determined to be strong, it is accepted – and then stored on the backend. Just as passwords are salted and hashed to reduce the risk of internal exposure, so are fastwords. More specifically, if the credential is a k-tuple of words, $W = (w_1, w_2, \ldots, w_k)$, then $hash(W, salt)$ is stored, along with the unique value *salt*.

- **Normalization of credentials.** We assume the use of some amount of normalization, whether for robustness or to add system features. An example of the former type of normalization is for all credentials to be converted to lower-case representations before they are salted and hashed. As an example of a feature-extending type of normalization, one may sort the words of the fastwords in order to obtain order invariance.

- **Conventional authentication.** The server looks up the appropriate user record (given the user name or other identifier), and salts and hashes the normalized credential to be verified, comparing the result with the stored result. More specifically, the value *salt* is extracted from the database, $hash(W, salt)$ is computed, where W is the normalized credential to be verified. If the result of the hash matches the stored result, the authentication is said to succeed.

- **Application.** The technique we describe can be used both to authenticate from handsets to remote sites, and to the handsets themselves. In the latter case, an external service could to be involved during the fastword registration phase in order to verify the strength of the credential – it is not practically feasible to house this database on the handset. If no strength check is needed, this outsourcing is also not required.

We report on relative recall rates for different types of credentials in section refrec; analyze the security of our construction in section 6; and the speed on credential entry in section 7. In the next section, we describe an extended feature set based on the techniques we have just described.

4. EXTENDED FEATURE SET

There is an array of new features that are made possible by the new structure we use – and, in particular, by the decomposability of the credential. We will now describe some of these features.

Use of conceptual equivalence classes. One can use conceptual equivalence classes to allow for *variants* of a word to be accepted, which aims as establishing the intent of the user when she enters a credential. The use of conceptual equivalence classes[7] addresses a situation in which some words are largely interchangeable to users, a situation which could otherwise potentially create difficulties if a user has to remember the exact word she used. As a simple example, an equivalence class may contain different tenses of a given verb – in order, for example, to avoid a distinction to be made between the word "run" and the word

[7]It is straightforward to generate some conceptual equivalence relations, e.g., for tenses and synonyms, but not clear how to generate a complete collection.

"running". Equivalence classes may also be used to allow substitution of words of similar meaning. For example, a user entering a fastword (mother, stroke, wedding) during enrollment may later attempt to authenticate using the sequence (mom, stroke, wedding) or (mother, rub, wedding) – depending on whether the person uses multiple terms to refer to his/her mother, and based on the intended meaning of "stroke". (There is no attempt to infer the meaning of a word on the backend in the current proof-of-concept implementation.)

Given a credential $W = (w_1, w_2, \ldots, w_k)$, the backend computes $E(W) = E(w_1), E(w_2), \ldots, E(w_k))$, where E is the function that maps a word to its equivalence class. Instead of computing $hash(W, salt)$ for a given value $salt$, the backend would compute $hash(E(W), salt)$ – whether for the purposes of enrollment or authentication.

Use of homophonic equivalence classes. One can use a normalization corresponding to homophonic equivalence classes to simplify voice-based[8] entry of credentials. We assume the use of standard dictation tools to create a mapping from the audio sample to a homophonic equivalence class; this corresponds to the error-correction processing of text inputs. To avoid having to train the tool on individual speakers (as dictation tools need), we will combine this with wide equivalence classes. This will map a large number of words to the same equivalence class, which will result in the same selection of equivalence class for different pronunciations and accents.[9] The resulting equivalence classes are phonetic representations of the words of the fastword. To process the credential, the backend would salt and hash the sequence of phonetic representations to create the credential record.

The creation of homophonic equivalence classes, and the associated credential records could be done *in addition* to the other credential records created and maintained on the backend. During voice-based authentication, the candidate credential would be verified by being represented by its phonetic description, salted, hashed, and compared to the stored record.

Implementing fuzzy authentication. Instead of storing a salted hash of the *full* credential W during the enrollment process, the backend server stores salted hashes of all acceptable variants, using the same salt for each such variant. This is done in a manner that does not reveal the number of words k of the credential, where $k \leq k_{max}$. If we set $k_{max} = 4$ words, then there are no more than four subsets of the credential in which one word has been omitted – and one verbatim credential. If a credential has fewer than k_{max} words, then the remaining slots in the record would store random strings of the correct format. This way, one cannot infer the value k from a user record on the backend.

To perform *fuzzy* verification of a submitted credential during an attempted login, it is checked against each of the stored credentials by salting and hashing it, comparing the result to the stored values. One can also verify that it has no more than k_{max} words, and test all subsets of size $k_{max} - 1$ to see if either matches the stored values.

If the submitted credential, after being salted and hashed, matches the stored full credentials then the login attempt is successful. If either of the comparisons with the subset-credentials results in equality, then the submitted credential is known to be *almost* correct. It is a matter of policy how to react to *almost* correct credentials: The backend server may consider it a successful login attempt; may permit limited access; or other system actions may be taken. Users are not given any feedback describing the degree to which their credential was correct, or this could be used as a password breaking oracle.

Implementing hints. If a user forgets her fastword, she can be given a hint, which is one of the words in her fastword. It would always be the same word for a particular user and fastword; it could be either the first word or the word with the lowest frequency – this is fairly likely to provide the most help to the user. The hint is also selected so that the frequency measures of the *remaining* words in the fastword correspond to a sufficiently secure credential[10]. In section 5, we report on the extent to which hints help users recall a credential, based on a user study we performed.

Implementing fuzzy blacklists. If a given user's credentials are believed or known to have been compromised (e.g., by a phisher or malware), then the exposed credential can be placed on a user-specific blacklist. This would block the user from using this credential – or one with a large overlap – later on. This is to address the problem that users have "credential classes" and there is a large degree of reuse – even after a credential has been corrupted!

5. RECALL RATES

We performed an experiment in which we recruited users to set up a collection of different credentials, and to attempt to authenticate between 2-3 weeks later. (On average, the authentication took place 20 days after the setup.) To incentivize participation, we raffled off an iPad2 among all subjects that completed the study, independently of "performance". A total of 147 subjects enrolled in the study; 105 completed it.

The aim of the study was to determine what types of credentials are easiest to recall, relative to each other. For this reason, we asked subjects not to reuse credentials from elsewhere, as this would bias their ability to recall these (and provide the authors with valid real-life credentials for these subjects, which we did not want to obtain.) Furthermore, we asked the subjects to promise that they would not write down any credentials.

We asked subjects to create five types of credentials: a "simple password" (such as what they might use for a social networking service); a "strong password" (such as what they may use for financial transactions; a 4-digit PIN; a 6-digit PIN; and a three-word fastword. We also asked them to remember a "super-strong password" – a complex password we assigned to them. For each credential, we asked them to

[8]The issue of eavesdropping is an orthogonal problem to the management of audio authentication, and is not addressed in this paper.

[9]One might argue that it is enough that the sequence sounds *the same* during registration and authentication; however, the use of wide equivalence classes permits text-based registration followed by voice-based authentication.

[10]We note that if a hint has *ever* been given for a given fastword, without a successful authentication following in the same session, then the word that corresponds to the hint should be considered public. This must be considered in the context of fuzzy verification.

assess how likely they would be to remember it after two-three weeks. In the second phase of the experiment, we asked them to recall the credentials and state whether they believe they managed to do so.

Subjects were considered to have succeeded with an authentication if they managed to enter the credential verbatim during the authentication stage, except for the fastwords, where capitalization, tense, and order were not considered, and where subjects who entered at least two of the three words of the fastword were passed. This matches the way real authentication would be performed. Subjects who failed the fastword authentication were given a hint – the first word of their fastword – and asked to try again. Subjects who failed the strong password authentication were given a second chance, too, but no hint. This, to matches the way real authentication would be performed.

Table 1: Recall rates for various types of credentials.

	Recall rates	
Credential	User estimate	Measured
Simple passwords	24%	14%
Strong passwords	22%	6%
– addl. after reminder	10%	0%
Super-strong passwords	5%	2%
4-digit PIN	47%	26%
6-digit PIN	28%	29%
3-word fastword	25%	36%
– addl. after hint	65%	48%

The recall rates of the various credentials are shown in table 1. The subjects's guess whether they correctly recalled a credential was done on a 5-point Likert scale, where we count the responses "I think I did", "very likely" and "certain" as a vote of confidence, while "maybe" and "I did not" were counted as a lack of confidence. We see that people remember passwords to a lesser extent than they expect, and fastwords to a greater extent. The lower success rates for the 4-digit PINs in comparison to the 4-digit PINs (in spite of the subject's expectations) could be explained by the fact that a greater number of subjects reused[11] their 4-digit PIN than their 6-digit PIN – probably since 4-digit PINs are more common than 6-digit PINs – and then failed to remember *which* PIN they used.

While this does not show how well people would recall any of these credentials in a real-life scenario, it shows how well they remember them relative to each other, in a setting where they are not strongly incentivized to do well. The experiment shows that users are able to recall fastwords to a much greater extent than passwords, and that close to half of those who forget their fastword are helped by the hint they are given. For actual success rates, it may be necessary to perform a more realistic experiment in which users are better incentivized to recall their credentials.

As part of the study, we collected some demographic information. Among other things, we asked subjects to indicate their profession. We could not identify any relation between profession and performance, and in particular, did not see different recall rates among technical people, who were over-represented in the study in relation to their relative number in society.

[11]In spite of being instructed not to reuse credentials, 20% of the subjects admitted to reusing a 4-digit PIN, in contrast to only 10% of for 6-digit PINs.

6. SECURITY ANALYSIS

We want to compare the strength of the fastword with that of traditional passwords. We will begin by reviewing the approximate security of passwords (section 6.1), followed by an adversarial model for our context (section 6.2), and an analysis of the security of fastwords (section 6.3).

6.1 The Security of Passwords

NIST [2] estimates that the distribution of passwords corresponds to an entropy of 4 bits for the first character, 2 bits for the next 7 characters, and 1.5 bits per character for the 9th to the 20th character, and 1 bit per character for the remainder of the password. 6 bits of entropy is added when the user is forced to use both upper case and non-alphabetic characters. This is for traditional passwords – mobile passwords are likely to have lower entropy due to the complications of entering them – at least in contexts where the user is aware of later having to enter the password on a mobile platform when first selecting it. It is also an average: There are indications that users select passwords of different strength on different types of sites. Analysis [6] of passwords from raided dropboxes suggests that the average password length was 7.9 characters, which corresponds to an entropy of approximately 18 bits. While this indicates that the *average* probability of guessing a password is 2^{-18}, an attacker can gain access to a fairly large portion of accounts simply by trying the most common credentials – this probability is on the order of $0.22 - 0.9\%$ [17, 5]. (While we do not have any evidence to support it, there are plenty of indications that passwords used on handsets are weaker than traditional passwords.)

6.2 Understanding the Adversary

We consider a remote adversary attempting to gain illegitimate access to an account. We assume that the adversary knows the rules used to approve and reject fastwords as they are first established, and that he knows system-wide weights and parameters. We also assume that he knows the frequencies of individual words and N-grams. We make the pessimistic assumption that the system does *not* know the true frequencies, but that it mis-estimates the true frequencies by up to a factor c.

The adversary wishes to guess the fastword of a given user. Since we may assume that an adversary will behave rationally, we know that he will try the most likely candidate fastwords (that would be accepted by the system) in order of decreasing likelihood, and that he will try as many as he is allowed before the account gets locked down.

The adversary will request to get the hint for the fastword (claiming to be the user to be targeted and claiming to have forgotten the fastword.) Let's say that the hint is displayed to the adversary adversary – as opposed to being sent to an email address associated with the account. The adversary then attempts to guess the two missing words in a manner that maximizes his probability of success. We say that the adversary *wins* if he manages to get access to the targeted account.

Note that we do not focus on security against shoulder surfing or eavesdropping, nor attacks in which the adversary knows his victim[12]. These are interesting attacks to consider, but are not the primary threats in most systems, and are beyond the scope of this paper.

6.3 The Security of Fastwords

We have assumed an adversary who knows the true frequencies and distributions of words and fastwords, obtains the hint for a given fastword, and who then attempts to guess the remaining two words. Let us also assume that a person will be given n chances to log in to an account from an unknown IP address.

We will let $\hat{f}$ denote an upper bound of the the actual frequencies of the n most likely fastwords, conditional on the hint. This corresponds to a probability of success for the adversary of no more than $p = 1-(1-\hat{f})^n$. This is the same as stating that $\hat{f} = 1-(1-p)^{1/n}$. Since we have assumed that the system's understanding of frequencies would be off by a factor c, this corresponds to requiring that the system's belief of the conditional frequency of a fastword, given the hint, is $f \leq (1-(1-p)^{1/n})/c$.

For concreteness, we may set the maximum probability of success for the attacker to $p = 2^{-20}$. Recalling the analysis in section 6.1, this corresponds to a *minimum* security of the solution exceeding the *average* security of typical passwords by two bits. We further assume $n = 5$ and $c = 2$. These parameter choices corresponds to a conditional frequency of the fastword – given the hint – of at most $2^{-23.3}$.

In figure 2, we show the conditional probabilities of fastwords used by subjects in our study, given the word with the lowest frequency as a hint. In figure 3, we instead plot the conditional probability based on using the first keyword in the fastword as a hint. In both graphs, we draw a line at the probability of $2^{-23.3}$, as described above. We see that 61% vs. 64% of the fastwords have conditional probabilities (for the adversary to succeed) that correspond to a fastword security that is better than the *average* password security – and *much* better than the *lowest* password security. If this is the minimum acceptable security, then fastwords that do not comply can either be rejected during the enrollment phase, or the system may refuse to disclose hints for these values.

In figure 4, we show the cumulative distribution of fastwords in our study, where hints are *not* given or where these are sent to the account owner in a way that can be established not to be possible to intercept by a typical attacker. All of these measures use the minimum-security estimate after both the N-gram frequency and the product of frequencies are computed.

We have not discussed the security against an adversary who gains access to the salted and hashed fastwords, and to the hints (which must be stored in cleartext on the backend). However, this can easily be seen to correspond to the security shown in figures 2 and 3. We note that if the system policy is to never give out a hint if the resulting security would fall below a system threshold, however, then these hints do not need to be stored.

Turning now to the security of the extended feature set, we observe that this depends on the number of and sizes of the equivalence classes. For simplicity, we assume that *all* words belong to equivalence classes, and that each such class contains exactly s elements. The probability of being able to guess the sequence will be increased by a factor of s^n where n is a number of words in a sequence. For $n = 3$ words in a fastword, as we have used, and for equivalence classes of size $s = 8$ words, this means a reduction of security by a factor 2^9 for the entire fastword, and 2^6 for the fastword given a hint.

In other words, a fastword whose probability of being guessed is 2^{-42} would be guessable with a probability of 2^{-33} if equivalence classes are used, and these each have the maximum size of $s = 8$. Similarly, if the probability of success for an adversary would be 2^{-27} after seeing the hint, then the use of equivalence classes of size $s = 8$ would increase this to a probability of success of 2^{-21}.

[12]It is worth noting that most systems are rather vulnerable against adversaries who know the victim, due to the poor security of many password reset schemes [15].

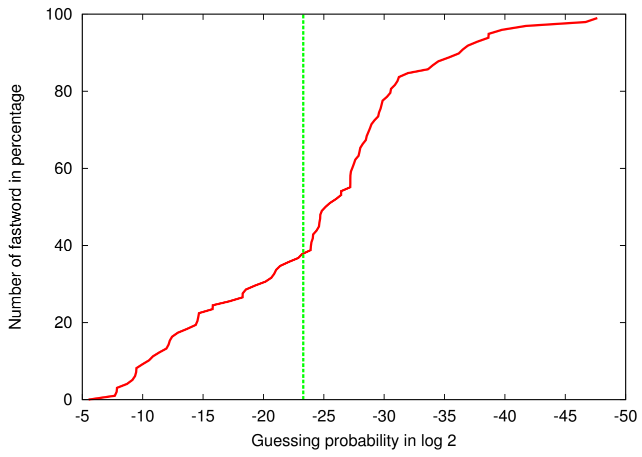

Figure 2: The figure shows a cumulative distribution of the conditional probabilities of the fastwords in our user study, after the word with the *lowest frequency* has been given as a hint. 61% of the fastwords have a security – after the hint is given – that exceeds the average security of passwords.

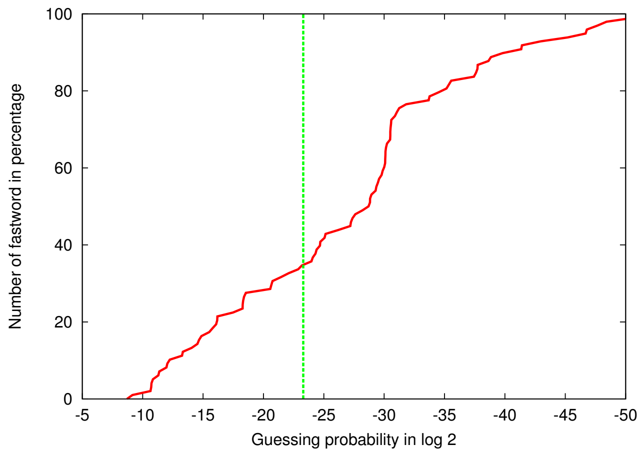

Figure 3: The figure shows a cumulative distribution of the conditional probabilities of the fastwords in our user study, after the *first* keyword of the fastword has been given as a hint. 64% of the fastwords have a security – after the hint is given – that exceeds the average security of passwords.

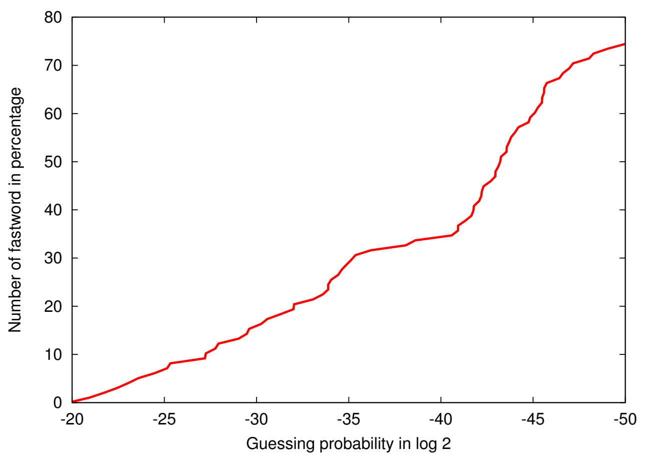

Figure 4: The figure shows a cumulative distribution of probabilities of the fastwords in our user study.

In figure 5, we show the effects on security of using such equivalence classes. The graph describes the conditional fastword probabilities, given the first keyword as a hint.

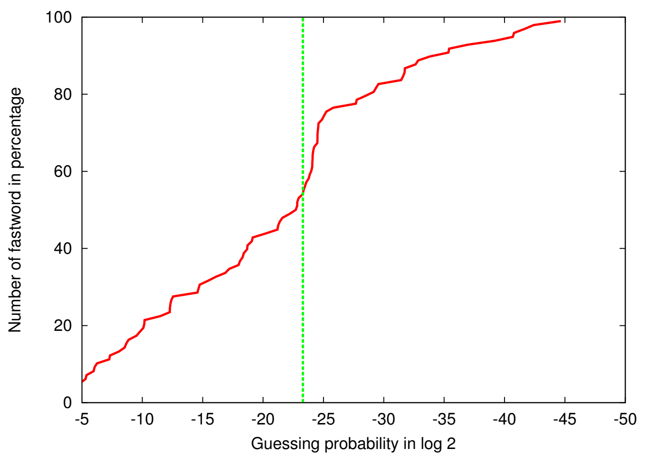

Figure 5: The figure shows a cumulative distribution of the conditional probabilities of the fastwords in our user study, with equivalence classes of size $s = 8$ added. This assumes that the adversary has obtained the *first* keyword of the fastword, after this is given as a hint. 45% of the fastwords have a security that exceeds the average security of passwords. We note that fastwords with lower security can be rejected at enrollment. Alternatively, hints can be disabled for these credentials, or smaller sets of equivalence classes be used.

We note that a realistic implementation will have differing sizes of equivalence classes. While we use a somewhat simplified analysis by assuming that all equivalence classes are of the same size, this does not affect the underlying principles. Moreover, we note that the sizes of equivalence classes can be set to balance usability needs and security expectations.

Similarly, if the system accepts a login attempt with only a partial match to the registered fastword, this affects security. It is possible to set a threshold for the minimum security

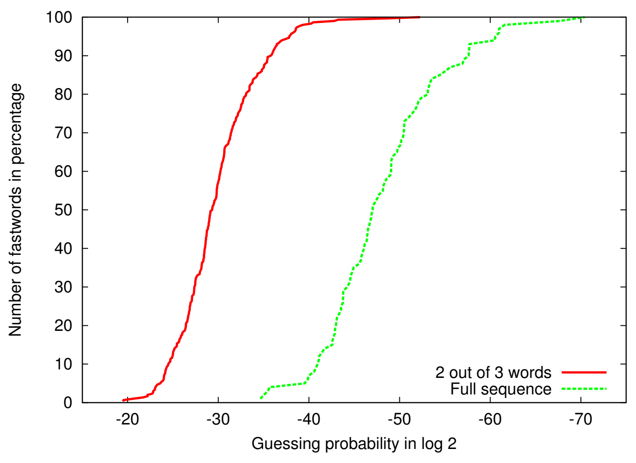

Figure 6: The figure shows the cumulative distribution of the strengths of partial fastwords of one hundred subjects where two out of the three keywords are entered and the third is left out. We show all combinations; thus, the above corresponds to the average security. For comparison, we also show the strengths of the complete fastwords for the same set of subjects. Recall that typical passwords are believed to have 18 bits [2] of security.

required. For example, if *one* subset of keywords correspond to a sufficiently low frequency measure, while *another* does not, then the first would be accepted but the second would not. (At least not without any additional support for the login attempt.) It is also possible that an authentication above one level gives access to certain resources, whereas an authentication above another level gives access to others. To illustrate the effect of partial matches on security, we plot the security resulting from inputting only two out of the three keywords for the fastwords of our one hundred subjects. In figure 6, we show all combinations of keyword selections herein – in other words, a total of 300 partial matches for our 100 subjects.

7. ENTRY SPEED

In addition to being able to assess the security and recall rates of fastwords relative to traditional passwords, we *also* wish to estimate how long it takes to enter these types of credentials on a typical handset.

This was done in a second user study. We recruited participants to enter three fastwords and three passwords on a device, where the credentials used were drawn at random from the credentials obtained from the study described in section 5. Half of the passwords were what we refer to as "simple" passwords, and the other half were "strong" passwords. This corresponds to passwords the participants in the first study gave as example passwords for social networking sites, and for financial sites. Subjects were told to enter the credentials as fast as they could, and were shown the times taken to enter these.

We recruited total 234 PC users and 45 mobile users from Amazon Mechanical Turk, friends and family, and by tweets. The browser agent was read to determine what type of device each subject used. The resulting timings are shown in figures 7 and 8. While error-correction can be added to desktop text entry using plugins or back-end correction, this was not done herein, and therefore, the timings for Fastwords on desktops are upper estimates.

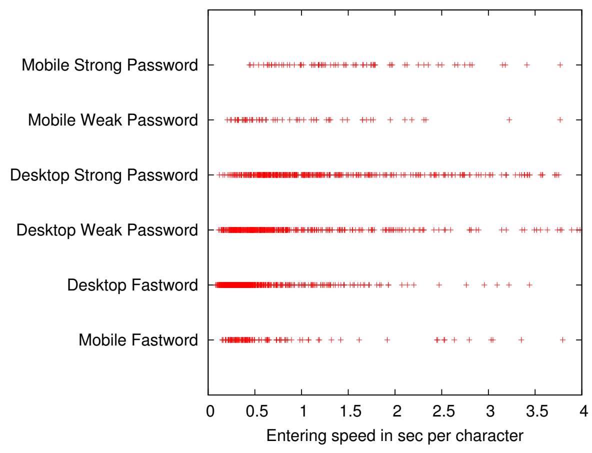

Figure 7: The figure shows a scatterplot of how long participants in the study took to enter passwords and fastwords on handsets and traditional keyboards. All times in seconds. Note that we did not implement error correction for the desktop fastwords in this study; this could easily be done in a real-world application, using a scripting language.

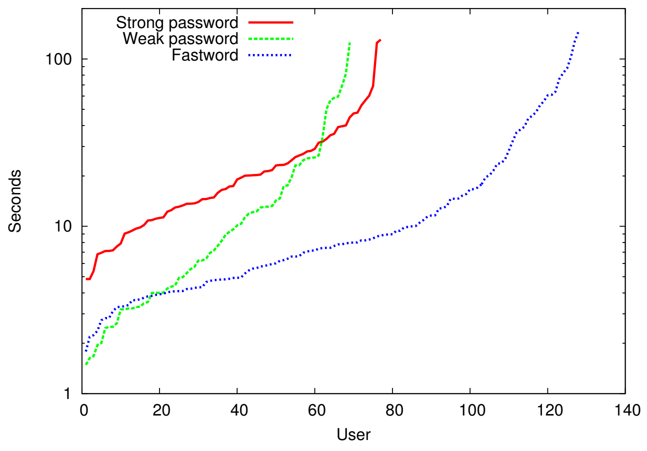

Figure 8: This plot shows the time taken to enter each type of credential (strong password, weak password, and fastword) by different users. The plot is sorted in ascending order. It should be noted that points with the same "User" number do not come from the same user. Fastwords are quicker to enter for most users; however, could be slower for users who do not use their phone to type; these users "peck out" any credential, and the entry time depends to a large extent on the length of the credential.

We note that the subjects entered unknown credentials – therefore, there is no speedup due to the "motoric memory" of having entered the same string time after time. That effect has to be determined through follow-up studies. However, we note that it is reasonable to expect that motoric memory will affect both fastwords and passwords – the extent to which it will reduce the entry times is likely to depend largely on the length of the credential.

Conclusion and Future Work.

We have presented a novel authentication scheme and argued that it offers benefits over traditional passwords in terms of speed of entry; security; and recall. It also enables features not currently available for passwords, such as voice-entry, credential strength checking, and robustness against mistakes.

In this paper, we measured recall rates *a posteriori*. It is an interesting problem how to automatically assess – *at the time of enrollment* – the likelihood that a user will be able to recall a given credential. For example, consider a user who thinks of using a rocket to fly out in to space and plant a flag on the moon. He could either enter a credential "fly space flag" or "rocket moon flag". The latter credential is arguably better from the perspective of recall, given a user who requests a hint. "Fly" could refer to an insect or what a bird does, and "space" could refer to a key on a keyboard or a distance between teeth. However, "rocket" and "moon" are less ambiguous, which makes them better memory joggers. This example suggests that one may be able to build *credential recall estimators* and ask users with credentials that do not seem likely to be remembered to provide another credential. Like many of the features described in this paper, such a feature is made possible by the fact that credentials can be broken down into components that can be processed.

It is also of interest to consider additional features. For example, if a user faces a camera and *mouths* his fastword (i.e., speaks it with no sound), can the device infer the credential from his lip movements with sufficient precision? If that were possible, it would provide us with an elegant way of avoiding attacks based on eavesdropping and shoulder surfing, given that the average person is not very good at lip reading.

We believe the novel structure we have proposed has the potential of offering many new helpful features; we have described some, but believe that there are many more possibilities to be unearthed, given the possibilities offered by being able to identify and process conceptual components of the credential. We hope that our paper will be a first step towards improving authentication, and in particular, authentication on input-constrained devices.

Acknowledgments

Many thanks to Richard Chow, Lucky Green, Bill Leddy, Debin Liu, Alon Nir and Liu Yang for helpful feedback on previous versions of the manuscript, and to Hampus Jakobsson for helping to recruit subjects.

8. REFERENCES

[1] G. Bard. Spelling-error tolerant, order-independent pass-phrases via the Damerau-Levenshtein string-edit distance metric. In *Proceedings of the Fifth Australasian Information Security Workshop (Privacy Enhancing Technologies) (AISW 2007)*, 2007.

[2] W. E. Burr, D. F. Dodson, R. A. Perlner, W. T. Polk, S. Gupta, E. A. Nabbus, C. M. Gutierrez, J. M. Turner, and A. Director. Draft i draft special publication 800-63-1 electronic authentication guideline, 2008.

[3] H. Ebbinghaus. Memory: A contribution to experimental psychology, 1885.

[4] C. Herley, P. C. van Oorschot, and A. S. Patrick. Passwords: If we're so smart, why are we still using them? In *Financial Cryptography*, pages 230–237, 2009.

[5] Imperva. Consumer password worst practices, http://www.imperva.com/docs/wp_consumer_password_worst_practices.pdf.

[6] M. Jakobsson and D. Liu. Bootstrapping mobile PINs using passwords. In *W2SP*, 2011.

[7] M. Jakobsson, E. Shi, P. Golle, and R. Chow. Implicit authentication for mobile devices. In *HotSec'09: Proceedings of the 4th USENIX conference on Hot topics in security*, pages 9–9, Berkeley, CA, USA, 2009. USENIX Association.

[8] C.-M. Karat, C. Halverson, D. Horn, and J. Karat. Patterns of entry and correction in large vocabulary continuous speech recognition systems. In *CHI '99: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 568–575, New York, NY, USA, 1999. ACM.

[9] P.-O. Kristensson and S. Zhai. Relaxing stylus typing precision by geometric pattern matching. In *IUI '05: Proceedings of the 10th international conference on Intelligent user interfaces*, pages 151–158, New York, NY, USA, 2005. ACM.

[10] C. Kuo, S. Romanosky, and L. F. Cranor. Human selection of mnemonic phrase-based passwords. In *Proceedings of the second symposium on Usable privacy and security*, SOUPS '06, pages 67–78, New York, NY, USA, 2006. ACM.

[11] S. Lee and S. Zhai. The performance of touch screen soft buttons. In *CHI '09: Proceedings of the 27th international conference on Human factors in computing systems*, pages 309–318, New York, NY, USA, 2009. ACM.

[12] I. S. MacKenzie and W. Soukoreff. Text entry for mobile computing: Models and methods, theory and practice. In *Human-Computer Interaction*, volume 17, pages 147–198, 2002.

[13] F. Monrose, M. K. Reiter, Q. Li, and S. Wetzel. Cryptographic key generation from voice. In *In Proceeedings of the 2001 IEEE Symposium on Security and Privacy*, pages 12–25, 2001.

[14] M. Rothman and B. Wilson. Authentication death match: Mobility vs. passwords (and why passwords will lose), Webcast, May 17, 2011.

[15] S. Schechter, A. J. B. Brush, and S. Egelman. It's no secret. measuring the security and reliability of authentication via secret questions. In *In Proceedings of IEEE Symposium on Security and Privacy*, pages 375–390, 2009.

[16] S. Schechter, C. Herley, and M. Mitzenmacher. Popularity is everything: A new approach to protecting passwords from statistical-guessing attacks. In *Proceedings of HotSec 2010*, 2010.

[17] B. Schneier. Myspace passwords aren't so dumb, Wired, Dec. (2006).

[18] K. Wang, C. Thrasher, E. Viegas, X. Li, and B. Hsu. An overview of Microsoft web N-gram corpus and applications. In *Proceedings of the NAACL HLT 2010 Demonstration Session*, HLT '10, pages 45–48, Morristown, NJ, USA, 2010. Association for Computational Linguistics.

APPENDIX

A. SAMPLE STORIES AND FREQUENCIES

We are interested in understanding how people choose fastwords. In a separate experiment not reported on in the above, we asked subjects to write down a memorable story and then, from this, select three words that would constitute their credential. We did this to understand how well people can extract the important words from their memorable stories. We list a representative sample of the stories here. For each story, we underline the three keywords (provided by the user) that make up the fastword for that story and provide the associated single-word frequencies in parenthesis. Each story starts with a label that is the 3-gram frequency of fastword and the maximum of the 3-gram frequency and the product of the single-word frequency.

- ($2^{-53.0}$,$2^{-48.8}$) "Back in February I lost my job due to the company downsizing ($2^{-20.1}$) I was worried ($2^{-16.7}$) about how to pay my bills but managed to find a better ($2^{-12.1}$) job in 3 days."
- ($2^{-55.2}$,$2^{-43.2}$) "My mother ($2^{-13.8}$) had a stroke ($2^{-15.7}$) one week before my wedding ($2^{-13.7}$)".
- ($2^{-61.6}$,$2^{-53.4}$) "Up until recently, I always thought penguins ($2^{-17.5}$) were the size of humans ($2^{-15.9}$). I'm in my twenties ($2^{-20.1}$)."
- ($2^{-49.9}$,$2^{-49.9}$) "When I was in college I went to a wedding which didn't happen. The bride ($2^{-16.3}$) was left at the altar. The minister performing the ceremony had to tell the assembled family and friends that there would be no wedding ($2^{-13.7}$). Since all the food was prepared and the hall rented, we had the reception anyway. The jilted ($2^{-22.7}$) bride-to-be walked around wistfully saying : I just wish he'd come back"
- ($2^{-34.9}$,$2^{-32.9}$) "My good friend Shawn *(not a dictionary word; frequency not available)* hung ($2^{-17.2}$) himself 10 years ago this Halloween ($2^{-15.8}$) because he was depressed about a girl he liked who didn't have the same feelings toward him." – *This would not have been accepted as a fastword, since "Shawn" is not a dictionary word.*
- ($2^{-43.5}$,$2^{-43.5}$) "I left my house one day to find my bicycle ($2^{-16.4}$) had been stolen ($2^{-16.5}$). Later, I saw it outside the convenience store, so I waited there for the thief. That's how I met my wife ($2^{-14.3}$)!"

- ($2^{-53.0}$,$2^{-50.4}$) "One day I was driving on the freeway in a major city. I was driving an older model car that only had a lap belt for a seat belt. As I was driving along, a highway patrol pulled up in the lane next to me, pointed at me, and then indicated his shoulder strap. He was motioning me to put on my seat belt. I tried to motion back to him that I was wearing a seat belt by pointing down at my lap. The cop immediately put on his lights and made me pull over. When he came up to my car window, I was puzzled by why I was pulled over. He said in an angry voice, 'You think you're pretty funny?' I then realized: When I had pointed down at my lap, he thought I was making an obscene gesture! He thought I was pointing down at my private parts and was making a crude suggestion. When he realized I was only trying to explain I was wearing a lap safety belt, he burst out laughing and so did I. 'Okay,' he said. 'Get the hell out of here'." – *This subject provided the keywords "cops, driving, goofy". While these keywords are actually not part of the expanded story, that is not a problem – the real system would not ask for the long story, but only for the keywords. The frequencies of the three keywords are (*$2^{-17.2}$*,*$2^{-14.1}$*,and* $2^{-19.2}$*).*

A quick look at these potential fastwords suggest that the way subjects are selecting keywords from the story in a meaningful manner.