NISTIR 8040

Measuring the Usability and Security of Permuted Passwords on Mobile Platforms

Kristen K. Greene
John Kelsey
Joshua M. Franklin

This publication is available free of charge from:
http://dx.doi.org/10.6028/NIST.IR.8040

National Institute of Standards and Technology
U.S. Department of Commerce

NISTIR 8040

Measuring the Usability and Security of Permuted Passwords on Mobile Platforms

Kristen K. Greene
Information Access Division
Information Technology Laboratory

John Kelsey
Computer Security Division
Information Technology Laboratory

Joshua M. Franklin
Applied Cybersecurity Division
Information Technology Laboratory

This publication is available free of charge from:
http://dx.doi.org/10.6028/NIST.IR.8040

April 2016

U.S. Department of Commerce
Penny Pritzker, Secretary

National Institute of Standards and Technology
Willie May, Under Secretary of Commerce for Standards and Technology and Director

National Institute of Standards and Technology Internal Report 8040
65 pages (April 2016)

This publication is available free of charge from:
http://dx.doi.org/10.6028/NIST.IR.8040

Comments on this publication may be submitted to:

National Institute of Standards and Technology
Attn: Information Access Division, Information Technology Laboratory
100 Bureau Drive (Mail Stop 8940) Gaithersburg, MD 20899-8940
Email: nistir8040@nist.gov

Reports on Computer Systems Technology

The Information Technology Laboratory (ITL) at the National Institute of Standards and Technology (NIST) promotes the U.S. economy and public welfare by providing technical leadership for the Nation's measurement and standards infrastructure. ITL develops tests, test methods, reference data, proof of concept implementations, and technical analyses to advance the development and productive use of information technology. ITL's responsibilities include the development of management, administrative, technical, and physical standards and guidelines for the cost-effective security and privacy of other than national security-related information in Federal information systems.

Abstract

Password entry on mobile devices significantly impacts both usability and security, but there is a lack of usable security research in this area, specifically for complex password entry. To address this research gap, we set out to assign strength metrics to passwords for which we already had usability data, in an effort to have a more meaningful comparison between usability and security. This document reports a method of optimizing the input of randomly generated passwords on mobile devices via password permutation to allow for a comparison of password usability data. We found that the number of keystrokes saved—the efficiency gained—via permutation depends on the number of onscreen keyboard changes required in the original password rather than on password length. Additionally, we created and are releasing Python scripts (publicly available from https://github.com/usnistgov/PasswordMetrics) for the experiments on entropy loss we conducted across passwords ranging in length from 5 to 20 characters.

Keywords

authentication; mobile devices; onscreen keyboards; password entry; password generation; password permutation; security-usability balance; text entry; usable security

Acknowledgments

The authors wish to thank their colleagues who reviewed drafts of this report and contributed to its technical content, including Sharon Laskowski, Kerry McKay, Ray Perlner, Andrew Regenscheid, Mary Theofanos, and Meltem Sönmez Turan of NIST. Thank you to Kenneth Thompson and others for assistance with obtaining screenshots of mobile platforms. A very special thanks to Jim Filliben and Andrew Rukhin for their mathematical consultation, and especially to Andrew for his formulas presented in Appendix D.

Audience

This document is intended for those researchers in the usable security field, regardless of whether their background is primarily usability or primarily security.

Trademark Information

Table of Contents

List of Appendices

List of Figures

List of Tables

1 Introduction

Passwords are rarely lauded as an effective authentication mechanism, yet their use is widespread. Since their inception, passwords have been a bane to the individuals using them. Users constantly forget and reset passwords. Organizations attempt to ensure that users' passwords meet minimum complexity requirements and are periodically changed as often as deemed necessary. Building upon these problems, scores of password databases have been exfiltrated from various websites since electronic commerce became commonplace. These leaked password datasets, combined with specialized hardware, provide an optimum environment to make modern password cracking software effective and efficient.

The introduction of mobile computing platforms further complicates these issues, since the traditional problems of authenticating via passwords are transferred to mobile devices. Given the ubiquity of mobile devices and the need to use passwords on these systems, it is critical that we understand their security, usability, and any potential tradeoffs associated with this smaller form factor. While in an ideal world, there would be no tradeoffs between security and usability, in the real world, there often are (with security requirements historically trumping usability considerations, although recognition of the importance of usability is growing).

1.1 Purpose and Scope

This document proposes a measurement method for quantifying the effects on security resulting from optimizing the usability of password entry specifically for constrained input environments, i.e., the mobile touchscreen. Password entry on mobile devices significantly impacts input errors and time to completion, in large part due to device constraints such as smaller keys and lack of tactile feedback (as opposed to a desktop keyboard where it is possible to feel individual keys and thus type purely by touch). While such mobile device constraints impact general text entry tasks as well, there are certain issues unique to the entry of complex, randomly generated passwords due to their inclusion of numbers and symbols, which are located on different onscreen keyboards.

Our larger purpose is twofold: 1) explore the current state of both usability and security metrics applicable to passwords, and 2) discuss our experiences in attempting to use these metrics in a real world situation. Our specific initial goal was to assign strength metrics to passwords for which we already had usability metrics, in an effort to have a more meaningful comparison between the two. Too often, "usable security" research is primarily performed from a single dominant perspective (i.e., focused on security or focused on usability, but rarely both) depending on the background of the researchers. This research attempts to provide more equitable treatment of the usability and security aspects of the problem at hand in an effort to have a true *usable security* project.

1.2 Document Structure

In order to assess both the usability and security of system-generated passwords pre- and post-permutation, it is necessary to first define the metrics and measurement methodology for usability and security. This document begins with a discussion of the keyboard of multiple mobile platforms, exploring what makes them distinct from their desktop counterparts. This is followed by a discussion of usability and security measures and terminology, followed by an explanation of why password permutation would increase usability while decreasing security. We include our methodology to explore the question "how much security is lost as a result of permuting system-generated passwords for mobile devices?" The methodology section includes a practical means of answering the question via Monte Carlo simulations.

The mathematical formulas by which one could derive the true mathematical probability answer are included in Appendix D. In the results section, we present metrics for both the original (non-permuted)

passwords and the permuted passwords. We also present metrics from an entirely new set of randomly generated passwords created with a program developed in-house (all code available at https://github.com/usnistgov/PasswordMetrics). Finally, we conclude with discussion of the larger issues and misconceptions we uncovered during the course of this project.

1.3 Document Conventions

The following conventions are used throughout the Interagency Report:

- All references to NIST Special Publication (SP) 800-63 are references to NIST SP 800-63-2 [9].
- Note that NIST SP 800-63 refers to user selected and randomly selected passwords. However, as this could imply that users are selecting passwords from a list of provided passwords, we instead use the terms *user generated* and *randomly generated* in this document. Within the context of this document, we use the terms randomly generated and system generated interchangeably to describe passwords that were created algorithmically or by other password generation software.
- Unless otherwise noted, all text and figures that refer to iOS keyboards refer to iOS 8. Android keyboards were taken from Android 5.0 (Lollipop). Windows Phone keyboards were taken from Windows Phone 8.1.
- Passwords included in-line with the text are italicized and are not offset with double quotation marks (“ ”). The authors believed that using quotation marks would have caused confusion since it is possible for quotation marks to be included as part of a password.

2 Exploring Mobile Platforms

2.1 Literature Review

In our experience, there is a misconception that both sides of the usable security field can easily acquire data, assign metrics, and analyze the data. This work should assist individuals of the usable security community in understanding the challenges both sides face. For instance, some may believe that assigning security metrics to passwords is a solved problem, when in reality there is a menagerie of open research questions.

Although we set out simply to measure the loss in security versus the gain in usability for passwords that were permuted to be more "mobile device friendly," we believe we achieved much more than our initial measurement goal. By having usability and security experts working so closely together on this project, we uncovered several important—and likely widespread—misunderstandings experts from one field initially held about the other field; we identified core concepts and vocabulary that are fundamental to each field's measurement methodology, yet were initially foreign or only vaguely understood by the other field. We hope this report can serve as a type of primer or refresher for each field to learn more about the other, in an effort to foster more informed dialogue and facilitate better collaboration between experts with differing backgrounds.

We permuted passwords from prior mobile usability research [1] and calculated the theoretical entropy lost as a result of grouping character classes (i.e., uppercase, lowercase, numbers, symbols) together within a password. By rearranging/grouping the password contents in this way, it reduces the number of keystrokes required to enter the password (see Table 2), since the user does not have to continually switch back and forth between three different onscreen keyboards (see Figure 2). We know unequivocally that for at least one facet of usability (efficiency), the permuted passwords are better. On the other hand, this restructuring of password contents obviously decreases security by adding predictable structure: in the permuted passwords, uppercase letters are always first, followed by lowercase letters, numbers, and finally symbols.

We argue that only by empirically quantifying the security-usability tradeoff can we hope to measure and understand effects of changing passwords along either or both dimensions. Although alternative—and arguably better—mobile authentication mechanisms exist [2], the unfortunate reality is that passwords are too deeply ingrained in our current digital world to be fully replaced in the near term. In the longer term, research efforts such as the National Strategy for Trusted Identities in Cyberspace (NSTIC) aim to ultimately replace passwords as a primary authentication mechanism [3]. In the interim, however, our work is focused on evaluating ways to improve password usability for mobile devices without an unacceptably large sacrifice to security.

2.2 Background

The stimuli we used for this effort were taken from a set of passwords reported in a recent behavioral study on mobile password entry for complex, system-generated passwords [1], which was a replication of a desktop password entry study [4]. In both studies, participants had to learn and enter 10 complex, system-generated passwords 10 times each, then complete a surprise recall test; their entry times, error frequencies, and recall failures were recorded. The 10 passwords used in those studies ranged in length from 6 to 14 characters, and were generated using the *Advanced Password Generator* from BinaryMark[1] with the following password policy rules: must consist of at least one upper-case character, one lower

[1] http://www.binarymark.com/Products/PasswordGenerator/default.aspx

case character, one number, and one special character, cannot begin with an upper-case character, nor end with an exclamation mark. In addition to the 10 passwords (see Table 2) from previous studies, we also used new randomly generated passwords via a program developed in-house (see Section 5.1 for details).

Prior work [1] suggested that the combination of complex password requirements and mobile keyboard constraints seems to result in emergent cognitive costs associated with the interruptive nature of transitioning back and forth between multiple onscreen keyboards. This implies that rearranging complex passwords to minimize the number of onscreen keyboard changes required should increase their usability on mobile devices, but at what cost to security?

The work presented in [1] was focused on iOS devices only (using an iPhone 4S and iPad II both running iOS 6), and did not compare alternate mobile platforms such as Android or Windows, nor did it include a direct (i.e., within-subjects) comparison with desktop computers. Such comparisons may ultimately be necessary to better understand the interaction of password policy and device-specific constraints, and to understand the ramifications of using the same policy in vastly different computing environments (mobile vs. traditional desktop).

In sharp contrast to the standard desktop QWERTY keyboard layout [18], there is no *de facto* onscreen keyboard for mobile touchscreen devices.[2] While a standard onscreen keyboard layout may not be necessary, the variability in onscreen keyboard sizes, layouts, and functionality complicates mobile password research, at least where randomly generated passwords are concerned. Onscreen keyboards vary widely between manufacturers and operating systems, but may also vary between devices with the same operating system. For instance, there are symbols, such as , . ? ! available on the primary iPad alphabetic keyboard that are unavailable on the corresponding iPhone alphabetic keyboard in Figure 1.[3]

[2] Onscreen keyboards can differ by phone model, manufacturer, and operating system.
[3] Note: We do not discuss third-party keyboards or built-in sliding keyboards.

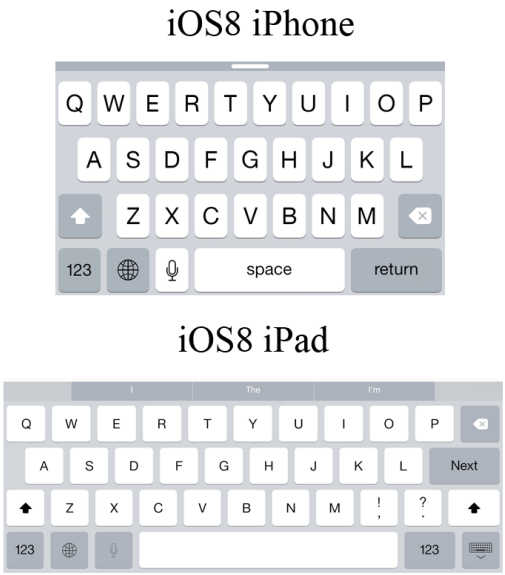

Figure 1 - Demonstrating the differences between mobile keyboards of the same OS

Regardless of specific device, there are some very high-level commonalities between onscreen keyboards as shown in Figure 2. While exact layouts differ, there is some basic similarity in the sense that they follow a core pattern of three keyboards—roughly containing letters, numbers, and symbols—that users must navigate between, as only one keyboard is displayed at a time.

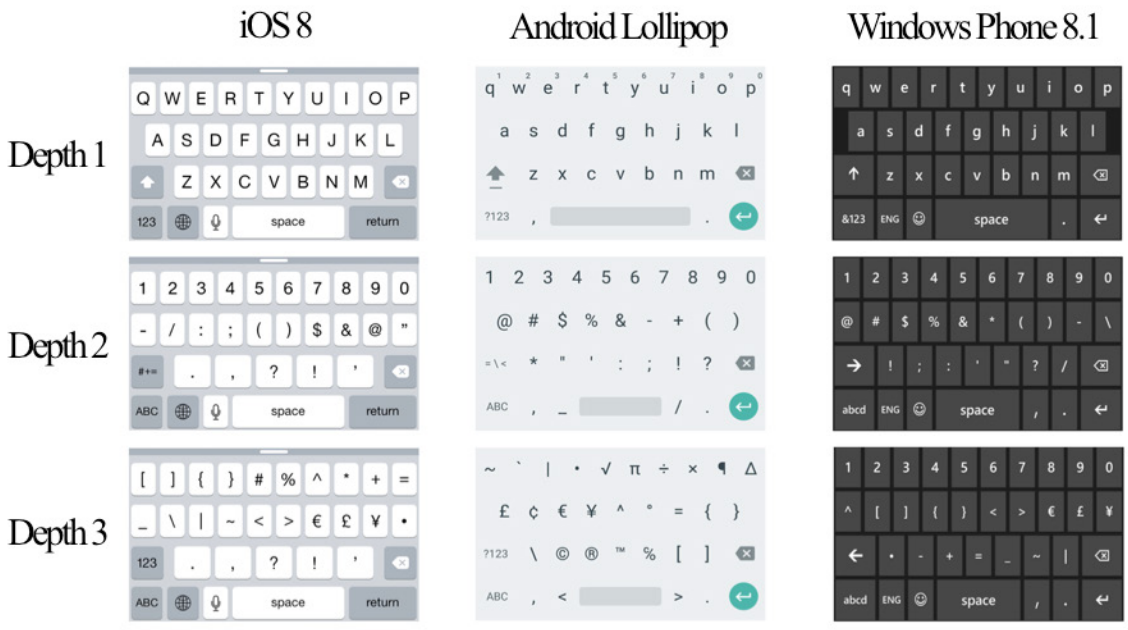

Figure 2 - Keyboard screen depths on modern mobile operating systems

Another high-level similarity is the inclusion of press-and-hold functionality, shown in Figure 3, where a sustained keypress brings up additional character options. Although this press-and-hold key option may not be obvious to users, the issue of hidden functionality is not unique to onscreen keyboards, or even mobile devices—hidden functionality is an issue in desktop computing as well. Note that on the Android keyboard, the press-and-hold functionality for the top row of letters (q, w, e, r, t, y, u, i, o, p) allows one to access numbers (1, 2, 3, 4, 5, 6, 7, 8, 9, 0). There is even a visible indication of those keys' functionality—note the small numbers in the upper right corner of the aforementioned Android letter keys.

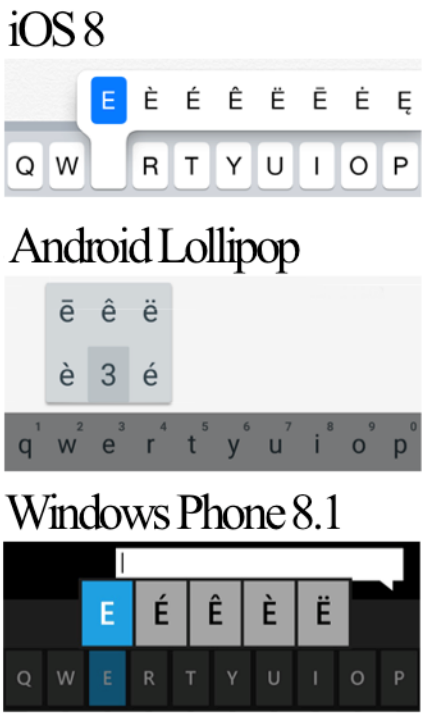

Figure 3 - Press-and-hold functionality on mobile keyboards

Although they are not actual letter keys, there are two keys on the iPad letter keyboard that do have visible indicators of their alternate functionality. These are the previously mentioned comma/exclamation mark key, and the period/question mark key, but note that the secondary characters are not accessed via press-and-hold, but rather by a single tap on the shift key (see Figure 1).

3 Password Usability Background

Note to readers: Although this section may be a review of core concepts already well understood by usability practitioners, many of the concepts described below are new for those usable security researchers coming from the field of computer security.

In this section we first provide an overview of the standard definition of usability and its components. This is followed by more detailed discussion of each of the facets of usability, as well as examples of usability measurement specifically as it pertains to passwords.

While there is an international standard for the definition of product usability [5], there is no corresponding standard definition of password usability; we are by no means suggesting that there should be one, for the standard definition of usability suffices as-is. Although the traditional definition of usability was developed in the context of product evaluation and testing, its wider applicability and adaptability should be clear from the following section(s).

Usability is defined as "the extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use. [5]" Effectiveness, efficiency, and satisfaction are measurable attributes that combine to form the larger construct of usability. The former measures (effectiveness and efficiency) are the objective usability metrics, while the latter (satisfaction) is more subjective; this objective/subjective distinction is often referred to as "preference versus performance," where preference refers to satisfaction, and performance refers to effectiveness and efficiency. All three—effectiveness, efficiency, and satisfaction—have well-defined general methods of measurement, which can be tailored specifically for measuring usability in a particular task domain, such as for password entry tasks.

Finally, it should be noted that objective and subjective usability are not always correlated with one another; there can be disagreement between performance versus preference [6].

As defined in ISO 9241 [5]:

> **Usability**: The extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use.
>
> **Effectiveness**: Accuracy and completeness with which users achieve specified goals.
>
> **Efficiency**: Resources expended in relation to the accuracy and completeness with which users achieve goals.
>
> **Satisfaction**: Freedom from discomfort, and positive attitudes towards the use of the product.
>
> **Context of use**: Users, tasks, equipment (hardware, software and materials), and the physical and social environments in which a product is used.
>
> **User**: Person who interacts with the product.
>
> **Goal**: Intended outcome.
>
> **Task**: Activities required to achieve a goal. Note that activities can be physical or cognitive.

> **Product**: Part of the equipment (hardware, software and materials) for which usability is to be specified or evaluated.

Mapping measures of effectiveness, efficiency, and satisfaction to password entry tasks is straightforward with a solid understanding of general usability. Effectiveness can be measured as the number (and class) of errors a user made during password entry, while efficiency can be measured as the time taken by a user to enter a password. Satisfaction can be measured via post-task questionnaires.

As with many dependent variables, usability can be measured at different levels of granularity, some of which can only be captured in the laboratory setting. Offering the capability for such granular measurement is an important advantage and contribution of conducting laboratory experiments; some of the measures described in subsequent sections would be impossible to capture outside of the laboratory. The following focuses heavily on measures of effectiveness since errors are arguably most important in terms of password entry; while it is common for users to be locked out of their accounts due to too many erroneous login attempts, it is relatively rare for a user to be locked out for typing too slowly.

In 1999, Adams and Sasse [7] identified usability characteristics that users desire of passwords: easy to remember, able to be used across multiple systems, and rarely change. However, these are usability characteristics at a higher level, and not usability metrics at the level of granularity appropriate for measuring usability of passwords in the context of individual password entry tasks (i.e., password typing tasks) per se, which is the focus of the current work. The following subsections describe how usability can be measured at different levels of granularity specifically in the context of password entry tasks.

3.1 Effectiveness

> **Effectiveness**: Accuracy and completeness with which users achieve specified goals. [5]

In general usability research, effectiveness is typically measured via error rates, which can be captured at a high level (e.g., overall rates of task completion/success versus failure), intermediate level (e.g., per subtask errors), and/or very detailed low-level (e.g., per keystroke or per mouseclick errors). Beyond simply counting the number of errors made relative to the number of opportunities for error, it is also critical to identify the specific nature of the error, and if possible, its root cause.

For password entry specifically, effectiveness can be measured in multiple ways, at the password level and at the character level. At the password level, effectiveness can be measured via per-password login failure rates. At the more granular character level, effectiveness can be measured via per-keystroke errors.

Measuring only overall login failure rates is most analogous to the real world (when as long as a password contains at least one error, login fails), yet this level of measurement ignores rich and important information regarding the cause behind a login failure. Measuring effectiveness at the password level as only a binary success/failure would ignore both the number and nature of errors committed during password entry; it would also not capture any corrective actions taken by a user.

As both the frequency and nature of errors can differ greatly by device, it is much more informative to also measure effectiveness at the individual character level by capturing per-keystroke errors. Categorizing the nature of the error at an individual character level has been done in text entry research for decades. Although password entry does differ from traditional transcription typing tasks (e.g., where the use of numbers and symbols is rare) commonly used in text entry research, it still makes sense to use the error classification from that field. Not only do passwords—specifically user generated ones—often contain mostly letters, but the traditional text entry error categories (e.g., transposition, omission) largely apply regardless of whether the characters are letters, numbers, or symbols.

Ref. [1] categorized errors at this level based on the common classes of errors used in text entry research as well as the frequency of certain errors found in that particular experiment. If duplicate or additional characters were entered into a password field, they were categorized as Extra Character errors. If characters were omitted from entry, they were categorized as Missing Character errors. There were four classes of substitution errors: substitution of the correct character with an Incorrectly Shifted character, with a Wrong Character, or with an Adjacent Key character (i.e., a character adjacent to it on the keyboard), and finally, substitution of the letter "o" for the number zero and vice versa (although this particular error could also be categorized as a Wrong Character error, its high frequency of occurrence warranted its own category).

Why is it important to capture such granular error data? Why bother to categorize individual character errors in this manner? Only by understanding the nature of the errors can we determine exactly how they relate to device constraints versus password characteristics. For example, the frequency of adjacent key errors was significantly higher for the smartphone than the tablet in the Greene et al. (2014) study [1], obviously due to device constraints given that the onscreen keys are much smaller targets for an iPhone (especially in portrait orientation) than for an iPad. On the other hand, the propensity for users to substitute the letter "o" for the number zero and vice versa is related to characteristics of the password rather than constraints of the device per se, as this error was common both in mobile and desktop password typing studies ([1] and [4], respectively).

3.2 Efficiency

> **Efficiency**: Resources expended in relation to the accuracy and completeness with which users achieve goals. [5]

Efficiency is generally measured via time on task. As with measures of effectiveness, efficiency measures can also be captured at different levels of granularity, from overall task completion times, to subtask times, and all the way down to individual keystroke times. Similarly, efficiency for password entry tasks can be measured at the password level (time to type the entire password) and at the character level (time to type an individual character).

In laboratory experiments where participants are assigned passwords and must memorize and practice them, it is possible to distinguish efficiency during the initial learning phase (i.e., time to initially memorize a password and practice typing it a specified number of times to meet a certain performance criterion) from efficiency during subsequent entry tasks (i.e., time to type a well-practiced password). By measuring differences in password learning efficiency compared with differences in password entry efficiency, one could investigate questions such as whether there are passwords that are initially more time-consuming to learn, but faster to enter in the long run, and vice versa.

Efficiency does not exist in a vacuum however, as it is greatly influenced by effectiveness. If users are making numerous errors that require correction, they will obviously be slower during password entry than if they were error-free. That is why when discussing efficiency and effectiveness, researchers often talk about a speed-accuracy tradeoff function. In terms of mobile password entry, some users may have a more aggressive speed-accuracy tradeoff function, preferring to type more quickly at the cost of higher probability of errors. Conversely, other users may have a more conservative speed-accuracy tradeoff function, preferring to type more slowly and cautiously in order to reduce their probability of committing errors. Given the higher cost of errors in password entry tasks (e.g., having to retype the entire password and/or deal with account lockouts) relative to regular text entry tasks, it is quite possible that some users even change their normal text entry strategies to be more conservative during password entry on mobile devices. This may be especially likely for users that normally rely heavily upon features like autocorrect and autocomplete during regular text entry.

3.3 Satisfaction

Satisfaction: Freedom from discomfort, and positive attitudes towards the use of the product. [5]

Although user satisfaction itself is a subjective construct, there are nonetheless standardized questionnaires (also called instruments) and objective ways to measure it—indeed, there are entire subfields of research devoted to questionnaire design and measurement theory. In addition to standardized instruments for measuring user satisfaction, it is possible for researchers to create their own customized questionnaires; the following section discusses each in turn.

There are numerous standardized usability questionnaires with which to assess user satisfaction in a rigorous, repeatable manner. They utilize predefined questions in a specific order and format, with specific scoring rules based on a user's responses to the questions. Furthermore, standardized questionnaires have undergone psychometric testing to evaluate their reliability, validity, and sensitivity. Psychometrics is a field of study surrounding psychological measurement; it involves the objective measurement of various human capabilities and characteristics (e.g., knowledge, abilities, personalities, attitudes), as well as statistical research on measurement theory. In addition to variability in their psychometric properties, these standardized instruments vary in length and cost (e.g., some are free and others must be purchased for use).

As with effectiveness and efficiency, satisfaction can also be measured at different levels of granularity. There are both post-task questionnaires[4] and post-study questionnaires.[5] There are general usability questionnaires appropriate for almost any product, as well as more targeted questionnaires,[6] such as those designed specifically for website evaluation.[7] Whereas the aforementioned questionnaires are (largely speaking) appropriate for usability testing with a wide variety of products and scenarios, their direct applicability to passwords is less clear.

As in other areas of usability research, it is common for password usability researchers to create their own customized experiment-specific post-task and/or post-study questionnaires. However, knowing the exact psychometric properties of such customized questionnaires is difficult since they have not been used and validated by the larger usability research community. Given the numerous—and sometimes subtle—considerations for a well-constructed questionnaire, a detailed discussion of questionnaire design principles and psychometric theory is out of scope for the current paper. However, a brief description and example of one of the more common satisfaction questionnaire items (Likert[8] item) is appropriate.

Likert scales and Likert items are commonly used in questionnaire research; although often used interchangeably, there are technically distinctions between a true Likert scale, an individual Likert item, and the rating scale presentation format. A scale comes from grouped responses to a set of individual items. An individual Likert item is often a statement where a respondent indicates their level of disagreement/agreement on a symmetric disagree/agree scale.

---

[4] Examples of post-task questionnaires include the ASQ (After-Scenario Questionnaire), SEQ (Single Ease Question), SMEQ (Subjective Mental Effort Questionnaire), ER (Expectation Ratings), and UME (Usability Magnitude Estimation).

[5] Examples of post-study questionnaires include the QUIS (Questionnaire for User Interface Satisfaction), SUMI (Software Usability Measurement Inventory), PSSUQ (Post-Study System Usability Questionnaire), SUS (System Usability Scale), and UMUX and UMUX-LITE (Usability Metric for User Experience).

[6] Examples of more targeted questionnaires include the CSUQ (Computer System Usability Questionnaire) and HQ (Hedonic Quality).

[7] Examples include the WAMMI (Website Analysis and Measurement Inventory) and SUPR-Q (Standardized Universal Percentile Rank Questionnaire).

[8] Named after psychologist Rensis Likert, developer of the scale. Likert is pronounced "lick-ert".

For example, the statement might be "Password entry on mobile devices is too onerous with current password requirements." For a five-level Likert item the response options might then be: strongly disagree, disagree, neither agree nor disagree, agree, strongly agree. The rating scale presentation format would then typically consist of a horizontal line with five equidistant markers (each labeled with its corresponding response option), and the respondent would then circle or draw an X next to their desired response. Note that this is but a single example question; if designing a questionnaire, one would want to balance negatively phrased and positively phrased items to help reduce bias. Another example might be "Rate the difficulty of memorizing this password" or "Rate the difficulty of typing this password," with five response options then ranging from very easy to very hard for each question.

It is possible to use different levels of response options (e.g., five versus seven), and it is also possible to use an even-point scale where the neutral middle option is removed. It is common for scales to have an equal number of positively keyed and negatively keyed statements in order to reduce bias. Regardless of whether a traditional Likert item or scale is used, careful phrasing of questions and response options is absolutely critical in all cases to reduce bias.

3.4 Across Effectiveness, Efficiency, and Satisfaction Measures

> **Usability**: The extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use. [5]
>
> **User**: Person who interacts with the product. [5]
>
> **Goal**: Intended outcome. [5]
>
> **Context of use**: Users, tasks, equipment (hardware, software and materials), and the physical and social environments in which a product is used. [5]

There is much more to the ISO 9241 definition of usability beyond simply the words "effectiveness, efficiency, and satisfaction." The surrounding words "specified users," "specified goals," and "specified context of use" are all extremely important, both to the field of usability in general and usable security in particular. Here we briefly consider each of those three phrases in turn as they relate to various aspects of usable security research.

Specified users: while password research traditionally focuses on end users (as does our current work), one could—and we argue, should—conduct research with other specified user populations, such as system administrators and IT support staff.

Specified goals: goals will change depending on both the user and the context of use; these concepts can influence one another. Is the goal of password creation/assignment to maximize security or memorability, for a single password or across a password portfolio? [8]

Specified context of use: authentication usability may change significantly depending on the current context of use. Imagine a means of mobile authentication (e.g., simple pin entry) that is relatively easy for users in an office setting; that same means of authentication would not only be unusable, but actually impossible, for gloved firefighters in an emergency response situation, for gloved soldiers in combat situations, etc.

3.5 The Science of Measuring Usability

Usability can be measured in a rigorous, reliable, and repeatable manner via metrics and measurement methodologies that have been well-established and used in research across a variety of topic areas. It should be emphasized that there are rigorous methods for both qualitative research and quantitative research, for formative and summative usability testing, etc. Regardless of whether it is a smaller qualitative study (e.g., n=5) or a much larger quantitative study (e.g., n=5000), there are fundamental concepts and research methods from the behavioral sciences that are already appropriate for—or easily adapted to—conducting usable security research. Many usability studies are, in fact, cognitive psychology experiments that make use of inferential statistics and experimental design principles common to other research fields; they utilize control variables, manipulate experimental variables, use randomization and counterbalancing to control for order effects, measure power and sensitivity, examine main effects and interactions between variables, etc.

This is not to suggest that all usability research is conducted in this manner (nor should it be, as different study designs and sample sizes are more appropriate to answer different research questions), nor is this to suggest that all usability researchers have the background and training to conduct such studies (just as there are many security specializations, so too are there many specializations in usability and human factors), but merely to give an example of experimental methodology with which other fields may already be more familiar. The final point is this: no matter the sample size, it is both possible and necessary to conduct human subjects research with rigor. We urge both usability and security practitioners to examine each other’s work through the lens of experimental design and be critical consumers of others' research; many of the same principles apply regardless of whether one is measuring human behavior, system behavior, or combined human-system performance.

4 Password Security Background

Note to readers: Although this section may be a review of core concepts already well understood by individuals from the field of computer security, many of the concepts described below are new for those usable security researchers coming from the usability field.

Broadly speaking, passwords are used as a means of confirming an identity in order to access protected data or information systems, but are also used to derive cryptographic keys used for encrypting individual files or whole drives. Passwords play a part in the authentication process, often divided into the identification, authentication, and authorization of individuals.[9] Identification is the process of making an identity claim, such as *"I am Alice."* Authentication is the process of establishing confidence in an identity by providing evidence via a token[10] (e.g., password, PIN, smartcard, biometric). Finally, authorization is the act of granting privileges to the person or entity associated with the proven identity, once they have already been authenticated. Therefore, it could be stated that the focus of this work is authentication via a specific type of token, the password, being used in a mobile environment.

The authentication process uses identities, credentials, and tokens to provide assurance in a person or entity's identity claims. Simple authentication schemes involve two parties: an entity asserting an identity claim (the claimant) and an entity verifying that the claim is accurate (the verifier). Modern authentication schemes typically use cryptographic protocols to achieve this end. The tokens used to provide assurance in an entity's identity are categorized as follows:

- *Something you know*: Passwords and PINs are common examples,
- *Something you have*: Such as a driver's license or a cryptographic key, and
- *Something you are*: An iris, fingerprint, or other biometric data.

Passwords are *Something you know* and are further classified by NIST's guidance on electronic authentication in NIST SP 800-63 as *memorized secret tokens* [9]. Memorized secret tokens are defined as "A secret shared between the user and the party issuing credentials. Memorized Secret Tokens are typically character strings (e.g., passwords and passphrases) or numerical strings (e.g., PINs)." Memorized secret tokens are passwords, passphrases, passcodes, and PINs, although industry standard definitions do not exist for each of these tokens. In general, passphrases are groups of words concatenated together to create short sentences or sayings. PINs are composed of numbers and are typically shorter, only 4-6 characters, than passwords. Finally, passcodes may be longer than PINs, and although they typically contain only numbers, complex passcodes may also be alphanumeric.

4.1 Password Generation

When discussing passwords, one overarching distinction exists: passwords generated by a user, and passwords generated by a computer program. Password generation has a significant impact on the usability and security of passwords and may be performed by a user or by a computer program. User

[9] It is possible to authenticate non-person entities such as resource, process, and information.

[10] NIST SP 800-63 defines token: Something that the Claimant possesses and controls (typically a cryptographic module or password) that is used to authenticate the Claimant's identity.

generated passwords are often requested from a user when they create an account. Randomly generated passwords are often created in conjunction with a pseudo-random number generator, although this is not the only method. For example, published schemes exist for generating random passwords using lists of words and dice [10]. The following describes the characteristics of user and randomly generated passwords.

Randomly generated passwords are more complex than user generated passwords and are generally considered to offer a higher level of security. Randomly generated passwords are generally more difficult for users to remember and input than user generated passwords while user generated password are often easier to remember and input, but the security of the password is more difficult to measure. While there are many ways to measure the security of user generated passwords [9] [11], the field of computer security lacks a universally agreed upon measurement standard with sufficient evidence to prove the merit of the standard.

When users create passwords, it is common for the password to be compared against a word list to ensure it does not contain common words, which are likely to be guessed by an attacker. These wordlists are referred to as dictionaries. If the user generated password contains a word within the dictionary, the password may be rejected and the user asked to generate a different password. In addition to lists of words, user generated passwords can be compared against lists of known passwords. These password lists, also known as password sets or password dictionaries, contain passwords previously obtained from users. It is common for these password lists to be illegally obtained by malicious entities via attacking websites and online databases. Although ethical considerations exist when using password lists, these password lists have been used throughout academic literature for research purposes.

4.2 Password Usage

Authentication protocols use passwords to provide assurance to a verifying party. Strong authentication protocols are cryptographic[11] in nature and exchange messages between the verifier and claimant in an effort to assist the verifier in arriving at an authentication decision. A myriad of authentication protocols exist, each using passwords in different ways and offering varying levels of assurance in an individual's identity.[12] These protocols may provide services beyond authentication, such as cryptographic key generation and key exchange.

Two common ways of using passwords are to access a web application or operating system. The authentication protocols used to access these systems are quite different from each other. These are not the sole methods of using passwords, but instead encompass a large majority of use cases.

When authenticating to a web application, registration is generally required to obtain a password. Once a password is either chosen by, or assigned to, a user, it must be stored. The user's plaintext password is concatenated with random data referred to as a salt. The password and salt are then cryptographically hashed using a cryptographic hashing algorithm (e.g., SHA-1, SHA-256) on the server. For instance, the password dragonfootball hashes to the value 33fe8ce02a88f2168bfdec233dcbda366e61aba2. When the password and salt are hashed together there is a different result: SHA-1(dragonfootball + &D6HTpU`r2Y5yai+xLkp;1Y!M=!u~3#0Fde^Q|q) = 7dbc12b064306bf61e5fe1af8fb00671d7ffd88c.

Only the final hash and salt are stored within the server's database—the plaintext password should never

[11] Cryptography: The discipline that embodies principles, means and methods for providing information security, including confidentiality, data integrity, non- repudiation, and authenticity.

[12] Examples of authentication protocols include Kerberos, PEAP (Protected Extensible Authentication Protocol), and TACACS+ (Terminal Access Controller Access-Control System Plus).

be stored in the database. When a user needs to authenticate, they type and submit their password, which is transmitted to the verifier, often in an encrypted format. The verifier decrypts the password and queries the database for the salt and hash. The password and salt the system receives from the user are concatenated, hashed, and the output is compared with the stored hash. If they match, the user is authenticated to the system.

This authentication method implies that web-based systems must store authentication-related information (e.g., usernames, salts, salted hashes of passwords) that allows the system to check whether the user's password is correct. An attacker who compromises a web application is in a much better position to obtain a specific user's password, but systems that store authentication information in cleartext are vulnerable to having all of their users' authentication information (e.g., usernames, passwords) exposed. This type of security breach is considered quite serious for a user or group of users, since the attackers can now try various username/password couplets on the compromised web application and other systems on the internet.

Passwords used in an operating system context use drastically different backend authentication systems. While web based authentication requires a network, authenticating to an operating system does not necessarily require a network connection depending on how the system and the enterprise are configured. When locally authenticating to general purpose operating systems (e.g., Windows, OS X), the method(s) by which passwords are used and stored are different based on the operating system, and are often version specific (e.g., Windows and *nix variants). In Unix and Linux variants, hashed passwords are often stored in a file called a *password file*, which often has corresponding operating system level logical access control mechanisms (e.g., file permissions) in place to prevent the reading and writing of this file. The passwords of many users are often stored in the same password file alongside their associated username.

4.3 Classes of Attacks on Passwords

There are a variety of ways of attacking passwords and we will discuss the following classes of attacks:

- Password guessing:
  - Brute force,
  - Intelligent guessing;
- Eavesdropping;
- Social Engineering; and
- Physical attacks.

One of the largest classes of attacks is password guessing, in which unauthorized individuals attempt to guess the password. One type of password guessing attack is a brute force attack, which is an attempt to exhaustively guess all possible passwords to gain access to a resource. This attack begins with simple passwords and attempts more complex passwords over time. Dictionary attacks are another type of password guessing attack, which draw passwords from a list of words (i.e., a dictionary) and try to authenticate via these words. This attack is effective since users often use simple dictionary words as their password. Dictionary attacks can be made more intelligent by concatenating additional dictionary words or by using character substitution. By concatenating the dictionary words *dragon* and *football* together the password *dragonfootball* is generated. Character substitution can build upon this tactic by replacing the characters of a password with common alternatives, such is the case when the word *dragonfootball* is restyled as *Dr@g0nFo07Ba1l*.

It's useful to note that the environment in which password guessing attacks are conducted plays a large role in defending these systems. If an attacker is actively trying to guess a password against a live

authentication system in real-time, this is referred to as an *online attack*. A common defense mechanism for online attacks is restricting the number or speed of password guesses, which renders these types of attacks impractical. One can also “lock out" an account after some predetermined number of incorrect guesses, and an administrator is commonly required to reset the account. A related defense mechanism is referred to as “exponential backoff,” where every incorrect authentication attempt exponentially increases the duration of time required before the account can be used again.

When systems used for performing authentication are compromised, it is commonplace for an attacker to retrieve password files or databases. If the attacker has a database or file of password hashes, they typically have the opportunity to guess as many passwords as they would like (i.e., an offline attack) for as long as they would like. They are only limited by the speed of their guesses and the amount of time and other resources they wish to invest in password guessing activities.

A useful model for thinking about password security is to consider a password’s resistance to offline attacks. For online systems, this includes assuming that a system’s password file may be compromised at some point in the future, which helps to give a useful "worst case" estimate of password security. However, an online system in which the password file isn't compromised will provide users with an enormously better level of security. Security against offline attacks is based on the technical measures of how the password file or password-based key derivation is done, and on the quality of the password used.

Eavesdropping attacks occur when a user is monitoring the traffic on a network and is able to obtain the password via intercepting network traffic. This is mitigated by establishing a protected connection, such as using Secure Sockets Layer (SSL) or Transport Layer Security (TLS) between two communicating parties.

Social engineering is defined as "the act of deceiving an individual into revealing sensitive information by associating with the individual to gain confidence and trust." Social engineering attacks, such as phishing, completely bypass well implemented authentication systems.

Physical attacks, such as installing a hardware key logger on an information system, are all ways of circumventing even well-implemented password-based authentication schemes and protocols. Shoulder surfing occurs when an attacker is physically present can view a user's keyboard while a password is being typed without their knowledge. This can occur on desktops and may be easier or more common on a mobile device with keyboards engineered for easy viewing. Automated shoulder surfing can also occur by using video recording devices to watch password entry, and use computer vision algorithms to identify the password after the fact [12].

4.4 Password Strength Metrics

Methods of measuring password strength have been devised and applied to passwords for some time. In general, a strength estimate can be applied to either user generated passwords or randomly generated passwords. In this section we will explore a number of password strength metrics.

Password entropy is one of the most common ways of measuring password strength. NIST SP 800-63 notes that the term password entropy is "at most only loosely related to the use of the term in thermodynamics" [9]. Password entropy was originally suggested by Claude Shannon in 1948 [13]. These types of entropy estimates can only be performed for randomly generated passwords, and user generated passwords cannot be measured in this manner. Calculating strength for user generated passwords is somewhat different and considered a much more difficult task. NIST SP 800-63 provides two methods for

estimating the strength of user selected (a.k.a., user generated) passwords: guessing entropy and min-entropy.

It's important to note that password entropy is only useful in password guessing attacks. Entropy is not an effective metric for determining resistance against keylogging and phishing attacks. Higher entropy passwords may be slightly more secure against shoulder surfing attacks as longer and more complex passwords are often more difficult to capture by the human eye from a distance. However, as technology advances, even these longer passwords may not be secure against automated shoulder surfing tools and new computer vision algorithms [12].

Shannon Entropy
Applies to: Randomly Generated Passwords

In NIST SP 800-63, Shannon entropy is expressed as $H = \log_2(b^L)$, where H is entropy, b is the number of characters available to be used for the password (e.g., numbers, letters, special characters), and L is the number of characters in the password [9]. For instance, to measure the strength of the initial 10 passwords used within our study, we used Shannon entropy where we assumed a 94-character keyboard (b) and (L) was the number of characters in the password. Our original 10 entropy measurements are shown in Table 1.

Table 1 - Examples of Shannon Entropy

Password	Password Length	Entropy Estimate
5c2'Qe	6	39.32753311
3.bH1o	6	39.32753311
a7t?C2#	7	45.88212196
m3)61fHw	8	52.43671081
p4d46*3TxY	10	65.54588852
q80<U/C2mv	10	65.54588852
d51)u4;X3wrf	12	78.65506622
6n04%Ei'Hm3V	12	78.65506622
m#o)fp^2aRf207	14	91.76424392
4i_55fQ$2Mnh30	14	91.76424392

Although useful in some scenarios, Shannon entropy is inappropriate for our purposes because it measures the average storage and transmission requirements of the password, not the difficulty in guessing the password.

Using Password Guessing Attacks as a Strength Indicator

Suppose we start out knowing the probability distribution from which a user's password is drawn. For example, we know that:

- The password *GU3$Sme* has probability 0.1,
- The password *$ecretC0de* has probability 0.08,
- The password *123456* has probability 0.07, and so on.

Given that knowledge, one could create a list of the probabilities of each password, in descending order: $d[1]$, $d[2]$, $d[3]$, ... so that $d[1] \geq d[2] \geq ... \geq d[n]$.

Now, imagine an attacker also knows the distribution from which the passwords are drawn, and wants to guess the password. Each guess of the password takes some resources, such as time on a computer, and the attacker would like to minimize the resources spent. One obvious strategy is to try the passwords in descending order of probability: first attempt *GU3$Sme*, then *$ecretC0de*, then *123456*, and so on, until at the end they begin guessing unlikely passwords like *ckJahgU33#ffz12-^:*.

To understand how secure a user's password is against this kind of attack, one needs to know how likely it is to have guessed a user's password after each number of guesses. This is the cumulative distribution of all passwords, which can be notated with uppercase *D* like this:

$$D[1] = d[1]$$

$$D[2] = d[1] + d[2]$$

$$D[3] = d[1] + d[2] + d[3]$$

...

$$D[k] = d[1] + d[2] + ... + d[k] = D[k\text{-}1] + d[k].$$

If we graph the cumulative distribution of password guessing probabilities, we have what John Pliam called the "work function": the relationship between the number of guesses the attacker makes and his probability of success [14]. How can we use our knowledge of these probability distributions (e.g., *d, D*) to analyze the user's password security? There are three answers worth thinking about: Guessing entropy, min-entropy, and $D[w]$ where w = the maximum number of guesses we believe the attacker is capable of.

Guessing Entropy
Applies to: User Generated Passwords

Guessing entropy attempts to estimate the amount of work an attacker is expected to do in order to guess the password. This is written as $H[\text{guessing}]$. Guessing entropy is the base-2 logarithm of the expected number of guesses needed to learn a user's password, defined as:

$$H[\text{guessing}] = \log_2(1 \cdot p[1] + 2 \cdot p[2] + 3 \cdot [p3] + ... + N \cdot p[N]).$$

Imagine an attacker who has a set of encrypted files from different users, and needs to crack most or all of them. The cost of his attack is given by $H[\text{guessing}]$. Specifically, to crack N passwords, he expects to spend $N \times 2^{H[\text{guessing}]}$ work.

Min-Entropy
Applies to: User Generated Passwords

One problem with guessing entropy as a measure of password security is that it only talks about average behavior. Perhaps most users of some system have pretty good passwords, but a few users have extremely weak passwords that are easily guessed. In the example above, that last password appears difficult to guess, and although it may be that most of the passwords in the distribution are like that, the attacker's first guess still gives them a 10 % chance of getting into the system.

This leads to another natural question to ask: "how likely is the attacker to guess the password on his first try?" The answer to that is given by the min-entropy, written as $H[\min]$. Min-entropy is defined as

$$H[\min] = \log_2(p[1])$$

where $p[1]$ is the probability of the most-likely password.

Imagine an attacker who is simply trying to get a login to some system, and thus who is occasionally connecting to the system, and trying some account ID with a guessed password. If we want to know how many login attempts we can expect him to need in order to get into *any* account, we can learn that from $H[\min]$. Specifically, the expected number of tries is $2^{H[\min]}$.

$H[\min] \leq H[\text{guessing}]$, so it's easier to focus on $H[\min]$ if a single metric of password security is needed.

Security After W Guesses

Finally, we might be interested in the following question: "Assuming the attacker gets only W guesses at a single password, what are his chances of getting in." For example, perhaps a user's smartcard has been stolen, but the smartcard will lock itself up after W incorrect password attempts. We want to know how likely the attacker is to get into the smartcard.

The answer to that question is given by the cumulative distribution of the password, P. Specifically, $P[W]$ gives the probability that the attacker will successfully guess the password given W tries.

For randomly-generated passwords, it's usually possible to come up with $p[1,2,3,...]$ and $P[1,2,3,...]$, so all these metrics are fairly easy to calculate. (They still are often ignored in real-world password systems, unfortunately.) In fact, most schemes for generating passwords randomly have $p[1] = p[2] = p[3] = ... = p[N]$, which makes the analysis quite easy. For example, a password generator that simply produces 4-digit random PINs has:

$$H[\min] = -\log_2(1/10000)$$

and

$$H[\text{guessing}] = \log_2(5000)$$

and

$$P[10] = 1/1000$$

For user generated passwords, there is not a known distribution from which the passwords are drawn. Different user populations, on different systems, with different guidance likely draw their passwords from different distributions. This has led to the use of a number of password heuristics which attempt to make some kind of estimate on the likely guessing- or min-entropy of user generated password, based on the length and other characteristics of the password.

An influential set of these heuristics are found in NIST SP 800-63. These attempt to approximate guessing entropy based on the length of the user generated password, and whether it makes use of both upper- and lowercase letters as well as non-alphabetic characters. The rest of the research described in this report is based on randomly generated passwords, and so these heuristics aren't necessarily relevant.

NIST SP 800-63 Guessing Entropy
Applies to: User Generated Passwords

It is of note that in NIST SP 800-63 the definition of guessing entropy is not strictly the academic definition of guessing entropy, and a collision of terms exists.

NIST SP 800-63 notes that guessing entropy is "arguably the most critical measure of the strength of a password system, since it largely determines the resistance to targeted, online password guessing attacks" [9]. Additionally, the document defines guessing entropy as "A measure of the difficulty that an attacker has to guess the average password used in a system. When a password has n-bits of guessing entropy then an attacker has as much difficulty guessing the average password as in guessing an n-bit random quantity. The attacker is assumed to know the actual password frequency distribution" [9]. Guessing entropy is used to measure user generated passwords and calculated via the following ruleset provided by NIST SP 800-63 [9]:

- *The entropy of the first character is taken to be 4 bits;*
- *The entropy of the next 7 characters are 2 bits per character; this is roughly consistent with Shannon's estimate that "when statistical effects extending over not more than 8 letters are considered the entropy is roughly 2.3 bits per character;"*
- *For the 9th through the 20th character the entropy is taken to be 1.5 bits per character;*
- *For characters 21 and above the entropy is taken to be 1 bit per character;*
- *A "bonus" of 6 bits of entropy is assigned for a composition rule that requires both upper case and non-alphabetic characters. This forces the use of these characters, but in many cases these characters will occur only at the beginning or the end of the password, and it reduces the total search space somewhat, so the benefit is probably modest and nearly independent of the length of the password;*
- *A bonus of up to 6 bits of entropy is added for an extensive dictionary check. If the Attacker knows the dictionary, he can avoid testing those passwords, and will in any event, be able to guess much of the dictionary, which will, however, be the most likely selected passwords in the absence of a dictionary rule. The assumption is that most of the guessing entropy benefits for a dictionary test accrue to relatively short passwords, because any long password that can be remembered must necessarily be a "pass-phrase" composed of dictionary words, so the bonus declines to zero at 20 characters.*

Weir et al. [15] showed that NIST's guessing entropy measurement has no relation to the guessing entropy of the password. In other words, how vulnerable a password actually is to a password guessing attack. Komanduri et al. [16] also found that the guessing entropy metric devised by NIST may be a poor predictor of password strength. Shay et al. took issue with the assumptions made by the NIST guessing entropy metric, identifying assumptions that were inconsistent with their findings, stating "...This means that the space of passwords actually used is much smaller than the space of theoretically available passwords, dramatically increasing the likelihood of an attacker being able to discover a given password, through a brute-force attack or even guessing" [17].

NIST SP 800-63 Min-Entropy
Applies to: User Generated Passwords

It is of note that in NIST SP 800-63 the definition of min-entropy is not strictly the academic definition of min-entropy, and a collision of terms exists.

Besides guessing entropy, min-entropy is another method of estimating the strength of a user generated password. NIST defines min-entropy as "A measure of the difficulty that an Attacker has to guess the most commonly chosen password used in a system. When a password has n-bits of min-entropy then an Attacker requires as many trials to find a user with that password as is needed to guess an n-bit random quantity. The Attacker is assumed to know the most commonly used password(s)" [9]. A dictionary test is used to ensure at least 10 bits of min-entropy. NIST SP 800-63 defines the following ruleset for the min-entropy test [9]:

- *Upper case letters in passwords are converted to entirely lower case and compared to a dictionary of at least 50 000 commonly selected otherwise legal passwords and rejected if they match any dictionary entry, and*
- *Passwords that are detectable permutations of the username are not allowed.*

5 Towards a New Approach

5.1 General Methodology

We defined a *password permutation* in an effort to make randomly generated passwords easier to enter on mobile devices. This was performed by grouping like character categories together in order to minimize the number of times a user must switch back and forth between onscreen keyboards. In our current scheme, the permutation categorizes the characters of a password into four sets: uppercase (*U*), lowercase (*L*), numbers (*N*), and symbols (*S*). The rearranged password is then created by concatenating each set in the order *U + L + N + S*. We created a python script to perform the permutation[13] which ensures all characters retain the order in which they were parsed by our tool. For instance, the password *5c2'Qe* becomes *Qce52'* and not *Qec25'* (notice the order of the lowercase letters). For a number of randomly-generated passwords, we processed the passwords with our permutation, and then measured the impact of the permutation on both usability and security.

5.2 Password Permutation and Its Effects on Usability

Our password permutation re-orders (permutes) the characters in a given password, so that characters likely to be on the same on-screen keyboard will appear together in the password. This makes the password easier to enter since the user needs fewer screen touches to enter the re-ordered password. However, the permutation also makes the password more predictable. In our current password permutation scheme, characters are grouped together by classes—uppercase letters, lowercase letters, digits, and symbols. Thus the password *3.bH1o* is converted to *Hbo31***.** An attacker trying to guess the permuted password has an easier job, because he knows that the permuted password will always have like characters grouped together.

The loss in security comes from the fact that there are fewer possible permuted passwords than original passwords. All of the input passwords shown below (and many more we haven't shown) will yield the same permuted password, see Table 3 for an illustration. An attacker trying to guess the user's password simply has fewer possibilities to try after the password permutation. Below, we will discuss how to measure the impact of permuting the password characters on password security.

In a randomly-generated password, members of the different character classes are intermingled. These passwords are hard for the users of mobile devices to enter, because different character classes (like digits and uppercase letters) usually require different on-screen keyboards. Putting like character classes together makes a password much easier for the user to enter—the permuted passwords require fewer screen touches and fewer changes of the onscreen keyboard.

It is possible to quantitatively measure how much easier this permutation makes password entry. By reducing the number of keystrokes required to type a password, efficiency is improved—this usability improvement is undeniable. To measure efficiency gained, simply measure the number of keystrokes saved via permutation. While human data can be gathered to measure the exact password entry time improvement, we can make this efficiency improvement claim regardless, as we know that fewer keystrokes requires less entry time. We predict that ease of learning and memorability will also improve (thereby improving effectiveness and potentially satisfaction as well). Indeed, recent research found that across platforms (smartphone, tablet, and desktop computer), people rated longer (length 14) permuted passwords as easier to type than shorter (length 10) non-permuted passwords [19].

[13] Publicly available from https://github.com/usnistgov/PasswordMetrics.

The 10 passwords used in the Greene et al. [1] study are provided below in Table 2, along with their permutations and iOS keystroke counts. For per-password keystroke sequences in their entirety, see Appendix A. Note that we only include iOS keystroke counts here since that was the operating system used in the study upon which the current work is based.

Table 2 - Original and permuted passwords and iOS keystroke counts.

Original Password	Permuted Password	Length	Keystrokes: Original, Permuted	Screen Depth Changes: Original, Permuted	Keystrokes Saved via Permutation
5c2'Qe	*Qce52'*	6	11, 8	4, 1	3
3.bH1o	*Hbo31.*	6	11, 8	4, 1	3
a7t?C2#	*Cat72?#*	7	14, 10	6, 2	4
m3)61fHw	*Hmfw361)*	8	11, 10	2, 1	1
*p4d46*3TxY*	*TYpdx4463**	10	18, 14	6, 2	4
q80<U/C2mv	*UCqmv802</*	10	19, 15	7, 3	4
d51)u4;X3wrf	*Xduwrf5143);*	12	19, 14	6, 1	5
6n04%Ei'Hm3V	*EHVnim6043%'*	12	24, 17	9, 2	7
m#o)fp^2aRf207	*Rmofpaf2207#)^*	14	24, 19	10, 4	6
4i_55fQ$2Mnh30	*QMifnh455230_$*	14	25, 19	9, 3	6

It should be obvious from Table 2 that the number of keystrokes saved—the efficiency gained—via permutation depends on the number of onscreen keyboard changes in the original password rather than on the length of the original password per se. The number of screen depth changes in turn depends on the frequency and placement of symbols and numbers, as those are the two character categories that require switching back and forth between onscreen keyboards. In other words, the more symbols there are and the more intermixed they are with other character classes in the original password, the more screen depth changes that will be saved via permutation. Note the variability in the *Keystrokes Saved* column of Table 2. Keystrokes saved range from one to seven, even in this small sample of only ten passwords.

For example, the password *m#o)fp^2aRf207* is 14 characters long, but on an iOS device, actually requires 24 keystrokes (i.e., taps on the onscreen keyboard), including 10 screen depth changes (i.e., switches between the first, second, and third iOS keyboards). Permuting it yields *Rmofpaf2207#)^* which reduces it to 19 keystrokes and four screen depth changes. Note that this type of generic permutation (upper, lower, number, symbol) could be further optimized for each mobile device operating system's specific onscreen keyboard layout. For example, were we to relax the permutation rule of preserving within-class character order, we could better rearrange the symbols in password *Rmofpaf2207#)^* from *#)^* to *)#^*. This would save an additional two keystrokes (two screen depth changes) on an iOS device, since the) symbol is on the second keyboard, while the # and ^ symbols are on the third keyboard. Similarly, we could further optimize password *UCqmv802</* specifically for iOS keyboards by reversing the order of the last two

symbols (changing </ to />), thereby saving one keystroke (one screen depth change). An analogous reversal of the last two symbols in password *QMifnh455230_$* would also save an additional keystroke/screen depth change on iOS devices.

5.3 Password Permutation and its Effects on Security

Once our password permutation is applied to the 10 passwords the Shannon entropy measurement can no longer be used due to the order imposed on the string. As previously stated, with the permutation, character groups must appear in a pre-specified order. Our permutation tool[14] creates a new permuted string by parsing each character in the original password, assigning it a character class, and then concatenating each set in the order *U* + *L* + *N* + *S*.

We determined it is possible to identify the entropy loss by determining the probability of a given mix of various character classes in randomly generated passwords of various lengths. We then compute how many ways we could generate the permuted passwords by putting those different kinds of characters in different orders. For our purposes, we defined the entropy loss due to the password permutation as equal to:

$$Entropy\ Loss = \binom{Length}{Upper} \cdot \binom{Length - Upper}{Lower} \cdot \binom{Length - Upper - Lower}{Number}$$

It is not necessary to include the S character category in the above equation, for once the *U*, *L*, and *N* character categories have been accounted for, the remaining *S* category is fixed, always appended to the end of the permuted string. It is of note that for a fully-optimized approach, instead of these four character categories (*U*, *L*, *N*, *S*) we would instead use the probabilities of the appearance of characters on each keyboard provided by a mobile operating system. The method to calculate entropy loss would then need to be revised to account for the different character classes. Below we walk through more detailed logic and examples behind the above entropy loss estimation, as well as report the results of three experiments.

We want to know the min-entropy (as defined in Section 3.4) of our newly permuted passwords, so that we can compare it with the min-entropy of the original, non-permuted passwords. We can then take the difference of the two entropy measures to determine the entropy lost as a result of our password permutation. Let us write the permutation function as $q = F(p)$, where p is the original password and q is the new one. Further, let P be the set of all passwords p, and let Q be the set of all derived (i.e., permuted) passwords q.

The min-entropy is the base two log of the probability of the most likely password, so we are really trying to work out the maximum probability password. Beginning with some password p generated from a uniform distribution, ensuring that there are 2^{40} equally likely passwords that can come out of whatever process generates p. That means the maximum probability is 2^{-40} and thus that it has 40 bits of min-entropy.[15]

Each password in *P* maps to one password in *Q*. This is another way of saying that the transformation is deterministic—the same input always goes to the same output. However, some passwords in *Q* can be mapped to multiple passwords in *P*.

[14] Publicly available from https://github.com/usnistgov/PasswordMetrics.

[15] This is also referred to as "password collision" and "mapping down".

Table 3 demonstrates multiple passwords being mapped to the same password.

Table 3 - Examples of Password Collision

Original Password	Permuted Password
abCde3f#g	*Cabdefg3#*
aC3bdefg#	*Cabdefg3#*
Ceda3g#bf	*Cabdefg3#*
S^2z5J1sw	*SJzsw251^*
^2zJ51wsS	*SJzsw251^*
2Sz51sw^J	*SJzsw251^*

As evidenced in Table 3, it is possible for collisions such as these to occur during the password permutation. By identifying the number of collisions, we can measure what we term the "fan-out" effect. Specifically, we are asking *how many different random passwords would we expect to map down to the same user-friendly password?*

In order to figure out the probability of a given password in *Q*, we need to figure out how many passwords in *P* map to it, i.e., our fan-out effect. The probability of a password *q* in *Q* is just the sum of the probabilities of the passwords in *P* that map to it, so if there are 16 passwords in *P* that map to *q*, and each password in *P* has a probability of 2^{-40}, then *q* has a probability of $16 \times 2^{-40} = 2^4 \times 2^{-40} = 2^{-36}$.

The interesting question is how to determine the maximum number of passwords in *P* that will ever map to any password in *Q*. Let us walk through an example, starting by randomly generating a password of length 14. This original password is 14 characters, and is created by randomly selecting each character from a 93-character set. We are going to sort these into four non-overlapping categories (upper, lower, number, symbol).

For each character, we have the following probabilities:

- Prob[lowercase] = 26/93
- Prob[uppercase] = 26/93
- Prob[number] = 10/93
- Prob[symbol] = 31/93

Note that the number of symbols and their probabilities would change on different devices, depending on the layout of the onscreen keyboards. In the above example, the number of symbols used was 31 because it is the number of symbols available on the iOS keyboards (second and third screen depths combined), not including the four symbols that are not visible on standard desktop keyboards.

To generate the password:

i. Randomly choose how many uppercase letters to have. The number of uppercase characters, *U*, in the password is drawn from a binomial distribution with trials = 14 and probability = 26/93.

- *U* ~ binomial(trials = 14, prob = (26/93))

ii. Randomly choose how many lowercase letters to have. The number of lowercase characters, *L*, in the password is drawn from a binomial distribution with trials = 14-*U* and probability = 26/93.

- *U* ~ binomial(trials = 14-*U*, prob = (26/93))

iii. Randomly choose how many numbers to have. The number of numeric characters, *N*, in the password is drawn from a binomial distribution with trials = 14-*U*-*L* and probability = 10/93.

- *U* ~ binomial(trials = 14-*U*-*L*, prob = (10/93))

iv. The number of special characters, *S*, in the password is:

$$S = 14\text{-}U\text{-}L\text{-}N$$

Probability theory can analytically compute what this distribution is going to be, so we reached out to experts from NIST's Statistical Engineering Division, who graciously consulted with us, providing us with probability formulas (see Appendix D) and advice regarding whether we should attempt to use said formulas versus simulating our distributions. Since computing the exact probabilities would have required specialized mathematical software and more powerful computing hardware than we had access to, we instead decided to use Monte Carlo simulations to obtain good approximate probabilities for each mix of upper, lower, numbers, and symbols.

With such simulations, we could then obtain a table of probabilities for each possible mix of upper, lower, numbers, and symbols. Given the mix, we could then compute how many ways we could generate the permuted password by putting those different kinds of characters in different orders.

i. There are 14 places to put our *U* (uppercase) letters. However, it doesn't matter which order we put them in within those places. So, the number of different ways to arrange the uppercase letters is:
$\binom{14}{U}$ = 14 choose *U* = 14!/((14-*U*)! *U*!)

ii. There are 14-*U* places to put our *L* lowercase letters, and the number of ways to arrange them is:
$\binom{14-U}{L}$

iii. There are 14-*U*-*L* places for our *N* numbers, and the number of ways to arrange them is:
$\binom{14-U-L}{N}$

iv. At this point, we know where our characters from the *S* character category must go, as only once choice remains.

So, for a password with *U* uppercase, *L* lowercase, *N* numbers, and *S* symbols, the permutation algorithm loses $W = \binom{14}{U} \cdot \binom{14-U}{L} \cdot \binom{14-U-L}{N}$ choices. Another way of saying this: there are *W* possible input passwords that would all get mapped to the same output (iOS friendly) password.

5.3.1 Experiment 1, Fan-Out

After the password permutation, how many passwords would we expect to "map down" to the same user-friendly password? For this experiment, a random password generator was created[16] that adheres to the following rules:

i. Each password must have at least one member of each character class (uppercase, lowercase, number, symbol).

ii. No password may start with an uppercase letter.

iii. No password may end with a ".", "?", or "!" character.[17]

Our value for estimating fan-out is approximate rather than exact, because we do not account for the rejection rules. That is, when counting how many passwords will map to the same user-friendly password as "aBCD!123", one of the passwords we count is "BaCD123!", which would not be allowed by the password rules. The error introduced here should be very small, and it also leads us to slightly overestimate the entropy loss. That is, our approximation can only lead us to think we're losing slightly more entropy than we're really losing, not less. Table 4 helps to answer the question "How many passwords map to the same user-friendly password?"

[16] Publicly available from https://github.com/usnistgov/PasswordMetrics.

[17] Note that this third rule is somewhat more stringent than the rules used in the original 2014 Stanton and Greene desktop password typing study [4] and the 2014 Greene et. al mobile password typing study [1], which only precluded ending in an exclamation mark.

Table 4 - Fan-out by Password Length

Length	10th Percentile	Median	90th Percentile	Average
5	60	60	60	60.0
6	120	180	180	159.8
7	210	420	630	451.5
8	840	1680	1680	1329.0
9	1512	3780	7560	3936.2
10	5040	12600	25200	12659.5
11	13860	34650	69300	38946.9
12	27720	110880	277200	132492.4
13	102960	360360	900900	414875.6
14	360360	1261260	3153150	1438513.1
15	675675	3153150	9459450	4729531.2
16	2402400	12108096	40360320	17187712.2
17	6806800	34306272	142942800	55421383.7
18	24504480	147026880	514594080	208414540.2
19	99768240	399072960	1955457504	709865504.5
20	221707200	1551950400	6518191680	2327087101.0

This first experiment measures fan-out, looking at 1000 sample passwords for each length and using the rules for rejecting passwords without one of every character type, or with a capital letter in front or a sentence-ending punctuation character at the end. Average fan-out tells us that, for example, the average 14-character random password will have about 1.4 million other 14-character passwords that will all map to the same user-friendly password. The 90th percentile is a kind of upper-bound: 90 % of the time, there will be fewer than this many passwords that map to the same user-friendly password. The 10th percentile is a lower-bound—only 10 % of the time will there be fewer than this many passwords that map to the same user-friendly password.

5.3.2 Experiment 2, Entropy Loss by Password Length

This second experiment examines how many bits of entropy lost by password length, again on randomly generated passwords subjected to the aforementioned rejection rules. In Table 5, one can see that on average, a 17-character password loses 25.7 bits of entropy. That's quite a bit—we would need to add 6 random lowercase letters back to the password to make up for the lost entropy. The following table

shows our result for this experiment and answers the question “How much entropy is lost by mapping to user-friendly password?” and the related question "How many randomly chosen lowercase letters would need to be added to the password to make up for the loss of entropy?"

Table 5 - Entropy Loss by Password Length

Length	10th Percentile	Median	90th Percentile	Average	Additional Letters
5	5.9	5.9	5.9	5.9	2
6	6.9	7.5	7.5	7.3	2
7	7.7	8.7	9.3	8.8	2
8	9.7	10.7	10.7	10.4	3
9	10.6	11.9	12.9	12.0	3
10	12.3	13.6	14.6	13.6	3
11	13.8	15.1	16.1	15.3	4
12	14.8	16.8	18.1	17.0	4
13	16.7	18.5	19.8	18.7	4
14	18.0	19.9	21.6	20.4	5
15	19.8	21.8	23.2	22.2	5
16	21.5	23.3	25.0	24.0	6
17	22.7	25.0	27.1	25.7	6
18	24.5	27.1	28.9	27.6	6
19	26.1	29.0	30.6	29.3	7
20	27.9	30.7	32.6	31.2	7

To measure usability, specifically efficiency, we evaluated time to complete the task based on keystrokes, so this is a nice apples-to-apples comparison here. For example, a 12-character password needs about 4 additional lowercase letters to make up for the loss in entropy from mapping the password down, so we

might expect that the mapped-down password will be worth it if the new format of the password means that we save more than 4 keystrokes. The mapped-down format should be even better than that, because nothing is going to be easier to type in than lowercase letters, and changing keyboards is difficult for users. Each keyboard change is like a mini task interruption, so reducing keyboard changes not only reduces keystrokes required, but also yields cognitive benefits by reducing working memory load.

It is actually possible to enhance our original permutation scheme, where the order of character categories was fixed: uppercase, lowercase, numbers, symbols (four choices). By allowing the mapped-down passwords to be in any of these arrangements, as long as all characters are kept in the same class (that is lowercase, uppercase, numbers, symbols or numbers, lowercase, symbols, uppercase, etc.) it will increase the number of possible selections. That's 4 x 3 x 2 = 24 additional choices, and so it's worth about one lowercase letter's worth of entropy to select the order of the character classes randomly.

5.3.3 Experiment 3, Additional Length Required for All-Lowercase Passwords

How much more password length would we need to just change over to all-lowercase letters? (Understanding of course that for IT departments to accept this, it would obviously need to preclude the use of dictionary words.) Table 6 shows how long an all-lowercase password must be, in order to give us the same entropy as one of these randomly selected passwords from 93 possible characters.

Table 6 - Required Additional Length for All-Lowercase Passwords

Complex Password Length	All-Lowercase Password Length	Required Extra Letters
5	7	2
6	9	3
7	10	3
8	12	4
9	13	4
10	14	4
11	16	5
12	17	5
13	18	5
14	20	6
15	21	6
16	23	7
17	24	7
18	25	7

19	27	8
20	28	8

What this tells us is that we could simply replace that 12-character multi-symbol password with a 17-character all-lowercase password. To compare things, that's:

- 12-character password from 93 possible characters:
  - ***}EZh@CAMokZ0***
- 16-character password from 93 possible characters, sorted into character classes in a fixed order: Here, we've generated the additional lowercase string *ihxu*:
  - ***hok****ihxu****EZCAMZ0}@***
- 15-character password from 93 possible characters, sorted into character classes with the order of the classes adding back a few bits of entropy: Here, we only needed three extra lowercase characters to get back the entropy we lost by mapping the password down: *ihx*
  - ***0hok****ihx****}@EZCAMZ***
- 17-character password that's all lowercase:
  - ***tvpllxtfdjuhpyugl***

5.4 Considerations of Password Permutation

Ideally, one should generate passwords from the start to be mobile-device friendly rather than permuting them post-generation. This would certainly make sense, if investigating the usability of password generation for mobile devices was the initial research goal. However, recall that this work was based on previously conducted password usability studies, the first of which was a desktop study [4] where mobile device constraints were not considered during stimuli generation—nor should they have been, as it was a desktop study. When those same stimuli were used in a subsequent mobile study [1], the negative effects of onscreen keyboard changes on participant performance were enormous. This led to the password permutation idea and corresponding question on entropy loss, the joint focus of the current work.

6 Discussion and Conclusions

We set out to answer what seemed a simple question from a usability researcher: how much entropy is lost as a result of permuting system generated passwords to be more usable for mobile devices? Along the way, we discovered that each side—usability and security—had much to learn in terms of language and concepts that we each take for granted in our respective fields of study. By having security and usability researchers working closely together, each learned core vocabulary and principles from the other's field. By documenting these fundamental concepts as we have, we hope to support future usable security collaborations, both internal and external.

All too often, researchers propose a means of improving usability or improving security without actually measuring the resulting change on the opposing axis[18]. Here, we proposed a means of improving usability for system-generated passwords (i.e., permuting them by grouping like character classes together to minimize onscreen keyboard changes and thereby improve ease of entry on mobile devices), and a means of measuring the resulting change in security. Usability is well-defined, with an associated ISO standard for the definition of usability, ISO 9241-11 [5], as are methods for its measurement, whereas no single standard definition of security exists. This makes usable security research both interesting and challenging, particularly within constraints of the mobile domain.

No single mobile platform has a native onscreen keyboard where all characters are available at all times. Given the screen size of mobile devices, the mobile equivalent of the standard QWERTY desktop keyboard would be quite difficult to implement without having keys that were even smaller and more difficult for people to tap. Therefore, we must accept that onscreen keyboard constraints are inevitable given the current state of mobile device technology, and that complex passwords are particularly difficult to enter on mobile devices. Permuting passwords in order to make them easier for mobile entry is one option for dealing with this issue. The number of keystrokes saved—the efficiency gained—via permutation depends on the number of screen depth changes in the original password rather than on the length of the original password. The number of screen depth changes in turn depends on the frequency and placement of symbols and numbers, as those are the two character categories that require switching back and forth between onscreen keyboards.

The fact that passwords containing numbers and symbols require changing onscreen keyboards is a fundamental difference between desktop and mobile platforms, yet it does not appear that current password policies have taken this fundamental difference into account. The effects that mobile device constraints have on both usability and security of passwords need to be systematically explored and measured in order to better inform password policies for the increasingly mobile work environment. With the current research, we have accomplished an initial piece of such systematic exploration.

We have created publicly available code from which other researchers may benefit, and we certainly hope that others will expand upon our initial three experiments. In our first experiment, we measured fan-out, looking at 1000 sample passwords for each length (ranging from 5 to 20 characters) to see how many passwords that when permuted, map down to the same user-friendly password. Average fan-out tells us

[18] We use the term "opposing axis" not because we feel that the two concepts—usability and security—should theoretically be in opposition to one another, but merely for purposes of exposition. Envisioning usability and security on opposing axes of a graph makes visualization easier, as does the mental image conjured by the commonly used phrase "the intersection of usability and security," where one visualizes two intersecting lines.

that, for example, the average 14-character random password will have about 1.4 million other 14-character passwords that will all map to the same user-friendly password.

In our second experiment, we measured entropy loss by password length, with password length again ranging from 5 to 20 characters. We found that on average, a 14-character random password loses 20.4 bits of entropy due to the password permutation. For a 14-character password, the entropy loss could be mitigated by adding five additional lowercase letters back to the password. Since lowercase letters are so much easier for people to type—not only do they require fewer keystrokes on mobile devices, but letters in general are more practiced than numbers or symbols—this prompted our third experiment, where we examined how much additional length we would need to simply change over to all lowercase letters.

As with the first two experiments, we examined passwords ranging in length from 5 to 20 characters in our third experiment. We found that a 14-character random password would need to be 20 characters in length if it were composed only of lowercase letters. Adding 6 extra characters (to a 14-character random password) may easily be preferable to typing numbers and symbols. Using all lowercase mobile passwords would remove the need to change onscreen keyboards at all, which would likely have significant usability benefits (human data would be needed to measure the extent of such usability improvements).

As passwords are still the most prevalent means of authentication, this work has implications for helping protect against online attacks and mitigating offline attacks. While there are many metrics for password security, it may be that the most meaningful goal is to understand the number of guesses it will likely take to take for the attacker to guess this password. While we anxiously await the day when the password will truly be dead, we seek to improve password usability now, without an unacceptably high cost to security. We hope this work will foster ongoing collaborations between usability and security researchers everywhere.

6.1 Ethical Considerations

As the current work deals with system-generated passwords and theoretical measurements, the ethical considerations are quite different from those that would apply to research with user generated passwords. Nonetheless, we feel that discussion of the ethical considerations of user generated password research is important and should be included here given the larger goals of the paper (e.g., fostering more informed dialogue between usability and security researchers). Furthermore, although much can be learned via basic research on system-generated passwords (e.g., exploration of device constraints and typing times and error rates), this work may be considered a stepping stone to future work with user generated passwords. User generated passwords may be obtained in laboratory experiments that must always undergo Institutional Review Board (IRB) approval for human-subjects testing, a process that may be unfamiliar to some security researchers.

6.2 Future Work

There are numerous research studies one could run to expand upon this work, ranging from studies with system generated passwords to studies with user generated passwords. For example, one could test the permuted passwords, or a small subset of them, in a laboratory study to gather measures of effectiveness (i.e., error rates) and satisfaction in addition to the currently reported measure of efficiency. Having behavioral data on the other two facets of usability would be helpful to fully understand the benefits of our proposed permutation. Recent research has already addressed the benefits of our proposed permutation for subjective usability, finding that across platforms (smartphone, tablet, and desktop computer), people rated longer (length 14) permuted passwords as easier to type than shorter (length 10)

non-permuted passwords [19]. It would also be useful to investigate how easy it would be to write rules for password cracking tools to guess our permuted passwords.

A laboratory study that directly compares longer, all lowercase passwords with shorter, mixed-character class passwords (i.e., passwords containing a mixture of uppercase, lowercase, numbers, symbols) is another obvious future direction. After so many years of password policies requiring uppercase letters, numbers, and symbols, it is unlikely that organizations would be willing to change their password policies to permit all lowercase passwords without more data. Such a study could initially test system generated passwords for greater stimuli control. A subsequent study could test user generated passwords, where users are given different password requirement sets and asked to generate, practice, then log in with their generated passwords; one password requirement set would allow all lowercase passwords of a certain minimum length (not allowing dictionary words), and another would require that uppercase, numbers, and symbols also be included in the passwords.

All uppercase passwords could also be studied. Since both desktops and mobile devices have the equivalent of a "caps lock" function, all uppercase passwords really only require one additional key press to execute on a desktop computer, and two extra key presses on a mobile device (on an iOS device, one must typically press the shift key twice to turn caps lock on). It would be interesting to see whether people are now so accustomed to generating passwords containing numbers and symbols that asking them to generate letter-only passwords actually proves more difficult for them if it breaks their now ingrained password generation strategies.

There is new research on user generated passwords being conducted in NIST's Information Access Division's (IAD) Visualization and Usability Group. That usability research is examining the effects of different password requirements and formatting on the passwords users generate on desktop computers, already finding uneven distributions for character frequencies [20], meaning that some letters or symbols are rarely if ever used, while others are extremely common. This should be of great interest to the security community, as it aligns with the growing concern that the typical use of "94 possible starting characters"[19] in much password security research is incorrect, and the practical starting character space is much smaller. Having usability and security researchers working together to computer strength or quality metrics for user generated passwords would complement both fields.

Future studies could expand upon the aforementioned desktop password generation research by conducting it with mobile devices. If people are sensitive to their mobile device constraints, the starting character space for user generated passwords may be even further reduced. In order to avoid navigating to the third (and even the second) onscreen keyboard, people may be unlikely to choose symbols from those screen depths, especially when they are not required to.

The previously mentioned research ideas are focused on password usability for the end user, but there are other types of users who are vitally important for system security: system administrators and software developers. Little usable security research has been conducted with these users, yet they are often responsible for implementing protocols and software that are vital to the realized security of a system. Having security and usability researchers working together on topics such as these would greatly benefit

[19] Note that the space of 94 starting characters applies to desktop computers, since 26 uppercase letters, 26 lowercase letters, 10 numbers, and 32 symbols are available on traditional desktop keyboards. As previously mentioned, starting character pools for mobile devices should be computed based on their individual keyboard implementations.

the field of usable security. Only by considering both perspectives—usability and security—can we hope to strike an optimal balance between the two.

Appendix A: Acronyms and Abbreviations

HCI	Human-Computer Interaction
IAD	Information Access Division
IEC	International Electrotechnical Commission
iOS	iPhone Operating System
IRB	Institutional Review Board
ISO	International Organization for Standardization
ITL	Information Technology Laboratory
NIST	National Institute of Standards and Technology
NISTIR	NIST Interagency Report
NSTIC	National Strategy for Trusted Identities in Cyberspace
OS	Operating System
PIN	Personal Identification Number
SP	Special Publication
SSL	Secure Sockets Layer
TLS	Transport Layer Security

Appendix B: References

[1] K. K. Greene, M. A. Gallagher, B. Stanton, and P. Y. Lee, "I Can't Type That!, P@$$w0rd Entry on Mobile Devices," in *Human Aspects of Information Security, Privacy, and Trust*, Lecture Notes in Computer Science 8533, 2014, pp. 160-171. http://dx.doi.org/10.1007/978-3-319-07620-1_15.

[2] M. Jakobsson and R. Akavipat, "Rethinking Passwords to Adapt to Constrained Keyboards," *Mobile Security Technologies 2012 (MoST 2012)*, San Francisco, California, United States, May 24, 2012. http://www.mostconf.org/2012/papers/5.pdf [accessed 4/19/2016].

[3] *National Strategy for Trusted Identities in Cyberspace, Enhancing Online choice, Efficiency, Security, and Privacy*, 2011. http://purl.fdlp.gov/GPO/gpo7204.

[4] B. Stanton and K. Greene, "Character strings, memory, and passwords: What a recall study can tell us," in *Human Aspects of Information Security, Privacy, and Trust*, Lecture Notes in Computer Science 8533, 2014, pp. 195-206. http://dx.doi.org/10.1007/978-3-319-07620-1_18.

[5] International Organization for Standardization, *Ergonomic requirements for office work with visual display terminals (VDTs) -- Part 11: Guidance on usability*, ISO 9241-11:1998.

[6] E. Frøkjær, M. Hertzum, and K. Hornbæk, "Measuring usability: Are effectiveness, efficiency, and satisfaction really correlated?" in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '00)*, The Hague, The Netherlands, April 1-6, 2000, pp. 345-352. http://dx.doi.org/10.1145/332040.332455.

[7] A. Adams, and M. A. Sasse, "Users are not the enemy," *Communications of the ACM*, vol. 42, no. 12 (December 1999), pp 40-46. http://dx.doi.org/10.1145/322796.322806.

[8] D. Florêncio, C. Herley, and P. C. van Oorschot, "Password Portfolios and the Finite-Effort User: Sustainably Managing Large Numbers of Accounts," in *Proceedings of the 23rd USENIX Security Symposium*, San Diego, California, United States, August 20-22, 2014, pp. 575-590. https://www.usenix.org/conference/usenixsecurity14/technical-sessions/presentation/florencio [accessed 4/19/2016].

[9] National Institute of Standards and Technology, *Special Publication (SP) 800-63-2, Electronic Authentication Guideline*, 2013. http://dx.doi.org/10.6028/NIST.SP.800-63-2.

[10] A. G. Reinhold, *The Diceware Passphrase Home Page* [Web site], 2016. http://world.std.com/~reinhold/diceware.html [accessed 4/19/2016].

[11] M. Jakobsson and M. Dhiman, "The Benefits of Understanding Passwords," *7th USENIX Workshop on Hot Topics in Security (HotSec'12)*, Bellevue Washington, United States, August 7, 2012, 6 pp. https://www.usenix.org/conference/hotsec12/workshop-program/presentation/jakobsson [accessed 4/19/2016].

[12] Q., Yue, "My Google Glass Sees Your Password!" presented at *Blackhat USA*, Las Vegas, Nevada, United States, August 2-7, 2014. https://www.blackhat.com/docs/us-14/materials/us-14-Fu-My-Google-Glass-Sees-Your-Passwords.pdf [accessed 4/19/2016].

[13] C. E. Shannon, "A Mathematical Theory of Communication," *Bell Systems Technical Journal*, vol. 27, no. 3 (July 1948), pp. 379-423. https://dx.doi.org/10.1002%2Fj.1538-7305.1948.tb01338.x.

[14] J. O. Pliam, "Guesswork and Variation Distance as Measures of Cipher Security," in *Selected Areas in Cryptography 6th Annual International Workshop (SAC'99)*, Lecture Notes in Computer Science 1758, 2000, pp. 62–77. http://dx.doi.org/10.1007/3-540-46513-8_5.

[15] M. Weir, S. Aggarwal, M. Collins, and H. Stern , "Testing Metrics for Password Creation Policies by Attacking Large Sets of Revealed Passwords," in *Proceedings of the 17th ACM Conference on Computer and Communications Security (CCS '10)*, Chicago, Illinois, United States, October 4-8, 2010, pp. 162-175. http://dx.doi.org/10.1145/1866307.1866327.

[16] S. Komanduri, R. Shay, P. G. Kelley, M. L. Mazurek, L. Bauer, N. Christin, L. F. Cranor, and S. Egelman, "Of Passwords and People: Measuring the Effect of Password-Composition Policies," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '11)*, Vancouver, British Columbia, Canada, May 7-12, 2011, pp. 2595-2604. http://dx.doi.org/10.1145/1978942.1979321.

[17] R. Shay, S. Komanduri, P. G. Kelley, P. G. Leon, Michelle L. Mazurek, L. Bauer, N. Christin, and L. F. Cranor, "Encountering Stronger Password Requirements: User Attitudes and Behaviors," in *Proceedings of the Sixth Symposium on Usable Privacy and Security (SOUPS '10)*, Redmond, Washington, United States, July 14–16, 2010, article no. 2. http://dx.doi.org/10.1145/1837110.1837113.

[18] International Organization for Standardization/International Electrotechnical Commission, *Information technology -- Keyboard layouts for text and office systems -- Part 1: General principles governing keyboard layouts*, ISO/IEC 9995-1:2009.

[19] K. K. Greene, "Effects of Password Permutation on Subjective Usability Across Platforms," in *Human Aspects of Information Security, Privacy, and Trust*, Lecture Notes in Computer Science 9190, 2015, pp. 59-70. http://dx.doi.org/10.1007/978-3-319-20376-8_6.

[20] P. Y. Lee, and Y. Choong, "Human Generated Passwords – the Impacts of Password Requirements and Presentation Styles," in *Human Aspects of Information Security, Privacy, and Trust*, Lecture Notes in Computer Science 9190, 2015, pp. 83-94. http://dx.doi.org/10.1007/978-3-319-20376-8_8.

Appendix C: Keystroke Counts

The following tables demonstrate the key sequences a user would need to enter in order to type the 10 passwords from the prior Greene et. al. study [1]. This information is provided for both the original passwords in the study, and the permuted password using the permutation described within this document. All sequences are given in reference to an iPhone running iOS 6. Note that passwords and its associated permuted forms are available in table captions.

Table 7 - Keystroke counts & key sequences for *5c2'Qe* and *Qce52'*

Keystroke count	Key sequence, original: *5c2'Qe*	Key sequence, permuted: *Qce52'*
1	<.?123>	shift
2	5	Q
3	<ABC>	c
4	C	e
5	<.?123>	<.?123>
6	2	5
7	‘	2
8	<ABC>	‘
9	shift	
10	Q	
11	E	
total	11	8

Table 8 - Keystroke counts & key sequences for *3.bH1o* and *Hbo31.*

Keystroke count	Key sequence, original: *3.bH1o*	Key sequence, permuted: *Hbo31.*
1	<.?123>	shift
2	3	H
3	.	b
4	<ABC>	o
5	B	<.?123>
6	shift	3
7	H	1
8	<.?123>	.
9	1	
10	<ABC>	
11	O	
total	11	8

Table 9 - Keystroke counts & key sequences for *a7t?C2#* and *Cat72?#*

Keystroke count	Key sequence, original: *a7t?C2#*	Key sequence, permuted: *Cat72?#*
1	a	shift
2	<.?123>	C
3	7	a
4	<ABC>	t
5	t	<.?123>
6	<.?123>	7
7	?	2
8	<ABC>	?
9	shift	<#+=>
10	C	#
11	<.?123>	
12	2	
13	<#+=>	
14	#	
total	14	10

Table 10 - Keystroke counts & key sequences for *m3)61fHw* and *Hmfw361)*

Keystroke count	Key sequence, original: *m3)61fHw*	Key sequence, permuted: *Hmfw361)*
1	m	shift
2	<.?123>	H
3	3	m
4	)	f
5	6	w
6	1	<.?123>
7	<ABC>	3
8	f	6
9	shift	1
10	H	)
11	w	
totals	11	10

Table 11 - Keystroke counts & key sequences for *p4d46*3TxY* and *TYpdx4463

Keystroke count	Key sequence, original: *p4d46*3TxY*	Key sequence, permuted: *TYpdx4463**
1	p	shift
2	<.?123>	T
3	4	shift
4	<ABC>	Y
5	d	p
6	<.?123>	d
7	4	x
8	6	<.?123>
9	<#+=>	4
10	*	4
11	<123>	6
12	3	3
13	<ABC>	<#+=>
14	shift	*
15	T	
16	x	
17	shift	
18	Y	
total	18	14

Table 12 - Keystroke counts & key sequences for *q80<U/C2mv* and *UCqmv802</*

Keystroke count	Key sequence, original: *q80<U/C2mv*	Key sequence, permuted: *UCqmv802</*	Key sequence, iOS-optimized permutation: *UCqmv802/<*
1	q	shift	shift
2	<.?123>	U	U
3	8	shift	shift
4	0	C	C
5	<#+=>	q	q
6	<	m	m
7	<ABC>	v	v
8	shift	<.?123>	<.?123>
9	U	8	8
10	<.?123>	0	0
11	/	2	2
12	<ABC>	<#+=>	/
13	shift	<	<#+=>
14	C	<123>	<
15	<.?123>	/	
16	2		
17	<ABC>		
18	m		
19	v		
total	19	15	14

Table 13 - Keystroke counts & key sequences for *d51)u4;X3wrf* and *Xduwrf5143);*

Keystroke count	Key sequence, original: *d51)u4;X3wrf*	Key sequence, permuted: *Xduwrf5143);*
1	d	shift
2	<.?123>	X
3	5	d
4	1	u
5	)	w
6	<ABC>	r
7	u	f
8	<.?123>	<.?123>
9	4	5
10	;	1
11	<ABC>	4
12	shift	3
13	X	)
14	<.?123>	;
15	3	
16	<ABC>	
17	w	
18	r	
19	f	
Totals	19	14

Table 14 - Keystroke counts & key sequences for *6n04%Ei'Hm3V* and *EHVnim6043%'*

Keystroke count	Key sequence, original: *6n04%Ei'Hm3V*	Key sequence, permuted: *EHVnim6043%'*
1	<.?123>	shift
2	6	E
3	<ABC>	shift
4	n	H
5	<.?123>	shift
6	0	V
7	4	n
8	<#+=>	i
9	%	m
10	<ABC>	<.?123>
11	shift	6
12	E	0
13	i	4
14	<.?123>	3
15	‘	<#+=>
16	<ABC>	%
17	shift	‘
18	H	
19	m	
20	<.?123>	
21	3	
22	<ABC>	

23	shift	
24	V	
total	24	17

Table 15 - Keystroke counts & key sequences for *m#o)fp^2aRf207* and *Rmofpaf2207#)^*

Keystroke count	Key sequence, original: *m#o)fp^2aRf207*	Key sequence, permuted: *Rmofpaf2207#)^*	Key sequence, iOS-optimized permutation: *Rmofpaf2207)#^*
1	m	shift	shift
2	<.?123>	R	R
3	<#+=>	m	m
4	#	o	o
5	<ABC>	f	f
6	o	p	p
7	<.?123>	a	a
8	)	f	f
9	<ABC>	<.?123>	<.?123>
10	f	2	2
11	p	2	2
12	<.?123>	0	0
13	<#+=>	7	7
14	^	<#+=>	)
15	<123>	#	<#+=>
16	2	<123>	#
17	<ABC>	)	^
18	a	<#+=>	
19	shift	^	
20	f		
21	<.?123>		
22	2		

23	0		
24	7		
total	24	19	17

Table 16 - Keystroke counts & key sequences for *4i_55fQ$2Mnh30* and *QMifnh455230_$*

Keystroke count	Key sequence, original: *4i_55fQ$2Mnh30*	Key sequence, permuted: *QMifnh455230_$*	Key sequence, iOS-optimized permutation: *QMifnh455230$_*
1	<.?123>	shift	shift
2	4	Q	Q
3	<ABC>	shift	shift
4	i	M	M
5	<.?123>	i	i
6	<#+=>	f	f
7	_	n	n
8	<123>	h	h
9	5	<.?123>	<.?123>
10	5	4	4
11	<ABC>	5	5
12	f	5	5
13	shift	2	2
14	Q	3	3
15	<.?123>	0	0
16	$	<#+=>	$
17	2	_	<#+=>
18	<ABC>	<123>	_
19	shift	$	
20	M		
21	n		
22	h		

23	<.?123>		
24	3		
25	0		

Appendix D: Probability Formulas

These formulas were provided by Andrew Rukhin of NIST's Statistical Engineering Division. Since computing the exact probabilities in the following formulas would have required specialized mathematical software and more powerful computing hardware than we had access to, we instead decided to use Monte Carlo simulations to obtain good approximate probabilities for each mix of upper, lower, numbers, and symbols.

March 18, 2014

Entropy Comparison of Passwords After a Usability Transform

Andrew Rukhin

The goal here is to derive a formula for the entropy decrease after the usability transform which consists in ordering the random entries of a password according to their classes and also within each class.

Let us start with just one class consisting of, say, m symbols, and passwords of length n. Then the total number of passwords is m^n, while the number of ordered passwords is

$$\sum_{1\le k\le m} k(m-k+1)^{n-1}.$$

Indeed if k is the first symbol in the password after ordering, then there are $(m-k+1)$ choices for each of the remaining $n-1$ symbols (which can be equal one to another).

Now assume that there are I classes (like lower case alphabetic characters, upper case alphabetic characters, numeric characters, and special symbols so that $I=4$). The i-th class is supposed to have m_i symbols (like $m_1=m_2=26, m_3=10, m_4=32$).

Let us evaluate entropies provided that the password must have exactly n_i symbols from the class $i, i=1,\dots,I$ and thus to have the length $N=n_1+\cdots+n_I$.

The total number of unordered passwords is then $(m_1+\cdots+m_I)^N$. There are

$$\sum_i \sum_{1\le k_i\le m_i} k_i(m_i-k_i+1)^{n_i-1}$$

choices for ordered password after the usability transform.

Thus for given $n_1,\dots,n_i$ the entropy decrease is given by

$$N\log(m_1+\cdots+m_I)-\log\left(\prod_{i=1}^{I}\sum_{1\le k_i\le m_i} k_i(m_i-k_i+1)^{n_i-1}\right). \qquad (1)$$

To get the final formula (which is not very useful for calculations) assume that non-negative probabilities $p_{n_1,\dots n_I}$, $\sum_{n_1,\dots n_I} p_{n_1,\dots n_I}=1$, are given.

Thus $p_{n_1,\ldots n_I}$ represent the weights assigned to this particular combination of frequencies of symbols from each class. These probabilities can be used to reflect given restrictions, e.g., the rule according to which there must be at least one upper-case character means that $p_{0n_2\cdots n_4} = 0$.

The logarithm of the probability of a truly random password,

$$\sum_{n_1,\ldots n_I} \frac{p_{n_1,\ldots n_I}}{(m_1+\cdots+m_I)^N} = \frac{1}{(m_1+\cdots+m_I)^N}$$

is to be contrasted with that of the probability

$$\sum_{n_1,\ldots n_I} \frac{p_{n_1,\ldots n_I}}{\prod_i \sum_{1\le k_i\le m_i} k_i(m_i-k_i+1)^{n_i-1}}.$$

These formulas ignore some of existing restriction rules, e.g. the password cannot end with an exclamation mark. This can be taken into account by replacing $N\times\log(m_1+\cdots+m_I)$ by $(N-1)\times\log(m_1+\cdots+m_I)+log(m_1+\cdots+m_I-1)$, and by subtracting in (1)

$$-\log\left(\left[\prod_{i=1}^{I-1}\sum_{1\le k_i\le m_i} k_i(m_i-k_i+1)^{n_i-1}\right]\sum_{1\le k_I\le m_I} k_I(m_I-k_I+1)^{n_I-1}(m_I-k_I)\right).$$

Passwords with Maximum Entropy under Restrictions:Formulas and Estimation
Discussion Draft by Andrew. L. Rukhin

1 On the Calculation of Password's Maximum Entropy under Restrictions

The goal is to find the maximum entropy of a random password of, say, length M for a given set of possible keyboard characters which has cardinality (the number of elements) n. Then the set of all possible passwords has cardinality n^M. Assume that there are K restriction rules $A_1, \ldots, A_K$, so that a password must satisfy all K conditions. In mathematical terms each A_i is a subset of the set of all passwords, and an admissible password must belong to the set $A_1 \bigcap \cdots \bigcap A_K$. We denote by $W(A_i)$ the cardinality of A_i. and by $W(B_i) = n^M - W(A_i)$ the cardinality of $B_i = A_i^c$ (the complement of A_i.) For example, the rule A_1 can be that a password must have at least one numeric character, in which case $W(A_1) = n^M - p^M$, $W(B_1) = p^M$, where p is the number of non-numeric characters. The rule A_2 can be that a password cannot belong to a certain dictionary. Then, $W(A_2) = n^M -$card(dictionary).

As a primary setting, let us think that the set of possible keyboard characters is split into $K+1$ different mutually exclusive categories (classes), like alphabetic characters, numeric characters, etc. There are n_i characters in the i-th class, so that $n_1 + \cdots + n_K \leq n$. In a real life application $n = 92, K = 3, M = 8, n_1 = 26$ (alphabetic characters), $n_2 = 10$ (numeric characters), $n_3 = 20$ (special characters), $n_4 = 36$ (remaining characters). The restriction rule A_i has the form: a password must have at least one character from the i-th class, $i = 1, \ldots, K$. We may add the rule, A_{K+1}: no passwords belonging to a dictionary B_{K+1}.

The cardinality m of the set of all admissible passwords can be evaluated by the *inclusion-exclusion principle* (Hall 1986, p 8)

$$m = n^M - \sum_i W(B_i) + \sum_{i<j} W(B_i \bigcap B_j) + \cdots$$

$$+(-1)^k \sum_{i_1<i_2<\cdots<i_k} W(B_{i_1} \bigcap \cdots \bigcap B_{i_k}) + \cdots + (-1)^K W(B_1 \bigcap \cdots \bigcap B_K).$$

In our setting (with no dictionary),

$$m = n^M - (n-n_1)^M - \cdots - (n-n_K)^M$$
$$+(n-n_1-n_2)^M + \cdots + (n-n_{K-1}-n_K)^M - \cdots$$
$$+(-1)^K (n-n_1-n_2-\cdots-n_K)^M.$$

The entropy of a random password then is $\log_2 m$. This is the largest possible entropy and it corresponds to the probability distribution under which each password has the probability $1/m$ to be chosen. In the application above it is $0.4556 * 92^8$ with the maximum entropy of 51.05 bits.

The inclusion-exclusion formula also is applicable in the situation when a password must satisfy *exactly* (or *at least*) $k, k \le K$, conditions out of $A_1, \ldots, A_K$, and this may suggest a more general setting for passwords restrictions.

When $n = K\ell$, with an integer ℓ, the optimal choice for $n_1, n_2, \ldots, n_K$ is $n_i \equiv \ell, n_{K+1} = 0$, in which case

$$m = n^M \sum_{j=0}^{K} (-1)^j \left(\begin{array}{c} K \\ j \end{array} \right) \left(\frac{K-j}{K} \right)^M = \frac{n^M K!}{K^M} \left\{ \begin{array}{c} M \\ K \end{array} \right\},$$

where $\left\{ \begin{array}{c} M \\ K \end{array} \right\}$ denotes Stirling's number of the second kind, i.e. the number of partitions of an M element set in K nonempty subsets. It follows that $m = 0$ if $K > M$, and $m \sim n^M$ as $M \to \infty$. In our example with $K = 3$, the optimal entropy is 52.01 bits.

The following table for $M = 8$, gives the values of the ratio $\frac{m}{n^M} = \frac{K!}{K^M} \left\{ \begin{array}{c} M \\ K \end{array} \right\}$, and its binary logarithm, with sign minis, i.e. the entropy of the random password with no restrictions minus the optimal entropy $D = M \log_2 n - \log_2 m$.

K	m/n^M	D
1	1.000	0
2	0.992	0.011
3	0.883	0.179
4	0.623	0.683
5	0.323	1.632
6	0.114	3.133
7	0.024	5.352
8	0.002	8.701

2 Sequential Design to Estimate Entropy

Let $T_1, T_2, \ldots$ be the observed sequence of random (hashed or not) passwords which are treated as observations on a random variable whose entropy is to be estimated. Clearly, this random variable has a very large number of outcomes with very small individual probabilities, so the classical estimation procedures are not applicable. However, we will assume that there are at least two identical passwords in this series.

Denote by τ_1 the waiting time until the first password repeats itself

$$\tau_1 = \min\Big\{j : T_{j+1} = T_1\Big\}.$$

After the random moment τ_1, one inspects τ_2 passwords until the third, identical to the first, password appears. Formally,

$$\tau_2 = \min\Big\{j : T_{j+\tau_1+1} = T_{\tau_1+1}\Big\},$$

so that in general, if $R_0 = 1, R_r = \tau_1 + \cdots + \tau_r + 1$, $r = 1, 2, \ldots$,

$$\tau_r = \min\Big\{j : T_{j+R_{r-1}} = T_{\tau_{r-1}}\Big\},$$

$r = 1, 2, \ldots$. Let m denote the largest value such that τ_m occurs in the sample. (If $m = 0$, i.e. if the first password is unique in this sequence, we'll have to start the clock at some other password.)

The suggested estimator of entropy $\hat{H}$ (Dobrushin, 1958, Maurer, 1992) has the form,

$$\hat{H} = \frac{1}{m}\sum_{r=1}^{m} \log \tau_r + C, \tag{1}$$

with $C = 0.577..$ denoting Euler's constant. It can be shown to have a fairly small bias and a tractable (normal) limiting distribution. If in the observed sequence the passwords are not in random order, the randomization process will be needed.

It may happen that the collected passwords will have "salts" representable as random numbers 12 bits long, so that each sequence as above leads to 2^{12} independent (?) replicas. In this situation the estimates $\hat{H}$ above can be profitably averaged.

In lieu of real life passwords, this study can be done for computer generated passwords, which may or may not obey several restriction rules.

3 References

1. R. L. Dobrushin, A simplified method of experimental estimation of entropy of a stationary sequence. *Probab. Theory Appl.*, 3, 462–464, 1958.

2. M. Hall. *Combinatorial Theory.* 2nd ed, Springer, NY, 1986.

3. U. M. Maurer. A Universal Statistical Test for Random Bit Generators. *Journal of Cryptology*, vol. 5, no. 2, 1992, pp. 89-105.