Practical security guidance for real-world users and designers of authentication systems

Thesis Proposal

Joshua Tan

September 6, 2019

Abstract

A wealth of security guidance is available to users and security system designers. Much of this guidance assumes an idealized world in which users are motivated and attentive when completing security tasks interjected in front of their primary task; and in which system designers are free to design security systems that subject their user base to however poor a usability experience deemed necessary in order to maximize security benefits for those users. However, these idealized-world assumptions do not reflect reality. Users perform security as a secondary task, typically unmotivated to exert more than the minimum possible effort to accomplish security tasks, which they ultimately perform in an inattentive, error-prone manner.

I will demonstrate how practical guidance can be given to users that assumes real-world settings but that does not abandon the worthwhile goal of improving security for those who do not want to or cannot take the security-optimal approach. First, I will show how users can be guided to create strong passwords while minimizing negative usability impacts, using password policies that incorporate a minimum neural-network-derived password-strength requirement. Second, I will show how latent features of passwords can be learned that can then be leveraged to improve password-creation guidance with more relevant and comprehensive text feedback. Lastly, I will show how practical guidance can be given to system designers in authentication domains other than passwords, using public-key verification as a case study. Specifically, I will examine ways of exposing fingerprint verification to ordinary users that remain effective at protecting against man-in-the-middle attacks in low-risk situations or usability-focused environments.

Contents

1 Introduction

More and more of our lives are becoming integrated into the digital world. This has enabled new, improved ways of living. Social media networks allow us to share life events privately with friends. We can send money near instantaneously across the world using cryptocurrencies. Secure messaging tools allow people to have private conversations through the Internet. Family photos and videos can be stored in a digital family archive and passed down through generations.

This digital migration of our lives also necessitates protections against adversaries operating in the digital world. Privately shared information should stay private. Money stored in digital wallets should be protected from thieves. Private online conversations should be free from government or ISP eavesdropping. Digital family archives should be safe from hackers, accessible only by family. A primary way this protection is achieved is through authentication systems, such as those based on passwords or public-key encryption.

Unfortunately authentication systems are often designed in ways that make them ineffective. A primary reason for this is that they don't consider the motivations and capabilities of typical end users. Average users do not understand nor have the time to contemplate specific ways in which an attacker might try to guess their account password when they are creating a password for that account. Many are more concerned with creating a password that they can remember and that satisfies the (annoying) password policy required for that particular account. Designers of authentication systems should embrace this reality and create tools and mechanisms that can be securely used without unrealistic expectations about user motivations, knowledge, and effort.

Just as system designers might make assumptions of end users that are unrealistic and impractical, the guidance provided by security experts for authentication system designers can also assume unrealistic conditions and thus be impractical for those designers to actually implement. Expert-recommended guidance for end-user authentication may inhibit the usability of the service it is meant to protect. This negative impact on the user experience often may be enough for a usability-focused organization to rule out security improvements based on that guidance.

This proposed thesis will explore authentication system designs that are easier for average users to use in a secure, effective way. Specifically, this work investigates methods that leverage password-strength neural networks at password-creation time to encourage stronger passwords without reducing password memorability or causing undue user annoyance. This work also examines how usability-focused organizations might incorporate public-key verification to leverage its security guarantees even for inattentive or distracted users, for low-risk scenarios in which software-automated comparison methods are unavailable.

2 Thesis themes

Develop neural-network-driven security mechanisms for improving password security

This thesis will explore ways of using neural-network-based security mechanisms to encourage users to create guess-resistant passwords. Prior work [1] has demonstrated how neural networks trained on leaked password data can be deployed in a browser setting and used to quantify and communicate to the user an estimate of password strength, via a password meter bar. We will extend this work by examining the effect of imposing minimum-strength password requirements within a password creation policy, where password strength is measured using a neural network model fine-tuned during training for a specific password policy. An important goal of this work is to understand how such minimum-strength password requirements affect both usability during password creation and the memorability of created passwords.

We will also explore ways of using neural networks to improve password-strength text feedback in password meters. Specifically, we will use neural networks to help identify novel password guessability heuristics, visually map these heuristics to the individual password characters comprising them, and create a linear model based on interpretable heuristics to predict password strength.

Provide practical recommendations to authentication system designers operating in real-world, usability-focused environments

We will examine usability and security challenges faced by security architects who must design systems in which users perform security tasks while distracted or otherwise inattentive. This is especially relevant for security architects who wish to introduce security mechanisms for a usability-focused user base. Using public-key verification as a case study, we will perform an online user study to examine the security and usability provided by different fingerprint representations and comparison modes under user-adverse conditions for simulated man-in-the-middle attacks.

3 Background and related work[1]

This section first describes related work on password security, including the frequency and types of password reuse, metrics for quantifying the strength of a password against guessing attacks, and tools researchers have created to assist users in creating strong passwords. Next, we discuss prior work on learning interpretable explanations for black-box machine-learning models, covering topics such as dimensionality reduction, cluster visualization, and methods for learning the relevance of input features in model predictions. Lastly, we provide an overview of public-key fingerprints, including actual application that use them, studies of their usability, and how entropy can be used to quantify the strength of fingerprint representations and attacks.

3.1 Passwords

3.1.1 Password reuse

Prior work has found that users heavily rely on password reuse (both exact and partial reuse) in order to manage the large number of online accounts they accumulate over time [3–6]. While this helps reduce memory burdens associated with having unique passwords for each account, it also can reduce the effective security of passwords against attackers.

Choong et al. studied the password management behaviors of U.S. Department of Commerce employees using an online survey [7]. They found that employees had an average of 9 work-related accounts that required login, with a median of 5 of those accounts being frequently used. Around 40% of employees wanted the workplace to transition to a SSO system to help them manage passwords. To reduce the burden of remembering or storing unique passwords for each workplace account, employees relied on a variety password reuse strategies: 67% made minor changes to existing passwords, 43% reused an existing password, 38% recycled a previously expired password, and 34% created new passwords by modifying a common "password root" (e.g., 2PwdRt&, PwdRt42%). When asked what an ideal login process would entail, employees did not show a strong preference for using PINs over passwords; although most employees preferred a system that used biometrics or the badges they already carried, employees mentioned passwords about as often as PINs, at approximately 5% in both cases.

Pearman et al. examined password reuse for online accounts in an in situ study of 154 participants [8]. In their study, the average participant used approximately ten unique passwords. Participants heavily relied on password reuse, with 40% of participants reusing 80-90% of passwords. For the average participant's unique passwords, 67% were exactly reused, 63% were partially reused, and 79% were reused in both ways (exactly and partially). Most of this partial reuse involved simple changes to existing passwords (the most common reuse strategy involved changing a single character) and more than half of the time these changes were consolidated at the beginning and end of the reused passwords. Passwords for work-related accounts were approximately three times as likely to be reused compared to passwords for non-work-related accounts. Participants also reused passwords

[1]Part of this is excerpted from the related work section of CHI 2017 [2].

across categories of websites, often in unsafe ways. For example, 85% of passwords on financial sites were reused (either partially or exactly), and of these, 95% were reused for other types of websites.

Zhang et al. studied password reuse in the context of password expiration policies [9]. They developed an algorithm that would use knowledge of a user's previously expired passwords combined with a set of password transformation rules to guess that user's newer replacement password. They evaluated their algorithm by testing it against password histories associated with defunct university accounts. Their results confirm that many users create new passwords by modifying old ones, and that this information can be leveraged by attackers to improve password guessing success. Even when limiting password guesses to five attempts, an attacker with knowledge of a previously used account password could use their algorithm to successfully guess a future account password for an average of 17% of accounts.

Haque et al. examined how easily passwords created for a user's low-value accounts could be leveraged to crack passwords created by the same user for high-value accounts [10]. Participants in their study were asked to create passwords for different categories of websites. Website categories differed according to the degree with which a user might be incentivized to secure an account in that category. For example, participants were asked to create a banking account password, as well as a password for a weather forecasting website. They then attempted to use the John The Ripper (JTR) password cracking tool to guess participant's high-value account passwords, using only their low-value account passwords (i.e., no other cracking dictionary). Nearly 20% of high-value account passwords were able to be guessed in this way. This supports the idea that users partially reuse password across account categories, even in unsafe directions (reusing low-value account passwords for high-value accounts), and that a significant portion of unsafe partial password reuse can be guessed using a default configuration of a freely available password cracking tool.

3.1.2 Measuring resistance to guessing attacks

Security researchers have attempted to quantify the *guessability* of secrets, which is the resistance of secrets (e.g., passwords, PINs) to guessing attacks. Current metrics for quantifying the guessability of secrets fall into two categories: metrics based on statistical properties of the secret distribution and metrics based on using (or simulating) actual secret-cracking tools.

Statistic-based metrics can be further divided into entropy-based metrics and guess-based metrics. Among entropy-based metrics, Shannon entropy has seen the most use. NIST provided a heuristic for estimating the Shannon entropy of passwords, which studies have used and extended [11]. Shannon entropy can be interpreted in many ways. It characterizes the "uncertainty" of a secret distribution, or equivalently how many bits are needed to optimally compress a distribution. While this can be a useful metric of resistance to guessing attacks, especially in the case of cryptographic keys that are randomly generated, in general it has no direct relationship to guessability and can be significantly influenced by rare events (e.g., a small set of users that choose large randomly-generated cryptographic keys as their password) [12]. This limits its use for quantifying the guessability of user-chosen secrets. Another useful metric is min-entropy, which quantifies security based only on the most probable secret.

Guess-based statistical metrics more closely align with a model of an attacker that makes sequential guesses, e.g., an attacker that tries to guess common passwords to break into a set of accounts. *Guesswork* quantifies the expected number of guesses that would be required to correctly guess a set of secrets, assuming the attacker orders her guesses in an optimal way, i.e. guessing the most probable secret first, with subsequent guesses in order of decreasing probability. Bonneau argues that guesswork does not reflect resistance to actual attackers, since it averages the number of guesses needed to compromise all accounts, whereas in practice attackers may ignore harder-to-guess secrets and where they may be limited by a security policy in the number of guesses they can attempt [12]. Partial guesswork metrics can capture this. *Marginal success rate* or *β-success-rate* measures the expected fraction of accounts that can be compromised (or, equivalently, the expected probability of breaking an individual account) if an attacker is limited to β guesses per account. *α-guesswork* measures the number of guesses required by an attacker to compromise a proportion α of accounts (or, equivalently, to have an expected probability α of breaking an individual account) [12]. [2]

The other general category of guessability metrics are metrics based on actual secret-cracking tools. A variety of cracking tools exist, all of which are tailored to cracking passwords: oclHashcat and John the Ripper use wordlists and mangling rules to quickly generate guesses, probabilistic context-free grammar (PCFG) tools generate guesses according to a grammar-based model that represents passwords using character class terminals (e.g., letters, digits, symbols), and Markov model tools generate guesses according to a trained character-level Markov model. The most recent development in cracking tools is the modeling of passwords using artificial neural networks. The features used by a neural network are generated as a result of training and not easily interpreted.[3] By attacking a password dataset and observing the point (guess number) at which each password is guessed by a particular tool, a distribution of guess numbers can be generated. This can be translated into useful metrics such as the expected percentage of passwords guessed for a given number of guesses, similar to the marginal success rate metric. A refinement of this approach is to combine guess numbers for a set a popular password-cracking tools into a summary metric. For example, researchers have created Password Guessability Service (PGS), a service which allows other researchers to generate *min-auto* guess numbers for sets of passwords [13]. For a given password, that password's min-auto guess number is defined as the smallest guess number among five password-cracking tools: oclHashcat, John the Ripper, and three password models (Markov, PCFG, neural network). Min-auto guess numbers were found to approximate the guessing success typical of expert attackers [13].

Guess-based statistical metrics and tool-based guessability metrics are closely related; both attempt to characterize resistance to an attacker. The primary differences between them are the assumptions about the particular guesses made by an attacker, and the order in which they are made. Guess-based statistical metrics are useful for characterizing security against a best-case attacker who has complete knowledge of the distribution of secrets

[2] This improves upon a related metric, *α-work-factor* (also known as marginal guesswork), by modeling the fact that an intelligent attacker will stop guessing for an account once one of those guesses succeeds.

[3] The features learned by a neural network that models passwords could theoretically include the same features used by PCFG or Markov models, as long as those features are useful for accurately modeling passwords in the training set.

and orders guesses for an account starting with the most probable secret. This provides a worst-case-scenario security analysis, which may be appropriate for high-risk applications. Their weakness is that they may overestimate security concerns; no practical attacker will have complete knowledge of the secret distribution, and these distributions will differ based on many contextual factors, including the sub-population being attacked [12]. More realistic characterizations of security can be provided by tool-based metrics, but the ability to generalize conclusions from these metrics will necessarily be limited; cracking tools can be configured and trained in many different ways besides those that were used to calculate metric values.[4]

3.1.3 Password-creation guidance

Prior work by Ur et al. [14] demonstrated that data-driven password meters are useful for helping users create more guess-resistant passwords. Besides the standard password strength meter, the password meter in that work introduced text feedback intended to guide users toward stronger passwords, by explaining aspects of the user's current password that could be improved. In that study, text feedback was the result of a regression of min-auto guess numbers onto 21 heuristic features associated with guessable passwords. For each heuristic feature, the meter would detect if that heuristic was present in the current password. It would then display up to three pieces of text feedback describing any detected heuristics and how addressing them could strengthen the password. Results from a Mechanical Turk user study in which participants created passwords using a password meter incorporating such text feedback showed that the feedback helped participants create stronger passwords.

Concrete password suggestions were also explored. These suggested improvements were optional, and took the form of a slightly modified password that users could choose to replace their current password. The slightly modified password was the result of an algorithm that made modifications in a random manner to remove heuristic features commonly used in passwords. The modifications that were made included those addressing aspects besides those shown in the actual text feedback, in order to increase the space of possible modifications utilized in the suggested improvements [14].

User study results reported by that same work found most participants did not choose to explicitly show suggested improvements. Among those that did, most found it useful, and a slight majority reported that it helped make their password stronger, albeit not by choosing suggested improvements verbatim, but more often by serving as inspiration for what to change in their existing password. Those that did not find the suggested improvement useful were either concerned about password memorability when using exact suggestions or about the trustworthiness of the algorithm creating those suggestions. Based on these results, Ur et al. recommended keeping concrete password improvements as optional [14].

[4]Researchers attempt to mitigate this by using configurations and training data representative of a sophisticated attacker (e.g., configurations recommended by Hashcat, publicly leaked passwords). However, it is still possible that an attacker could use more effective configurations or additional training data to increase guessing effectiveness.

3.2 Interpretable explanations for text-input black-box models

A variety of methods exist for extracting features from text that are useful for machine-learning predictions about that text. For character-levels models, where input instances take the form of strings of characters, straightforward representations include one-hot or index-integer encodings, which indicate each character in the instance by its index in the input alphabet. Character-level embeddings can be used to transform input characters to a lower dimensional, continuous feature space. Summary features based on n-grams can also be used, which quantify frequencies of each n-length character sequence in a given input instance. Features can also be hand-crafted based on expert knowledge or experimentation. For example, a heuristic feature indicating whether a password starts with an uppercase letter may be useful for predictions about that password's strength.

While these approaches can extract features useful for machine-learning algorithms, some are more easily interpretable than others (e.g., hand-crafted heuristic features compared to character embeddings). In addition, while explanations of a model's prediction may require reference to interpretable features, interpretable features themselves are not necessarily sufficient for an interpretable explanation.

3.2.1 Dimensionality reduction

One barrier to interpretable explanations is if explanations depend on a large number of (potentially interacting) features. To address this, dimensionality reduction techniques can be used. A common technique is to apply principal component analysis (PCA) and represent features in terms of their principal components. While useful for generating uncorrelated input features, principal component features can be difficult to interpret, which limits their use for explaining model predictions. An approach to dealing with this is to describe what each principal component represents in terms of more interpretable features, by observing which interpretable features correlate with each principal component. For example, Smith et al. used clustering on PCA features to investigate how dietary patterns associate with socio-demographic variables [15]. Pearman et al. also clustered study participants according to their password reuse behavior and described clusters in terms of the characteristics of passwords users in those clusters created [8]. A downside of this general approach is that it won't directly lead to discover of new interpretable features.

Autoencoders Another way of reducing dimensionality is to compress data using an *autoencoder*. Autoencoders are a type of neural network that are often used for unsupervised feature learning. In an autoencoder, the input to the network is also the target output of the network. They are typically used to compress input data by learning an encoding of that data in a lower dimensional space. An autoencoder consists of an encoder, decoder, and a bridge, where the bridge refers to the innermost hidden layer separating the encoder and decoder that captures the encoding. An important characteristic of encodings produced by autoencoders is that they are data-specific; they learn an encoding useful only to represent the specific data they were trained on (or data closely resembling that data).

Given a sufficiently high capacity encoder or decoder, autoencoders can simply learn the identity function without learning any useful features of the input data. Many possible approaches exist for reducing dimensionality in autoencoders, including undercomplete autoencoders, denoising autoencoders, variational autoencoders, and contractive autoencoders. These approaches help the autoencoder to capture only those input features most salient for reconstruction.

Visualization of clusters Dimensionality reduction can be combined with clustering algorithms and visualization techniques to elucidate high-level concepts that are represented in a reduced-dimensionality feature space. Principal components for a set of instances can be projected into two dimensions and visualized. Clusters of of data instances may then convey a type of data similarity captured by that feature space. This is similar to how the t-distributed stochastic neighbor embedding (t-SNE) visualization technique visualization works. Researchers have visualized hidden layer activations of neural networks using both t-SNE [16] and PCA projections [17] in order to understand features utilized by those networks. A challenge with this approach is that visualization of clustered instances may not reveal interpretable concepts associated with those clusters, especially for more complex concepts.

3.2.2 Approximate complex model with simpler, interpretable model

A general approach to explaining complex models is to train a simpler, but more interpretable model to approximate the predictions of the complex model. For example, Ur et al. utilize linear regression of password guessability on high-level heuristic password features to approximate a more complex neural network [14]. A downside of this approach is that predictions made by a globally-trained linear model may not reflect the complex model's behavior for specific input instances.

Ribeiro et al. introduce LIME, a technique for providing locally-faithful explanations for black-box models based on interpretable features from a simpler model, e.g., a sparse linear model [18]. This is similar to the approach taken by Ur et al. [14], except that in LIME, the simpler linear model is generated specifically for a given input instance. This means that learned weights for the linear model may differ from a globally learned model; the simpler model is trained to be faithful to the behavior of complex model locally around the input instance. A drawback of LIME is that it can be expensive to implement for networks with a densely-populated input feature space [19]. In addition, finding a suitable simpler model (including its features) is a requirement for LIME. For a given complex model, not all linear models will be powerful enough to explain the complex model's predictions. Further, a linear model may not be sufficient to produce locally-faithful explanations under LIME if the complex model is highly non-linear even in the restricted local space [18].

3.2.3 Relevance of input features in neural network predictions

Another approach to explaining the complex neural network predictions is to understand the relevance of input-space features for particular predictions output by the model.[5] Although

[5]Researchers have used "relevance," "attribution," and "contribution" to mean the same thing [20]. Leino et al. distinguish "influence" from "attribution," reserving "influence" for measures based on gradient-based sensitivity [21].

these approaches each utilize neuron activations in the network to characterize input feature relevance, the manner in which they are utilized differs between approaches. Most prior work has explored input feature relevance in the context of pixel-wise explanations for convolutional neural network predictions, however, many of the same methods can be applied to explain predictions for text-input neural networks, e.g., the relevance of characters in a password instance on that password's strength classification.

Identify input instances that activate hidden units Prior work has attempted to determine the relevance of input features using hidden unit activations invoked by specific sets of input instances. Karpathy et al. examined character-level recurrent neural networks models for text. In that work, they searched for high-level features learned by neurons, as expressed by patterns in their activation levels as they process text input exhibiting specific high-level characteristics [22]. Although many cells did not have interpretable patterns, they did find some interpretable cells, including cells that identified line length, cells that tracked quotes or brackets, and cells that turned on inside source code comments. A similar approach is to identify sets of input instances that trigger similar activations in a set of hidden units; similarities in those input instances may reveal the function of the hidden units, which can help explain network predictions. Researchers have built interactive hidden-state visualization interfaces to help identify high-level semantic concepts learned by recurrent neural networks [23, 24]. While useful, the interactive nature of these tools is more suited for manual, hypothesis-driven feature space exploration and is also less useful for identifying character-level patterns in latent feature spaces with many dimensions. More critically, identifying high-level hidden space features learned by a network solely using unit activation levels is inadequate for discovering all relevant features; neurons can be highly activated without necessarily contributing to a prediction outcome [21].

Gradient-based sensitivity analysis Besides observing how much of a classification prediction can be attributed to input features, input feature relevance can be defined in terms how sensitive the classification prediction is to the particular value (sometimes referred to as gradient-based saliency). This approach utilizes the gradient of the output prediction to determine input feature relevance. Li et al. explored gradient-based sensitivity analysis for NLP models, examining which individual input words contribute to the prediction output by those models [25]. They also applied sensitivity analysis to hidden unit gradients as input words were consumed to understand the high-level features learned by hidden units. Although their analysis revealed some useful insights into internal network behavior, they found first order derivatives were unable to captures all the relevant information needed to interpret model classification. Specifically, if the units in the network were highly saturated at the time of gradient measurement, a relevant characteristic of the input may not be flagged as relevant for a given classification [25].

Integrated Gradients The potential for gradient-based sensitivity analysis to fail to identify all relevant input features can be explained by its lack of *sensitivity.* In order for an attribution method to satisfy sensitivity, instances that differ with respect to one feature only and have different predictions should assign a non-zero attribution to that feature [19].

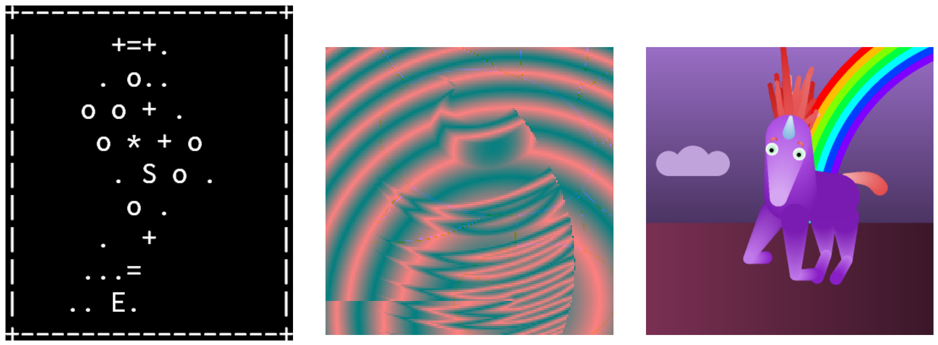
Sundararajan et al. propose *Integrated Gradients* to address this [19]. This approach also utilizes gradients of an output quantity for attribution, but instead of a single gradient, their attribution method accumulates the attributions of input variables at all points along the path from the baseline to the input instance (e.g., from the all-zero word embedding vector to the input instance word embedding). This approach was shown to be more effective at highlighting relevant input features for a given network output. Dhamdhere et al. utilize Integrated Gradients as well, but do so to identify the function and importance of hidden nodes in terms of input features [26]. Their attribution metric, *conductance*, is measured by the flow of Integrated Gradient attributions through particular hidden nodes.

Influence-directed explanations Leino et al. combine the gradient-based analysis and Integrated Gradients approaches to determine high-level features influential for a particular model outcome [21]. They also utilize network gradients, however, they calculate the gradient of an outcome of interest with respect to activations in a hidden layer of the network. They then identify class-specific "experts", sets of hidden layer neurons that are most influential for a given classification (those having high *internal influence*). Finally, they use the Integrated Gradients method to back-propagate expert activations to the input feature space, in order to visualize high-level concepts. A benefit of using internal influence to highlight input variable relevance is that high-level, rather than instance-specific, features can be learned and highlighted.

Layer-wise relevance propagation An alternative approach to identifying input feature relevance is to use relevance-score algorithms, in particular layer-wise relevance propagation (LRP) [27]. Ding et al. apply LRP-based visualization to explain internal network behavior in a neural machine translation model. Specifically, they examine hidden state activations in the network to infer the relevance of source and target word contexts at generating target words [28]. Arras et al. describe how LRP can be applied to recurrent neural network models containing LSTM layers [29]. Recently, Kauffmann et al. showed how LRP can be used to explain cluster assignments using NEON, using a two step approach in which a k-means model is first converted into a neural network, following by application of existing LRP techniques [30]. A criticism of LRP as an attribution method is that it breaks implementation variance, a property that requires two functionally equivalent networks to always produce identical attributions [19]. A consequence to this is that prediction explanations could be attributed to spurious correlations if collinear dimensions exist in the feature space.

3.3 Public-key fingerprints

Public-key encryption can be used to secure communications such as email or instant messaging. For Alice to send a message to Bob that only Bob can read, she needs to encrypt the message with Bob's public key. Bob will use his private key to decrypt the message. One type of attack against such communications is a man-in-the-middle (MITM) attack, in which an attacker inserts himself or herself between the two communicating parties in such a way that neither party is aware of the attacker. Once inserted, the attacker can eavesdrop or actively manipulate communications.



```
+-----------------+
|      +=+.       |
|     . o..       |
|    o o + .      |
|     o * + o     |
|      . S o .    |
|       o .       |
|     .  +        |
|    ...=         |
|   .. E.         |
+-----------------+
```

(a) OpenSSH Visual Host Key (b) Vash (c) Unicorn

Figure 1: Examples of graphical fingerprint representations.

In order to prevent MITM attacks, it is important that Alice encrypts her messages with the actual public key belonging to Bob (and vice versa). In many situations, there may not be a trusted and secure key server that Alice can use to obtain Bob's legitimate key online. In these cases, Bob can still host his public key online somewhere that Alice can retrieve it. Then, Bob only needs to provide Alice with his key *fingerprint*—a digest of the full key that makes manual key comparison feasible—through some secure channel (such as in person). By manually comparing fingerprints, both sides of the communication channel can detect when the other side has been replaced by an imposter.

3.3.1 Fingerprint applications

Well-known applications that make use of fingerprints include GnuPG [31], a tool for encrypting communications, and OpenSSH [32], commonly used for remote access to servers. Off-the-Record (OTR) Messaging applications provide multiple ways to authenticate message recipients, including fingerprint verification [33]. Many popular secure chat smartphone apps also use fingerprints, usually in a layered approach in which fingerprint comparison is optional [34, 35].

A variety of fingerprint representations and formats are used, the majority of which are textual. Examples of textual representations used in real systems are shown in Table 1. Graphical representations have seen limited use; examples include Peerio [36], which uses an avatar representation, and OpenSSH Visual Host Key, which resembles ASCII art (Figure 1).

3.3.2 Usability of fingerprints

Public key cryptography has long had usability and adoption issues [37–41]. Fingerprints have been part of the problem. In a user study on OTR messaging, participants were confused by fingerprints and struggled to verify them correctly [42]. In order to make fingerprint verification more usable, researchers have explored a variety of approaches.

One way to make comparison easier is to shorten fingerprints. This approach is used in Short Authentication Strings (SAS). SAS can provide reasonable security using only a 15-bit string [43], though they are primarily useful in synchronous environments. SAS-based meth-

GnuPG	`3A70 F9A0 4ECD B5D7 8A89`
	`D32C EDA0 A352 66E2 C53D`

OpenSSH	`ef:6d:bb:4c:25:3a:6d:f8:79:d3:a7:90:db:c9:`
	`b4:25`

bubblebabble	`xucef-masiv-zihyl-bicyr-zalot-cevyt-lusob-` `negul-biros-zuhal-cixex`
OTR	`4206EA15 1E029807 C8BA9366 B972A136 C6033804`

WhatsApp	`54040 65258 71972 73974`
	`10879 55897 71430 75600`
	`25372 60226 27738 71523`

Table 1: Examples of textual fingerprint representations used in actual applications.

ods are less useful for public-key fingerprint verification, which is traditionally asynchronous. Alternatively, the computational security of small fingerprints can be increased by slowing down the hashing algorithm using a stretching function.

Adding structure to a fingerprint representation may make it easier to compare. Textual fingerprint representations can be separated into smaller chunks. Representing fingerprints as pronounceable words or sentences may also facilitate comparison. For graphical representations, structured images resembling abstract art have been suggested for improving usability. These include Random Art [44] and OpenSSH Visual Host Key, which was inspired by Random Art [45]. Another way to add structure is to represent fingerprints as avatars, such as unicorns [46] or robots [47].

Different comparison modes have also been proposed, primarily for SAS-based device pairing or synchronous authentication [48,49]. These include compare-and-confirm (compare two strings and indicate if they match), compare-and-select (compare one string to a set of others and select the matching option), and copy-and-enter (copy a string from one device to another and let the device itself perform the check). Compare-and-select has also been used for anti-phishing tools that ask users to select the website they want to visit from a list, rather than ask for a yes/no answer to whether users would like to proceed to a given website [50]. Prior work has postulated that compare-and-select may help prevent users from "verifying" fingerprints without actually comparing them [48, 51].

A number of usability studies on device pairing have explored different fingerprint representations and comparison modes [52–54]. They considered representations such as numbers, images of visual patterns, and phrases, as well as various comparison modes. In each of these studies, participants performed fingerprint comparison tasks in a lab setting. In some cases, participants were also subjected to a simulated MITM attack [52, 54]. These studies had mixed findings; for example, Kainda et al. conclude that compare-and-confirm and compare-and-select should not be used because it is subject to security failures [52], while Kumar et al. recommend that compare-and-confirm with numbers be used due to its low error rate and comparison speed [54]. An important note about the fingerprints tested in these lab studies is the size of fingerprints tested, which ranged from 15-20 bits; for traditional asynchronous public-key fingerprint verification, a larger fingerprint size would be needed.

In a 400-participant online study, Hsiao et al. examined the speed and accuracy of fingerprint comparisons under different representations [55]. In contrast to lab studies, their methodology enabled statistical testing. A specific fingerprint representation they tested was Random Art [44], which can encode bit sizes large enough to be suitable for use as cryptographic key fingerprints. In order to generate attacks against Random Art fingerprints, researchers selected visually similar fingerprint pairs from 2000 randomly-generated fingerprints. They found Random Art performed well in both accuracy and speed, recommending its use for color-display devices with sufficient computational power.

Other than recent work by Dechand et al. [56], scant research has tested representations suitable for key fingerprints. That study involved a large-scale, within-subjects experiment on Mechanical Turk (MTurk) in which participants compared fingerprints displayed in different textual representations, including hexadecimal, numbers, words, and sentences. They measured comparison speed and accuracy, and they also recorded whether participants correctly compared fingerprints for multiple simulated 2^{80} attacks. They found that hexadecimal performed significantly worse than numbers and sentences in both attack detection rates and usability ratings. In addition, they found that sentences had a significantly higher attack detection rate than numbers, while also being rated as more usable.

3.3.3 Entropy as a security metric

Entropy can be used to measure the computational security afforded by different fingerprint representations. If users compare fingerprints fully, then the entropy of a fingerprint representation quantifies the average work needed to find a key whose fingerprint collides with the target fingerprint. If users do not compare fingerprints completely, but only compare certain aspects, then an intelligent attacker may attempt to only match the aspects he expects will be actually compared. This type of attack, in which the adversary attempts to find a visually similar fingerprint to the target fingerprint, has been explored for both the hexadecimal and graphical fingerprints used in OpenSSH [57, 58]. More recently, the previously mentioned study by Dechand et al. investigated users' ability to detect such attacks for different textual representations [56].

The reduction in entropy of the original representation (after fixing the matched aspects) can be used to quantify the work an attacker must spend to produce a key whose fingerprint matches specific aspects of the target fingerprint. This approach to quantifying attacker work has the added benefit of being independent of the particular binary encoding used for a fingerprint. A 2^{60} attack corresponds to an attacker that generates 2^{60} keys, computes the fingerprint for each, and then (manually or programmatically) selects the key whose fingerprint maximizes similarity according to some metric. Stevens et al. estimate the cost of renting CPU/GPU time from EC2 to find a SHA-1 collision, which requires resources similar to those to perform a 2^{60} attack on fingerprints. They estimate this cost to be between 75K and 120K USD, which they note is within the budget of criminal organizations [59].

4 Previous work

4.1 Can unicorns help users compare crypto key fingerprints?

4.1.1 Research goals

- Understand the security and efficiency of public-key fingerprint verification for users subjected to practical constraints, including habituation to benign scenarios, time pressure, and comparison difficulties.

- Explore fingerprint verification for graphical fingerprint representations and using the compare-and-select comparison method.

4.1.2 Methodology

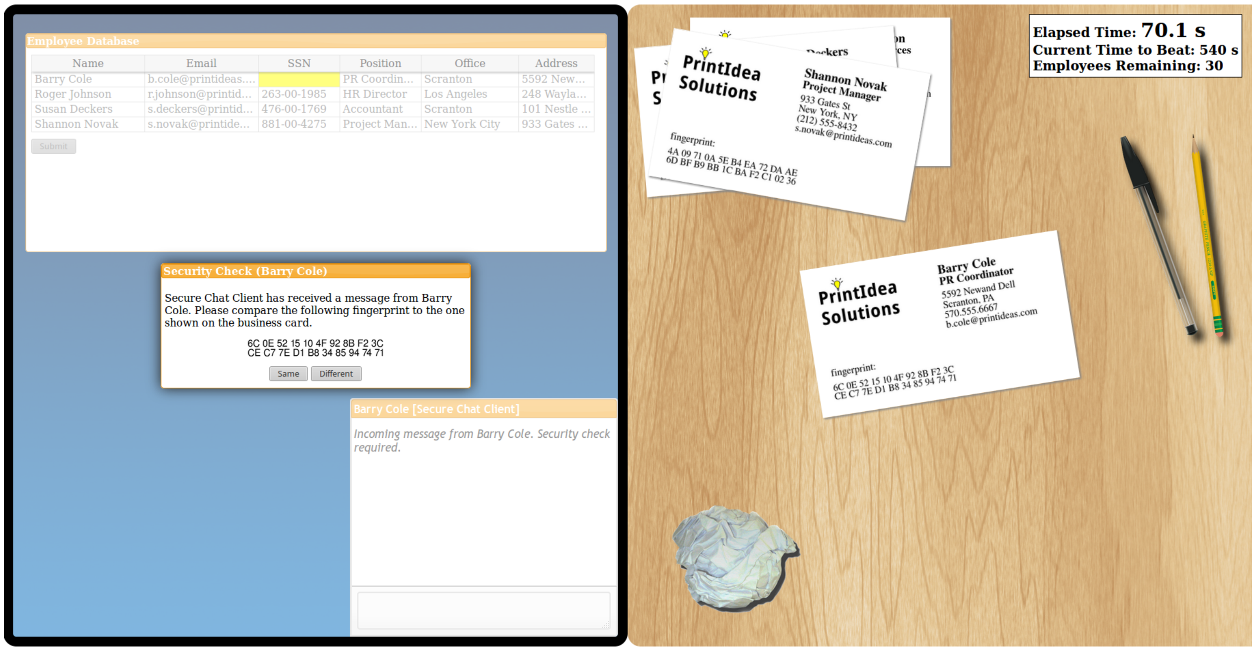
We conducted a between-subjects experiment to evaluate and compare the usability and security of fingerprint representations and configurations. We recruited participants from MTurk in August 2016. We required participants be 18 years or older and live in the United States. Our protocol was approved by our universitys IRB. We advertised the study as a "role-playing activity involving technology and communication in the workplace" that would take about 20 minutes. We compensated participants \$3, with the opportunity to earn a \$1 bonus. As our activity was not designed for use on tablets or smartphones, we asked that participants use a desktop or laptop computer. Participants were randomly assigned to a condition, which determined the fingerprint representation and configuration they saw.

We asked participants to imagine they worked as an accountant at a company that was updating its employee database. To perform this update, participants had to retrieve the social security numbers (SSNs) for 30 employees and enter them into a database. We chose SSNs to motivate the need for secure communication. In the U.S., SSNs are highly sensitive because knowing an individuals SSN can enable identity theft and have other financial ramifications. Our activity web page divided the browser window into two sections that mimicked the appearance of a computer screen and a desk (Figure 2). The computer screen section displayed a spreadsheet-like database where participants would need to fill in missing SSN details for employees. Several business cards were sitting on the desk. A stopwatch and information about the participants progress completing the task appeared in the top right corner.

We provided the instructions for the activity via an interactive tutorial. Participants could repeat the tutorial any number of times until they were comfortable proceeding, and the on-screen stopwatch did not begin until after the tutorial ended.

At the start of a task, a chat window appeared on the simulated computer screen informing participants of an incoming message from one of the 30 employees. To proceed, they had to perform a security check. Shortly afterward, a dialog box was displayed on the computer screen containing a fingerprint and instructions to compare it to the fingerprint on the employees business card, which appeared simultaneously on the desk.

For our baseline configuration, a security check involved comparing two fingerprints (one in the security check dialog box and one on the business card) and pressing a button to indicate if they were the same. We informed participants that this check was needed to ensure



Employee Database

Name	Email	SSN	Position	Office	Address
Barry Cole	b.cole@printideas....		PR Coordin...	Scranton	5592 New...
Roger Johnson	r.johnson@printid...	263-00-1985	HR Director	Los Angeles	248 Wayla...
Susan Deckers	s.deckers@printid...	476-00-1769	Accountant	Scranton	101 Nestle ...
Shannon Novak	s.novak@printide...	881-00-4275	Project Man...	New York City	933 Gates ...

Security Check (Barry Cole)

Secure Chat Client has received a message from Barry Cole. Please compare the following fingerprint to the one shown on the business card.

```
6C 0E 52 15 10 4F 92 8B F2 3C
CE C7 7E D1 B8 34 85 94 74 71
```

Barry Cole [Secure Chat Client]

Incoming message from Barry Cole. Security check required.

Shannon Novak
Project Manager
933 Gates St
New York, NY
(212) 555-8432
s.novak@printideas.com

fingerprint:
4A 09 71 0A 5E B4 EA 72 DA AE
6D BF B9 BB 1C BA F2 C1 02 36

Barry Cole
PR Coordinator
5592 Newand Dell
Scranton, PA
570.555.6667
b.cole@printideas.com

fingerprint:
6C 0E 52 15 10 4F 92 8B F2 3C
CE C7 7E D1 B8 34 85 94 74 71

Elapsed Time: 70.1 s
Current Time to Beat: 540 s
Employees Remaining: 30

Figure 2: Screenshot of the task. This example is the compare-and-confirm, simultaneously visible, hexadecimal condition.

a secure chat session and avoid potential eavesdroppers. Depending on which fingerprint representation was shown, we provided guidance on what differences participants should look for when comparing fingerprints. If the participant indicated that the fingerprints matched, the chat window displayed a message from the employee with her SSN. The participant was instructed to type the SSN into the database. If the participant instead indicated that fingerprints did not match, the chat message instructed the participant to instead enter "ERROR" in the database. Each participant repeated this task for 30 employees. For one of these employees, the comparison task involved a simulated MITM attack, in which the legitimate key fingerprint was replaced with a visually similar, but distinct fingerprint. The extent of fingerprint similarity was limited according to computational resources required to produce a key with that fingerprint, for various attacker strength levels.

We instrumented our activity to record participants database entries, security-task decisions, and detailed timing information. We also recorded their browser user agent strings to determine whether participants used a tablet or smartphone. For one condition in which users had to toggle between two views, we recorded the number and timing of toggles.

Afterwards, participants filled out a survey. In this survey, we told participants whether they had missed our attack and asked them to explain why they thought they missed or detected that attack. To aid in memory, we showed the fingerprint pair corresponding to the attack alongside these questions. We asked participants to describe their strategy for comparing fingerprints, respond on a Likert scale to statements about the fingerprints they saw, and provide general demographic data.

4.1.3 Results

- Compare-and-select approach in which fingerprint options are dissimilar performs poorly for key-fingerprint comparisons.
- For high-risk situations, none of the tested fingerprint representations or configurations are suitable, due to successful attack rates. In such situations, manual comparison should be avoided in favor of automated approaches (e.g., using QR codes).
- For low-risk situations where usability is paramount, graphical fingerprint representations may provide an intuitive and fast method of manual key verification. Given participants' tendency to look for large rather than small differences, an ideal graphical representation should make it computationally difficult to find visually similar fingerprint instances.

This study is complete and was published at CHI 2017 [2].

5 Ongoing and future work

5.1 Minimum-password-strength requirements vs. blacklists [ongoing]

Blacklists are an established approach used to prevent users from creating weak passwords, where a password might be considered weak because it is known to be commonly used (and thus a likely candidate an attacker would try) or because common configurations of popular password-cracking tools would guess that password. For example, NIST recommends blacklisting passwords known to be compromised [60].

In current work, we are taking a closer look at blacklists and neural networks (NN) in the context of password-creation meters. The high-level goal of this experiment is to help users create strong but memorable passwords. Specifically, we plan to experimentally test the effectiveness of two types of mechanisms for accomplishing this goal: blacklists and minimum neural-network-derived password-strength requirements. We examine the feasibility of replacing blacklist requirements with a requirement based on a neural-network-derived strength estimate (minNN). Neural-network-strength-based requirements may lead to stronger passwords, while also being easier to manage and deploy, especially when system administrators wish to prevent the creation of a large number of passwords.

A secondary goal of our study is to provide concrete recommendations to the Carnegie Mellon University Security Information Office on components to include in a future update to their password meter. One of our conditions constitutes the actual CMU password-creation policy (CMU-current), including the blacklist (and blacklist algorithm) it applies.

5.1.1 Experimental factors

We consider three experimental factors in which password policies can vary: blacklist (including blacklist matching algorithm), minimum neural-network-estimated strength, and composition policy specifying the number of required character classes and minimum password length.

Blacklist There are many ways to implement a blacklist. Blacklist implementations can be characterized by the list of strings in which checks are made and the matching algorithm determining whether a check returns as a positive or negative match. For blacklist lists, we test a wordlist constructed by prior work [14] (*CHI17 blacklist*); this blacklist also performed well in retrospective analyses on leaked data, by rejecting weak passwords with minimal impact to strong passwords. We also consider a blacklist consisting of 555 million passwords known to have been previously exposed as part of a data breach [61] (*HIBP blacklist*). Lastly, we test the blacklist used by Carnegie Mellon University for student, staff, and faculty university accounts.

We consider four types of blacklist matching algorithms: case-insensitive full-string (*ci-fullstring*), case-sensitive full-string (*fullstring*), an algorithm that strips numbers and digits and then performs case-insensitive full-string (*strip-ci-fullstring*), and a case-insensitive substring matching algorithm considering substrings five or more characters long (*ci-substring-5+*). The *strip-ci-fullstring* algorithm is similar to a recommendation made in prior work [62] and to the current CMU blacklist matching algorithm.

minNN For minimum neural-network-estimated password-strength cutoffs (*minNN*), we chose to experiment with 10^6 and 10^{12}. We came up with these thresholds based on results from retrospective-policy experiments on leaked passwords, according to two criteria: (1) the smallest threshold that would result in a cumulative percentage of passwords guessed that was lower than the percentage guessed under CMU-current at all guessing points from 10^0 to 10^{14}, and (2) similar to the first but with respect to a corresponding policy that replaced the minNN cutoff with a standard blacklist.[6]

Composition policy We experiment with composition policies ranging from the basic policy requiring passwords to be at least 8 characters long (*1c8*) to stricter policies requiring all four character classes (lowercase letters, uppercase letters, symbols, and digits) and a length of at least 8 characters (*4c8*). Besides 1c8, the particular policies we selected were either recommended in prior work (1c16, 2c12, 3c12, 4c8) [63–68] or seemed promising based on our retrospective-policy experiments (3c8).

5.1.2 Experimental conditions

Here are the 15 experimental conditions we will test:

1. 3c8
2. 4c8
3. 1c8
4. 3c8, minNN=10^6
5. 1c8, minNN=10^{12}

[6]In the current PGS implementation, enumeration-based approaches (JTR, Hashcat) make between 10^{13} and 10^{15} guesses.

6. 1c8, minNN=10^6
7. 3c12, minNN=10^6
8. 1c16, minNN=10^6
9. 2c12, minNN=10^6
10. CMU's current password policy[7]
11. 3c8, CHI17 blacklist w/ ci-fullstring
12. 1c8, CHI17 blacklist w/ ci-fullstring
13. 1c8, HIBP blacklist w/ fullstring[8]
14. 1c8, CHI17 blacklist w/ strip-ci-fullstring[9]
15. 1c8, CHI17 blacklist w/ ci-substring-5+

We chose not to test password policies that included combinations of minNN and blacklist requirements, as retrospective leaked password study analyses suggested the addition of a blacklist tended to do very little with respect to guessed passwords once a minNN threshold was imposed. We also chose to only test conditions that including a minimum password length of at least 8 characters, as our PGS password strength estimates assume this requirement. In addition, passwords consisting of less than 8 characters can be easily bruteforced, making them unsuitable for protection against offline attackers.

5.1.3 Neural network training

Different NN models were trained according to the following process: first, models were trained on PGS-compatible, 1c8-compliant password training data. Then, for non-1c8-policy models, we used transfer learning. Specifically, we started training those models from the final 1c8-model weights, potentially with feature layer freezing (we chose whichever guessed more effectively in initial experiments).

NN models were trained on a dataset that we constructed and denote $PGS3$, consisting of PGS-compliant passwords in the LinkedIn, Mate1, RockYou, and 000webhost datasets. Tables 2 and 3 describe these models. In the prior $PGS++$-trained NN model, training began from all PGS++ data (i.e. 1c1) and secondary training was performed on the subset of data satisfying the specific policy-model being trained (e.g., 1c8, 1c16). Freezing of feature layers was experimented with for 1c16. Another major difference for the $PGS3$ models is that they each include a word embedding layer as the first layer, instead of a one-hot encoding layer. There are also some differences between the JavaScript model architecture used in prior work by Melicher et al. [1] and our study; our TensorflowJS models use LSTM layers instead of the JRZ2 and time-distributed dense layers applied in their models.

We also trained two sizes of models, each trained using code extended from work by Melicher et al. [1] to use a Tensorflow-based backend: a large Keras model with 1000 fully-connected units and 512 LSTM units, and a small Keras model with 200 units each of fully-connected and LSTM units. We also create model variants that could be included in a webpage by converting the small Keras models into corresponding TensorflowJS models.

Currently we use a length-based heuristic score (length was the most impactful heuristic

[7]4c8, CMU blacklist w/ strip-ci-fullstring unless len(password) >= 20, max same character = 3 unless len(password) >= 20, max len = 100

[8]similar to NIST recommendation

[9]USEC17 [62] recommendation

Policy	PGS++ (old)	PGS3 (new)
1c8	73.4 million	32.8 million
1c16	2.5 million	1.5 million
2c12	13.3 million	5.4 million
3c8	13.6 million	5.7 million
3c12	4.1 million	1.8 million
4c8	311 thousand	599 thousand

Table 2: Training data used for PGS3 models. Each PGS3 model was trained on the subset of training data corresponding to the model's character class policy.

Policy	Large model	Small model
1c8	From scratch	From scratch
1c16	From 1c8 weights, w/ freezing	From 1c8 weights, w/ freezing
2c12	From 1c8 weights, w/o freezing	From 1c8 weights, w/o freezing
3c8	From 1c8 weights, w/o freezing	From 1c8 weights, w/o freezing
3c12	From 3c8-no-freeze weights, w/ freezing	From 3c8-no-freeze weights, w/ freezing
4c8	From 3c8-no-freeze weights, w/ freezing	From 3c8-no-freeze weights, w/o freezing

Table 3: Training methods used for large and small PGS3 models.

found in prior work [14]) to rate a password not meeting the character class and length requirements of a given password-policy model (the neural network was not trained on such data and so would produce inaccurate estimates for such passwords).

Our hypothesis is that minimum neural network strength requirements are a better mechanism for guiding users toward strong and memorable passwords than blacklists are. Initial retrospective experiments on leaked passwords, in which we take a random sample of leaked passwords, partition them into allowed and rejected sets, and then visualize the resulting guess curve, suggest that weak passwords are more often rejected when using minimum strength cutoffs. Our retrospective testing datasets included 000webhost, Yahoo, and Comcast password leaks.

5.1.4 Methodology

We plan to verify this hypothesis using a user study.[10] This will allow us to evaluate passwords created in a controlled environment created under a specific policy (as opposed to only retroactively applied). Prior work has shown that retrospective password studies can be misleading, e.g., retroactively subsetted 4c8 passwords originally from a 1c8 policy tend to be stronger than those created under a 4c8 policy [69]. Without a user study, we do not know what password a user would create in place of a blacklisted creation attempt; the best we could do to account for this would be to normalize allowed-password guess curves, which assumes that, after attempted a blacklisted password, users would pick new passwords randomly with respect to the allowed-password distribution. Also, a user study will allow us

[10]This user study has recently finished running; collected data is ready for analyses.

to investigate password creation usability and password memorability, which retrospective studies cannot examine. Additionally there are potential issues with the use of password leak data itself: we don't always know which particular constraints or requirements were imposed on passwords at creation time (and if this changed over time, i.e. effective policies differed for passwords in the same leak); and leaked passwords may be for accounts that users view as non-important, as so put less effort into protecting.

Changes to the password meter from the Ur et al. study [14] include: different implementation of NN models, no specific-password suggestions, randomized order of character classes in text feedback, and no use of heuristic scoring unless the neural network is incompatible with a particular user's web browser client (such data will be analyzed separately). In prior work, the lower of the log10 of the (scaled) neural network score and heuristic score was chosen as the final password score, which then was then translated into a bar fill amount [1]. In our work, we chose to no longer use heuristic score as a determining factor in password meter bar fill, unless the neural network did not function for technical reasons. Initial results suggests our updated neural network models password guesses more accurate, and so the heuristic score was deemed unnecessary.

We will measure frustration with particular password policies via study dropout rates between different conditions, as well as password creation and recall sentiment questions in the user surveys. When comparing password recall, we will only analyze data for participants who: typed in their password from memory (as self-reported in the survey); who said they didn't reuse their study password (as self-reported); and who didn't copy and paste their password in the recall tasks (based on keystroke data).

One potential confounding factor in comparing neural-network cutoff to blacklist requirements is that using different text feedback for the two types of mechanisms may lead to results that differ because of the different text feedback, not only because of differences in which sets of passwords were allowed or rejected. To address this, one approach would be to use the exact same text feedback for both, in order to evaluate the rejection mechanism separately from the feedback text. We chose not to follow this approach and instead show slightly different text feedback based on the rejection mechanism. Certain rejection mechanisms may lend themselves to easier-to-understand text feedback explanation and we wanted to capture this aspect in our evaluation.

In our current password experiment, feedback on whether a password meets requirements or not is shown interactively in real-time as the user types their password. Because of this, we don't know exactly when a given string constitutes something a user would have actually tried submitting versus when it represents a string they had no intention of submitting (i.e., they were still typing). This would be useful to know because it sheds light on password creation usability. We attempt to mitigate this issue using a few approaches: we instrument when the user presses the enter or tab keys while inside the password field (pressing the enter key does nothing in the meter, but users might press it out of habit when trying to submit a form), and in the post-creation survey we directly ask users if a password they wanted to create was rejected due to blacklist or minNN reasons (we show a screenshot of the relevant feedback; if they remember, we ask them to attempt to recall the rejected passwords as further support).

5.1.5 Planned statistical analyses

Specific statistical analyses will be conducted to answer research questions. We will compare the CMU-current condition to potential replacements, which include 3c8+minNN=10^6 and 1c8+minNN=10^{12}. In order to be a suitable replacement candidate, we hope to find a strict improvement (in terms of proportion of passwords guessed) over CMU-Current up through the 10^{14} range. If our user study does not show convincing evidence that such minNN policies are viable replacement candidates, our blacklist-based conditions may still suggest an improved policy; such policies may include "3class8+CHI17 blacklist w/ case-insensitive-fullstring matching" and "1c8+HIBP blacklist".

Other research questions will be answered by comparing two specific conditions differing with respect to only one factor, in order to understand the impact of that factor.

We will compare *class-composition+length* policies to *class-composition+length+minNN* policies to answer the question of whether introducing a minNN requirement leads to stronger passwords (as expected) without: making password creation take significantly longer (in a practical sense) or a frustrating process; resulting in passwords being created that users have a difficult time remembering or which they store insecurely; or leading to either strong passwords being rejected or weak passwords being allowed.

We will compare minNN conditions to corresponding blacklist conditions to understand whether minNN requirements can replace blacklists without reducing security against guessing attacks. An important aspect we want to consider is whether minNN requirements are more or less frustrating to users during password creation. In the case that a system administrator incorporates a minNN requirement into their organization's password policy, we also want to determine the appropriate configurations (composition policy and threshold value) that should be be recommended in order to provide comparable security against guessing attacks (by comparison to effective security blacklist-based policies would provide). For this will compare:

- "3c8 policy + CHI17 blacklist" to "3c8+minNN=10^6"
- "1c8 policy + CHI17 blacklist" to "1c8+minNN=10^{12}"

We want to understand the appropriate length requirements for policies incorporating both a character-class-composition requirement and a minNN requirement. We are interested in answering these questions: Do longer length requirements or more character-class requirements improve the security of passwords policies incorporating minNN requirements, or are length requirements captured by minNN requirements? Are minNN requirements more frustrating to users during creation or do they result in less memorable passwords? To answer these questions, we will compare the following conditions):

- "3c12+minNN=10^6" to "3c8+minNN=10^6"
- "1c16+minNN=10^6" to "1c8+minNN=10^6"
- "2c12+minNN=10^6" to "3c12+minNN=10^6"

A secondary question we will explore is: in the case that an organization cannot or will not use a minNN requirement, which specific blacklist configurations should be used?[11] To answer this, we will compare the following conditions to each other:

[11]We focus on recommendations for 1c8 policies. 1c8 policies with only a blacklist may not be appropriate for high-security settings, but in that case a minNN requirement may be appropriate anyway.

- 1c8 + CHI17 blacklist w/ case-insensitive fullstring matching
- 1c8 + HIBP blacklist w/ fullstring matching
- 1c8 + CHI17 blacklist w/ strip-symbols-and-numbers then case-insensitive fullstring matching
- 1c8+CHI17 blacklist w/ case-insensitive substring matching, with the minimal considered substring length being 5 characters

For each of the above, the specific statistical tests planned will compare both security and usability aspects. We will use the following tests:

- Pairwise (comparing 2 conditions)
  - Comparing proportion of weak passwords (strength $< 10^7$) that are allowed: Fishers exact test
  - Comparing proportion of medium-strength passwords ($10^7 <=$ strength $< 10^{13}$) that are rejected: Fishers exact test
  - Comparing proportion of strong passwords that are rejected (strength $>= 10^{13}$): Fishers exact test
  - Comparing percent of passwords guessed between at different guessing cutoffs: Fishers exact test (Possible cutoffs include 10^{14} (bruteforce, offline attack) and 10^3 (extended/gradual online attack)
  - Comparing guess curves: Log rank test
- Omnibus
  - Comparing proportion of medium-strength/strong passwords that are rejected, or proportion of weak passwords allowed: Chi-squared test of independence

Lastly, we wish to determine the impact of each individual factor in a password policy on password strength. This includes interactions between factors (e.g., in the case a blacklist only impacts guessability if there is the policy lacks a minNN requirement). To answer this, we will conduct a Cox proportional hazards regression using data from all conditions except CMU-current (since it is unusual in how it differs from the other policies). In this regression, the dependent variable will be password guessability as measured by PGS min-auto. We could also explore other dependent variables such as whether participants recalled their password in part 2 (using a logistic regression model). The independent variables in the regression will be:

- Minimum password length (8, 12, 16)
- Number of required character classes (1-4)
- minNN threshold (0/none, 10^6, 10^{12})
- Blacklist file and matching algorithm (none, CHI17+ci-fullstring, HIBP blacklist w/ fullstring, CHI17 blacklist w/ strip-ci-fullstring, CHI17 blacklist w/ ci-substring-5+)

We plan to consider all two-way interactions between independent variables and apply backwards model selection based on BIC.

5.2 Modeling password guessability with autoencoder-derived heuristic features

5.2.1 Motivation

Participants in the study by Ur et al. [14] reacted overall positively to password meter text feedback. However, there remain ways in which this feedback might be improved to encourage stronger passwords. This includes showing password-creation text feedback based on a more comprehensive set of commonly used password patterns and highlighting the specific parts of the candidate password that text feedback pertains to.

Identify additional guessability heuristics The heuristic features leveraged in prior work for text feedback [14] were identified by experts based on commonly found patterns in leaked passwords. Although these heuristics were found sufficient to model password guessability reasonably well, the identification of that set of heuristics was a manual process and thus the resulting heuristic set may not be comprehensive. Other commonly used patterns may exist that have evaded detection by experts but which an intelligent adversary might nonetheless leverage to increase the chances of attack success. In order to find such patterns, a more data-driven method for identifying common password patterns is desirable.

Highlight password characters associated with identified heuristics A component of the password meter used in prior work [14] that is related to text feedback was concrete suggested improvements to users' candidate passwords. Although suggested concrete improvements to passwords may not always be desired or trusted by users, a component of them that was largely viewed positively in that work was how the suggestions highlighted specific parts of users' existing password that could be changed. A possible modification to text feedback is to incorporate this aspect, by highlighting the specific part(s) of the password that feedback is associated with. This might be especially useful for conveying complex heuristic patterns in users' passwords that a data-driven approach to identifying heuristics patterns might identify. In addition, initial results from user studies on password meters incorporating minimal password-strength requirements suggest users can become frustrated when they do not know what to change to satisfy such strength requirements. A mechanism for highlighting the pertinent parts of users' existing passwords that they could then modify to satisfy minimum strength requirements would be useful in these cases.

5.2.2 Proposed methodology: Overview

Details of the proposed methodology differ depending on the specific plan chosen. For all plans, an autoencoder will be trained in order to perform unsupervised feature learning of high-level password patterns. The next step will involve grouping a set of password into clusters based on the high-level patterns they contain. This will be accomplished using a clustering approach (by training a deep k-means neural network or by applying standard k-means to the learned encoding).

In order to explain cluster patterns using semantic, interpretable descriptions of password patterns, we will have experts manually inspect the passwords assigned to each cluster. If deep k-means was used, we will first implement LRP to visualize the specific characters attributed to each cluster, which may help identify complex patterns. Regardless of the clustering technique used, we can also visualize cluster soft-assignments (distances to each cluster) for a set of passwords using two-dimensional projections.

Next, for the set of heuristics we were able create meaningful and interpretable explanations for, we will perform a linear regression of NN guess numbers onto those heuristics. We will also update the regression models used by prior work [14] to reflect guess numbers for newly constructed NN models. Finally, we will evaluate the ability of our newly constructed heuristics to model password strength by evaluating and comparing goodness-of-fit measures for each model. We will also analyze safe and unsafe errors to explore possible future model improvements.

Here are outlines of three possible study plans, in terms of the specific study components each plan would require. The following section will expand on each study component.

Plan A Identify heurstics with help of LRP visualizations

1. Train autoencoder
2. Train deep k-means neural network
3. Implement LRP visualization
4. Expert identification of heuristics associated with each cluster
5. Perform regression of NN guess numbers onto new set of heuristics
6. Perform error analysis and goodness-of-fit evaluations

Plan B Use deep k-means for heuristic clustering

1. Train autoencoder
2. Train deep k-means neural network
3. Expert identification of heuristics associated with each cluster
4. Perform regression of NN guess numbers onto new set of heuristics
5. Perform error analysis and goodness-of-fit evaluations

Plan C Use standard k-means for heuristic clustering

1. Train autoencoder
2. Perform standard k-means clustering
3. Expert identification of heuristics associated with each cluster
4. Perform regression of NN guess numbers onto new set of heuristics
5. Perform error analysis and goodness-of-fit evaluations

5.2.3 Proposed methodology: Component details

Train autoencoder (all plans; 4 weeks) As a first step in generating password-specific text feedback, we plan to use a password autoencoder. This autoencoder will take a password as input, encode it into a reduced-dimensionality representation, and decode that password encoding back into the original input space. The architecture of the encoder will consisted of embedding layer, stacked bi-directional LSTM layers, following by a fully-connected layer as the autoencoder bridge (Figure 7).

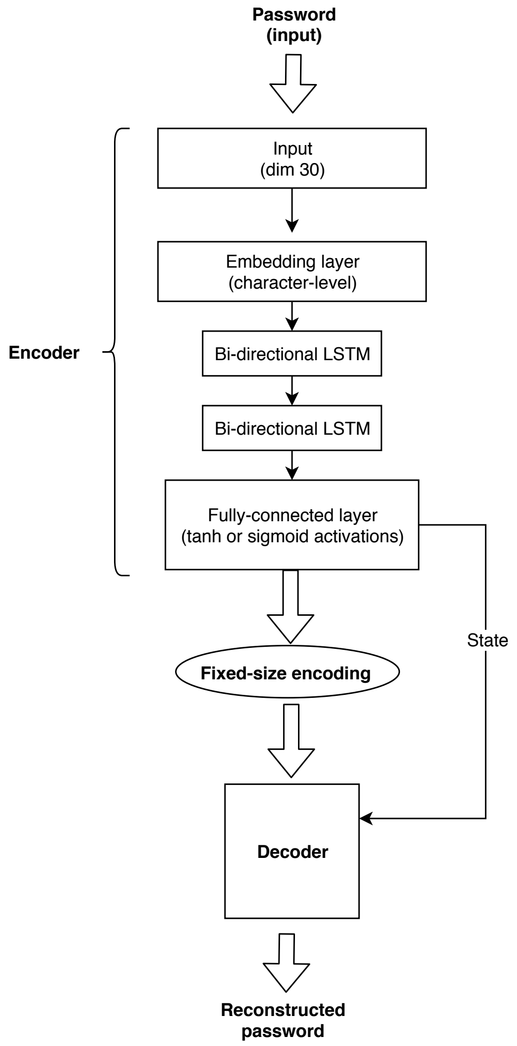


Figure 3: Proposed autoencoder architecture. The decoder weights will be initialized with the state of the encoder weights. Once trained, the encoder could have it weights frozen and be used independently of the decoder.

We plan to implement our autoencoder using the seq2seq encoder-decoder framework for Tensorflow [70]. We will experiment with differently-sized models, varying in number of nodes in the encoding, decoding, and embedding layers. All models will incorporate a bidirectional recurrent neural network (RNN) encoder and a unidirectional RNN decoder. Both encoder and decoder will use either LSTM or GRU cells for activation units. For all models, a single fully-connected hidden layer (the bridge layer) will separate the encoder and decoder. The variant of autoencoders we will use is undercomplete autoencoders, which reduce dimensionality by constraining the bridge layer to have a dimension smaller than that of the input. This forces the autoencoder to capture only the input features most salient for reconstruction.

We will quantify the ability of the autoencoder to recreate passwords using the BLEU (Bilingual Evaluation Understudy) metric, commonly used for evaluating the accuracy of machine translation. Although we can compare our password autoencoder to the performance of other autoencoders, our ultimate goal is not to reconstruct passwords exactly, but rather to reconstruct passwords in terms of their high-level features.

Train deep k-means network (Plans A, B; 2-3 weeks) To understand password concepts represented in the compressed feature space and to enable concepts to be mapped to specific parts of input passwords, we will apply deep k-means clustering [71] to the password encoder model (Figure 4). Prior work has demonstrated how to perform LRP for recurrent neural networks with LSTM layers [29]. Recent work has also shown how k-means clustering can be "neuralized," enabling LRP-based explanations for cluster assignments [30]. For the neuralized k-means network, the optimization problem is:

$$\min \sum_{ik} \delta_{ik} ||Enc(x) - \mu_k||^2$$

where δ_{ik} is an indicator variable for whether data point i is assigned to cluster k, $Enc(x)$ is the encoding of password x, and μ_k is the centroid of cluster k [30].

The benefit of neuralizing k-means is that LRP can then be used to highlight specific characters in a given password that are associated with each learned cluster (Figure 5).

An alternative approach would be to set the encoder to have k units in the bridge and avoid an additional clustering step. However, this would be less flexible for modifying the number of bridge layer units. By introducing the deep k-means step, we do not need to retrain the autoencoder in order to change the number of learned clusters.

The clustering algorithm will be applied to randomly selected subset of the autoencoder training data.

Perform standard k-means (Plan C; 2 weeks) We will perform k-means clustering not using the deep k-means network, but instead using the standard k-means algorithm on a set of password encodings (the output of the encoder layers).

Implement LRP visualization (Plan A; 2-3 weeks) We will apply LRP to back-propagate soft-assignments to clusters (distances to cluster centroids) to the input layer of the network. This will allow us to create heatmaps of relevant input characters for each heuristic-mapped cluster. Figure 6 demonstrates how LRP-based cluster-assignment explanations could be applied to passwords.

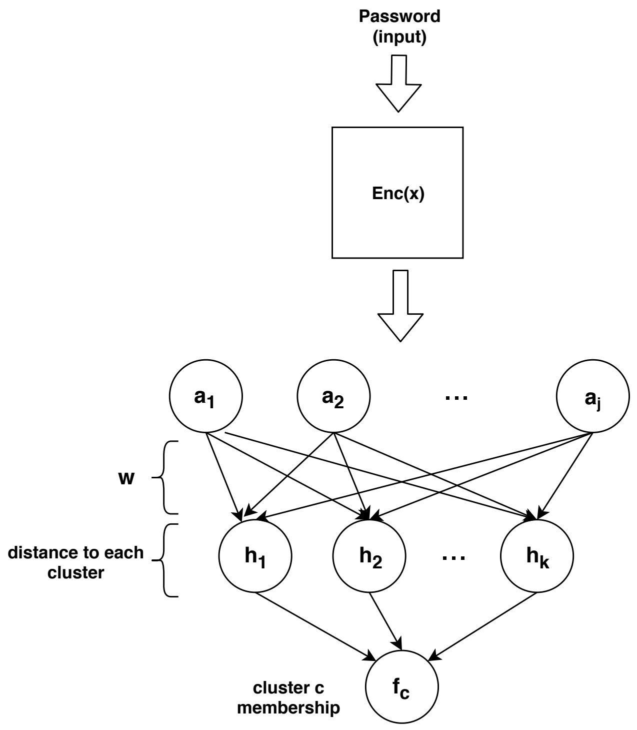


Figure 4: Architecture of neural network for performing deep k-means.

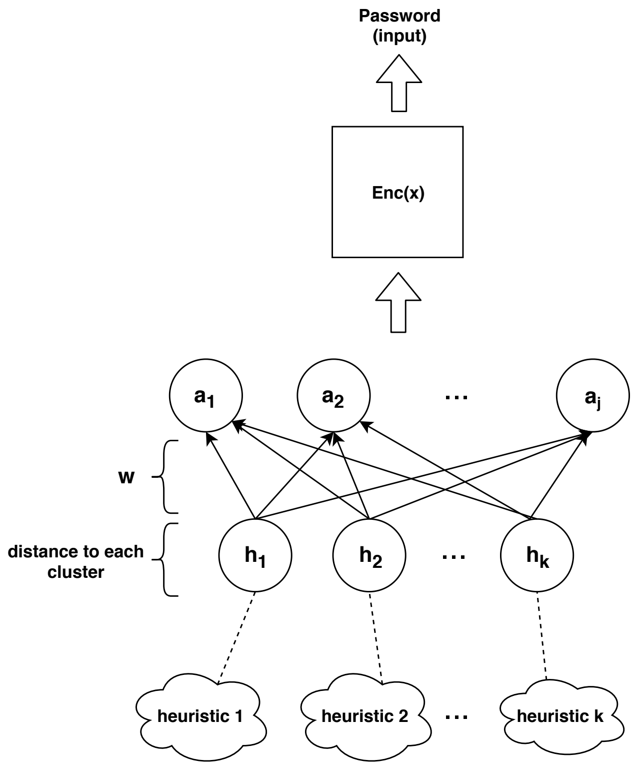


Figure 5: Illustration of LRP for associating individual password characters with a learned heuristic.

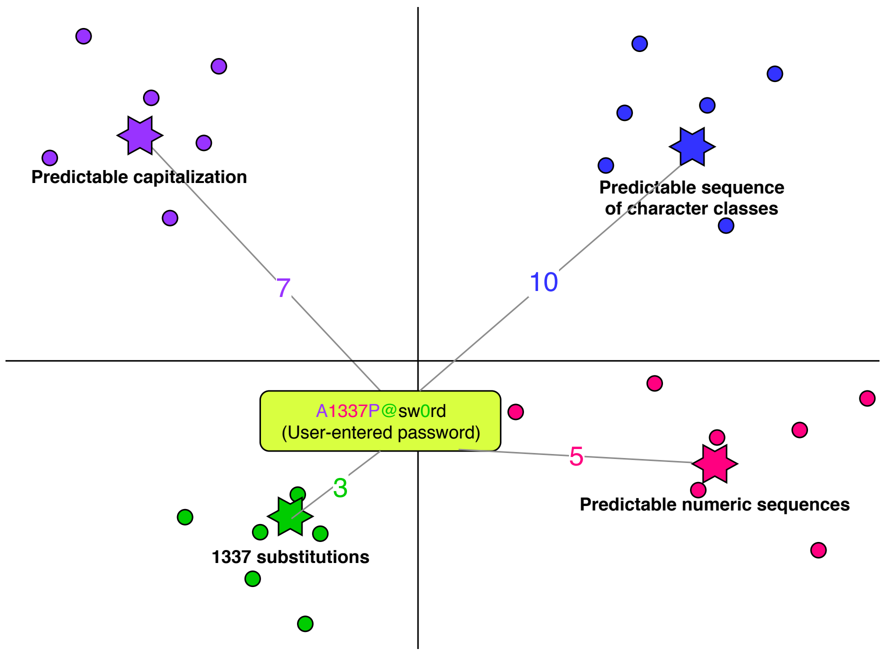

Predictable capitalization
Predictable sequence of character classes
7
10
A1337P@sw0rd (User-entered password)
5
3
Predictable numeric sequences
1337 substitutions

Figure 6: Illustration of how high-level patterns in a password could be measured and attributed to specific input characters. The colored numbers connecting the user-entered password to cluster centroids represents the distance to each of those centroids (i.e. how much that pattern is manifested in the password).

Expert identification of heuristics (all plans; 3 weeks) We plan to use the decoder portion of the autoencoder to decode cluster centroids, which will provide a prototype describing each learned cluster [72]. We will have one or more researchers examine clusters of password instances to identify patterns. Projections of cluster soft-assignments into two dimensions will help in this process. In Plan A, we can also make use of LRP highlighting to help identify high-level patterns.

It will be important to validate that the high-level password concepts identified by the autoencoder have been correctly identified, in order for feedback based on those concepts to be relevant. Methods for doing so include individually removing or substituting characters in the password identified by the network as being associated with the high level concept [72]. If the high-level concept has been correctly identified, then the encoding of the modified password should reflect this. For example, if a concept for keyboard patterns has been associated with a given cluster and evidence of this concept is found in the password encoding of “qwertyababab” (via small distance to the corresponding cluster centroid), then the encoding of “ababab” should reflect the absence of that concept. Related, passwords constructed to contain similar concepts should produce similar encodings (e.g., “qwertyababab” and “zxcvbnababab”) [72].

Perform guess number regression onto heuristics (all plans; 2 weeks) In order to make use of newly identified heuristic features, we need to understand their overall effect on password strength. This step will estimate these effects using linear regression for a given set of passwords. If a learned password pattern has no effect on password strength, a heuristic based on that pattern would not be useful for text feedback and so that pattern could be discarded from the regression model as part of the model selection process. Similarly, heuristics we are unable to map to interpretable heuristics would not be useful for text feedback and could be excluded from the regression model.

This component will also require updated heuristic weights for the regression model used in prior work [14]. These will need to be updated because we want to compare our new regression model to prior work using the more accurate TensorflowJS models, as well as using password-policy-specific models.[12]

Perform error analysis and goodness-of-fit evaluations (all plans; 1 week) Using previously trained, large neural networks modeling guessability as ground truth, we will examine the extent of safe or unsafe prediction errors made by the new heuristic regression model. This may shed light on potential ways to improve our model, including missing regression features or interactions between features.

We will also evaluate and compare goodness-of-fit measures (e.g., adjusted R^2) for the new regression model as well as for models based on prior work [14]. This will provide insight into how well password guessability can be expressed using our learned heuristics. We plan to perform this analysis for different sets of passwords (randomly-selected, weak, medium, and strong); this will explore how well our heuristics can model guessability across the password-strength spectrum. We may explore performing similar regressions for sets of passwords differing according to a single dimension other than password strength, e.g., number of character classes, to explore how model coefficients and goodness-of-fit change between different levels off that dimension. Corrections for multiple testing will be applied as appropriate for any hypothesis tests that are made.

6 Thesis outline

1. Introduction
2. Background and related work
   (a) Passwords
   (b) Interpretable explanations for text-input black-box models
   (c) Public-key fingerprints
3. Password-creation policies with neural-network-driven minimum-password-strength requirements
4. Modeling password strength using neural-network-derived heuristic features
5. Public-key fingerprint representations that help users detect MITM attacks
6. Conclusion

[12]These neural network models have already been trained.

7 Timeline

August 2019

- Begin statistical analysis for minimum-strength vs. blacklist study

September 2019

- Finish statistical analysis for minimum-strength vs. blacklist study in early September
- Write minimum-strength vs. blacklist study paper
- Begin training autoencoder before end of September

October 2019

- Finish training autoencoder by end of October
- Begin training deep k-means neural network as soon as autoencoder training is completed

November 2019

- Begin implementation of LRP algorithm as soon as deep k-means training is completed
- If deep k-means training not finished by 11/15
  - Switch to standard k-means
  - Skip LRP implementation
- Begin expert identification of heuristics as soon as either LRP implementation is finished or standard k-means has been implemented

December 2019

- Finish expert identification of heuristics by 12/22
- Begin linear regression analyses and good-of-fit evaluations as soon as expert identification has finished

January 2020

- Finish linear regressions and evaluations by 1/15
- Begin autoencoder paper as soon as linear regressions and evaluations have finished
- Prepare for job interviews

February 2020

- Finish autoencoder paper
- Begin job interviews

March 2020

- Write concluding chapters of thesis
- Continue job interviews

April 2020

- Defend thesis

May 2020

- Graduate

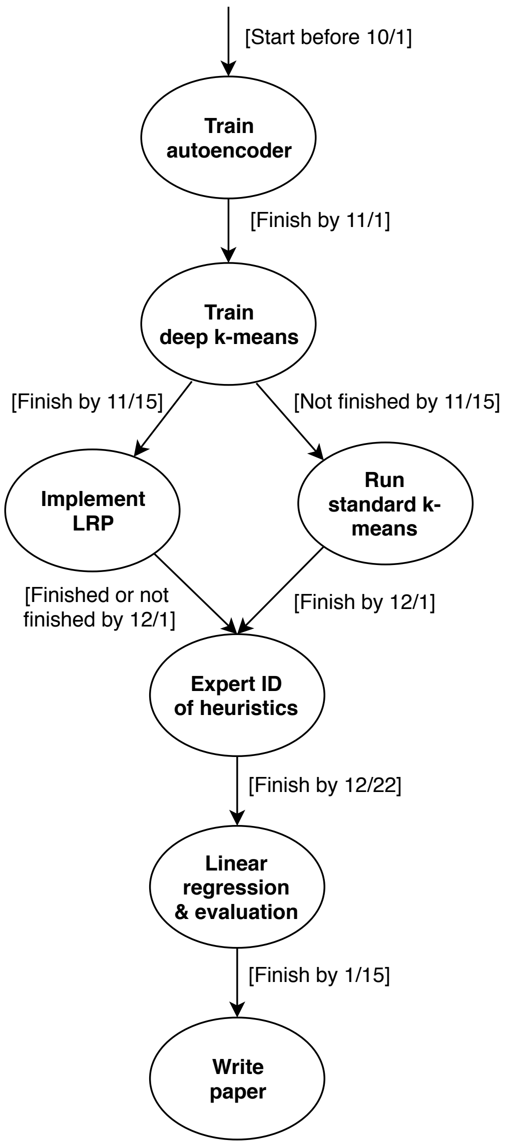


Figure 7: Diagram of timeline with task decision points.

References

[1] W. Melicher, B. Ur, S. M. Segreti, S. Komanduri, L. Bauer, N. Christin, and L. F. Cranor, "Fast, lean, and accurate: Modeling password guessability using neural networks," in *Proceedings of the 25th USENIX Security Symposium*, Aug. 2016.

[2] J. Tan, L. Bauer, J. Bonneau, L. F. Cranor, J. Thomas, and B. Ur, "Can unicorns help users compare crypto key fingerprints?" in *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems.* New York, NY, USA: ACM, 2017, pp. 3787–3798.

[3] D. Florencio and C. Herley, "A large-scale study of web password habits," in *Proceedings of the 16th International Conference on World Wide Web*, ser. WWW '07. New York, NY, USA: ACM, 2007, pp. 657–666. [Online]. Available: http://doi.acm.org/10.1145/1242572.1242661

[4] R. Wash, E. Rader, R. Berman, and Z. Wellmer, "Understanding password choices: How frequently entered passwords are re-used across websites," in *Twelfth Symposium on Usable Privacy and Security (SOUPS 2016).* Denver, CO: USENIX Association, 2016, pp. 175–188. [Online]. Available: https://www.usenix.org/conference/soups2016/technical-sessions/presentation/wash

[5] E. Hayashi and J. Hong, "A diary study of password usage in daily life," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ser. CHI '11. New York, NY, USA: ACM, 2011, pp. 2627–2630. [Online]. Available: http://doi.acm.org/10.1145/1978942.1979326

[6] S. Gaw and E. W. Felten, "Password management strategies for online accounts," in *Proceedings of the Second Symposium on Usable Privacy and Security*, ser. SOUPS '06. New York, NY, USA: ACM, 2006, pp. 44–55. [Online]. Available: http://doi.acm.org/10.1145/1143120.1143127

[7] Y.-Y. Choong, M. Theofanos, and H.-K. Liu, *United States Federal Employees' Password Management Behaviors: A Department of Commerce Case Study.* US Department of Commerce, National Institute of Standards and Technology, 2014.

[8] S. Pearman, J. Thomas, P. E. Naeini, H. Habib, L. Bauer, N. Christin, L. F. Cranor, S. Egelman, and A. Forget, "Let's go in for a closer look: Observing passwords in their natural habitat," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '17. New York, NY, USA: ACM, 2017, pp. 295–310. [Online]. Available: http://doi.acm.org/10.1145/3133956.3133973

[9] Y. Zhang, F. Monrose, and M. K. Reiter, "The security of modern password expiration: An algorithmic framework and empirical analysis," in *Proceedings of the 17th ACM Conference on Computer and Communications Security*, ser. CCS '10. New York, NY, USA: ACM, 2010, pp. 176–186. [Online]. Available: http://doi.acm.org/10.1145/1866307.1866328

[10] S. T. Haque, M. Wright, and S. Scielzo, "A study of user password strategy for multiple accounts," in *Proceedings of the Third ACM Conference on Data and Application Security and Privacy*, ser. CODASPY '13. New York, NY, USA: ACM, 2013, pp. 173–176. [Online]. Available: http://doi.acm.org/10.1145/2435349.2435373

[11] R. Shay, S. Komanduri, P. G. Kelley, P. G. Leon, M. 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*. ACM, 2010, p. 2.

[12] J. Bonneau, "The science of guessing: Analyzing an anonymized corpus of 70 million passwords," in *Proceedings - IEEE Symposium on Security and Privacy*, 2012, pp. 538–552.

[13] B. Ur, S. M. Segreti, L. Bauer, N. Christin, L. F. Cranor, S. Komanduri, D. Kurilova, M. L. Mazurek, W. Melicher, and R. Shay, "Measuring real-world accuracies and biases in modeling password guessability," in *24th USENIX Security Symposium (USENIX Security 15)*. Washington, D.C.: USENIX Association, Aug. 2015, pp. 463–481. [Online]. Available: https://www.usenix.org/conference/usenixsecurity15/technical-sessions/presentation/ur

[14] B. Ur, F. Alfieri, M. Aung, L. Bauer, N. Christin, J. Colnago, L. F. Cranor, H. Dixon, P. E. Naeini, H. Habib, N. Johnson, and W. Melicher, "Design and evaluation of a data-driven password meter," in *CHI'17: 35th Annual ACM Conference on Human Factors in Computing Systems*. ACM, May 2017, pp. 3775–3786.

[15] A. D. Smith, P. Emmett, P. Newby, and K. Northstone, "A comparison of dietary patterns derived by cluster and principal components analysis in a UK cohort of children," *European journal of clinical nutrition*, vol. 65, no. 10, p. 1102, 2011.

[16] P. E. Rauber, S. G. Fadel, A. X. Falcao, and A. C. Telea, "Visualizing the hidden activity of artificial neural networks," *IEEE Transactions on Visualization and Computer Graphics*, vol. 23, no. 1, pp. 101–110, 2016.

[17] M. Aubry and B. C. Russell, "Understanding deep features with computer-generated imagery," *arXiv e-prints*, vol. abs/1506.01151, 2015.

[18] M. T. Ribeiro, S. Singh, and C. Guestrin, ""Why should I trust you?": Explaining the predictions of any classifier," in *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. New York, NY, USA: ACM, 2016, pp. 1135–1144.

[19] M. Sundararajan, A. Taly, and Q. Yan, "Axiomatic Attribution for Deep Networks," in *Proceedings of the 34th International Conference on Machine Learning - Volume 70*. JMLR.org, 2017, pp. 3319–3328.

[20] M. Ancona, E. Ceolini, C. Öztireli, and M. Gross, "Towards better understanding of gradient-based attribution methods for Deep Neural Networks," *arXiv e-prints*, vol. cs.LG, p. arXiv:1711.06104, Nov. 2017.

[21] K. Leino, S. Sen, A. Datta, M. Fredrikson, and L. Li, "Influence-Directed Explanations for Deep Convolutional Networks," *arXiv e-prints*, vol. cs.LG, p. arXiv:1802.03788, Feb. 2018.

[22] A. Karpathy, J. Johnson, and F. F. Li, "Visualizing and Understanding Recurrent Networks," *arXiv e-prints*, vol. abs/1506.02078, 2015.

[23] Y. Ming, S. Cao, R. Zhang, Z. Li, Y. Chen, Y. Song, and H. Qu, "Understanding Hidden Memories of Recurrent Neural Networks," in *2017 IEEE Conference on Visual Analytics Science and Technology (VAST)*, Oct. 2017, pp. 13–24.

[24] H. Strobelt, S. Gehrmann, H. Pfister, and A. M. Rush, "LSTMVis: A Tool for Visual Analysis of Hidden State Dynamics in Recurrent Neural Networks," *IEEE Transactions on Visualization and Computer Graphics*, vol. 24, no. 1, pp. 667–676, Jan. 2018.

[25] J. Li, X. Chen, E. H. Hovy, and D. Jurafsky, "Visualizing and Understanding Neural Models in NLP," *arXiv e-prints*, vol. abs/1506.01066, 2015.

[26] K. Dhamdhere, M. Sundararajan, and Q. Yan, "How Important Is a Neuron?" *arXiv e-prints*, vol. cs.LG, p. arXiv:1805.12233, May 2018.

[27] S. Bach, A. Binder, G. Montavon, F. Klauschen, K.-R. Müller, and W. Samek, "On Pixel-Wise Explanations for Non-Linear Classifier Decisions by Layer-Wise Relevance Propagation," *PLOS ONE*, vol. 10, no. 7, pp. 1–46, Jul. 2015.

[28] Y. Ding, Y. Liu, H. Luan, and M. Sun, "Visualizing and understanding neural machine translation," in *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2017, pp. 1150–1159.

[29] L. Arras, G. Montavon, K.-R. Müller, and W. Samek, "Explaining recurrent neural network predictions in sentimenft analysis," *arXiv preprint arXiv:1706.07206*, 2017.

[30] J. Kauffmann, M. Esders, G. Montavon, W. Samek, and K.-R. Müller, "From Clustering to Cluster Explanations via Neural Networks," *arXiv e-prints*, vol. abs/1906.07633, 2019.

[31] J. Callas, L. Donnerhacke, H. Finney, D. Shaw, and R. Thayer, "Openpgp message format," 2007. [Online]. Available: https://tools.ietf.org/html/rfc4880

[32] J. Galbraith and R. Thayer, "The secure shell (ssh) public key file format," 2006. [Online]. Available: https://www.ietf.org/rfc/rfc4716.txt

[33] Off-the-Record Messaging, "Fingerprints," 2016. [Online]. Available: https://otr.cypherpunks.ca/help/fingerprint.php

[34] WhatsApp, "Whatsapp encryption overview: Technical white paper," April 2016. [Online]. Available: https://www.whatsapp.com/security/WhatsApp-Security-Whitepaper.pdf

[35] Wickr, "What is the key verification feature?" 2016. [Online]. Available: https://wickr.desk.com/customer/en/portal/articles/2342342-what-is-the-key-verification-feature-

[36] S. Nagao, "Avatars," Oct 2016. [Online]. Available: https://peerio.zendesk.com/hc/en-us/articles/202729949-Avatars

[37] A. Whitten and J. D. Tygar, "Why johnny can't encrypt: A usability evaluation of pgp 5.0," in *Proceedings of the 8th Conference on USENIX Security Symposium*, ser. SSYM'99, 1999. [Online]. Available: http://dl.acm.org/citation.cfm?id=1251421.1251435

[38] S. L. Garfinkel and R. C. Miller, "Johnny 2: A user test of key continuity management with s/mime and outlook express," in *Proceedings of the 2005 Symposium on Usable Privacy and Security*, ser. SOUPS '05, 2005. [Online]. Available: http://doi.acm.org/10.1145/1073001.1073003

[39] S. Clark, T. Goodspeed, P. Metzger, Z. Wasserman, K. Xu, and M. Blaze, "Why (special agent) johnny (still) can't encrypt: A security analysis of the apco project 25 two-way radio system," in *Proceedings of the 20th USENIX Conference on Security*, ser. SEC'11, 2011. [Online]. Available: http://dl.acm.org/citation.cfm?id=2028067.2028071

[40] S. Ruoti, N. Kim, B. Burgon, T. van der Horst, and K. Seamons, "Confused johnny: When automatic encryption leads to confusion and mistakes," in *Proceedings of the Ninth Symposium on Usable Privacy and Security*, ser. SOUPS '13, 2013. [Online]. Available: http://doi.acm.org/10.1145/2501604.2501609

[41] S. Gaw, E. W. Felten, and P. Fernandez-Kelly, "Secrecy, flagging, and paranoia: Adoption criteria in encrypted email," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ser. CHI '06, 2006. [Online]. Available: http://doi.acm.org/10.1145/1124772.1124862

[42] R. Stedman, K. Yoshida, and I. Goldberg, "A user study of off-the-record messaging," in *Proceedings of the 4th Symposium on Usable Privacy and Security*, ser. SOUPS '08, 2008. [Online]. Available: http://doi.acm.org/10.1145/1408664.1408678

[43] S. Vaudenay, "Secure communications over insecure channels based on short authenticated strings," in *Proceedings of the 25th Annual International Conference on Advances in Cryptology*, ser. CRYPTO'05, 2005. [Online]. Available: http://dx.doi.org/10.1007/11535218_19

[44] A. Perrig and D. Song, "Hash visualization: A new technique to improve real-world security," 1999. [Online]. Available: https://users.ece.cmu.edu/~adrian/projects/validation/

[45] OpenSSH, "Openssh 5.1 release announcement," 2008. [Online]. Available: https://www.openssh.com/txt/release-5.1

[46] B. D. v. d. Ehe, "Unicornify! how does it work?" 2012. [Online]. Available: https://unicornify.appspot.com/making-of

[47] C. Davis, "Robohash," 2016. [Online]. Available: https://robohash.org/

[48] M. Farb, Y.-H. Lin, T. H.-J. Kim, J. McCune, and A. Perrig, "Safeslinger: Easy-to-use and secure public-key exchange," in *Proceedings of the 19th Annual International Conference on Mobile Computing & Networking*, ser. MobiCom '13, 2013. [Online]. Available: http://doi.acm.org/10.1145/2500423.2500428

[49] M. Shirvanian and N. Saxena, "On the security and usability of crypto phones," in *Proceedings of the 31st Annual Computer Security Applications Conference*, ser. ACSAC 2015, 2015. [Online]. Available: http://doi.acm.org/10.1145/2818000.2818007

[50] M. Wu, R. C. Miller, and G. Little, "Web wallet: Preventing phishing attacks by revealing user intentions," in *Proceedings of the Second Symposium on Usable Privacy and Security*, ser. SOUPS '06, 2006. [Online]. Available: http://doi.acm.org/10.1145/1143120.1143133

[51] N. Unger, S. Dechand, J. Bonneau, S. Fahl, H. Perl, I. Goldberg, and M. Smith, "Sok: Secure messaging," in *Proceedings of the 2015 IEEE Symposium on Security and Privacy*, ser. SP '15, 2015. [Online]. Available: http://dx.doi.org/10.1109/SP.2015.22

[52] R. Kainda, I. Flechais, and A. W. Roscoe, "Usability and security of out-of-band channels in secure device pairing protocols," in *Proceedings of the 5th Symposium on Usable Privacy and Security*, ser. SOUPS '09, 2009. [Online]. Available: http://doi.acm.org/10.1145/1572532.1572547

[53] A. Kobsa, R. Sonawalla, G. Tsudik, E. Uzun, and Y. Wang, "Serial hook-ups: A comparative usability study of secure device pairing methods," in *Proceedings of the 5th Symposium on Usable Privacy and Security*, ser. SOUPS '09, 2009. [Online]. Available: http://doi.acm.org/10.1145/1572532.1572546

[54] A. Kumar, N. Saxena, G. Tsudik, and E. Uzun, "Caveat eptor: A comparative study of secure device pairing methods," in *2009 IEEE International Conference on Pervasive Computing and Communications*, 2009. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4912753

[55] H.-C. Hsiao, Y.-H. Lin, A. Studer, C. Studer, K.-H. Wang, H. Kikuchi, A. Perrig, H.-M. Sun, and B.-Y. Yang, "A study of user-friendly hash comparison schemes," in *2009 Annual Computer Security Applications Conference*, 2009. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5380523

[56] S. Dechand, D. Schürmann, K. Busse, Y. Acar, S. Fahl, and M. Smith, "An empirical study of textual key-fingerprint representations," in *25th USENIX Security Symposium (USENIX Security 16)*, 2016. [Online]. Available: https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/dechand

[57] Plasmoid, "Fuzzy fingerprints: Attacking vulnerabilities in the human brain," 2003. [Online]. Available: https://www.thc.org/papers/ffp.pdf

[58] D. Loss, T. Limmer, and A. von Gernler, "The drunken bishop: An analysis of the openssh fingerprint visualization algorithm," 2009. [Online]. Available: http://dirk-loss.de/sshvis/drunken_bishop.pdf

[59] M. Stevens, P. Karpman, and T. Peyrin, "Freestart collision for full sha-1," in *Proceedings of the 35th Annual International Conference on Advances in Cryptology*, ser. EUROCRYPT 2016, 2016. [Online]. Available: http://dx.doi.org/10.1007/978-3-662-49890-3_18

[60] National Institute of Standards and Technology (NIST), "SP 800-63B: Digital identity guidelines: Authentication and lifecycle management," https://doi.org/10.6028/NIST.SP.800-63-3, June 2017, updated Dec 2017.

[61] T. Hunt, "Pwned Passwords API," 2019. [Online]. Available: https://haveibeenpwned.com/Passwords

[62] H. Habib, J. Colnago, W. Melicher, B. Ur, S. M. Segreti, L. Bauer, N. Christin, and L. F. Cranor, "Password Creation in the Presence of Blacklists," in *Workshop on Usable Security*. Reston, VA: Internet Society, 2017.

[63] 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*, ser. CHI '11. New York, NY, USA: ACM, 2011, pp. 2595–2604. [Online]. Available: http://doi.acm.org/10.1145/1978942.1979321

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

[65] R. Shay, S. Komanduri, A. L. Durity, P. S. Huh, M. L. Mazurek, S. M. Segreti, B. Ur, L. Bauer, N. Christin, and L. F. Cranor, "Designing password policies for strength and usability," *ACM Trans. Inf. Syst. Secur.*, vol. 18, no. 4, pp. 13:1–13:34, May 2016. [Online]. Available: http://doi.acm.org/10.1145/2891411

[66] ——, "Can long passwords be secure and usable?" in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ser. CHI '14. New York, NY, USA: ACM, 2014, pp. 2927–2936. [Online]. Available: http://doi.acm.org/10.1145/2556288.2557377

[67] W. Melicher, D. Kurilova, S. M. Segreti, P. Kalvani, R. Shay, B. Ur, L. Bauer, N. Christin, L. F. Cranor, and M. L. Mazurek, "Usability and security of text passwords on mobile devices," in *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, ser. CHI '16. New York, NY, USA: ACM, 2016, pp. 527–539. [Online]. Available: http://doi.acm.org/10.1145/2858036.2858384

[68] National Institute of Standards and Technology (NIST), "SP 800-63 Ver. 1.0: Electronic authentication guideline," https://csrc.nist.gov/publications/detail/sp/800-63/ver-10/archive/2004-06-30, June 2004.

[69] M. L. Mazurek, S. Komanduri, T. Vidas, L. Bauer, N. Christin, L. F. Cranor, P. G. Kelley, R. Shay, and B. Ur, "Measuring password guessability for an entire university," in *Proceedings of the 2013 ACM SIGSAC Conference on Computer &#38; Communications Security*, ser. CCS '13. New York, NY, USA: ACM, 2013, pp. 173–186. [Online]. Available: http://doi.acm.org/10.1145/2508859.2516726

[70] D. Britz, A. Goldie, T. Luong, and Q. Le, "Massive Exploration of Neural Machine Translation Architectures," *ArXiv e-prints*, Mar. 2017.

[71] J. Xie, R. B. Girshick, and A. Farhadi, "Unsupervised Deep Embedding for Clustering Analysis," *arXiv e-prints*, vol. abs/1511.06335, 2015.

[72] G. Montavon, W. Samek, and K.-R. Müller, "Methods for Interpreting and Understanding Deep Neural Networks," *arXiv e-prints*, vol. abs/1706.07979, 2017.