Usable Human Authentication: A Quantitative Treatment

Jeremiah Blocki

CMU-CS-14-108

June 30, 2014

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Thesis Committee:
Manuel Blum, Co-Chair
Anupam Datta, Co-Chair
Luis von Ahn
Ron Rivest, MIT

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

This research was sponsored by the Naval Surface Warfare Center under grant number N00164-03-DG6623-0009, the National Science Foundation under grant numbers CNS-0831178, CCR-0122581, DGE-0750271, DGE-1252522, and CCF-0424422, and the Air Force under grant number FA95501210040. The views and conclusions contained in this document are those of the author and should not be interpreted as representing the official policies, either expressed or implied, of any sponsoring institution, the U.S. government or any other entity.

Keywords: Human Authentication, Passwords, Password Management Scheme, Usability Model, Security Model, Password Composition Policy, GOTCHA

I would like to thank my beautiful wife Heather for her love and support. I would also like to thank my parents for raising me and caring for me over the years. Finally, I would like to thank my Lord and Savior Jesus Christ for creating me, saving me and placing all of the people I mentioned above into my life.

Abstract

A typical computer user today manages passwords for many different online accounts. Users struggle with this task — often forgetting their passwords or adopting insecure practices, such as using the same passwords for multiple accounts and selecting weak passwords. While there are many books, articles, papers and even comics about selecting strong individual passwords, there is very little work on password management schemes — systematic strategies to help users create and remember multiple passwords. Before we can design good password management schemes it is necessary to address a fundamental question: How can we quantify the usability or security of a password management scheme. One way to quantify the usability of a password management scheme would be to conduct user studies evaluating each user's success at remembering multiple passwords over an extended period of time. However, these user studies would necessarily be slow and expensive and would need to be repeated for each new password management scheme. Our thesis is that user models and security models can guide the development of password management schemes with analyzable usability and security properties. We present several results in support of this thesis. First, we introduce Naturally Rehearsing Password schemes. Notably, our user model, which is based on research on human memory about spaced rehearsal, allows us to analyze the usability of this family of schemes while experimentally validating only the common user model underlying all of them. Second, we introduce Human Computable Password schemes, which leverage human capabilities for simple arithmetic operations. We provide constructions that make modest demands on users and we prove that these constructions provide strong security: an adversary who has seen about 100 10-digit passwords of a user cannot compute any other passwords except with very low probability. Our password management schemes are precisely specified and publishable: the security proofs hold even if the adversary knows the scheme and has extensive background knowledge about the user (hobbies, birthdate, etc.). They do not require any significant server-side changes. In further support of our thesis, we show that user models and security models can also be used to develop server-side defenses against online and offline attacks.

Acknowledgments

I am indebted to my advisors Manuel Blum and Anupam Datta for all of their guidance and encouragement over the years. I would like to thank Luis Von Ahn and Ron Rivest for serving on my thesis committee and for giving me helpful comments and suggestions about my work. I would like to thank Ariel Procaccia, Saranga Komanduri and Or Sheffet who coauthored the paper Optimizing Password Composition Policies with me. Saranga Komanduri has also collaborated with me to run user studies to empirically test several of the usability models presented in this thesis. Calvin Beideman, a bright high school student, worked with me on the paper Set Families with Low Pairwise Intersection, and Shikun Zhang has been working with me to develop a password management application based on ideas from this thesis. I would also like to thank Santosh Vempala for many helpful discussions about statistical algorithms and random planted satisfiability problems, and Ryan O'Donnell for pointing me to recent general hypercontractivity results. These discussions helped me to obtain the main technical result in my work on Human Computable Passwords.

I am thankful to Steven Rudich for running Andrew's Leap, a summer program at Carnegie Mellon University for high school students interested in computer science, and for his continued interest in my research. The Andrew's Leap program helped me to decide to study computer science as an undergraduate at CMU. I am thankful to Lenore Blum for helping me find a summer Research Experience for Undergraduates program while I was a sophomore at CMU, and I am thankful to Ryan Williams and Manuel Blum for advising me during the summer. My experience during the summer REU convinced me to pursue a PhD in computer science.

Some passages have been quoted verbatim from the following sources:
Optimizing Password Composition Policies [34, 35] by Jeremiah Blocki, Saranga Komanduri, Ariel Procaccia and Or Sheffet.
GOTCHA Password Hackers! [31, 32] by Jeremiah Blocki, Manuel Blum and Anupam Datta.
Naturally Rehearsing Passwords [30, 33] by Jeremiah Blocki, Manuel Blum and Anupam Datta.
Human Computable Passwords [36] by Jeremiah Blocki, Manuel Blum and Anupam Datta.
Set Families with Low Pairwise Intersection[25] by Calvin Beideman and Jeremiah Blocki.

The material from Chapter 4 is based on ongoing research with Anupam Datta, Saranga Komanduri and Lorrie Cranor.

Contents

1 Overview **1**

1.1 Introduction 1

1.2 Statement of Thesis 4

1.2.1 User Models 4

1.2.2 Quantitative Security Model 8

1.2.3 Developing Human Authentication Schemes with Analyzable Security and Usability Properties 8

1.3 Usable and Secure Password Management 11

1.3.1 Overview 12

1.4 Human Computable Passwords 16

1.4.1 Overview 17

1.5 Empirical Validation of User Model 22

1.6 A Defense against Online Attacks 23

1.6.1 Overview 23

1.7 A Defense Against Offline Attacks 25

1.7.1 Background 25

1.7.2 GOTCHAs 26

1.7.3 Overview 26

2 Naturally Rehearsing Passwords **29**

2.1 Introduction 29

2.2 Related Work. 32
2.3 Definitions 35
2.3.1 Associative Memory and Cue-Association Pairs 35
2.3.2 Visitation Schedules and Rehearsal Requirements 36
2.3.3 Password Management Scheme 37
2.4 Usability Model 38
2.4.1 Rehearsal Requirements 39
2.4.2 Visitation Schedules. 40
2.5 Security Model 42
2.6 Our Construction 45
2.6.1 Constructing (n, ℓ, γ)-sharing set families 46
2.6.2 Shared Cues 48
2.6.3 Usability and Security Analysis 49
2.7 Discussion and Future Work 51

3 Human Computable Passwords **53**
3.1 Introduction 53
3.2 Related Work 57
3.3 Definitions 59
3.3.1 Notation 59
3.3.2 Requirements for a Human Computable Function 61
3.3.3 Password Unforgeability 62
3.3.4 Security Parameters of f 63
3.4 Statistical Adversaries and Lower Bounds 64
3.4.1 Statistical Algorithms 64
3.4.2 Statistical Dimension Lower Bounds 66
3.5 Security Analysis 68
3.5.1 Breaking UF-RCA is Equivalent to Secret Recovery 68
3.5.2 Gaussian Elimination 71

3.6 Candidate Secure Human Computable Functions 72
3.6.1 Candidate Scheme 1 74
3.6.2 Candidate Scheme 2 75
3.6.3 Usability: 75
3.6.4 Statistical Algorithms: Security Upper Bound 78
3.7 Discussion 79
3.7.1 Human Computable Passwords Challenge 79
3.7.2 Security Under Continuous Leakage 80
3.7.3 Open Questions 80

4 Empirical Validation of User Model **83**
4.1 Introduction 83
4.2 Related Work 85
4.3 Study Design 87
4.3.1 Recruitment Text 88
4.3.2 Memorization Phase 88
4.3.3 Rehearsal Phase 93
4.3.4 Follow Up Survey 94
4.3.5 Rehearsal Schedules 95
4.3.6 List of People, Actions and Objects from the User Study . . 96
4.4 Preliminary Results 97
4.4.1 Discussion 103

5 Password Composition Policies: A Defense Against Online Attacks **105**
5.1 Introduction 105
5.1.1 Our Model 106
5.1.2 Our Results 108
5.1.3 Related Work 110
5.2 A Model of Password Composition Policies 111

5.3 Ranking Model: Complexity Results . . . 113
5.3.1 Positive Rules: Efficient Algorithm for Constant k . . . 114
5.3.2 Special Case $k = 1$ . . . 115
5.3.3 The General Case . . . 116
5.3.4 Singleton Rules: Hardness for Large k . . . 118
5.3.5 Negative Rules: Hardness of Approximation for $k = 1$ . . . 120
5.4 Normalization Model: Complexity Results . . . 123
5.4.1 Singleton Rules: Efficient Algorithm for large k . . . 123
5.4.2 Negative Rules: Hardness for $k = 1$ . . . 124
5.4.3 Positive Rules: Hardness of Approximation for Large k . . . 126
5.5 Efficient Sampling Algorithms . . . 126
5.6 Experiments . . . 128
5.6.1 Experiment Rules . . . 129
5.6.2 Baselines . . . 130
5.6.3 Performance . . . 130
5.7 Discussion . . . 131

6 GOTCHAs: A Defense Against Offline Attacks 135
6.1 Introduction . . . 135
6.1.1 Related Work . . . 138
6.2 Definitions . . . 141
6.2.1 Password Storage and Offline Attacks . . . 145
6.3 Inkblot Construction . . . 146
6.3.1 GOTCHA Authentication . . . 148
6.3.2 User Study . . . 151
6.3.3 An Open Challenge to the AI Community . . . 153
6.4 Analysis: Cost of Offline Attacks . . . 154
6.5 Discussion . . . 156

7 Appendix: Naturally Rehearsing Passwords **161**
7.1 Missing Proofs . . . 162
7.2 Varying the Association Strength Constant . . . 165
7.3 Baseline Password Management Schemes . . . 166
7.3.1 Security Of Baseline Password Management Schemes . . . 169
7.3.2 Usability of Baseline Schemes . . . 170
7.3.3 Sources of Randomness . . . 170
7.4 Other Measures of Password Strength . . . 170
7.4.1 Password Strength Meters . . . 171
7.4.2 Entropy . . . 172
7.4.3 Minimum Entropy . . . 173
7.5 Economics . . . 174
7.5.1 Password Storage . . . 174
7.5.2 Attack Cost and Benefit . . . 175
7.5.3 Cost of Guessing . . . 175
7.5.4 Benefit . . . 176
7.6 Associative Memory and Sufficient Rehearsal Assumptions . . . 177
7.6.1 Squared Rehearsal Assumption . . . 178
7.7 (n, ℓ, γ)-sharing Set Families . . . 179
7.7.1 Improved Constructions . . . 180
7.7.2 Applications to Pseudorandom Number Generators . . . 182
7.7.3 Upper Bounds . . . 189
7.7.4 Open Questions . . . 193

8 Appendix: Human Computable Passwords **195**
8.1 Human Computable Passwords Challenge . . . 196
8.2 Statistical Dimension . . . 197
8.3 Security Proofs . . . 205
8.4 Proofs of Claims and Facts . . . 209

8.4.1 Security Upper Bounds 214

9 Appendix: Password Composition Policies **217**

9.1 Optimizing Password Composition Policies: Missing Proofs 218

9.2 Impossibility of constant-factor universal approximation 223

10 Appendix: GOTCHA Password Hackers **225**

10.1 Missing Proofs 226

10.2 HOSP: Pre-Generated CAPTCHAs 227

Bibliography **229**

List of Figures

1.1 XKCD: correct, horse, battery staple Source: `http://xkcd.com/936/` 7

1.2 Example 1. Memorizing a Person-Action-Object Story 13

1.3 Example 2. Memorizing a Person-Action-Object Story 13

1.4 Login Example 1 14

1.5 Login Example 2 15

1.6 Secret Random Mapping from Pictures to Digits 17

1.7 Mnemonics to help memorize the secret mapping σ 17

1.8 A single-digit challenge 18

1.9 Computing the response ($f(\sigma(C)) = 6$) to a single-digit challenge . 19

1.10 Login Screen 20

1.11 Login Screen after the user responds to the first single-digit challenge 20

1.12 GOTCHA Authentication Example 27

2.1 Person Action Object Story with Cue 45

2.2 Account A_{19} using *Shared Cues* with the $(43, 4, 1)$-sharing set family **CRT** $(90, 9, 10, 11, 13)$. 45

3.1 Mnemonics to help memorize the secret mapping σ 73

4.1 Memorization Step 0. Scene and Person. 90

4.2 User Study: Non-Mnemonic Group Memorization Phase 93

4.3 Total Survival with Failures Carried Forward $\left(\frac{\mathbf{Survived}(i)}{\mathbf{Survived}(t)+\sum_{j=1}^{t}\mathbf{Failed}(j)}\right)$ vs **Time** (i). 99
4.4 Conditional Survival: **Survived** (i) /**Returned** (i) vs **Time** (i). 100
4.5 **EstimatedSurvival** (i) vs **Time** (i) 101
4.6 Survey: Which of the following reasons best describes why you were unable to return to take the follow up test? 102

6.1 Randomly Generated Inkblot Image—An evil clown? 137
6.2 GOTCHA User Study: Phase 1 159
6.3 GOTCHA User Study: Phase 2 160
6.4 Random Walk Inkblots 160

7.1 mmmmmmmmmmmmmmmmmmmmmmmmmmmm: sounds delicious, but is it really a strong password? 172

List of Tables

2.1 Visitation Schedules 40

2.2 Extra Rehearsals for Baseline Schemes 41

2.3 $\mathbb{E}[XR_{365}]$: Extra Rehearsals over the first year for SC-0,SC-1 and SC-2. 50

2.4 *Shared Cues* $(q_{\$10^6}, \delta, m, s, r, h)$-Security: δ vs h and r using a (n, ℓ, γ)-sharing family of m public cues. 50

3.1 Extra Rehearsals to Remember Secret Mapping 77

3.2 Example: Single-Digit Challenge 77

4.1 Rehearsal Schedules 96

4.2 Survived 98

5.1 Summary of Complexity Results. 109

5.2 Rankings used in the proof of Theorem 15. 122

5.3 Rules Used in Sampling Experiments 133

5.4 Baseline probabilities for the RockYou dataset 134

5.5 Performance of Sampling Algorithms with Positive Rules 134

5.6 Performance of Sampling Algortihms with Negative Rules 134

6.1 GOTCHA User Study: Completion Times 152

6.2 Usability Results: Fraction of Participants who would have authenticated with accuracy parameter α 152

7.1 Expanding Rehearsal Assumption: $\mathbb{E}[XR_{365,c}]$ vs. λ_c and σ 166
7.2 Constant Rehearsal Assumption: $\mathbb{E}[XR_{365,c}]$ vs. λ_c and σ 166
7.3 Upper Bound: q_B for BCRYPT, MD5 and SHA1 174
7.4 Guessing Costs 176
7.5 $q_{\$1,000,000}$ 177
7.6 $\mathbb{E}[XR_{365}]$: Extra Rehearsals over the first year under the Squared Rehearsal Assumption — $\sigma = 1$.
B+D: *Lifehacker*
SRI: *Strong Random and Independent* 179
7.7 (n, ℓ, γ)-sharing set family constructions 180

8.1 Human Computable Password Challenges n — Secret Length m—# Challenge-Response Pairs 196

Chapter 1

Overview

1.1 Introduction

A typical computer user has many different online accounts which require some form of authentication. While passwords are still the dominant form of authentication, users struggle to remember their passwords. As a result users often adopt insecure password practices (e.g., reusing the same password, selecting common passwords) [39, 52, 75, 102] or end up having to frequently reset their passwords. There have been numerous recent examples of major password breaches [3, 6, 8, 9, 10, 11, 12, 13, 14, 28, 52, 141].

An adversary may crack a weak password in an *online attack* where he pretends to login as a legitimate user and tries as many password guesses as the site permits him to try before he is locked out. If the cryptographic hash of a password is leaked or stolen an adversary will be able to mount a more dangerous attack known as an *offline attack*, in which he can continue guessing the user's password indefinitely. Unfortunately, these attacks are commonplace (e.g., breaches at Zappos, LinkedIn, Sony, Gawker and Adobe have affected millions of users [6, 8, 9, 11, 13, 14, 28]). Users are often advised to pick long passwords that include numbers, special characters and capital letters to protect themselves in the event of an offline attack [132]. Even the strongest passwords can be compromised via a *plaintext password leak attack*, which could occur because the user fell prey to a phishing attack or because the user signed into his account on an infected computer or due to software misconfigurations (e.g., [5, 10, 12, 141]). Users are typically advised against reusing the same password to protect themselves in the event of a plaintext password leak.

Password Management Schemes. Informally, a password management scheme is a strategy that a user could follow to create and remember each of his passwords. One of the central goals of this thesis is to develop password management schemes which can be implemented on "human hardware." A good password management scheme should be usable and secure. Informally, a password management scheme is usable if a human can create and recall passwords without too much effort. A secure password management scheme must provide concrete security guarantees even against an adversary who has already learned one or more of the user's passwords. Before we can design good password management schemes it is necessary to address two fundamental questions: How can we quantify the usability of a password management scheme? How can we quantify the security of a password management scheme? While it is straightforward to introduce a quantitative security model based on the attack scenarios described earlier, it is more challenging to quantify usability because our understanding of human memory is incomplete. One way to evaluate the usability of a candidate password management scheme is to conduct a large user study. However, this would make the design process slow and expensive as the user study would have to evaluate each user's success at remembering multiple passwords over an extended period of time. Our goal is to develop a quantitative usability model so that the design process for password management schemes could be separated from the validation of the usability model. In Chapter 2 we introduce a mathematical framework for quantifying the usability of a password management scheme, as well as a mathematical framework for quantifying the security of a password management scheme. Our usability model builds on cognitive psychology literature about spaced repetition and human memory. Using these models we develop a novel password management scheme, *Shared Cues*, which balances security and usability. In Chapter 3 we develop several password management schemes with even stronger security properties than *Shared Cues* by leveraging the user's capacity to perform simple computations (e.g., addition modulo 10) in his head, and we apply our usability model from Chapter 2 to help quantify the usability of our human computable password management schemes.

Defenses for a System Administrator. In this thesis we also suggest several defenses that a system administrator could adopt to mitigate the threat of online and offline attacks. One way to defend against online attacks is to adopt a password composition policy which specifies the passwords that a user may/may not select (e.g., one common policy says that each password must contain at least one capital letter and at least one number). The goal of these policies is to ensure that an

online adversary's first few guesses are likely wrong by disallowing overly popular passwords[1]. One natural question to ask is whether or not we can efficiently compute the best password composition policy given sufficient data about the password preferences of our users. In Chapter 5 we initiate the algorithmic study of password composition policies and present an algorithm to find the optimal policy with positive rules (e.g., one potential positive rules policy specifies that a password is allowed if it satisfies one of the following conditions: 1) It is longer than 15 characters, 2) It is longer than 12 characters and contains upper and lower case letters, or 3) It is longer than 9 characters and contains numbers as well as upper and lower case letters.). In Chapter 6 we present a defense against offline attacks called GOTCHAs. The basic idea behind GOTCHAs is to exploit hard artificial intelligence problems to ensure that human feedback is necessary to validate *each* different password guess. This dramatically increases the adversary's cost during an offline attack.

Organization. We first state our thesis in Section 1.2. In the remainder of this chapter we briefly summarize each of the remaining chapters in this thesis. In these summaries we emphasize how each of our proposed defenses would be used in practice, while postponing a discussion of the technical details to later chapters. Chapter 2 — based on the work of Blocki et al. [33] — takes the perspective of the user who is given the complex task of creating and remembering passwords for multiple accounts. We overview Chapter 2 in Section 1.3. In Chapter 4 we describe an ongoing user study that we are conducting to quantify the effects of rehearsal and the use of mnemonic techniques on long term memory retention. We give a brief overview of this user study in Section 1.5. Chapter 3 — based on the work of Blocki et al. [36] — continues the line of research from Chapter 2. We develop even more secure password managment schemes by considering schemes in which the user must perform a few simple computations in his head to compute each of his passwords. We overview Chapter 3 in Section 1.4. In contrast to Chapters 2 and 3, Chapters 5 and 6 take the perspective of a system administrator at a large company who is trying to protect users against online and offline attacks. Chapter 5 — based on the work of Blocki et al. [34] — deals with online attacks. We overview Chapter 5 in Section 1.6. Chapter 6 — based on the work of Blocki et al. [31] — deals with offline attacks. We overview Chapter 6 in Section 1.7.

[1]After a few incorrect guesses the server can temporarily lock the user's account to stop the online adversary.

1.2 Statement of Thesis

We argue in support of the following thesis:

> User models and security models can guide the development of human authentication schemes with analyzable usability and security properties.

We present two sets of results in support of this thesis. We present our first set of results in Chapters 2 and 3. While there are many articles, books, papers and even comics about selecting strong individual passwords [4, 46, 49, 82, 109, 132, 146, 162], there is very little work on *password management schemes*—systematic strategies to help users create and remember multiple passwords—that are both usable and secure. One of the primary goals of this thesis is to develop theoretical models to help quantify the security and usability of a password management scheme and to use these models to develop better password management schemes. A good password management scheme should be provably usable and should provably result in secure passwords. Furthermore, the password management scheme needs to be publishable, meaning that the security proof should hold even if the adversary knows the password management scheme that our user is following. We present models to quantify the usability and the security of a password management scheme, and we use these models to develop password management schemes that provably balance security and usability. We present our second set of results in support of our thesis in Chapters 5 and 6, where we develop and analyze defenses for offline and online attacks against passwords. The defenses we propose follow naturally from our user models and our security models.

1.2.1 User Models

A user model may either specify capabilities of the user or describe how the user will behave in different scenarios; possibly both. In Chapter 2 our user model consists of a memory assumption and a visitation schedule for each of the user's accounts. The memory assumption states that a user is capable of remembering a story if he follows a particular rehearsal schedule, and our visitation schedule specifies how often a user will naturally rehearse each of his passwords. This user model allows us to quantify the usability of a password management scheme by

predicting how much extra effort a user would need to expend to remember all of his passwords. In Chapter 3 we expand the user model of Chapter 2 by assuming that the user can also perform simple computations (e.g., addition modulo 10) in his head, and we show how to develop even more secure human computable password management schemes. In Chapter 5 our user model predicts how users will update their passwords in response to restrictions. While our user model in this chapter is simple and intuitive it allows us to develop a novel algorithm to find the optimal password composition policy to defend against online attacks. In Chapter 6 our user model specifies a task that a human can do, but a computer cannot (e.g., imagine objects in an randomly generated inkblot image and recognize those same objects later). We certainly do not claim to provide a comprehensive list of tasks that a human can do, but a computer could not. However, we are able to develop a novel defense against offline attacks called GOTCHAs based on a simple security assumption (e.g., given two randomly generated inkblot images and a human-generated label for one of the inkblot images the computer cannot accurately predict which inkblot image was labeled).

Each of our user models consists of an assumption about the user's capabilities (e.g., a user is able to remember a story if he follows a given rehearsal schedule, a user is able to imagine objects in an inkblot image and recognize those same objects later, a user is able to add two digits modulo 10 in his head) and/or a description of the user's behavior (e.g., how will a user change his password in response to a composition policy, how often will a user login to each of his accounts). Borrowing terminology from logic, our goal is develop user models that are sound (e.g., users truly possess the capabilities specified by our model), but not necessarily complete (e.g., users may have many other capabilities not considered by our models).

Related Work

A distinctive goal of our work is to develop a quantitative usability model so that the design process for password management schemes can be separated from the validation of the usability model. In contrast, a line of prior work on usability has focused on empirical studies of user behavior including their password management habits [52, 75, 102], the effects of password composition rules (e.g., requiring numbers and special symbols) on individual passwords [34, 101], the memorability of individual system assigned passwords [140], graphical passwords [27, 48], and passwords based on implicit learning [38]. These user studies have been limited in duration and scope (e.g., study retention of a single password over a short

period of time) and can only test a very specific hypothesis.

Example. As an example, consider the suggestion that Randall Munroe gave in his popular webcomic XKCD (See Figure 1.1). He suggested that users create their passwords by picking four random words from the dictionary and creating a story[2]. A user study conducted by Shay et al. [140] indicated that users had more difficulty when asked to remember three to four random words than when they were asked to remember 5 to 6 random characters. However, the user study was only able to test a very specific hypothesis: that users have less difficulty remembering 5 to 6 random characters than remembering 3 to 4 random words from a specific small dictionary when the users are not given instructions about how to memorize their passwords. This leaves open a host of other questions. What if the users were required to follow specific instructions about how to memorize their words (e.g., by making up a story)? Would it still be harder to remember 4 random words? What if we used a larger dictionary with fewer words per password? Is it still easier to remember 6 random characters than to remember 2 random words from a larger dictionary? Most importantly, what if the user is memorizing multiple passwords?

Usability Models. One way to evaluate the usability of a candidate password management scheme would be to conduct a large user study. However, this would make the design process slow and expensive as the user study would have to evaluate each user's success at remembering multiple passwords over an extended period of time. In Chapter 2 we introduce a mathematical framework for quantifying the usability of a password management scheme, as well as a mathematical framework for quantifying the security of a password management scheme. Our usability model allows us to separate the design process from the validation of the usability model. Our usability model builds on cognitive psychology literature about spaced repetition and human memory[160]. In Chapter 4 we describe our ongoing user study to test the usability model itself.

[2]Munroe does not deal with the more complicated problem of creating and remembering multiple passwords.

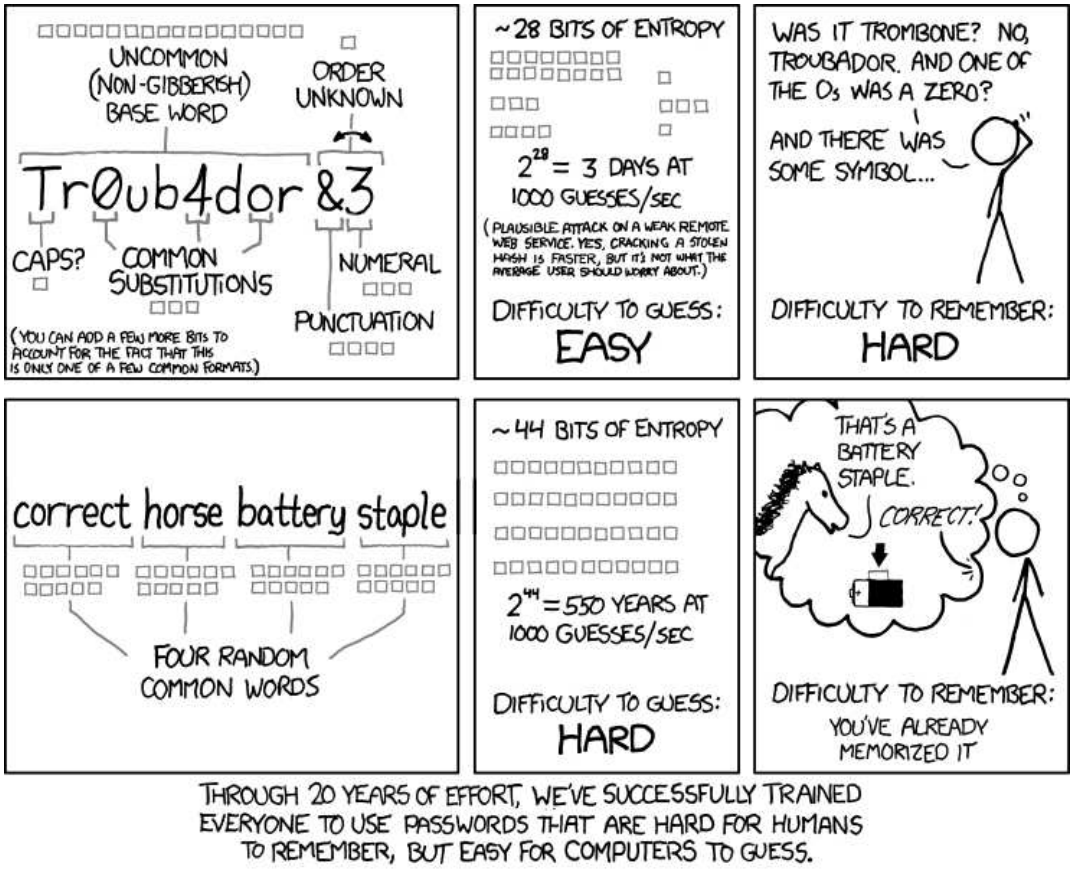


Figure 1.1: XKCD: correct, horse, battery staple
Source: `http://xkcd.com/936/`

1.2.2 Quantitative Security Model

We present a game based security model for a password management scheme in the style of exact security definitions of Bellare and Rogaway [26]. The game is played between a user ($\mathcal{U}$) and a resource-bounded adversary ($\mathcal{A}$) whose goal is to guess one of the user's passwords. Our game models three commonly occurring breaches (online attack, offline attack, plaintext password leak attack). Our security model is fundamentally different from metrics like guessing entropy (e.g., How many guesses does an adversary need to guess all of passwords in a dataset [107]?) and partial guessing entropy (e.g., How many guesses does the adversary need to crack α-fraction of the passwords in a dataset [39, 121]? How many passwords can the adversary break with β guesses per account [45]?), which take the perspective of a system administrator who is trying to protect many users with password protected accounts on his server. For example, a system administrator who wants to evaluate the security effects of a a new password composition policy may be interested in knowing what fraction of user accounts are vulnerable to offline attacks. By contrast, our security model takes the perspective of the user who has many different password protected accounts. Our user wants to evaluate the security of various password management schemes that he could choose to adopt. He is not worried about how many Amazon passwords could be cracked in three guesses. Instead, he will be worried about whether or not his personal accounts are vulnerable.

1.2.3 Developing Human Authentication Schemes with Analyzable Security and Usability Properties

No user model will perfectly capture all of the intricacies of human memory and behavior. As George E.P. Box famously observed[44], "essentially, all models are wrong, but some are useful." We argue that our quantitative security and usability models are useful because they allow us to develop usable and secure human authentication schemes. In Chapter 2.6 we develop a novel password management scheme, *Shared Cues*, which balances security and usability, and in Chapter 3 we develop several password management schemes with even stronger security properties than *Shared Cues* by leveraging the user's capacity to perform simple computations (e.g., addition modulo 10) in his head. In both of these cases insights from our usability model in Chapter 2 helped us to optimize for usability while we were developing our password management schemes. In Chapter 5 we

also propose a simple model of how users change their passwords in response to a password composition policy, and we use this model to develop an efficient sampling algorithm to find the most secure password composition policy.

Analyzable Usability and Security. Because our goal is to develop quantitative security and usability models we focus on password management schemes that are precisely specified. It is not always possible to quantify the security or usability of a password management scheme that is not precisely defined. As an example, consider the advice provided by Computing Facilities at the School of Computer Science at Carnegie Mellon University[3]. They recommend that users follow the following steps to create their passwords:

> 1. Make up a sentence you can easily remember. Some examples:
>
> I have two kids: Jack and Jill.
>
> I like to eat Dave & Andy's ice cream.
>
> No, the capital of Wisconsin isn't Cheeseopolis!
>
> 2. Now take the first letter of every word in the sentence, and include the punctuation. You can throw in extra punctuation, or turn numbers into digits for variety. The above sentences would become:
>
> Ih2k:JaJ.
>
> IlteD&A'ic.
>
> N,tcoWi'C!
>
> ...Please don't use one of the sentences above to generate your password.

These instructions do not clearly specify how the user should select the sentences for each of his passwords. Does our user select similar (or identical) sentences for several (all) of his passwords? If he selects similar sentences for each of his passwords then it may be easier to remember all of his passwords, but now a plaintext password breach at one of the user's accounts will leave our user's other passwords vulnerable. Does he pick his sentence(s) from a favorite poem, book or movie instead of stringing together truly random words? If he does then the sentence may be more memorable, but the resulting password(s) will be easier

[3]Source: `http://www.cs.cmu.edu/~help/security/choosing_passwords.html` (Retrieved May 5,2014.

for an adversary to break, especially if the adversary has background knowledge about the user. By contrast, the password management schemes that we present in Chapters 2 and 3 are defined precisely (e.g., if we ask users to memorize a story we specify the random distribution from which the story should be drawn, and we even give the user instructions about how to memorize the story).

1.3 Usable and Secure Password Management

A typical computer user may have many password protected accounts (e.g., Amazon, Google, LinkedIn) that require authentication, and the user may wish to authenticate using a diverse set of computing devices (e.g., desktop, personal laptop, public computer, friend's computer, smartphone). The user needs to account for all three types of attacks: online attacks, offline attacks and plaintext password leak attacks. If there is a password breach at LinkedIn [13] will the user's password resist offline cracking attempts? If the user borrows his friend's malware-infected computer to login to Google will the adversary also be able to recover the user's password for Amazon? Our user also needs to worry about remembering his password(s) because the password-reset process is often costly [158].

In Chapter 2 we introduce quantitative usability and security models to guide the design of *password management schemes* — systematic strategies to help users create and remember multiple passwords. In the same way that security proofs in cryptography are based on complexity-theoretic assumptions (e.g., hardness of factoring and discrete logarithm), we quantify usability by introducing *usability assumptions*. In particular, password management relies on assumptions about human memory, e.g., that a user who follows a particular rehearsal schedule will successfully maintain the corresponding memory. These assumptions are informed by research in cognitive science and can be tested empirically. To quantify the usability of the password scheme we predict how much 'extra effort' a user would have to expend to remember all of his passwords. We say that a user rehearses a secret *naturally* whenever he recalls that secret to log into one of his accounts. If a user does not get sufficient natural rehearsal for a secret then he will need to be reminded to rehearse that secret. We call this an *extra rehearsal*. Given rehearsal requirements and a user's visitation schedule for each account, we can predict how many times our user will need to be reminded to perform *extra rehearsals* to ensure that he remember all of his passwords. Our usability model lead us to a key observation: password reuse benefits users not only by reducing the number of passwords that the user has to memorize but, more importantly, by increasing the natural rehearsal rate for each password. We also present a security model which accounts for the complexity of password management with multiple accounts and associated threats, including online, offline, and plaintext password leak attacks. Observing that current password management schemes are either insecure or unusable, we present *Shared Cues* — a novel password management scheme in which the underlying secrets that the user memorizes are

strategically shared across accounts to ensure that most rehearsal requirements are satisfied naturally while simultaneously providing strong security guarantees. Our construction uses the Chinese Remainder Theorem to strategically share the secrets in a way that achieves these competing goals.

1.3.1 Overview

We are developing an application which implements the *Shared Cues* password management scheme.

Memorizing Person-Action-Object Stories. To begin using our application the user first memorizes several randomly generated Person-Action-Object (PAO) stories. Figures 1.2 and 1.3 illustrate this process. To memorize each PAO story we show the user four images: a person, an action, an object and a scene. We instruct the user to imagine the PAO story taking place inside the scene.

> Spend 10 seconds visualizing each story in your head, and try to make it as vivid as possible by thinking of details. For example, suppose that you see the story President Bush is flipping a leaf. When you are picturing this story in your head you should try to answer questions like the following: Is the leaf big or small? What color is the leaf? Is President Bush laughing or frowning?

After the user has memorized a story the application stores the images of the person and the scene, but discards the images of the action and object. The images of the person and the scene will be used as a public cue to help the user remember the secret action and object. We emphasize that actions and the objects in each of these stories are randomly chosen by the computer not by the user. If the user selected the action and the the object then he might pick words that are correlated with person or the scene (e.g., in Figure 1.2 the user might pick the object 'apple' or 'penguin' because these words are correlated with Steve Jobs and the glacier respectively). By having the computer select the story we can ensure that the action and object are not correlated with the scene or the person that the user is shown.

Creating an account. To help the user create a password for an account the application first chooses four of the PAO stories (see Chapter 2 for more details).

Figure 1.2: Example 1. Memorizing a Person-Action-Object Story

Figure 1.3: Example 2. Memorizing a Person-Action-Object Story

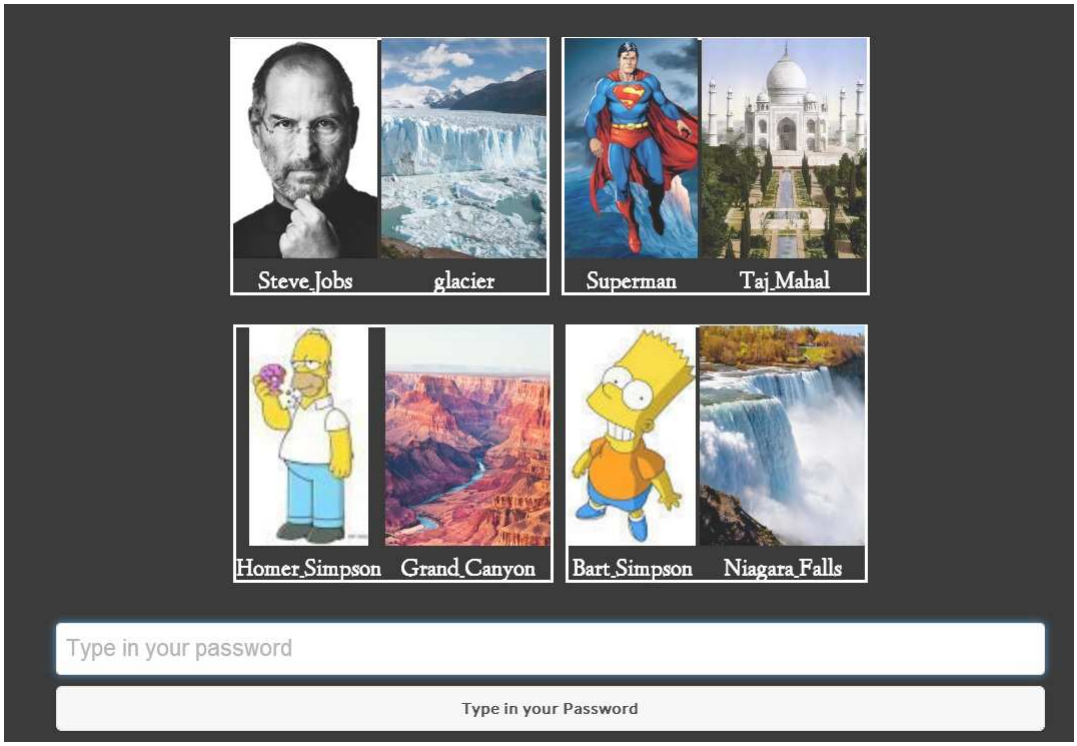


Figure 1.4: Login Example 1

The user is shown the images of the person and scene in each of the four stories. To form his password the user remembers the secret action and object associated with each story and concatenates all of these words together. Figures 1.4 and 1.5 illustrate this process for two different accounts. Observe that these two accounts have one common PAO story (the story involving Bart Simpson at the Niagara Falls). By sharing stories across accounts we can cut down on the number of PAO stories that the user has to memorize. In Chapter 2 we show that this can be done in a way that preserves strong security properties. The application keeps track of which stories are used for each account, but does not store any of the user's passwords.

Logging into an account. Logging into an account is similar to creating an account. When the user wants to login to an account the application displays the images of the person and scene in each of the four stories associated with that account (e.g., see Figure 1.4). We stress that these images will be the same images

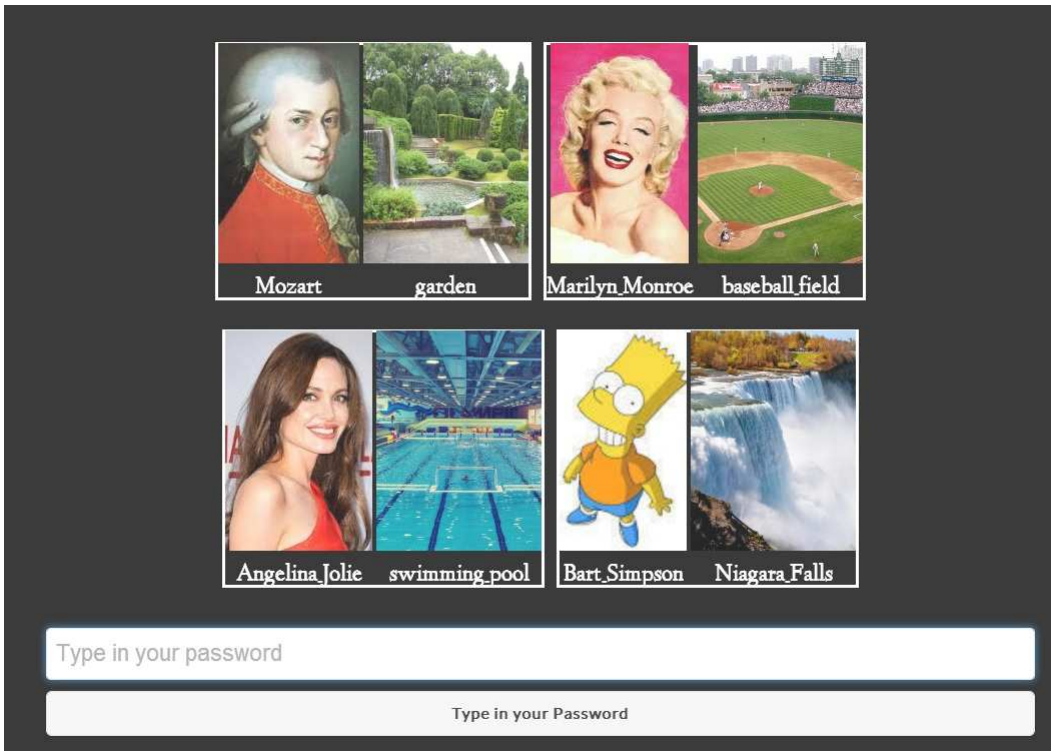


Figure 1.5: Login Example 2

that the user saw when he created the account. To recreate his password the user remembers the secret action and object associated with each story and concatenates all of these words together.

Helping the user remember all his stories. Our application keeps track of when the user rehearses each of his PAO stories. A user naturally rehearses a PAO story whenever he uses that story to login to one of his accounts. If a user has not rehearsed a PAO story in a long time then our application will remind that user to rehearse the story. We call this an extra rehearsal. Observe that the two accounts illustrated in Figures 1.4 and 1.5 have one common PAO story (the story involving Bart Simpson at the Niagara Falls). By reusing stories for different accounts we can minimize the total number of stories that the user needs to remember and maximize the frequency of natural rehearsal for each story. Whenever possible, our application ensures that each story is used as part of the password for a frequently visited account.

Security Guarantees. As an example suppose that the user is willing to memorize 9 PAO stories. Our application can help the user generate 126 different passwords, while providing our user with the following modest security guarantee: any adversary who has seen one of your passwords will not be able to break any of your other passwords in an online attack except with small probability. If the user is willing to memorize 43 PAO stories then our application can help the user generate 110 different passwords, while providing the following much stronger security guarantee: any adversary who has seen one of your passwords will not be able to break any of your other passwords in an offline attack.

1.4 Human Computable Passwords

While the *Shared Cues* password management scheme from Chapter 2 only relies on the human capacity to memorize and retrieve information, *Shared Cues* is secure against at most a constant number of plaintext password leak attacks. Could we improve security (or usability) by having the user perform simple computations to recover his passwords? In Chapter 3 we propose a human computable password scheme and provide strong evidence that the user's passwords will remain secure even after *many* (e.g., 50–100) breaches.

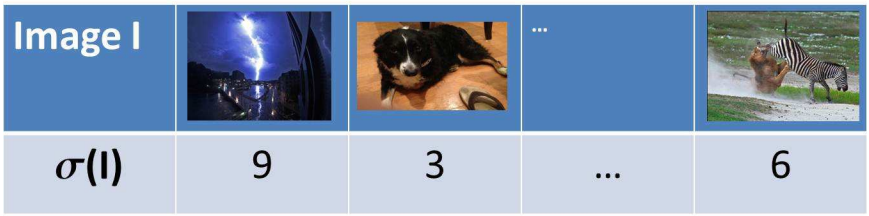


Figure 1.6: Secret Random Mapping from Pictures to Digits

(a) Original photo (an eagle).

(b) Mnemonic to help the user remember $\sigma(eagle) = 2$.

(c) Mnemonic to help the user remember $\sigma(eagle) = 6$.

Figure 1.7: Mnemonics to help memorize the secret mapping σ

1.4.1 Overview

Memorizing a Random Mapping. To begin using our human computable password schemes the user begins by memorizing a secret random mapping $\sigma : [n] \to \{0, \ldots, 9\}$ from n objects (e.g., letters, pictures) to digits. See Figure 1.6 for an example.

The computer can provide the user with mnemonics to help memorize the secret mapping σ — see Figures 1.7b and 1.7c. For example, if we wanted to help the user remember that $\sigma(eagle) = 2$ we would show the user Figure 1.7b. We observe that a $10 \times n$ table of mnemonic images would be sufficient to help the user memorize *any* random mapping σ. We stress that the computer will only save the original image (e.g., Figure 1.7a). The mnemonic image (e.g., Figure 1.7b or 1.7c) would be discarded after the user memorizes $\sigma(eagle)$.

Single-Digit Challenges. In our scheme the user computes each of his passwords by responding to a sequence of single-digit challenges. A single-digit challenge is a tuple $C \in [n]^{14}$ of fourteen objects. See Figure 1.8 for an example. To compute the response $f(\sigma(C))$ to a challenge $C = \{x_0, \ldots, x_{13}\}$ the user

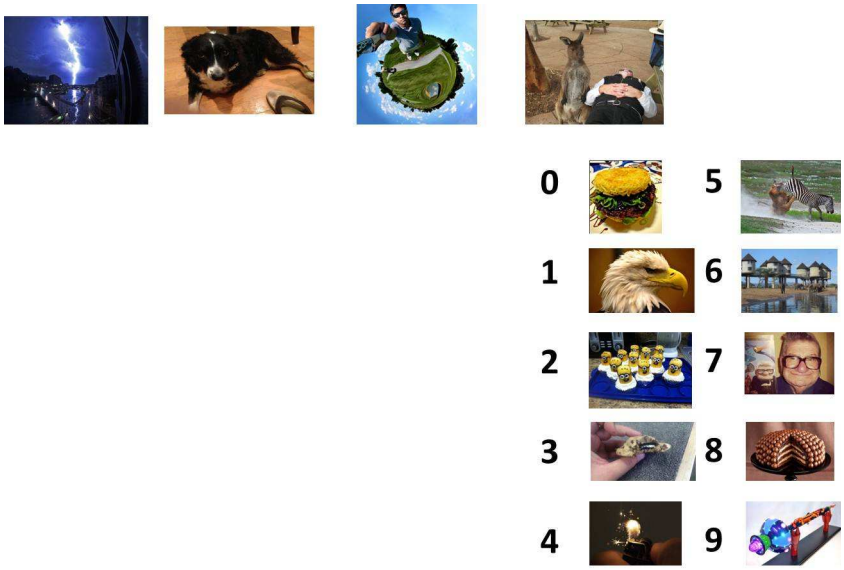


Figure 1.8: A single-digit challenge

computes $f(\sigma(C)) = \sigma\left(x_{\sigma(x_{10})+\sigma(x_{11}) \mod 10}\right) + \sigma(x_{12}) + \sigma(x_{13}) \mod 10$. Observe that this computation involves just three addition operations modulo ten. See Figure 1.9 for an example. In this example the response to the challenge $C = \{x_0 = \text{burger}, x_1 = \text{eagle}, \ldots, x_{10} = \text{lightning}, x_{11} = \text{dog}, x_{12} = \text{man standing on world}, x_{13} = \text{kangaroo}\}$ is

$$\begin{aligned}
f(\sigma(C)) &= \sigma\left(x_{\sigma(x_{10})+\sigma(x_{11}) \mod 10}\right) + \sigma(x_{12}) + \sigma(x_{13}) \mod 10 \\
&= \sigma\left(x_{\sigma(\text{lightning})+\sigma(\text{dog}) \mod 10}\right) \\
&\quad +\sigma(\text{man standing on world}) + \sigma(\text{kangaroo}) \mod 10 \\
&= \sigma\left(x_{9+3 \mod 10}\right) + \sigma(\text{man standing on world}) + \sigma(\text{kangaroo}) \mod 10 \\
&= \sigma(\text{minions}) + \sigma(\text{man standing on world}) + \sigma(\text{kangaroo}) \mod 10 \\
&= 7 + 4 + 5 \mod 10 = 6\,.
\end{aligned}$$

We stress that this computation is done entirely in the user's head. It takes the author of this thesis 7.5 seconds on average to compute each response.

Creating an Account. To help the user create an account the computer would first pick a sequence of single-digit challenges $C_1, \ldots, C_t$, where the security parameter

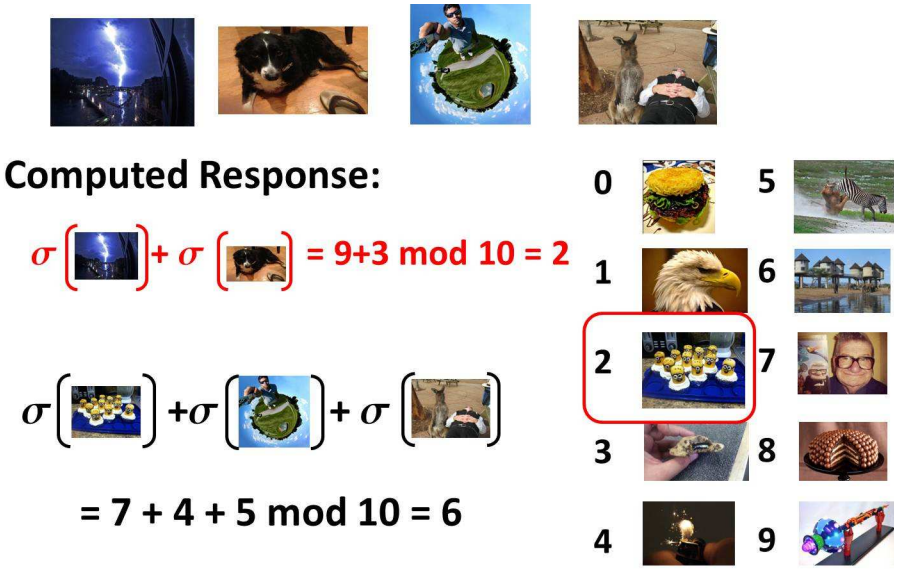


Figure 1.9: Computing the response ($f(\sigma(C)) = 6$) to a single-digit challenge

is typically $t = 10$, and would display the first challenge C_1 to the user — see Figure 1.10 for an example. To compute the first digit of his password the user would compute $f(\sigma(C_1))$. After the user types in the first digit $f(\sigma(C_1))$ of his password the computer will display the second challenge C_2 to the user — see Figure 1.11. After the user creates his account the computer will store the challenges $C_1, \ldots, C_{10}$ in public memory. The password $pw = f(\sigma(C_1)) \ldots f(\sigma(C_t))$ will not be stored.

Authentication. Authenticating is very similar to creating an account. To help the user recompute his password for an account the computer first looks up the challenges $C_1, \ldots, C_t$ which were stored in public memory, and the user authenticates by computing his password $pw = f(\sigma(C_1)) \ldots f(\sigma(C_t))$. We stress that the single-digit challenges the user sees during authentication will be the same single-digit challenges that the user saw when he created the account.

Helping the user remember his secret mapping. As before the computer keeps track of when the user rehearses each value of his secret mapping (e.g., $(i, \sigma(i))$ for each $i \in [n]$), and reminds the user to rehearse any part of his secret mapping that he hasn't used in a long time. One advantage of our human computable password scheme (compared with the *Shared Cues* scheme of Chapter 2) is that most users

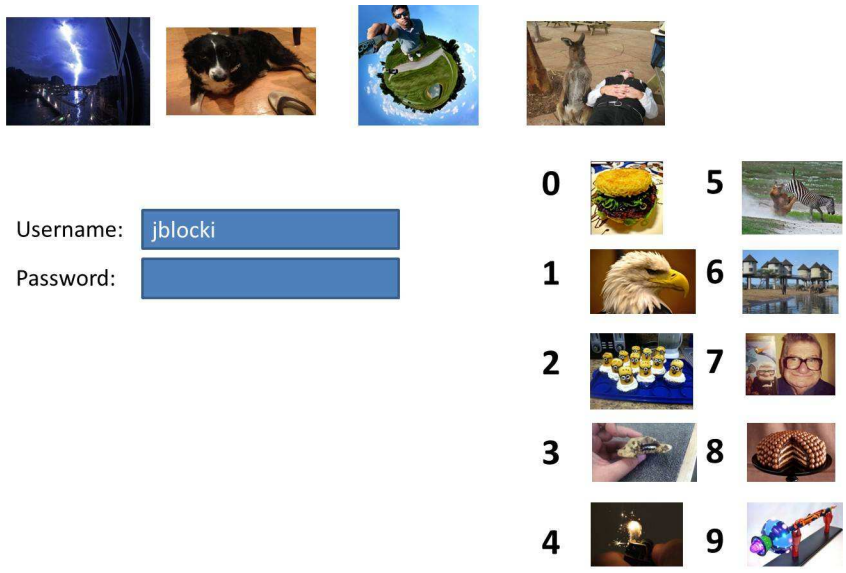


Figure 1.10: Login Screen

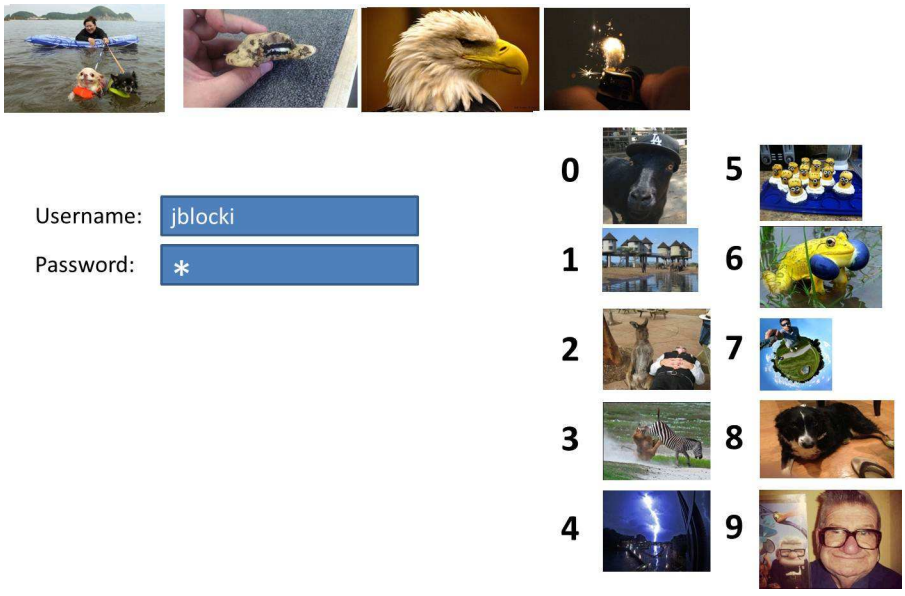


Figure 1.11: Login Screen after the user responds to the first single-digit challenge

will use each part of their secret mapping often enough that they will not need to be reminded to rehearse — see discussion in Chapter 3.6. The disadvantage is that we require the user to spend extra effort computing his passwords each time he authenticates.

Security Guarantees. The security guarantees of our human computable password scheme are much stronger than the security guarantees of *Shared Cues*. We provide strong evidence that any polynomial time adversary needs to see at least $\tilde{\Omega}\left(n^{1.5}/t\right)$ of the user's passwords before he can start to predict the user's passwords at other accounts. For example, if the user memorized a secret mapping from one-hundred pictures to digits then the adversary would need to see approximately one-hundred of the user's ten digit passwords before he could start predicting the user's passwords for other accounts.

Technical Contributions. We develop a general framework for analyzing the security of a human computable password management scheme and we propose two candidate human computable password management schemes in Chapter 3. We give evidence that a human computable password management scheme will remain secure until the adversary has seen at least $\tilde{\Omega}\left(n^{s(f)}\right)$ challenge-response pairs $(C, f(\sigma(C)))$ (see Theorem 6 in Chapter 3). Here, $s(f) = \min\{r(f)/2, g(f)+1\}$ is a composite security parameter which captures $g(f)$ (how many inputs to f need to be fixed to make f linear?) and $r(f)$ (what is the largest value of r such that the distribution over challenge-response pairs are $(r-1)$-wise independent?). We show that $s(f) = 1.5$ for our first scheme and $s(f) = 2$ for our second scheme. In particular we prove that any statistical adversary needs to see at least $\tilde{\Omega}\left(n^{r(f)/2}\right)$ challenge-response pairs $(C, f(\sigma(C)))$ before he can even approximately recover the secret mapping σ. Our lower bound is based on the *statistical dimension* of the distribution over challenge-response pairs induced by f and σ. We stress that our analysis of the statistical dimension applies to arbitrary functions $f : \mathbb{Z}_d^k \to \mathbb{Z}_d$, not just functions that are easy for humans to compute. Our analysis of the statistical dimension generalizes recent results of Feldman et al. [73], which only applied to binary predicates (e.g., $d = 2$), and may be of independent interest.

1.5 Empirical Validation of User Model

We are currently running user studies to empirically test the usability model we presented in Chapter 2.4 and obtain estimates of rehearsal parameters that are specific to the password setting. We are collaborating with the CyLab Usable Privacy and Security Laboratory (CUPS) to conduct these online user studies using Amazon's Mechanical Turk framework. In Chapter 4 we describe the design of the user study and present initial results from the study. Briefly, each participant in the study is asked to memorize several randomly selected actions (e.g., 'swallowing,' 'kicking') and several randomly selected objects (e.g., 'bike,' 'car'). Participants assigned to the mnemonic group were given specific instructions about how to memorize the actions following the *Shared Cues* password management scheme in Chapter 2.6. To help our participants memorize one of their action(s) and object(s) each participant was shown two additional photos of a person and a scene and was asked to imagine the corresponding person-action-object story taking place inside the scene (e.g., the user might be shown a photos of Bill Gates and a beach and asked to imagine "Bill Gates swallowing a bike on the beach."). Other participants were assigned to the standard group and were simply instructed to memorize their actions and objects (e.g., by typing in their words several times). After participants memorized their words we periodically asked them to return to rehearse their words. During each rehearsal participants in the mnemonic group were shown the photos of the person and the scene as a cue to help them remember the associated action and object. Participants in the standard group were simply asked to recall their actions and objects. Each participant was assigned a specific rehearsal schedule (e.g., participants in the aggressive rehearsal group were reminded to rehearse on the following days: $1, 2, 4, 8, 16, 32, 64$). Because the duration of the study is up to one-hundred days we do not yet have the full results from the user study. In Chapter 4.4 we report the results for rehearsals that have been completed. Our results support the hypothesis that recall is significantly improved by asking users to follow specific mnemonic techniques to memorize their actions and objects. Our results also demonstrate the benefit of having several early rehearsals.

1.6 A Defense against Online Attacks

To guard against online attacks many organizations adopt a k-strikes policy in which the user is locked out of his account after k incorrect guesses[4]. However, this defense is often insufficient because many users select trivial passwords like 'password' and '123456' [5]. To provide further protection against online attacks organizations often adopt password composition policies. A password composition policy is a set of rules specifying which passwords users are allowed to select and which passwords users are not allowed to select.

In Chapter 5 we initiate the *algorithmic* study of password composition policies. Such policies restrict the space of passwords to a subset of allowed passwords and force each user to pick a password in this subset. Thus, n users induce a distribution over passwords where for a password w, $\Pr[w] = \frac{1}{n}\left|\{i \ : \ i \text{ picks } w\}\right|$. By declaring different subsets of allowed passwords, different password composition policies induce different distributions. Our work formalizes and addresses the algorithmic problem a server administrator faces when designing a password composition policy; we ask:

> *In what settings can the information about the users' preferences over passwords allow us to design a password composition policy that is* guaranteed *to induce a password distribution as close to uniform as possible?*

1.6.1 Overview

Suppose that a server administrator has created m candidate positive rules $R_1, \ldots, R_m$, where each positive rule $R_i \subseteq \mathcal{P}$ specifies a subset of passwords that the server administrator expects to be strong (e.g., "the set of all passwords that are at least 10 characters long and include at least one number, at least one lowercase character and at least one uppercase character," or "the set all passwords longer than 12 characters"). A password composition policy is given by a subset $S \subseteq [m]$ of active rules. The user is allowed to choose only passwords from the set $\bigcup_{i \in S} R_i$ of permitted passwords (e.g., any password contained in an active rule). Observe that there are 2^m different password composition policies that we could form. Which

[4]Other organizations instead require the user to solve a CAPTCHA [152] after several wrong guesses [60]. This prevents an adversary from maliciously locking out another user (e.g., a competing bidder on eBay).

of these policies is optimal for security? In Chapter 5 we give a sampling algorithm to find the optimal password composition policy. A sampling algorithm is an algorithm that is allowed to sample a random user and ask that user what password he would select under a particular password composition policy $\bigcup_{i \in S} R_i$. Our algorithm is efficient both in its sample complexity and in its running time.

Negative Rules. We could also specify a password composition policy using negative rules (e.g., the user can only select passwords in the set $\mathcal{P} - \bigcup_{i \in S} R_i$). A negative rule specifies a subset of passwords that the server administrator expects to be weak (e.g., "the set of all passwords that are not longer than 12 characters"). In Chapter 5 we show that it is computationally intractable to find the optimal policy in this negative rules setting.

Experiments. We tested our algorithm on a dataset of 32 million user passwords using a small set of rules to find the optimal password composition policy in the positive rules setting. Because we only considered twenty-one different rules we were also also able to find the optimal password composition in the negative rules setting by brute force search (If we used the positive rule "all passwords longer than 9 characters" then the negative version of that rule would be "all passwords that are not longer than 9 characters"). The optimal policy in the positive rules setting was to allow any password $pw \in \mathcal{P}$ that satisfies any of the following conditions: 1) pw is at least 14 characters long, 2) pw contains at least 2 special symbols (e.g., !,*,&,@), OR 3) pw is at least 8 characters long and contains at least one upper case letter and at least one digit. The optimal policy in the negative rules setting was to allow any password $pw \in \mathcal{P}$ that satisfies all of the following conditions: 1) pw is at least 10 characters long, 2) pw contains at least 2 digits, 3) pw contains at least one special symbol (e.g., !,*,&,@), 4) pw contains at least one lowercase letter, AND 5) pw is not in the dictionary. The optimal positive rules policy was nearly as good as the optimal negative rules policy. We also used an efficient heuristic algorithm to find a good policy in the negative rules setting. The optimal positive rules policy was consistently far better than the negative rules policies returned by our heuristic algorithm. Our experiments indicate that it may be advantageous to find the optimal positive rules policy whenever we have a large set of potential rules and are not able to find the optimal negative rules policy by brute force. One limitation of these experiments is that they rely on an assumption about the way user's select passwords. See Section 5.1 for more discussion about this normalized probabilities assumption, and see Section 5.6 for

more details about the experiments.

1.7 A Defense Against Offline Attacks

1.7.1 Background

Any adversary who has obtained the cryptographic hash of a user's password can mount an automated brute-force attack to crack the password by comparing the cryptographic hash of the user's password with the cryptographic hashes of likely password guesses. This attack is called an offline dictionary attack, and there are many password crackers that an adversary could use [63]. Offline dictionary attacks against passwords are — unfortunately — powerful and commonplace [87]. Offline attacks are becoming increasingly dangerous as computing hardware improves (e.g., a modern GPU can evaluate a cryptographic hash function like SHA2 about 250 million times per second [165]) and as more and more training data (e.g., leaked passwords from prior breaches) becomes available [87]. Adversaries have been able to compromise servers at large companies (e.g., Zappos, LinkedIn, Sony, Gawker [5, 9, 10, 11, 13, 28]) resulting in the release of millions of cryptographic password hashes[5]. Symantec reported that compromised passwords have significant economic value to an adversary (e.g., compromised passwords are sold on the black market for between $4 and $30 each) [79].

Because cryptographic hash functions like SHA1, SHA2 and MD5 were designed for fast hardware computation they are poor choices for a password hash function, as they allow an offline adversary to evaluate millions of password guesses per second. One simple way that an organization can mitigate the threat of offline attacks is by using a hash function like BCRYPT [122] which is intentionally designed to be slow to compute. The BCRYPT hash function takes a parameter which allows the programmer to specify how costly the hash computation should be. The downside to this approach is that it also increases costs for the company that stores the passwords (e.g., if we want it to cost the adversary $1,000 for every million guesses then it will also cost the company at least $1,000 for every million login attempts). Another practical way for an organization to help defend against offline attacks is to adopt the practice of password salting (e.g., instead of storing the cryptographic hash of the password $\mathbf{H}(pw)$ the server stores $(\mathbf{H}(pw, r), r)$ for a

[5] In a few of these cases [5, 10] the passwords were stored in the clear.

random string r [16]), which can help mitigate the threat of offline attacks[6].

However, even these defenses will fail to protect many users against offline attacks. It has been repeatedly demonstrated that users tend to select easily guessable passwords [39, 66, 92], and password crackers are able to quickly break many of these passwords[136].

1.7.2 GOTCHAs

In Chapter 6 we introduce GOTCHAs (Generating panOptic Turing Tests to Tell Computers and Humans Apart) as a way of preventing automated offline dictionary attacks against user selected passwords. A GOTCHA is a randomized puzzle generation protocol, which involves interaction between a computer and a human. Informally, a GOTCHA should satisfy two key properties: (1) The puzzles are easy for the human to solve. (2) The puzzles are hard for a computer to solve even if it has the random bits used by the computer to generate the final puzzle — unlike a CAPTCHA [152]. Our main theorem demonstrates that GOTCHAs can be used to mitigate the threat of offline dictionary attacks against passwords by ensuring that a password cracker must receive constant feedback from a human being while mounting an attack. Finally, we provide a candidate construction of GOTCHAs based on inkblot images. This construction relies on the usability assumption that users can *recognize* the phrases that they originally used to describe each inkblot image — a much weaker usability assumption than previous password systems based on inkblots which required users to recall their phrase exactly [147]. We conducted a user study to evaluate the usability of our GOTCHA construction and generated a GOTCHA challenge where artificial intelligence and security researchers are encouraged to try to crack several passwords protected with our scheme.

1.7.3 Overview

Creating an account. To create an account in our scheme the user first selects a username u and a password pw, and sends (u, pw) to the server. After verifying that the username u is available and that pw is permitted under the password

[6]Rainbow tables, which consist of precomputed hashes, are often used by an adversary to significantly speed up a password cracking attack because the same table can be reused to attack each user when the passwords are unsalted [117].

Figure 1.12: GOTCHA Authentication Example

composition policy the server generates ten random Inkblot images $I_1, \ldots, I_{10}$ using (u, pw) as a random seed and sends these images to the user. The user responds by sending back ten labels $\ell_1, \ldots, \ell_{10}$ (one for each Inkblot image). The server stores these labels in a random order.

Authenticating. To authenticate the user sends his username and password to the server (u, pw). The server responds by regenerating the ten random Inkblot images $I_1, \ldots, I_{10}$ using (u, pw) as a random seed and sends these images to the user along with the labels $\ell_1, \ldots, \ell_{10}$ (in a random order). The user matches each label ℓ_i with the appropriate Inkblot image. Figure 1.12 illustrates this process. The server authenticates the user if the password is correct and all (most) of the Inkblot images are matched correctly. We stress that if the user's password is incorrect (e.g., if the user sends (u, pw') where $pw \neq pw'$) then the user will see different Inkblot images $I'_1, \ldots, I'_{10}$ ($I_i \neq I'_i$ for each $i \leq 10$). The labels $\ell_1, \ldots, \ell_{10}$ will be the same in either case.

Server. After the user labels each of his Inkblots the server selects a random permutation $\pi : [10] \to [10]$ and a random salt value $s \in \{0,1\}^*$ and stores the tuple:

$\left(u, s, \ell_{\pi(1)}, \ldots, \ell_{\pi(10)}, \mathbf{H}(u, s, pw, \pi)\right)$. To authenticate the user the server verifies that $\mathbf{H}(u, s, pw, \pi) = \mathbf{H}(u, s, pw', \pi')$, where pw' and π' are the password and permutation provided by the user during authentication. We stress that the server does not store the Inkblot images $I_1, \ldots, I_{10}$ or the random permutation π so an adversary would need to simultaneously guess pw and π to crack the user's password in an offline attack.

Chapter 2

Naturally Rehearsing Passwords

2.1 Introduction

A typical computer user today manages passwords for many different online accounts. Users struggle with this task—often forgetting their passwords or adopting insecure practices, such as using the same password for multiple accounts and selecting weak passwords [39, 52, 75, 102]. While there are many articles, books, papers and even comics about selecting strong individual passwords [4, 46, 49, 82, 109, 132, 146, 162], there is very little work on *password management schemes*—systematic strategies to help users create and remember multiple passwords—that are both usable and secure. In this chapter, we present a rigorous treatment of password management schemes. Our contributions include a formalization of important aspects of a usable scheme, a quantitative security model, and a construction that provably achieves the competing security and usability properties.

Usability Challenge. We consider a setting where a user has two types of memory: *persistent memory* (e.g., a sticky note or a text file on his computer) and *associative memory* (e.g., his own human memory). We assume that persistent memory is reliable and convenient but not private (i.e., accessible to an adversary). In contrast, a user's associative memory is private but lossy—if the user does not rehearse a memory it may be forgotten. While our understanding of human memory is incomplete, it has been an active area of research [23] and there are many mathematical models of human memory [19, 100, 106, 150, 157]. These

models differ in many details, but they all model an associative memory with cue-association pairs: to remember $\hat{a}$ (e.g., a password) the brain associates the memory with a context $\hat{c}$ (e.g., a public hint or cue); such associations are strengthened by rehearsal. A central challenge in designing usable password schemes is thus to create associations that are strong and to maintain them over time through rehearsal. Ideally, we would like the rehearsals to be *natural*, i.e., they should be a side-effect of users' normal online activity. Indeed insecure password management practices adopted by users, such as reusing passwords, improve usability by increasing the number of times a password is naturally rehearsed as users visit their online accounts.

Security Challenge. Secure password management is not merely a theoretical problem—there are numerous real-world examples of password breaches [3, 6, 8, 9, 10, 11, 12, 13, 28, 52, 141]. Adversaries may crack a weak password in an *online attack* where they simply visit the online account and try as many guesses as the site permits. In many cases (e.g., Zappos, LinkedIn, Sony, Gawker [6, 8, 9, 11, 13, 28]) an adversary is able to mount an *offline attack* to crack weak passwords after the cryptographic hash of a password is leaked or stolen. To protect against an offline attack, users are often advised to pick long passwords that include numbers, special characters and capital letters [132]. In other cases even the strongest passwords are compromised via a *plaintext password leak attack* (e.g., [5, 10, 12, 141]), for example, because the user fell prey to a phishing attack or signed into his account on an infected computer or because of server misconfigurations. Consequently, users are typically advised against reusing the same password. A secure password management scheme must protect against all these types of breaches.

Contributions. We precisely define the password management problem in Section 2.3. A password management scheme consists of a *generator*—a function that outputs a set of public cue-password pairs—and a *rehearsal schedule*. The generator is implemented using a computer program whereas the human user is expected to follow the rehearsal schedule for each cue. This division of work is critical—the computer program performs tasks that are difficult for human users (e.g., generating random bits) whereas the human user's associative memory is used to store passwords since the computer's persistent memory is accessible to the adversary.

Quantifying Usability. In the same way that security proofs in cryptography are

based on complexity-theoretic assumptions (e.g., hardness of factoring and discrete logarithm), we quantify usability by introducing *usability assumptions*. In particular, password management relies on assumptions about human memory, e.g., that a user who follows a particular rehearsal schedule will successfully maintain the corresponding memory. These assumptions are informed by research in cognitive science and can be tested empirically. Given rehearsal requirements and a user's visitation schedule for each account, we use the total number of extra rehearsals that the user would have to do to remember all of his passwords as a measure of the usability of the password scheme (Section 2.4). Specifically, in our usability analysis, we use the *Expanding Rehearsal Assumption (ER)* that allows for memories to be rehearsed with exponentially decreasing frequency, i.e., rehearse at least once in the time-intervals (days) $[1, 2)$, $[2, 4)$, $[4, 8)$ and so on. Few long-term memory experiments have been conducted, but *ER* is consistent with known studies [144, 160]. Our memory assumptions are parameterized by a constant σ which represents the strength of the mnemonic devices used to memorize and rehearse a cue-association pair. Strong mnemonic techniques [78, 143] exploit the associative nature of human memory discussed earlier and its remarkable visual/spatial capacity [145].

Quantifying Security. We present a game based security model for a password management scheme (Section 2.5) in the style of exact security definitions [26]. The game is played between a user ($\mathcal{U}$) and a resource-bounded adversary ($\mathcal{A}$) whose goal is to guess one of the user's passwords. Our game models three commonly occurring breaches (online attack, offline attack, plaintext password leak attack).

Our Construction. We present a new password management scheme, which we call *Shared Cues*, and prove that it provides strong security and usability properties (see Section 2.6). Our scheme incorporates powerful mnemonic techniques through the use of public cues (e.g., photos) to create strong associations. The user first associates a randomly generated person-action-object story (e.g., Bill Gates swallowing a bike) with each public cue. We use the Chinese Remainder Theorem to share cues across sites in a way that balances several competing security and usability goals: 1) Each cue-association pair is used by many different web sites (so that most rehearsal requirements are satisfied naturally), 2) the total number of cue-association pairs that the user has to memorize is low, 3) each web site uses several cue-association pairs (so that passwords are secure) and 4) no two web sites share too many cues (so that passwords remain secure even after the adver-

sary obtains some of the user's other passwords). We show that our construction achieves an asymptotically optimal balance between these security and usability goals (Lemma 2, Theorem 3).

2.2 Related Work.

A distinctive goal of our work is to quantify the usability of password management schemes by drawing on ideas from cognitive science and leverage this understanding to design schemes with acceptable usability. We view the results of this paper–employing usability assumptions about rehearsal requirements—as an initial step towards this goal. While the mathematical constructions start from the usability assumptions, the assumptions themselves are empirically testable, e.g., via longitudinal user studies. In contrast, a line of prior work on usability has focused on empirical studies of user behavior including their password management habits [52, 75, 102], the effects of password composition rules (e.g., requiring numbers and special symbols) on individual passwords [34, 101], the memorability of individual system assigned passwords [140], graphical passwords [27, 48], and passwords based on implicit learning [38]. These user studies have been limited in duration and scope (e.g., study retention of a single password over a short period of time). Other work [43] articulates informal, but more comprehensive, usability criteria for password schemes.

Our use of cued recall is driven by evidence that it is much easier than pure recall [23]. We also exploit the large human capacity for visual memory [145] by using pictures as cues. Prior work on graphical passwords [27, 48] also takes advantage of these features. However, our work is distinct from the literature on graphical passwords because we address the challenge of managing multiple passwords. More generally, usable and secure password management is an excellent problem to explore deeper connections between cryptography and cognitive science.

Security metrics for passwords like (partial) guessing entropy (e.g., how many guesses does the adversary need to crack α-fraction of the passwords in a dataset [39, 107, 121]? how many passwords can the adversary break with β guesses per account [45]?) were designed to analyze the security of a dataset of passwords from many users, not the security of a particular user's password management scheme. While these metrics can provide useful feedback about individual passwords (e.g., they rule out some insecure passwords) they do not deal with the complexities of securing multiple accounts against an adversary who may have

gained background knowledge about the user from previous attacks — we refer an interested reader to Appendix 7.7 for more discussion.

Biometrics. Biometric factors like fingerprints and voice recognition have been proposed as an alternative to passwords. For example, the user's computer might record features from the user's fingerprint (a biometric template) and compare them with the features extracted later when the user tries to authenticate — authentication is successful if these biometric templates are 'close enough'. While biometrics do offer a usability advantage (e.g., there is usually nothing for the user to remember) there are many drawbacks: they require additional hardware support and biometric templates are difficult (or impossible) to change if they are compromised. Another security disadvantage is that biometric templates often have low entropy. For example, O'Gorman estimated that biometric templates based on fingerprints, iris scans and voice recognition contain just 13.3, 19.9 and 11.7 bits of entropy respectively[118]. Storing biometric templates is also a challenging research problem because biometric templates are not matched exactly like passwords, but based on the closeness of the two signals. There has been some work in the cryptographic community on developing *secure sketches* or *fuzzy extractors*[65, 104]. A secure sketch is a function that extracts a stable signal, which could be encrypted, from a noisy signal with high minimum entropy. If two noisy signals are 'close' then the secure sketch will extract the same stable signal from both noisy signals. However, if the noisy signal has low minimum entropy like most biometric templates then the stable signal we extract might not be random at all because these techniques entail a small loss in entropy. Consequently, biometric templates are often stored in the clear on the authentication server, which means that an adversary who breaches the server will learn the user's biometric template directly.

Password Managers. A password manager is a computer program that uses an initial password (often called a master password) to generate password(s) for the user. For example, if the user wanted to generate a password for a domain D using the initial password pw_D then the password manager PwdHash[130] would generate the password $\mathbf{H}(D, pw_D)$[1]. Even if the user reuses the same initial

[1]Other password managers like *1Password* and *LastPass* use a master password to encrypt a database of passwords, which could be stored in untrusted memory (e.g., USB sticks, the cloud). Gasti and Rasmussen showed that most of these password managers were vulnerable to attacks by adversary who could read the encrypted database – even in the user's master password was

password pw_D for a different domain D' the final password $\mathbf{H}(D', pw_D)$ will still be different. Password mangers like PwdHash[130] do provide users with several security advantages: the user can be sure that his initial password is always properly hashed and encrypted before it is sent to the authentication server[2] and an adversary who observes the password $\mathbf{H}(D', pw)$ for a domain D' will not be able to guess the password $\mathbf{H}(D, pw)$ for another domain D unless he can break the master password pw. However, password reuse is still a security problem even if the user adopts a password manager. If a user selects one master password to generate all of his passwords then an adversary who obtains this 'master password' would be able to compromise all of the user's accounts. This master password could be a tempting target for an adversary who is looking to maximize the return on investment of his attack. An adversary who has obtained the cryptographic hash of the user's final password for a domain D would still be able to execute an offline attack against the user's initial password (e.g., by applying the PwdHash function as an extra step to verify each guess). Even if the master password is strong enough to resist offline attacks the master password could still be exposed whenever the user types it in to generate one of his passwords[3]. A user study conducted by Chiasson et al. indicated that the master passwords of many PwdHash users may still be vulnerable to phishing attacks because of confusing user interfaces[54]. Unless the user can be sure that *every* device (e.g., laptop, smartphone, friend's computer, public computer) he ever uses to login is malware free and that there are no 'hidden cameras' at *any* location (e.g., library, home, friend's house, office, coffee shop) from which the user logs into an account then the user's master password may be vulnerable. We stress that the password management schemes we propose could be used in conjunction with a password manager like PwdHash (e.g., instead of using one master password for all of his accounts the user would create a different initial password for each domain D by following our password management scheme). In this case the user would get all of the security benefits of a password manager (e.g., by ensuring that passwords are properly encrypted and hashed before they are sent to a server) without the single-point of failure problem.

strong[81].

[2]Unfortunately, some sites do not always properly hash the passwords stored on their servers[5, 13, 40]. Other sites do not properly encrypt their users' passwords before they are transmitted over the Internet[40].

[3]While this is a concern with or without a password manager, the damage of a plaintext password leak attack is potentially much greater if the user only has one master password.

Combinatorial Designs. Our notion of (n, ℓ, γ)-sharing set families (definition 5) is equivalent to Nisan and Wigderson's definition of a (k, m)-design [115]. The problem of finding maximally sized (n, ℓ, γ)–sharing set families was considered at least as early as 1956 by Paul Erdős and Alfréd Rényi [69], and applications of some of these families may have been considered by Euler [71]. Erdős explored properties of these families several times [68] [70], and Rödl built on his work [127]. (n, ℓ, γ)–sharing set families were rediscovered by Nisan and Wigderson [115], who used them to design a pseudorandom number generator. Trevisan showed how to use (n, ℓ, γ)–sharing set families to construct pseudorandom extractors [149]. Because Nisan and Wigderson were focused on a different application (constructing pseudorandom bit generators) the range of parameters that they consider are not suitable for our password setting in which ℓ and γ are constants. In Appendix 7.7.2 we show that our construction of (n, ℓ, γ)-sharing set families has interesting applications in the construction of parallel pseudorandom number generators. See Appendix 7.7 for more discussion of (n, ℓ, γ)-sharing set families.

2.3 Definitions

We use $\mathcal{P}$ to denote the space of possible passwords. A password management scheme needs to generate m passwords $p_1, ..., p_m \in \mathcal{P}$ — one for each account A_i.

2.3.1 Associative Memory and Cue-Association Pairs

Human memory is associative. Competitors in memory competitions routinely use mnemonic techniques (e.g., the method of loci [143]) which exploit associative memory[78]. For example, to remember the word 'apple' a competitor might imagine a giant apple on the floor in his bedroom. The bedroom now provides a context which can later be used as a cue to help the competitor remember the word apple. We use $\hat{c} \in C$ to denote the cue, and we use $\hat{a} \in \mathcal{AS}$ to denote the corresponding association in a cue-association pair $(\hat{c}, \hat{a})$. Physically, $\hat{c}$ (resp. $\hat{a}$) might encode the excitement levels of the neurons in the user's brain when he thinks about his bedroom (resp. apples) [106].

We allow the password management scheme to store m sets of public cues $c_1, ..., c_m \subset C$ in persistent memory to help the user remember each password. Because these cues are stored in persistent memory they are always available to the adversary as well as the user. Notice that a password may be derived from

multiple cue-association pairs. We use $\hat{c} \in C$ to denote a cue, $c \subset C$ to denote a set of cues, and $C = \bigcup_{i=1}^{m} c_i$ to denote the set of all cues — $n = |C|$ denotes the total number of cue-association pairs that the user has to remember.

2.3.2 Visitation Schedules and Rehearsal Requirements

Each cue $\hat{c} \in C$ may have a rehearsal schedule to ensure that the cue-association pair $(\hat{c}, \hat{a})$ is maintained.

Definition 1. *A rehearsal schedule for a cue-association pair* $(\hat{c}, \hat{a})$ *is a sequence of times* $t_0^{\hat{c}} < t_1^{\hat{c}} < ....$ *For each* $i \geq 0$ *we have a* rehearsal requirement, *the cue-association pair must be rehearsed at least once during the time window* $\left[t_i^{\hat{c}}, t_{i+1}^{\hat{c}}\right) = \{x \in \mathbb{R} \,\middle|\, t_i^{\hat{c}} \leq x < t_{i+1}^{\hat{c}}\}$.

A rehearsal schedule is *sufficient* if a user can maintain the association $(\hat{c}, \hat{a})$ by following the rehearsal schedule. We discuss sufficient rehearsal assumptions in Section 2.4. The length of each interval $\left[t_i^{\hat{c}}, t_{i+1}^{\hat{c}}\right)$ may depend on the strength of the mnemonic technique used to memorize and rehearse a cue-association pair $(\hat{c}, \hat{a})$ as well as i — the number of prior rehearsals. For notational convenience, we use a function $R : C \times \mathbb{N} \to \mathbb{R}$ to specify the rehearsal requirements (e.g., $R(\hat{c}, j) = t_j^{\hat{c}}$), and we use $\mathcal{R}$ to denote a set of rehearsal functions.

A visitation schedule for an account A_i is a sequence of real numbers $\tau_0^i < \tau_1^i < \ldots$, which represent the times when the account A_i is visited by the user. We do not assume that the exact visitation schedules are known a priori. Instead we model visitation schedules using a random process with a known parameter λ_i based on $\mathbb{E}\left[\tau_{j+1}^i - \tau_j^i\right]$ — the average time between consecutive visits to account A_i. A rehearsal requirement $\left[t_i^{\hat{c}}, t_{i+1}^{\hat{c}}\right)$ can be satisfied naturally if the user visits a site A_j that uses the cue $\hat{c}$ $\left(\hat{c} \in c_j\right)$ during the given time window. Formally,

Definition 2. *We say that a rehearsal requirement* $\left[t_i^{\hat{c}}, t_{i+1}^{\hat{c}}\right)$ *is* naturally satisfied *by a visitation schedule* $\tau_0^i < \tau_1^i < \ldots$ *if* $\exists j \in [m], k \in \mathbb{N}$ *s.t* $\hat{c} \in c_j$ *and* $\tau_k^j \in \left[t_i^{\hat{c}}, t_{i+1}^{\hat{c}}\right)$. *We use*

$$XR_{t,\hat{c}} = \left|\left\{i \,\middle|\, t_{i+1}^{\hat{c}} \leq t \wedge \forall j, k. \left(\hat{c} \notin c_j \vee \tau_k^j \notin \left[t_i^{\hat{c}}, t_{i+1}^{\hat{c}}\right)\right)\right\}\right| ,$$

to denote the number of rehearsal requirements that are not naturally satisfied by the visitation schedule during the time interval $[0, t]$.

We use rehearsal requirements and visitation schedules to quantify the usability of a password management scheme by measuring the total number of extra rehearsals. If a cue-association pair $(\hat{c}, \hat{a})$ is not rehearsed naturally during the interval $\left[t_i^{\hat{c}}, t_{i+1}^{\hat{c}}\right)$ then the user needs to perform an extra rehearsal to maintain the association. Intuitively, $XR_{t,\hat{c}}$ denotes the total number of extra rehearsals of the cue-association pair $(\hat{c}, \hat{a})$ during the time interval $[0, t]$. We use $XR_t = \sum_{\hat{c} \in C} XR_{t,\hat{c}}$ to denote the total number of extra rehearsals during the time interval $[0, t]$ to maintain all of the cue-association pairs.

Usability Goal: Minimize $\mathbb{E}[XR_t]$.

2.3.3 Password Management Scheme

A password management scheme includes a generator $\mathcal{G}_m$ and a rehearsal schedule $R \in \mathcal{R}$. The generator $\mathcal{G}_m\left(k, b, \vec{\lambda}, R\right)$ utilizes a user's knowledge $k \in \mathcal{K}$, random bits $b \in \{0,1\}^*$ to generate passwords $p_1, ..., p_m$ and public cues $c_1, ..., c_m \subseteq \mathcal{C}$. $\mathcal{G}_m$ may use the rehearsal schedule R and the visitation schedules $\vec{\lambda} = \langle \lambda_1, ..., \lambda_m \rangle$ of each site to help minimize $\mathbb{E}[XR_t]$. Because the cues $c_1, ...c_m$ are public they may be stored in persistent memory along with the code for the generator $\mathcal{G}_m$. In contrast, the passwords $p_1, ...p_m$ must be memorized and rehearsed by the user (following R) so that the cue association pairs (c_i, p_i) are maintained in his associative memory.

Definition 3. *A password management scheme is a tuple $\langle \mathcal{G}_m, R \rangle$, where $\mathcal{G}_m$ is a function $\mathcal{G}_m : \mathcal{K} \times \{0,1\}^* \times \mathbb{R}^m \times \mathcal{R} \rightarrow \left(\mathcal{P} \times 2^{\mathcal{C}}\right)^m$ and a $R \in \mathcal{R}$ is a rehearsal schedule which the user must follow for each cue.*

Our security analysis is not based on the secrecy of $\mathcal{G}_m$, k or the public cues $C = \bigcup_{i=1}^{m} c_i$. The adversary will be able to find the cues $c_1, ..., c_m$ because they are stored in persistent memory. In fact, we also assume that the adversary has background knowledge about the user (e.g., he may know k), and that the adversary knows the password management scheme $\mathcal{G}_m$. The only secret is the random string b used by $\mathcal{G}_m$ to produce $p_1, ..., p_m$.

Example Password Management Schemes. Most password suggestions are too vague (e.g.,"pick an obscure phrase that is personally meaningful to you") to satisfy the precise requirements of a password management scheme — formal security proofs of protocols involving human interaction can break down when humans

behave in unexpected ways due to vague instructions [123]. We consider the following formalization of password management schemes: (1) *Reuse Weak* — the user selects a random dictionary word w (e.g., from a dictionary of $20,000$ words) and uses $p_i = w$ as the password for every account A_i. (2) *Reuse Strong* — the user selects four random dictionary words (w_1, w_2, w_3, w_4) and uses $p_i = w_1 w_2 w_3 w_4$ as the password for every account A_i. (3) *Lifehacker* (e.g., [4]) — The user selects three random words (w_1, w_2, w_3) from the dictionary as a base password $b = w_1 w_2 w_3$. The user also selects a random derivation rule d to derive a string from each account name (e.g., use the first three letters of the account name, use the first three vowels in the account name). The password for account A_i is $p_i = bd(A_i)$ where $d(A_i)$ denotes the derived string. (4) *Strong Random and Independent* — for each account A_i the user selects four fresh words independently at random from the dictionary and uses $p_i = w_1^i w_2^i w_3^i w_4^i$. Schemes (1)-(3) are formalizations of popular password management strategies. We argue that they are popular because they are easy to use, while the strongly secure scheme *Strong Random and Independent* is unpopular because the user must spend a lot of extra time rehearsing his passwords. See Appendix 7.6 for more discussion of the security and usability of each scheme.

2.4 Usability Model

People typically adopt their password management scheme based on usability considerations instead of security considerations [75]. Our usability model can be used to explain why users tend to adopt insecure password management schemes like *Reuse Weak*, *Lifehacker*, or *Reuse Strong*. Our usability metric measures the extra effort that a user has to spend rehearsing his passwords. Our measurement depends on three important factors: rehearsal requirements for each cue, visitation rates for each site, and the total number of cues that the user needs to maintain. Our main technical result in this section is Theorem 1 — a formula to compute the total number of extra rehearsals that a user has to do to maintain all of his passwords for t days. To evaluate the formula we need to know the rehearsal requirements for each cue-association pair as well as the visitation frequency λ_i for each account A_i.

2.4.1 Rehearsal Requirements

If the password management scheme does not mandate sufficient rehearsal then the user might forget his passwords. Few memory studies have attempted to study memory retention over long periods of time so we do not know exactly what these rehearsal constraints should look like. While security proofs in cryptography are based on assumptions from complexity theory (e.g., hardness of factoring and discrete logarithm), we need to make assumptions about humans. For example, the assumption behind CAPTCHAs is that humans are able to perform a simple task like reading garbled text [152]. A rehearsal assumption specifies what types of rehearsal constraints are sufficient to maintain a memory. We consider two different assumptions about sufficient rehearsal schedules: Constant Rehearsal Assumption (CR) and Expanding Rehearsal Assumption (ER). Because some mnemonic devices are more effective than others (e.g., many people have amazing visual and spatial memories [145]) our assumptions are parameterized by a constant σ which represents the strength of the mnemonic devices used to memorize and rehearse a cue association pair.

Constant Rehearsal Assumption (CR): The rehearsal schedule given by $R(\hat{c}, i) = i\sigma$ is sufficient to maintain the association $(\hat{c}, \hat{a})$.

CR is a pessimistic assumption — it asserts that memories are not permanently strengthened by rehearsal. The user must continue rehearsing every σ days — even if the user has frequently rehearsed the password in the past.

Expanding Rehearsal Assumption (ER): The rehearsal schedule given by $R(\hat{c}, i) = 2^{i\sigma}$ is sufficient to maintain the association $(\hat{c}, \hat{a})$.

ER is more optimistic than CR — it asserts that memories are strengthened by rehearsal so that memories need to be rehearsed less and less frequently as time passes. If a password has already been rehearsed i times then the user does not have to rehearse again for $2^{i\sigma}$ days to satisfy the rehearsal requirement $\left[2^{i\sigma}, 2^{i\sigma+\sigma}\right)$. ER is consistent with several long term memory experiments [144],[23, Chapter 7], [160] — we refer the interested reader to Appendix 7.6 for more discussion. We also consider the rehearsal schedule $R(\hat{c}, i) = i^2$ (derived from [18, 151]) in Appendix 7.6 — the usability results are almost indentical to those for ER.

Schedule λ	$\frac{1}{1}$	$\frac{1}{3}$	$\frac{1}{7}$	$\frac{1}{31}$	$\frac{1}{365}$
Very Active	10	10	10	10	35
Typical	5	10	10	10	40
Occasional	2	10	20	20	23
Infrequent	0	2	5	10	58

Table 2.1: Visitation Schedules - number of accounts visited with frequency λ (visits/days)

2.4.2 Visitation Schedules.

Visitation schedules may vary greatly from person to person. For example, a 2006 survey about Facebook usage showed that 47% of users logged in daily, 22.4% logged in about twice a week, 8.6% logged in about once a week, and 12% logged in about once a month[15]. We use a Poisson arrival process with parameter λ_i to model the visitation schedule for site A_i. We formally define a Poisson arrival process in Appendix 7.1 (see Definition 18). One nice property of a Poisson arrival process with parameter λ is that the value $\frac{1}{\lambda}$ represents the average time between consecutive arrivals (see Fact 4 in Appendix 10.1). We assume that the value of $1/\lambda_i$ — the average inter-visitation time — is known. For example, some websites (e.g., gmail) may be visited daily ($\lambda_i = 1/1$ day) while other websites (e.g., IRS) may only be visited once a year on average (e.g., $\lambda_i = 1/365$ days). The Poisson process has been used to model the distribution of requests to a web server [125]. While the Poisson process certainly does not perfectly model a user's visitation schedule (e.g., visits to the IRS websites may be seasonal) we believe that the predictions we derive using this model will still be useful in guiding the development of usable password management schemes. While we focus on the Poisson arrival process, our analysis could be repeated for other random processes.

We consider four very different types of internet users: very active, typical, occasional and infrequent. Each user account A_i may be visited daily (e.g., $\lambda_i = 1$), every three days ($\lambda_i = 1/3$), every week (e.g. $\lambda_i = 1/7$), monthly ($\lambda_i = 1/31$), or yearly ($\lambda_i = 1/365$) on average. See Table 2.1 to see the full visitation schedules we define for each type of user. For example, our very active user has 10 accounts he visits daily and 35 accounts he visits annually.

Extra Rehearsals. Theorem 1 leads us to our key observation: cue-sharing benefits users both by (1) reducing the number of cue-association pairs that the user

Assumption	CR ($\sigma = 1$)		ER ($\sigma = 1$)	
Schedule/Scheme	B+D	SRI	B+D	SRI
Very Active	≈ 0	23,396	.023	420
Typical	.014	24,545	.084	456.6
Occasional	.05	24,652	.12	502.7
Infrequent	56.7	26,751	1.2	564

Table 2.2: $\mathbb{E}[XR_{365}]$: Extra Rehearsals over the first year for both rehearsal assumptions.
B+D: *Lifehacker*
SRI: *Strong Random and Independent*

has to memorize and (2) by increasing the rate of natural rehearsals for each cue-association pair. For example, a active user with 75 accounts would need to perform 420 extra-rehearsals over the first year to satisfy the rehearsal requirements given by ER if he adopts *Strong Random and Independent* or just 0.023 with *Lifehacker* — see Table 2.2. The number of unique cue-association pairs n decreased by a factor of 75, but the total number of extra rehearsals $\mathbb{E}[XR_{365}]$ decreased by a factor of $8,260.8 \approx 75 \times 243$ due to the increased natural rehearsal rate.

Theorem 1. *Let* $i_{\hat{c}}* = \left(\arg\max_x t_x^{\hat{c}} < t\right) - 1$ *then*

$$\mathbb{E}[XR_t] = \sum_{\hat{c} \in C} \sum_{i=0}^{i_{\hat{c}}*} \exp\left(-\left(\sum_{j:\hat{c} \in c_j} \lambda_j\right)\left(t_{i+1}^{\hat{c}} - t_i^{\hat{c}}\right)\right)$$

Theorem 1 follows easily from Lemma 1 and linearity of expectations. Each cue-association pair $(\hat{c}, \hat{a})$ is rehearsed naturally whenever the user visits *any* site which uses the public cue $\hat{c}$. Lemma 1 makes use of two key properties of Poisson processes: (1) The natural rehearsal schedule for a cue $\hat{c}$ is itself a Poisson process, and (2) Independent Rehearsals - the probability that a rehearsal constraint is satisfied is independent of previous rehearsal constraints.

Lemma 1. *Let* $S_{\hat{c}} = \{i \mid \hat{c} \in c_i\}$ *and let* $\lambda_{\hat{c}} = \sum_{i \in S_{\hat{c}}} \lambda_i$ *then the probability that the cue* $\hat{c}$ *is not naturally rehearsed during time interval* $[a, b]$ *is* $\exp(-\lambda_{\hat{c}}(b - a))$.

2.5 Security Model

In this section we present a game based security model for a password management scheme. The game is played between a user ($\mathcal{U}$) and a resource bounded adversary ($\mathcal{A}$) whose goal is to guess one of the user's passwords. We demonstrate how to select the parameters of the game by estimating the adversary's amortized cost of guessing. Our security definition is in the style of the exact security definitions of Bellare and Rogaway [26]. Previous security metrics (e.g., min-entropy, password strength meters) fail to model the full complexity of the password management problem (see Appendix 7.3 for more discussion). By contrast, we assume that the adversary knows the user's password management scheme and is able to see any public cues. Furthermore, we assume that the adversary has background knowledge (e.g., birth date, hobbies) about the user (formally, the adversary is given $k \in \mathcal{K}$). Many breaches occur because the user falsely assumes that certain information is private (e.g., birth date, hobbies, favorite movie)[7, 134].

Adversary Attacks. Before introducing our game based security model we consider the attacks that an adversary might mount. We group the adversary attacks into three categories: *Online Attack* — the adversary knows the user's ID and attempts to guess the password. The adversary will get locked out after s incorrect guesses (strikes). *Offline Attack* — the adversary learns both the cryptographic hash of the user's password and the hash function and can try many guesses $q_{\$B}$. The adversary is only limited by the resources B that he is willing to invest to crack the user's password. *Plaintext Password Leak Attack* — the adversary directly learns the user's password for an account. Once the adversary recovers the password p_i the account A_i has been compromised. However, a secure password management scheme should prevent the adversary from compromising more accounts.

We model online and offline attacks using a guess-limited oracle. Let $S \subseteq [m]$ be a set of indices, each representing an account. A guess-limited oracle $O_{S,q}$ is a blackbox function with the following behavior: 1) After q queries $O_{S,q}$ stops answering queries. 2) $\forall i \notin S$, $O_{S,q}(i,p) = \perp$ 3) $\forall i \in S$, $O_{S,q}(i,p_i) = 1$ and 4) $\forall i \in S, p \neq p_i$, $O_{S,q}(i,p) = 0$. Intuitively, if the adversary steals the cryptographic password hashes for accounts $\{A_i \mid i \in S\}$, then he can execute an offline attack against each of these accounts. We also model an online attack against account A_i with the guess-limited oracle $O_{\{i\},s}$ with $s \ll q$ (e.g., $s = 3$ models a three-strikes policy in which a user is locked out after three incorrect guesses).

Game Based Definition of Security. Our cryptographic game proceeds as follows:
Setup: The user $\mathcal{U}$ starts with knowledge $k \in \mathcal{K}$, visitation schedule $\vec{\lambda} \in \mathbb{R}^m$, a random sequence of bits $b \in \{0,1\}^*$ and a rehearsal schedule $R \in \mathcal{R}$. The user runs $\mathcal{G}_m\left(k, b, \vec{\lambda}, R\right)$ to obtain m passwords $p_1, ..., p_m$ and public cues $c_1, ..., c_m \subseteq C$ for accounts $A_1, ..., A_m$. The adversary $\mathcal{A}$ is given k, $\mathcal{G}_m$, $\vec{\lambda}$ and $c_1, ..., c_m$.
Plaintext Password Leak Attack: $\mathcal{A}$ adaptively selects a set $S \subseteq [m]$ s.t $|S| \leq r$ and receives p_i for each $i \in S$.
Offline Attack: $\mathcal{A}$ adaptively selects a set $S' \subseteq [m]$ s.t. $|S'| \leq h$, and is given blackbox access to the guess-limited offline oracle $O_{S',q}$.
Online Attack: For each $i \in [m] - S$, the adversary is given blackbox access to the guess-limited offline oracle $O_{\{i\},s}$.
Winner: $\mathcal{A}$ wins by outputting (j, p), where $j \in [m] - S$ and $p = p_j$.

We use $\textbf{AdvWins}\left(k, b, \vec{\lambda}, \mathcal{G}_m, \mathcal{A}\right)$ to denote the event that the adversary wins.

Definition 4. *We say that a password management scheme $\mathcal{G}_m$ is (q, δ, m, s, r, h)-secure if for every $k \in \mathcal{K}$ and adversary strategy $\mathcal{A}$ we have*

$$\Pr_b\left[\textbf{AdvWins}\left(k, b, \vec{\lambda}, \mathcal{G}_m, \mathcal{A}\right)\right] \leq \delta\ .$$

Discussion: Observe that the adversary cannot win by outputting the password for an account that he already compromised in a plaintext password leak. For example, suppose that the adversary is able to obtain the plaintext passwords for $r = 2$ accounts of his choosing: p_i and p_j. While each of these breaches is arguably a success for the adversary the user's password management scheme cannot be blamed for any of these breaches. However, if the adversary can use this information to crack any of the user's other passwords then the password management scheme can be blamed for the additional breaches. For example, if our adversary is also able to use p_i and p_j to crack the cryptographic password hash $\mathbf{H}(p_t)$ for another account A_t in at most q guesses then the password management scheme could be blamed for the breach of account A_t. Consequently, the adversary would win our game by outputting (t, p_t). If the password management scheme is $(q, 10^{-4}, m, s, 2, 1)$-secure then the probability that the adversary could win is at most 10^{-4} — so there is a very good chance that the adversary will fail to crack p_t.

Economic Upper Bound on q. Our guessing limit q is based on a model of a resource constrained adversary who has a budget of $\$B$ to crack one of the user's

passwords. We use the upper bound $q_B = \$B/C_q$, where $C_q = \$R/f_{\mathbf{H}}$ denotes the amortized cost per query (e.g., cost of renting ($\$R$) an hour of computing time on Amazon's cloud [1] divided by $f_{\mathbf{H}}$ — the number of times the cryptographic hash function can be evaluated in an hour.) We experimentally estimate $f_{\mathbf{H}}$ for SHA1, MD5 and BCRYPT[122] — more details can be found in Appendix 7.5. Assuming that the BCRYPT password hash function [122] was used to hash the passwords we get $q_B = B\left(5.155 \times 10^4\right)$ — we also consider cryptographic hash functions like SHA1, MD5 in Appendix 7.5. In our security analysis we focus on the specific value $q_{\$10^6} = 5.155 \times 10^{10}$ — the number of guesses the adversary can try if he invests $\$10^6$ to crack the user's password.

Sharing and Security. In Section 2.4 we saw that sharing public cues across accounts improves usability by (1) reducing the number of cue-association pairs that the user has to memorize and rehearse, and (2) increasing the rate of natural rehearsals for each cue-association pair. However, conventional security wisdom says that passwords should be chosen independently. Is it possible to share public cues, and satisfy the strong notion of security from Definition 4? Theorem 2 demonstrates that public cues can be shared securely provided that the public cues $\{c_1, \ldots, c_m\}$ are a (n, ℓ, γ)-sharing set family. The proof of Theorem 2 can be found in Appendix 7.1.

Definition 5. *We say that a set family $\mathcal{S} = \{S_1, ..., S_m\}$ is (n, ℓ, γ)-sharing if (1) $\left|\bigcup_{i=1}^{m} S_i\right| = n$, (2)$|S_i| = \ell$ for each $S_i \in \mathcal{S}$, and (3) $\left|S_i \cap S_j\right| \leq \gamma$ for each pair $S_i \neq S_j \in \mathcal{S}$.*

Theorem 2. *Let $\{c_1, \ldots, c_m\}$ be a (n, ℓ, γ)-sharing set of m public cues produced by the password management scheme $\mathcal{G}_m$. If each $a_i \in \mathcal{AS}$ is chosen uniformly at random then $\mathcal{G}_m$ satisfies (q, δ, m, s, r, h)-security for $\delta \leq \frac{q}{|\mathcal{AS}|^{\ell-\gamma r}}$ and any h.*

Discussion: To maintain security it is desirable to have ℓ large (so that passwords are strong) and γ small (so that passwords remain strong even after an adversary compromises some of the accounts). To maintain usability it is desirable to have n small (so that the user doesn't have to memorize many cue-association pairs). There is a fundamental trade-off between security and usability because it is difficult to achieve these goals without making n large.

For the special case $h = 0$ (e.g., the adversary is limited to online attacks) the security guarantees of Theorem 2 can be further improved to $\delta \leq \frac{sm}{|A|^{\ell-\gamma r}}$ because the adversary is actually limited to sm guesses.

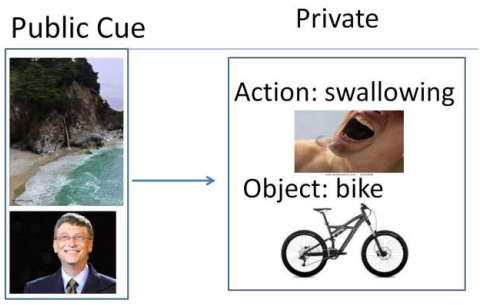


Figure 2.1: PAO Story with Cue

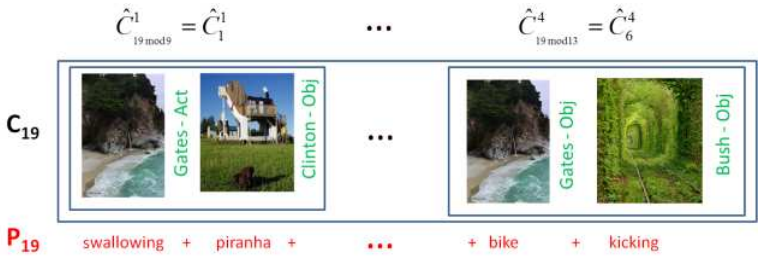


Figure 2.2: Account A_{19} using *Shared Cues* with the $(43, 4, 1)$-sharing set family **CRT** $(90, 9, 10, 11, 13)$. For convenience, we adopt the notation $\hat{c}^1_j = \hat{c}_j$ and $\hat{c}^4_j = \hat{c}_{9+10+11+j}$.

2.6 Our Construction

We present *Shared Cues*— a novel password management scheme which balances security and usability considerations. The key idea is to strategically share cues to make sure that each cue is rehearsed frequently while preserving strong security goals. Our construction may be used in conjunction with powerful cue-based mnemonic techniques like memory palaces [143] and person-action-object stories [78] to increase σ — the association strength constant. We use person-action-object stories as a concrete example.

Person-Action-Object Stories. A random person-action-object (PAO) story for a person (e.g., Bill Gates) consists of a random action $a \in \mathcal{ACT}$ (e.g., swallowing) and a random object $o \in \mathcal{OBJ}$ (e.g., a bike). While PAO stories follow a very simple syntactic pattern they also tend to be surprising and interesting because the story is often unexpected (e.g., Bill Clinton kissing a piranha, or Michael Jordan torturing

a lion). There is good evidence that memorable phrases tend to use uncommon combinations of words in common syntactic patterns [61]. Each cue $\hat{c} \in C$ includes a person (e.g., Bill Gates) as well as a picture. To help the user memorize the story we tell him to imagine the scene taking place inside the picture (see Figure 2.1 for an example). We use Algorithm 2.2 to automatically generate random PAO stories. The cue $\hat{c}$ could be selected either with the user's input (e.g., use the name of a friend and a favorite photograph) or automatically. As long as the cue $\hat{c}$ is fixed before the associated action-object story is selected the cue-association pairs will satisfy the independence condition of Theorem 2.

2.6.1 Constructing (n, ℓ, γ)-sharing set families

We use the Chinese Remainder Theorem to construct nearly optimal (n, ℓ, γ)-sharing set families. Our application of the Chinese Remainder Theorem is different from previous applications of the Chinese Remainder Theorem in cryptography (e.g., faster RSA decryption algorithm [64], secret sharing [21]). The inputs $n_1, ..., n_\ell$ to Algorithm 2.1 should be co-prime so that we can invoke the Chinese Remainder Theorem — see Figure 2.2 for an example of our construction with $(n_1, n_2, n_3, n_4) = (9, 10, 11, 13)$.

Algorithm 2.1 CRT $(m, n_1, ..., n_\ell)$

Input: m, and $n_1, ..., n_\ell$.
for $i = 1 \rightarrow m$ **do**
  $S_i \leftarrow \emptyset$
  for $j = 1 \rightarrow \ell$ **do**
    $N_j \leftarrow \sum_{i=1}^{j-1} n_j$
    $S_i \leftarrow S_i \cup \left\{\left(i \mod n_j\right) + N_j\right\}$
return $\{S_1, \ldots, S_m\}$

Algorithm 2.2 CreatePAOStories

Input: n, random bits b, images $I_1, ..., I_n$, and names $P_1, ..., P_n$.
for $i = 1 \rightarrow n$ **do**
  $a_i \xleftarrow{\$} \mathcal{ACT}$, $o_i \xleftarrow{\$} \mathcal{OBJ}$ %Using random bits b
%Split PAO stories to optimize usability
for $i = 1 \rightarrow n$ **do**
  $\hat{c}_i \leftarrow ((I_i, P_i, \text{'}Act\text{'}), (I_{i+1 \mod n}, P_{i+1 \mod n}, \text{'}Obj\text{'}))$
  $\hat{a}_i \leftarrow (a_i, o_{i+1 \mod n})$
return $\{\hat{c}_1, \ldots, \hat{c}_n\}, \{\hat{a}_1, \ldots, \hat{a}_n\}$

Lemma 2 says that Algorithm 2.1 produces a (n, ℓ, γ)-sharing set family of size m as long as certain technical conditions apply (e.g., Algorithm 2.1 can be run with any numbers $n_1, ..., n_\ell$, but Lemma 2 only applies if the numbers are pairwise co-prime.).

Lemma 2. *If the numbers $n_1 < n_2 < \ldots < n_\ell$ are pairwise co-prime and $m \leq \prod_{i=1}^{\gamma+1} n_i$ then Algorithm 2.1 returns a $(\sum_{i=1}^{\ell} n_i, \ell, \gamma)$-sharing set of public cues.*

Proof. Suppose for contradiction that $|S_i \bigcap S_k| \geq \gamma + 1$ for $i < k < m$, then by construction we can find $\gamma + 1$ distinct indices $j_1, ..., j_{\gamma+1} \in$ such that $i \equiv k \mod n_{j_t}$ for $1 \leq t \leq \gamma + 1$. The Chinese Remainder Theorem states that there is a unique number x^* s.t. (1) $1 \leq x^* < \prod_{t=1}^{\gamma+1} n_{j_t}$, and (2) $x^* \equiv k \mod n_{j_t}$ for $1 \leq t \leq \gamma + 1$. However, we have $i < m \leq \prod_{t=1}^{\gamma+1} n_{j_t}$. Hence, $i = x^*$ and by similar reasoning $k = x^*$. Contradiction!

□

Example: Suppose that we select pairwise co-prime numbers $n_1 = 9, n_2 = 10, n_3 = 11, n_4 = 13$, then **CRT** $(m, n_1, \ldots, n_4)$ generates a $(43, 4, 1)$-sharing set family of size $m = n_1 \times n_2 = 90$ (i.e. the public cues for two accounts will overlap in at most one common cue), and for $m \leq n_1 \times n_2 \times n_3 = 990$ we get a $(43, 4, 2)$-sharing set family.

Lemma 2 implies that we can construct a (n, ℓ, γ)-sharing set system of size $m \geq \Omega\left((n/\ell)^{\gamma+1}\right)$ by selecting each $n_i \approx n/\ell$. Theorem 3 proves that we can't hope to do much better — any (n, ℓ, γ)-sharing set system has size $m \leq O\left((n/\ell)^{\gamma+1}\right)$. We refer the interested reader to Appendix 7.1 for the proof of Theorem 3 and for discussion about additional (n, ℓ, γ)-sharing constructions.

Theorem 3. *Suppose that $\mathcal{S} = \{S_1, ..., S_m\}$ is a (n, ℓ, γ)-sharing set family of size m then $m \leq \binom{n}{\gamma+1} \Big/ \binom{\ell}{\gamma+1}$.*

2.6.2 Shared Cues

Our password management scheme — *Shared Cues*— uses a (n, ℓ, γ)-sharing set family of size m (e.g., a set family generated by Algorithm 2.1) as a hardcoded input to output the public cues $c_1, ...c_m \subseteq C$ and passwords $p_1, ..., p_m$ for each account. We use algorithm 2.2 to generate the underlying cues $\hat{c}_1, \ldots, \hat{c}_n \in C$ and their associated PAO stories. The computer is responsible for storing the public cues in persistent memory and the user is responsible for memorizing and rehearsing each cue-association pair $(\hat{c}_i, \hat{a}_i)$.

We use two additional tricks to improve usability: (1) Algorithm 2.2 splits each PAO story into two parts so that each cue $\hat{c}$ consists of *two* pictures and *two* corresponding people with a label (action/object) for each person (see Figure 2.2). A user who sees cue $\hat{c}_i$ will be rehearsing both the i'th and the $i+1$'th PAO story, but will only have to enter one action and one object. (2) To optimize usability we use GreedyMap (Algorithm 2.4) to produce a permutation $\pi : [m] \to [m]$ over the public cues — the goal is to minimize the total number of extra rehearsals by ensuring that each cue is used by a frequently visited account.

Algorithm 2.3 *SharedCues* $[S_1, \ldots, S_m,]$ $\mathcal{G}_m$

Input: $k \in \mathcal{K}$, b, $\lambda_1, ..., \lambda_m$, Rehearsal Schedule R.
$\{\hat{c}_1, \ldots, \hat{c}_n\}, \{\hat{a}_1, \ldots, \hat{a}_n\} \leftarrow$ **CreatePAOStories** $(\mathbf{n}, \mathbf{I_1}, ..., \mathbf{I_n}, \mathbf{P_1}, \ldots, \mathbf{P_n})$
for $i = 1 \to m$ **do**
    $c_i \leftarrow \left\{\hat{c}_j \,\middle|\, j \in S_i\right\}$, and $p_i \leftarrow \left\{\hat{a}_j \,\middle|\, j \in S_i\right\}$.
% `Permute cues`
$\pi \leftarrow$ *GreedyMap* $(m, \lambda_1, ..., \lambda_m, c_1, \ldots, c_m, R, \sigma)$
return $\left(p_{\pi(1)}, c_{\pi(1)}\right), \ldots, \left(p_{\pi(m)}, c_{\pi(m)}\right)$
User: Rehearses the cue-association pairs $(\hat{c}_i, \hat{a}_i)$ by following the rehearsal schedule R.
Computer: Stores the public cues $c_1, ..., c_m$ in persistent memory.

Once we have constructed our public cues $c_1, ..., c_m \subseteq C$ we need to create a mapping π between cues and accounts $A_1, ..., A_m$. Our goal is to minimize the total number of extra rehearsals that the user has to do to satisfy his rehearsal

requirements. Formally, we define the **Min-Rehearsal** problem as follows:
Instance: Public Cues $c_1, ..., c_m \subseteq C$, Visitation Schedule $\lambda_1, ..., \lambda_m$, a rehearsal schedule R for the underlying cues $\hat{c} \in C$ and a time frame t.
Output: A bijective mapping $\pi : \{1, ..., m\} \to \{1, ..., m\}$ mapping account A_i to public cue $S_{\pi(i)}$ which minimizes $\mathbb{E}[XR_t]$.
Unfortunately, we can show that **Min-Rehearsal** is NP-Hard to even approximate within a constant factor. Our reduction from Set Cover can be found in Appendix 7.1 of this paper. Instead GreedyMap uses a greedy heuristic to generate a permutation π.

Theorem 4. *It is NP-Hard to approximate* **Min-Rehearsal** *within a constant factor.*

Algorithm 2.4 GreedyMap

Input: $m, \lambda_1, ..., \lambda_m, c_1, \ldots, c_m$, Rehearsal Schedule R (e.g., CR or ER with parameter σ).
Relabel: Sort λ's s.t $\lambda_i \geq \lambda_{i+1}$ for all $i \leq m-1$.
Initialize: $\pi_0(j) \leftarrow \perp$ for $j \leq m$, *UsedCues* $\leftarrow \emptyset$.
$\%\pi_i$ `denotes a partial mapping` $[i] \to [m]$`,for` $j > i$`, the mapping is undefined (e.g.,` $\pi_i(j) = \perp$`). Let` $S_k = \{\hat{c} \mid \hat{c} \in c_k\}$`.`
for $i = 1 \to m$ **do**
  for all $j \in [m] -$ *UsedCues* **do**

$$\Delta_j \leftarrow \sum_{\hat{c} \in S_j} \mathbb{E}\left[XR_{t,\hat{c}} \middle| \lambda_{\hat{c}} = \lambda_i + \sum_{j:\hat{c} \in S_{\pi_{i-1}(j)}} \lambda_j \right] - \mathbb{E}\left[XR_{t,\hat{c}} \middle| \lambda_{\hat{c}} = \sum_{j:\hat{c} \in S_{\pi_{i-1}(j)}} \lambda_j \right] \% \ \Delta_j \text{:}$$

`expected reduction in total extra rehearsals if we set` $\pi_i(i) = j$
  $\pi_i(i) \leftarrow \arg\max_j \Delta_j$, *UsedCues* $\leftarrow$ *UsedCues* $\cup \{\pi_i(i)\}$
return π^m

2.6.3 Usability and Security Analysis

We consider three instantiations of *Shared Cues*: SC-0, SC-1 and SC-2. SC-0 uses a $(9, 4, 3)$-sharing family of public cues of size $m = 126$ — constructed by taking all $\binom{9}{4} = 126$ subsets of size 4. SC-1 uses a $(43, 4, 1)$-sharing family of public cues of size $m = 90$ — constructed using Algorithm 2.1 with $m = 90$ and $(n_1, n_2, n_3, n_4) = (9, 10, 11, 13)$. SC-2 uses a $(60, 5, 1)$-sharing family of public cues of size $m = 90$ — constructed using Algorithm 2.1 with $m = 90$ and $(n_1, n_2, n_3, n_4, n_5) = (9, 10, 11, 13, 17)$.

Assumption	CR ($\sigma = 1$)			ER ($\sigma = 1$)		
Schedule/Scheme	SC-0	SC-1	SC-2	SC-0	SC-1	SC-2
Very Active	≈ 0	1,309	2,436	≈ 0	3.93	7.54
Typical	≈ 0.42	3,225	5,491	≈ 0	10.89	19.89
Occasional	≈ 1.28	9,488	6,734	≈ 0	22.07	34.23
Infrequent	≈ 723	13,214	18,764	≈ 2.44	119.77	173.92

Table 2.3: $\mathbb{E}[XR_{365}]$: Extra Rehearsals over the first year for SC-0,SC-1 and SC-2.

Offline Attack?	$h = 0$			$h > 0$		
(n, ℓ, γ)-sharing	$r = 0$	$r = 1$	$r = 2$	$r = 0$	$r = 1$	$r = 2$
$(n, 4, 3)$ (e.g., SC-0)	2×10^{-15}	0.011	1	3.5×10^{-7}	1	1
$(n, 4, 1)$ (e.g., SC-1)	2×10^{-15}	4×10^{-11}	8×10^{-7}	3.5×10^{-7}	0.007	1
$(n, 5, 1)$ (e.g., SC-2)	1×10^{-19}	2×10^{-15}	4×10^{-11}	1.8×10^{-11}	3.5×10^{-7}	0.007

Table 2.4: *Shared Cues* $(q_{\$10^6}, \delta, m, s, r, h)$-Security: δ vs h and r using a (n, ℓ, γ)-sharing family of m public cues.

Our usability results can be found in Table 2.3 and our security results can be found in Table 2.4. We present our usability results for the very active, typical, occasional and infrequent internet users (see Table 2.1 for the visitation schedules) under both sufficient rehearsal assumptions CR and ER. Table 2.3 shows the values of $\mathbb{E}[XR_{365}]$ — computed using the formula from Theorem 1 — for SC-0, SC-1 and SC-2. We used association strength parameter $\sigma = 1$ to evaluate each password management scheme — though we expect that σ will be higher for schemes like *Shared Cues* that use strong mnemonic techniques [4].

Our security guarantees for SC-0,SC-1 and SC-2 are illustrated in Table 2.4. The values were computed using Theorem 2. We assume that $|\mathcal{AS}| = 140^2$ where $\mathcal{AS} = \mathcal{ACT} \times \mathcal{OBJ}$ (e.g., their are 140 distinct actions and objects), and that the adversary is willing to spend at most $\$10^6$ on cracking the user's passwords (e.g., $q = q_{\$10^6} = 5.155 \times 10^{10}$). The values of δ in the $h = 0$ columns were computed assuming that $m \leq 100$.

Discussion: Comparing Tables 2.3 and 2.2 we see that *Lifehacker* is the most usable password management scheme, but SC-0 compares very favorably! Unlike *Lifehacker*, SC-0 provides provable security guarantees after the adversary phishes one account — though the guarantees break down if the adversary can also ex-

[4] We explore the effect of σ on $\mathbb{E}[XR_{t,c}]$ in Appendix 7.6.

ecute an offline attack. While SC-1 and SC-2 are not as secure as *Strong Random and Independent* — the security guarantees from *Strong Random and Independent* do not break down even if the adversary can recover *many* of the user's plaintext passwords — SC-1 and SC-2 are far more usable than *Strong Random and Independent*. Furthermore, SC-1 and SC-2 do provide very strong security guarantees (e.g., SC-2 passwords remain secure against offline attacks even after an adversary obtains two plaintext passwords for accounts of his choosing). For the very active, typical and occasional user the number of extra rehearsals required by SC-1 and SC-2 are quite reasonable (e.g., the typical user would need to perform less than one extra rehearsal per month). The usability benefits of SC-1 and SC-2 are less pronounced for the infrequent user — though the advantage over *Strong Random and Independent* is still significant.

2.7 Discussion and Future Work

We conclude by discussing future directions of research.

Sufficient Rehearsal Assumptions. While there is strong empirical evidence for the Expanding Rehearsal assumption in the memory literature (e.g., [160]), the parameters we use are drawn from prior studies in other domains. In Chapter 4 we present preliminary results from user studies we are conducting to test the Expanding Rehearsal assumption in the password context, and obtain parameter estimates specific to the password setting.

Expanding Security over Time. Most extra rehearsals occur soon after the user memorizes a cue-association pair — when the rehearsal intervals are still small. Is it possible to start with a password management scheme with weaker security guarantees (e.g., SC-0), and increase security over time by having the user memorize additional cue-association pairs as time passes?

Secure Password Recovery Mechanism. Recently, we proposed a password recovery mechanism which would allow users who forget a few of their stories to recover them provided that they can still remember a couple of their other stories [164]. For example, suppose that we have the hash of the user's first six PAO

stories $\mathbf{H}(a_1o_1 \ldots a_6o_6)$. The entropy of the string $a_1o_1 \ldots a_6o_6$ is high enough that no adversary who manages to obtain this hash will be able to crack the password. However, if the user can remember five of these stories then it is trivial for our password recovery mechanism to find the sixth story by brute force search. By storing enough cryptographic hashes we can ensure that our user can recover *any* story that he forgets provided that he can remember *any* five of his other stories. These hashes could be deleted after the each of the stories are firmly entrenched in the user's memory (e.g., after the user has rehearsed each of his stories many times).

Login Time. One potential usability drawback of *Shared Cues* is that it might take a few seconds to type in each of his passwords because they are long (e.g., one password consists of four actions and four objects). One way to save time during authentication would be to instruct the user to form his password from the first three characters in each action and each object instead of typing in each word completely (most actions and objects in our set could be uniquely identified from the first two or three characters in the word). Alternatively, we could use auto-completion to help the user type in his passwords faster. Because we assume that the adversary already knows the set of actions and objects that we are using we would not reduce the adversary's search space by using only the first three characters of each word or by using auto-completion.

Human Computable Passwords. *Shared Cues* only relies on the human capacity to memorize and retrieve information, and is secure against at most $r = \ell/\gamma$ plaintext password leak attacks. Could we improve security (or usability) by having the user perform simple computations to recover his passwords? In Chapter 3 we present a candidate Human Computable Password scheme and provide strong evidence that this scheme will remain secure even after *many* (e.g., 50–100) plaintext password breaches.

Chapter 3

Human Computable Passwords

3.1 Introduction

Secure cryptographic protocols to authenticate humans typically assume that the human will receive assistance from trusted hardware or software. One interesting challenge for the cryptography community is to build authentication protocols that are so simple that a human can execute them without relying on assistance from a trusted computer. In this chapter we propose several candidate human authentication protocols in a setting in which the user can only receive assistance from a semi-trusted computer — a computer that can be trusted to store information and perform computations correctly, but cannot be trusted to ensure privacy. In our schemes, a semi-trusted computer is used to store and display public challenges $C_i \in [n]^k$. The user memorizes a random secret mapping $\sigma : [n] \to \mathbb{Z}_d$ and authenticates by computing responses $f(\sigma(C_i))$ to a sequence of public challenges, where $f : \mathbb{Z}_d^k \to \mathbb{Z}_d$ is a function that is easy for the human to evaluate. We prove that any statistical adversary needs to sample $m = \tilde{\Omega}\left(n^{s(f)}\right)$ challenge-response pairs to recover σ — for a security parameter $s(f)$ that depends on two key properties of f[1]. Our lower bound generalizes recent results of Feldman et al. [73], who proved analogous results for the special case $d = 2$. To obtain our results we apply the general hypercontractivity theorem [116] to lower bound the *statistical*

[1]Our guarantees are not information theoretic. Indeed, a computationally unbounded adversary would need to see at most $O(n)$ challenge-response pairs to break any such human computable password scheme. Our lower bounds provide strong evidence that any polynomial time adversary will need at least $m = \tilde{\Omega}\left(n^{s(f)}\right)$ challenge-response pairs — even if $s(f) > 1$.

dimension of the distribution over challenge-response pairs induced by f and σ. Our *statistical dimension* lower bounds apply to arbitrary functions $f : \mathbb{Z}_d^k \to \mathbb{Z}_d$ — not just functions that are easy for a human to evaluate — and may be of independent interest. For our particular schemes, we show that forging passwords is equivalent to recovering the secret mapping. We also show that $s(f_1) = 1.5$ for our first scheme and that $s(f_2) = 2$ in our second scheme. Thus, our human computable password schemes can maintain strong security guarantees even after an adversary has observed the user login to many different accounts (e.g., 100). We also issue a public challenge to the cryptography community to crack passwords that were generated using our human computable password schemes.

In Chapter 2 we initiated the rigorous study of usable and secure password management schemes — systematic strategies to help users create and remember multiple passwords. *Shared Cues*, the proposed password management scheme from Chapter 2, balances security and usability considerations. However, *Shared Cues* only maintains security for a small (constant) number of plaintext password breaches (e.g., 1 to 4). An adversary who has seen several of the user's passwords might be able to break the user's passwords at other accounts. This raises an important question: Is it possible to design a human authentication protocol that allows a user to authenticate to multiple untrusted parties and will remain secure even after many breaches (e.g., 50 to 100)?

In this chapter the goal is to develop a secure human computable password management scheme in which security guarantees are maintained after *many* breaches. In a human computable password management scheme the user reconstructs each of his passwords by *computing* the response to a public challenge. The computation may only involve a few very simple operations (e.g., addition modulo 10) over secret values (digits) that the user has memorized. More specifically, in our candidate human computable password schemes the user learns to compute a simple function $f : \mathbb{Z}_d^k \to \mathbb{Z}_d$ (in our candidate schemes we adopt the base $d = 10$ that is natural for most humans), and memorizes a secret mapping $\sigma : [n] \to \mathbb{Z}_d$. The user authenticates by responding to a sequence of single digit challenges — a challenge-response pair $(C, f(\sigma(C)))$ is a challenge $C \in X_k \subseteq [n]^k$ and the corresponding response.

Our first candidate human computable password scheme uses the function

$$f_1(x_0, x_1, x_2, x_3, x_4, x_5, \ldots, x_{13}) = x_{13} + x_{12} + x_{(x_{10}+x_{11} \mod 10)} \mod 10 .$$

To evaluate this function a human would only need to perform three addition operations modulo 10. While this function is quite simple we show that the attacker

would need to see $\tilde{\Omega}\left(n^{1.5}\right)$ challenge-response pairs before he can forge the user's passwords (accurately predict the responses to randomly selected challenges). In particular, if we ask the user to memorize a secret mapping of length $n = 100$ and the password for each account is ten digits then the adversary would need to breach about one-hundred of the user's accounts before he could obtain enough challenge-response pairs ($100^{1.5} = 10(100)$) to forge the user's passwords.

As in Chapter 2 we consider a setting where a user has two types of memory: *persistent memory* (e.g., a sticky note or a text file on his computer) and *associative memory* (e.g., his own human memory). We assume that persistent memory is reliable and convenient but not private (i.e., accessible to an adversary). In contrast, a user's associative memory is private but lossy—if the user does not rehearse a memory it may be forgotten. Thus, the user can store a password challenge $C \in X_k$ in persistent memory, but the mapping σ must be stored in associative memory (e.g., memorized and rehearsed). We allow the user to receive assistance from a semi-trusted computer. A semi-trusted computer will perform computations accurately (e.g., it can be trusted to show the user the correct challenge), but it will not ensure privacy of its inputs or outputs. This means that a human computable password management scheme should be based on a function f that the user can compute entirely in his head.

Contributions. We develop a general framework for analyzing the security of a human computable password management scheme and we propose two candidate human computable password management schemes. We give evidence that our schemes remain secure until the adversary has seen at least $\tilde{\Omega}\left(n^{s(f)}\right)$ challenge-response pairs $(C, f(\sigma(C)))$. Here, $s(f) = \min\{r(f)/2, g(f) + 1\}$ is a composite security parameter which captures $g(f)$ (how many inputs to f need to be fixed to make f linear?) and $r(f)$ (what is the largest value of r such that the distribution over challenge-response pairs are $(r-1)$-wise independent?). We show that $s(f) = 1.5$ for our first scheme and $s(f) = 2$ for our second scheme. In particular we prove that any statistical adversary needs to see at least $\tilde{\Omega}\left(n^{r(f)/2}\right)$ challenge-response pairs $(C, f(\sigma(C)))$ before he can even approximately recover the secret mapping σ. Our lower bound is based on the *statistical dimension* of the distribution over challenge-response pairs induced by f and σ. We stress that our analysis of the statistical dimension applies to arbitrary functions $f : \mathbb{Z}_d^k \to \mathbb{Z}_d$, not just functions that are easy for humans to compute. Our analysis of the statistical dimension generalizes recent results of Feldman et al. [73], which only applied to binary predicates (e.g., $d = 2$), and may be of independent interest. Because our function f is not a

binary predicate we cannot use the Walsh basis functions to express the Fourier decomposition of f and analyze the statistical dimension of our distribution over challenge-response pairs as Feldman et al. [73] do. Instead, we use a generalized set of Fourier basis functions to take the Fourier basis decomposition of f, and we apply the general hypercontractivity theorem [116] to obtain our bounds on the statistical dimension. Furthermore, we show that forging passwords and approximately recovering the secret mapping are equivalent for any 'reasonable' candidate human computable password scheme. This means that any adversary who can predict the response $f(C)$ to a random challenge C with better accuracy than random guessing can be used as a blackbox to approximately recover the secret mapping. These results imply that any statistical adversary needs to see at least $\tilde{\Omega}\left(n^{r(f)/2}\right)$ challenge-response pairs before he can accurately forge passwords. This is significant because almost all known algorithmic techniques have statistical analogues. In particular techniques like Expectation Maximization[62], local search, MCMC optimization[83], first and second order methods for convex optimization, PCA, ICA, k-means can be modeled as statistical algorithms — see [37] and [55] for proofs. While Gaussian Elimination is a notable exception our composite security parameter accounts for attacks based on Gaussian Elimination — we show that an adversary needs to see $m = \tilde{\Omega}\left(n^{1+g(f)}\right)$ challenge-response pairs to recover σ using Gaussian Elimination. To analyze the usability of our candidate human computable password schemes we use the usability model from Chapter 2 to quantify the effort that a user must expend to memorize and rehearse the secret mapping σ, and we use step counting to estimate the effort that a user must expend to compute each password. We also propose a mnemonic tool to help users memorize their secret mapping σ. Finally, we constructed public challenges for cryptographers to break our human computable password management schemes under various parameters (e.g., $n = 100$, $m = 1000$).

Organization. The rest of the paper is organized as follows: We first explore related work in Section 3.2. We then introduce preliminary notation and definitions in Section 3.3. We present our main technical results in Section 3.4 including an overview of our lower bound for statistical adversaries. We use these results to provide general security bounds for a human computable password scheme in Section 3.5. We introduce our candidate human computable password schemes in Section 3.6 and analyze the security and usability of these schemes. We conclude in Section 3.7 by presenting our human computable password challenge and discussing how a human computable password scheme could be used to defend

against an adversary who can always observe the user when he logs into any of his accounts (e.g., every time the user computes the response $f(\sigma(C))$ to a single-digit challenge the adversary observes the pair $(C, f(\sigma(C)))$).

3.2 Related Work

The literature on passwords has grown rapidly over the past decade. One line of prior work has focused on the effects of password composition rules (e.g., requiring a password to contain capital letters and numbers) on individual passwords [34, 101]. Another line of prior work has focused on empirical studies of user behavior in password management [39, 52, 75, 102] (e.g., How many different passwords do people have? How often do users reuse the same password? How strong are the passwords that people pick?). These studies consistently paint a grim picture. Many of the passwords that users select have low entropy and users frequently reuse their passwords. Shay et al. [140] empirically studied the usability of system assigned passwords and found that users often had difficulty remembering system assigned passwords. Some researchers have considered replacing text passwords with graphical passwords [27, 48] driven by evidence that humans have a large capacity for visual memories [145] and that cued-recall is easier than pure recall [23]. Fundamentally, both graphical passwords and text passwords rely solely on the user's ability to remember something (e.g., a string, a face or a location on a picture). Many security metrics have been proposed to analyze the security of a dataset of passwords or to estimate the security of an individual password [39, 45, 107, 121]. While these metrics can provide useful feedback about individual passwords (e.g., they rule out some insecure passwords) they do not deal with the complexities of securing multiple accounts against an adversary (e.g., they don't consider correlations between a user's passwords).

In Chapter 2 we considered the problem of developing usable and secure password management schemes — strategies for creating and remembering *multiple* passwords. We use the same usability model in this chapter to quantify the effort that a user will need to expend to remember his secret mapping in our human computable password schemes. We emphasize two key differences between the work in the previous chapter and the work in the previous chapter. First, *Shared Cues*, the password management scheme from Chapter 2, only maintains security for a small (constant) number of plaintext password breaches, while our goal in this chapter is to design protocols that maintain security guarantees even after

many password breaches. There are scenarios in which it may not be reasonable to assume that the adversary can only compromise a small number of the user's passwords (e.g., if the user's computer is infected with malware for a few days). Second, the *Shared Cues* scheme only requires users to remember several cue-association pairs to reconstruct their passwords while the password management schemes we consider in this chapter require users to perform a few additional computations in their head to reconstruct their passwords.

Hopper and Blum [91] designed a Human Identification Protocol based on a the noisy parity problem — a learning problem that is believed to be hard. Juels and Weis [93] modified the protocol of Hopper and Blum to design HB+ — a lightweight authentication protocol for pervasive devices like smartcards. Subsequent work has explored the security of the HB+ protocol under various threat models (e.g., man-in-the-middle attacks[47, 84], concurrent composition[95]). We emphasize a few fundamental differences between our work and the work of Hopper and Blum. First, they focus on the authentication setting where a human authenticates to a single trusted party with a shared secret. By contrast, we focus on the setting where a human user wishes to authenticate to multiple (possibly untrusted) parties without sharing his secret (e.g., by only sharing the cryptographic hashes of each password he computes). Second, computations in their protocol are randomized (e.g., the human occasionally flips his answer), while the computations in our protocol are deterministic. This is significant because humans are not good at consciously generating random numbers [74, 111, 154] (e.g., noisy parity could be easy to learn when humans are providing source of noise). It also means that their protocol would need to be modified in our setting so that the untrusted third party could validate a noisy response using only a cryptographic hash of the answer — invoking error correcting codes would increase the number of rounds needed to provide an acceptable level of security. Finally, we focus on computations of *very simple* functions over a *constant* number of variables so that a human can compute the response to each challenge quickly.

Naor and Pinkas[112] proposed using visual cryptography[113] to address a related problem: how can a human verify that a message he received from a trusted server has not been tampered with by an adversary? Their protocol requires the human to carry a visual transparency (a shared secret between the human and the trusted server in the visual cryptography scheme), which he will use to verify that messages from the trusted server have not been altered.

A related goal in cryptography, constructing pseudorandom generators in NC^0, was proposed by Goldreich [86] and by Cryan and Miltersen [58]. In

Goldreich's construction we fix $C_1, \ldots, C_m \in [n]^k$ once and for all, and a binary predicate $P : \{0,1\}^k \to \{0,1\}$. The pseudorandom generator is a function $G : \{0,1\}^n \to \{0,1\}^m$, whose i'th bit $G(x)[i]$ is given by P applied to the bits of x specified by C_i. O'Donnel and Witmer gave evidence that the "Tri-Sum-And" predicate ($TSA(x_1, \ldots, x_5) = x_1 + x_2 + x_3 + x_4 x_5 \mod 2$) provides near-optimal stretch. In particular, they showed that for $m = n^{1.5-\epsilon}$ Goldreich's construction with the TSA predicate is secure against subexponential-time attacks using SDP hierarchies. Our candidate human-computable password schemes use functions $f : \mathbb{Z}_{10}^k \to \mathbb{Z}_{10}$ instead of binary predicates. While our candidate functions are contained in NC^0, we note that an arbitrary function in NC^0 is not necessarily human computable.

Feldman et al. [73] considered the problem of finding a planted solution in a random binary satisfiability problem. They showed that any statistical algorithm — a class of algorithms that covers almost all known algorithmic techniques — needs to see at least $\tilde{\Omega}\left(n^{r/2}\right)$ random clauses to efficiently identify the planted solution when the distribution over clauses are $(r-1)$-wise independent[2]. Feldman et al. [73] also demonstrate that $\tilde{O}\left(n^{r/2}\right)$ clauses are sufficient. We extend the analysis of Feldman et al. [73] to cover non-binary planted satisfiability problems, and argue that our candidate human computable password schemes are secure.

3.3 Definitions

3.3.1 Notation

Given two strings $\alpha_1, \alpha_2 \in \mathbb{Z}_d^n$ we use $H(\alpha_1, \alpha_2) \doteq |\{i \in [n] \mid \alpha_1[i] \neq \alpha_2[i]\}|$ to denote the Hamming distance between them. We will also use $H(\alpha_1) \doteq H\left(\alpha_1, \vec{0}\right)$ to denote the Hamming weight of α_1. We use $\sigma : [n] \to \mathbb{Z}_d$ to denote a secret random mapping that the user will memorize. We will sometimes abuse notation and think of $\sigma \in \mathbb{Z}_d^n$ as a string which encodes the mapping, and we will use $\sigma \sim \mathbb{Z}_d^n$ to denote a random mapping chosen from $\mathbb{Z}_d^n$ uniformly at random.

[2]We note that after we have seen $O(n \log n)$ random clauses the planted solution is — with high probability — the only solution which satisfies all of the random clauses. The results of Feldman et al. [73] are evidence that we need $\tilde{\Omega}\left(n^{r/2}\right)$ examples to find the planted solution efficiently. We also note that $r = 3$ for the uniform distribution over clauses that satisfy the TSA predicate, which provides further evidence that the Goldreich's PRG is secure for $m = n^{1.5-\epsilon}$.

Definition 6. *We say that two mappings $\sigma_1, \sigma_2 \in \mathbb{Z}_d^n$ are ϵ-correlated if $\frac{H(\sigma_1,\sigma_2)}{n} \leq \frac{d-1}{d} - \epsilon$, and we say that a mapping $\sigma \in \mathbb{Z}_d^n$ is δ-balanced if*

$$\max_{i \in \{0,\ldots,d-1\}} \left| \frac{H\left(\sigma, \vec{i}\right)}{n} - \frac{d-1}{d} \right| \leq \delta \, .$$

Note that for a random mapping σ_2 we expect σ_1 and σ_2 to differ at $\mathbb{E}_{\sigma_2 \sim \mathbb{Z}_d^n}\left[H\left(\sigma_1, \sigma_2\right)\right] = n\left(\frac{d-1}{d}\right)$ locations, and for a random mapping σ and $i \sim \{0, \ldots, d-1\}$ we expect σ to differ from $\vec{i}$ at $\mathbb{E}_{i \sim \mathbb{Z}_d, \sigma \sim \mathbb{Z}_d^n}\left[H\left(\sigma, \vec{i}\right)\right] = n\left(\frac{d-1}{d}\right)$ locations. Thus, for any constant $\epsilon > 0$ a random mapping σ_2 will not be ϵ-correlated with σ_1 with probability $1 - o(1)$, but for any constant $\delta > 0$ a random mapping σ will be δ-balanced with probability $1 - o(1)$.

We let $X_k \subseteq [n]^k$ denote the space of ordered clauses of k variables without repetition. We use $C \sim X_k$ to denote a clause C chosen uniformly at random from X_k and we use $\sigma(C) \in \mathbb{Z}_d^k$ to denote the values of the corresponding variables in C. For example, if $d = 10$, $C = (3, 10, 59)$ and $\sigma(i) = (i + 1 \mod 10)$ then $\sigma(C) = (4, 1, 0)$.

We view each clause $C \in X_k$ as a *single-digit challenge*. The user responds to a challenge C by computing $f(\sigma(C))$, where $f : \mathbb{Z}_d^k \to \mathbb{Z}_d$ is a *human computable* function (see discussion below) and $\sigma : [n] \to \mathbb{Z}_d$ is the secret mapping that the user has memorized. For example, if $d = 10$, $C = (3, 10, 59)$, $\sigma(i) = (i + 1 \mod 10)$ and $f(x, y, z) = (x - y + z \mod 10)$ then $f(\sigma(C)) = (4 - 1 + 0 \mod 10) = 3$. A length-$t$ password challenge $\vec{C} = \langle C_1, \ldots, C_t \rangle \in (X_k)^t$ is a sequence of t single digit challenges, and $f(\sigma(\vec{c})) = \langle f(\sigma(C_1)), \ldots, f(\sigma(C_t)) \rangle \in \mathbb{Z}_d^t$ denotes the corresponding response (e.g., a password).

Let's suppose that the user has m accounts $A_1, \ldots, A_m$. In a human computable password management scheme we will generate m length-t password challenges $\vec{C}_1, \ldots, \vec{C}_m \in (X_k)^t$. These challenges will be stored in persistent memory so they are always accessible to the user as well as the adversary. When our user needs to authenticate to account A_i he will be shown the length-t password challenge $\vec{C}_i = \left\langle C_1^i, \ldots, C_t^i \right\rangle$. The user will respond by computing his password $p_i = \left\langle f\left(\sigma\left(C_1^i\right)\right), \ldots, f\left(\sigma\left(C_t^i\right)\right) \right\rangle \in \mathbb{Z}_d^t$.

3.3.2 Requirements for a Human Computable Function

In our setting we require that the composite function $f \circ \sigma : X_k \to \mathbb{Z}_d$ is human computable. A human computable function might involve several memory lookups (e.g., we can ask the user to recall the value $\sigma(i)$) as well as several simple operations. However, if we want the function $f \circ \sigma$ to be human computable then we cannot ask the user to perform too many operations.

Requirement 1. *A function f is $\hat{t}$-*human computable *for a human user H if H can evaluate f in his head in $\hat{t}$ seconds.*

Example: The function $f(x, y) = x + y \mod 10$ is 1-human computable for many humans.

Discussion Informally we say that a function f is *human-computable* if a human user can evaluate f *quickly* in his head. Intuitively, evaluation of a human computable function must only involve a few operations — otherwise a human will not be able to evaluate the function quickly. Furthermore, the operations must be extremely simple. A human computable function must only involve operations with a very low memory footprint as a typical person can only keep 7 ± 2 'chunks' of information in short-term memory [108] at any given time. If the memory footprint of a function is high then the user will need to store intermediate values in long-term memory and recall them mid-computation. We take the view that no human computable function should require users to store intermediate values in long-term memory because the memorization process would necessarily slow down computation[3]. Therefore, we can rule out operations involving large numbers. For example, expressions like $98423 + 498874 \mod 2345$ or $5432^{2340489} \mod 8156243869$ would be very difficult — if not impossible — for most humans to evaluate in their heads. Most humans would be capable of evaluating a long expressions like $7+1+6+0+8+3+4+7+2+7+8+9+5+3 \mod 10$ in their head — after receiving a few basic preliminary instructions (e.g., only worry about remembering the least significant digit). However, even this expression would take a while to evaluate because it involves many terms. Thus a human computable function involves 1) simple operations with a very small memory footprint 2) few terms, and 3) few operations.

[3]However, we do consider functions that require users to retrieve values from long-term memory. For example, the user might need to remember the value $\sigma(i)$ or the user might need to remember basic arithmetic facts that he memorized in grade school (e.g., 9+5=14).

3.3.3 Password Unforgeability

In the password forgeability game the adversary attempts to guess the user's password for a randomly selected account after he has seen the user's passwords at m other randomly selected accounts. We say that a scheme is UF-RCA (Unforgeability against Random Challenge Attacks) secure if any probabilistic polynomial time adversary fails to guess the user's password with high probability. In the password forgeability game we select the secret mapping $\sigma : [n] \rightarrow \mathbb{Z}_d$ uniformly at random along with challenges $C_1, \ldots, C_{mt+t} \sim X_k$. The adversary is given the function $f : \mathbb{Z}_d^k \rightarrow \mathbb{Z}_d$ and is shown the challenges $C_1, \ldots, C_{t(m+1)}$ as well as the values $f(\sigma(C_i))$ for $i \in \{t+1, \ldots, mt+t\}$. The game ends when the adversary $\mathcal{A}$ outputs a guess $\langle q_1, \ldots, q_t \rangle \in \mathbb{Z}_d^t$ for the value of $\langle f(\sigma(C_1)), \ldots, f(\sigma(C_t)) \rangle$. We say that the adversary wins if he correctly guesses the responses to all of the challenges $C_1, \ldots, C_t$. Formally, we use

$$\textbf{Wins}(\mathcal{A}, n, m, t) = \forall i \in \{1, \ldots, t\}. q_i = f(\sigma(C_i))$$

to denote the event that the adversary wins the game. We are interested in understanding how many example single digit challenge-response pairs the adversary needs to see before he can start breaking the user's passwords.

Definition 7. *(Security) We say that a function $f : \mathbb{Z}_d^k \rightarrow \mathbb{Z}_d$ is $\mathbf{UF-RCA}(n, m, t, \delta)$–secure if for every probabilistic polynomial time (in n, m) adversary $\mathcal{A}$*

$$\Pr[\textbf{Wins}(\mathcal{A}, n, m, t)] \leq \delta ,$$

where the randomness is taken over the selection of the secret mapping $\sigma \sim \mathbb{Z}_d^n$, the challenges $C_1, \ldots, C_{mt+t}$ as well as the adversary's coins.

Discussion Our security model in this chapter is different from the security model from Chapter 2 in which the adversary gets to adaptively select which accounts to compromise and which account to attack. While our security model may seem weaker at first glance because the adversary does not get to select which account to compromise/attack, we observe that the password management schemes of Chapter 2 are only secure against one to three adaptive breaches. By contrast, our goal is to design human computable password schemes that satisfy **UF−RCA** security for large values of m (e.g. 100), which means that it is reasonable to believe that the user has at most m password protected accounts. If the user has at most m accounts then even an adaptive adversary — who gets to compromise all but one account — will not be able to forge the password at any remaining account with probability greater than $m\delta$ (typically, $m \ll 1/\delta$).

3.3.4 Security Parameters of f

Given a function $f : \mathbb{Z}_d^k \to \mathbb{Z}_d$ we define the function $Q^f : \mathbb{Z}_d^{k+1} \to \{\pm 1\}$ s.t. $Q^f(x, i) = 1$ if $f(x) = i$; otherwise $Q^f(x, i) = -1$. We use Q_σ^f to define a distribution over $X_k \times \mathbb{Z}_d$ (challenge-response pairs) as follows

$$\Pr_{Q_\sigma^f}[C, i] \doteq \frac{Q^f(\sigma(C), i) + 1}{2|X_k|} .$$

Intuitively, Q_σ^f is the uniform distribution over challenge response pairs (C, j) s.t. $f(\sigma(C)) = j$. We also use $Q^{f,j} : \mathbb{Z}_d^k \to \{\pm 1\}$ ($Q^{f,j}(x) = Q^f(x, j)$) to define a distribution over X_k. We write the Fourier decomposition of a function $Q : \mathbb{Z}_d^k \to \{\pm 1\}$ as follows

$$Q(x) = \sum_{\alpha \in \mathbb{Z}_d^k} \hat{Q}_\alpha \cdot \chi_\alpha(x) ,$$

where our basis functions are

$$\chi_\alpha(x) \doteq \exp\left(\frac{-2\pi\sqrt{-1}(x \cdot \alpha)}{d}\right) .$$

We say that a function Q has degree ℓ if $\ell = \max\left\{H(\alpha) \,\middle|\, \alpha \in \mathbb{Z}_d^k\right\}$ — equivalently if $Q(x) = \sum_i Q_i(x)$ can be expressed as a sum of functions where each function $Q_i : \mathbb{Z}_d^k \to \mathbb{R}$ depends on at most ℓ variables.

Definition 8. *We use* $r(Q) \doteq \min\left\{H(\alpha) \,\middle|\, \exists \alpha \in \mathbb{Z}_d^k . \hat{Q}_\alpha \neq 0 \wedge \alpha \neq \vec{0}\right\}$ *to denote the distributional complexity of* Q*, and we use* $r(f) = \min\left\{r\left(Q^{f,j}\right) \,\middle|\, j \in \mathbb{Z}_d\right\}$ *to denote the distributional complexity of* f*. We use*

$$g(f) \doteq \min\left\{\ell \in \mathbb{N} \cup \{0\} \,\middle|\, \exists \alpha \in \mathbb{Z}_d^\ell, S \subseteq [k] . s.t\ |S| = \ell \text{ and } f_{|S,\alpha} \text{ is a linear function}\right\} ,$$

to denote the minimum number of variables that must be fixed to make f *a linear function. Here,* $f_{|S,\alpha} : \mathbb{Z}_d^{k-\ell} \to \mathbb{Z}_d$ *denotes the function* f *after fixing the variables at the indices specified by* S *to* α*. Finally, we use* $s(f) \doteq \min\{r(f)/2, g(f) + 1\}$ *as our composite security measure.*

We argue that a human computable password scheme — given by a function f — is secure against $m = \tilde{\Omega}\left(n^{s(f)}\right)$ breaches. In particular, we argue that any

statistical algorithm needs to see at least $m = \tilde{\Omega}\left(n^{r(f)/2}\right)$ challenge response pairs to (approximately) recover the secret mapping σ. We also demonstrate that any adversary that can break the security of our password scheme after seeing m challenge response pairs can be used to approximately recover the secret mapping using only $\tilde{O}(m)$ challenge response pairs.

3.4 Statistical Adversaries and Lower Bounds

Our main technical result (Theorem 6) is a lower bound on the number of single digit challenge-response pairs that a statistical algorithm needs to see to (approximately) recover the secret mapping σ. Our results are quite general and may be of independent interest. Given *any* function $f : \mathbb{Z}_d^k \to \mathbb{Z}_d$ we prove that *any* statistical algorithm needs $\tilde{\Omega}\left(n^{r(f)/2}\right)$ examples before it can find a secret mapping $\sigma' \in \mathbb{Z}_d^n$ such that σ' is ϵ-correlated with σ. We first introduce statistical algorithms in section 3.4.1 before stating our main lower bound for statistical algorithms in section 3.4.2. We also provide a high level overview of our proof in section 3.4.2.

3.4.1 Statistical Algorithms

Let $\mathcal{D}$ denote a set of distributions over a domain X, let $\mathcal{F}$ denote a set of solutions and $\mathcal{Z} : \mathcal{D} \to 2^{\mathcal{F}}$. The distributional search problem [72] $\mathcal{Z}$ over $\mathcal{D}$ and $\mathcal{F}$ is the following problem: Given access to m random samples from an unknown distribution $D \in \mathcal{D}$ find a solution $s \in \mathcal{Z}(D) \subseteq \mathcal{F}$. For a solution $s \in \mathcal{F}$ we will use $\mathcal{Z}^{-1}(s) \subseteq \mathcal{D}$ to denote the set of distributions for which s is a valid solution (e.g., $D' \in \mathcal{D}$ s.t. $s \in \mathcal{Z}(D')$). We can think of our planted constrained satisfiability problem as a distributional search problem. For example, in our context $X = X_k \times \mathbb{Z}_d \subseteq [n]^k \times \mathbb{Z}_d$ denotes the set of all possible single-digit challenge response pairs, and our solution space $\mathcal{F} = \mathbb{Z}_d^n$ is the set of possible mappings. Each $\sigma \in \mathcal{F}$ defines a unique distribution $D_\sigma = Q_\sigma^f$ over challenge response pairs and $\mathcal{Z}(D_\sigma)$ would denote the set of all assignments $\tau \in \mathbb{Z}_d^n$ that are ϵ-correlated with σ. Now the distributional search problem is to find an assignment $\tau \in \mathbb{Z}_d^n$ that is ϵ-correlated with our planted solution σ (the secret mapping) given m challenge-response pairs.

A statistical algorithm can access the input distribution by querying the 1-MSTAT oracle or by querying the VSTAT oracle.

Definition 9. *[73] [1-MSTAT(L) oracle and VSTAT oracle] Let D be the input distribution over the domain X. Given any function $h : X \to \{0, 1, \ldots, L-1\}$, 1-MSTAT(L) takes a random sample x from D and returns h(x). For an integer parameter $T > 0$ and any query function $h : X \to \{0, 1\}$, VSTAT (T) returns a value $v \in [p - \tau, p + \tau]$ where $p = \mathbb{E}_{x \sim D}[h(x)]$ and $\tau = \max\left\{\frac{1}{T}, \sqrt{\frac{p(1-p)}{T}}\right\}$.*

The *discrimination norm* [73] of a set of distributions $\mathcal{D}'$ relative to a distribution D is denoted by $\kappa_2(\mathcal{D}', D)$ and defined as follows:

$$\kappa_2(\mathcal{D}', D) \doteq \max_{h, \|h\|_D = 1} \{\mathbb{E}_{D' \sim \mathcal{D}'}[|\mathbb{E}_{D'}[h] - \mathbb{E}_D[h]|]\} \ ,$$

where $\|h\|_D = \sqrt{\mathbb{E}_{x \sim D}[h^2(x)]}$. In our setting $\mathcal{D}' \subseteq \left\{Q^f_\sigma \,\middle|\, \sigma \in \mathbb{Z}_d^n\right\}$ and our reference distribution D is the uniform distribution over $X_k \times \mathbb{Z}_d$ so we can write

$$\kappa_2(\mathcal{D}', D) = \max_{h, \|h\|_D = 1} \left\{\mathbb{E}_{\sigma \sim \left\{\sigma' \in \mathbb{Z}_d^n \,\middle|\, Q^f_{\sigma'}, \in \mathcal{D}'\right\}}[|\Delta(\sigma, h)|]\right\} \ ,$$

where

$$\Delta(\sigma, h) \doteq \mathbb{E}_{(C,j) \sim Q^f_\sigma}[h(C, j)] - \mathbb{E}_{(C,j) \sim X_k \times \mathbb{Z}_d}[h(C, j)] \ .$$

Definition 10. *[73] For $\kappa > 0$, $\eta > 0$, domain X and a search problem $\mathcal{Z}$ over a set of solutions $\mathcal{F}$ and a class of distributions $\mathcal{D}$ over X, let d′ be the largest integer such that there exists a* reference *distribution D over X and a finite set of distributions $\mathcal{D}_D \subseteq \mathcal{D}$ with the following property: for any solution $s \in \mathcal{F}$ the set $\mathcal{D}_s = \mathcal{D}_D \setminus \mathcal{Z}^{-1}(s)$ has size at least $(1 - \eta) \cdot |\mathcal{D}_D|$ and for any subset $\mathcal{D}' \subseteq \mathcal{D}_s$, where $|\mathcal{D}'| \geq |\mathcal{D}_s|/d'$, $\kappa_2(\mathcal{D}', D) \leq \kappa$. The* ***statistical dimension*** *with discrimination norm κ and error parameter η of $\mathcal{Z}$ is d′ and denoted by* $\mathrm{SDN}(\mathcal{Z}, \kappa, \eta)$.

Feldman et al. [73] proved the following lower bound on the number of 1-MSTAT(L) queries needed to solve a distributional search problem. Intuitively, Theorem 5 implies that many queries are needed to solve a distributional search problem with high statistical dimension. In Section 3.4.2 we argue that the statistical dimension our distributional search problem (finding σ' that is ϵ-correlated with the secret mapping σ given m samples from the distribution Q^f_σ) is high.

Theorem 5. *[73, Theorems 10 and 12] Let X be a domain and $\mathcal{Z}$ be a search problem over a set of solutions $\mathcal{F}$ and a class of distributions $\mathcal{D}$ over X. For $\kappa > 0$ and $\eta \in (0, 1)$, let $d' = \mathrm{SDN}(\mathcal{Z}, \kappa, \eta)$. Let D be the reference distribution and $\mathcal{D}_D$ be a set of distributions*

for which the value d' is achieved. Any randomized statistical algorithm that, given access to a VSTAT $\left(\frac{1}{3\kappa^2}\right)$ (resp. 1-MSTAT (L)) for a distribution chosen randomly and uniformly from $\mathcal{D}_D$, succeeds with probability $\Lambda > \eta$ over the choice of distribution and internal randomness requires at least $\frac{\Lambda-\eta}{1-\eta}d'$ (resp. $\Omega\left(\frac{1}{L}\min\left\{\frac{d'(\Lambda-\eta)}{1-\eta}, \frac{(\Lambda-\eta)^2}{\kappa^2}\right\}\right)$) calls to the oracle.

As Feldman et al. [73] observe, almost all known algorithmic techniques can be modeled within the statistical query framework. In particular, techniques like Expectation Maximization[62], local search, MCMC optimization[83], first and second order methods for convex optimization, PCA, ICA, k-means can be modeled as a statistical algorithm even with $L = 2$ — see [37] and [55] for proofs. One issue is that a statistical simulation might need polynomially more samples. However, for $L > 2$ we can think of our queries to 1-MSTAT(L) as evaluating L disjoint functions on a random sample. Indeed, Feldman et al. [73] demonstrate that there is a statistical algorithm for binary planted satisfiability problems using $\tilde{O}\left(n^{r(f)/2}\right)$ calls to 1-MSTAT $\left(n^{\lceil r(f)/2 \rceil}\right)$.

Remark 1. *We can also use the statistical dimension to lower bound the number of queries that an algorithm would need to make to other types of statistical oracles to solve a distributional search problem. For example, we could also consider an oracle MVSTAT(L, T) that takes a query $h : X \to \{0, \ldots, L-1\}$ and a set $\mathcal{S}$ of subsets of $\{0, \ldots, L-1\}$ and returns a vector $v \in \mathbb{R}^L$ s.t for every $Z \in \mathcal{S}$*

$$\left|\sum_{i \in Z} v[i] - p_Z\right| \leq \max\left\{\frac{1}{T}, \sqrt{\frac{p_Z(1-p_Z)}{T}}\right\},$$

where $p_Z = \Pr_{x \sim D}[h(x) \in Z]$ and the cost of the query is $|\mathcal{S}|$. Feldman et al. [73, Theorem 7] proved lower bounds similar to Theorem 5 for the MVSTAT oracle. In this paper we focus on the 1-MSTAT and VSTAT oracles for simplicity of presentation.

3.4.2 Statistical Dimension Lower Bounds

We are now ready to state our main technical result.

Theorem 6. *Let $\sigma \in \mathbb{Z}_d^n$ denote a secret mapping chosen uniformly at random and let $\mathcal{Z}_{Q^f}$ be a planted constrained satisfiability problem with distribution Q_σ^f over $X_k \times \mathbb{Z}_d$, where f has distributional complexity $r = r(f)$. Any randomized statistical algorithm that finds*

an assignment τ such that τ is $\left(\sqrt{\frac{-2\ln(\eta/2)}{n}}\right)$-correlated with σ with probability at least $\Lambda > \eta$ over the choice of σ and the internal randomness of the algorithm needs at least m calls to the 1-MSTAT(L) oracle (resp. VSTAT$\left(\frac{n^r}{2(\log n)^{2r}}\right)$)with $m \cdot L \geq c_1 \left(\frac{n}{\log n}\right)^r$ (resp. $m \geq n^{c_1 \log n}$) for a constant $c_1 = \Omega_{k,1/(\Lambda-\eta)}(1)$. In particular if we set $L = \left(\frac{n}{\log n}\right)^{r/2}$ then our algorithms needs at least $m \geq c_1 \left(\frac{n}{\log n}\right)^{r/2}$ calls to 1-MSTAT(L).

The proof of Theorem 6 follows from Theorems 7 and 5. Theorems 6 and 7 generalize results of Feldman et al. The results of Feldman et al. [73] only apply for functions $f : \{0,1\}^k \to \{0,1\}$. An interested reader can find our proofs in Appendix 8.2. At a high level our proof proceeds as follows: Given any function $h : X_k \to \mathbb{R}$ we show that $\Delta(\sigma, h)$ can be expressed in the following form:

$$\Delta(\sigma, h) = \sum_{\ell=r(f)}^{k} \frac{1}{|X_\ell|} b_\ell(\sigma) \ ,$$

where $|X_\ell| = \Theta\left(n^\ell\right)$ and each function b_ℓ has degree ℓ (Lemma 5). We then use the general hypercontractivity theorem [116, Theorem 10.23] to obtain the following concentration bound.

Lemma 3. *Let $b : \mathbb{Z}_d^n \to \mathbb{R}$ be any function with degree at most ℓ, and let $\mathcal{S} \subseteq \mathbb{Z}_d^n$ be a set of assignments for which $d' = d^n/|\mathcal{S}| \geq e^\ell$. Then $\mathbb{E}_{\sigma\sim\mathcal{S}}[|b(\sigma)|] \leq \frac{2(\ln d'/c_0)^{\ell/2}}{d^\ell} \|b\|_2$, where $c_0 = \ell\left(\frac{1}{2ed}\right)$ and $\|b\|_2 = \sqrt{\mathbb{E}_{x\sim\mathbb{Z}_d^n}\left[b(x)^2\right]}$.*

We then use Lemma 3 to bound $\mathbf{E}_{\sigma\sim\mathcal{S}}[\Delta(\sigma, h)]$ for any set $\mathcal{S} \subseteq \mathbb{Z}_d^n$ such that $|\mathcal{S}| = \left|\mathbb{Z}_d^k\right|/d'$ (Lemma 8). This leads to the following bound on $\kappa_2(\mathcal{D}', U_k) = O_k\left((\ln d'/n)^{r(f)/2}\right)$.

Theorem 7. *Let $\mathcal{Z}_{Q,\epsilon}$ denote the problem of finding for every $\sigma \in \mathbb{Z}_d^n$, an assignment $\tau \in \mathbb{Z}_d^k$ that is ϵ-correlated with σ given access to distribution Q_σ^f over $X_k \times \mathbb{Z}_d$. Then there exists a constant $c_Q > 0$ such that for any $\epsilon > 1/\sqrt{n}$ and $q \geq n$,*

$$\mathrm{SDN}\left(\mathcal{Z}_{Q,\epsilon}, \frac{c_Q (\log q)^{r/2}}{n^{r/2}}, 2e^{-n\cdot\epsilon^2/2}\right) \geq q \ ,$$

where $r = r(f)$ is the distributional complexity of f.

3.5 Security Analysis

In this section we analyze the security of a human computable password scheme using our statistical query lower bounds as a building block. In section 3.5.1 we show that any adversary that breaks UF-RCA security can also (approximately) recover the secret mapping σ. As we showed in Section 3.4 statistical algorithms need at least $m = \tilde{\Omega}\left(n^{r(f)/2}\right)$ challenge-response pairs to recover the secret mapping. This implies that no statistical adversary can break UF-RCA security. This is significant because most known algorithmic techniques can be modeled within the statistical query framework. While Gaussian Elimination is a notable exception, we show that an adversary needs $m = \tilde{\Omega}\left(n^{r(f)/2}\right)$ challenge-response pairs to recover σ using Gaussian Elimination in Section 3.5.2 .

3.5.1 Breaking UF-RCA is Equivalent to Secret Recovery

Theorem 6 only establishes that it is hard for a statistical adversary to properly learn the secret mapping σ. Could an adversary win our password security game without properly learning the secret mapping? In learning theory it is NP-hard to find a 2-term DNF that is consistent with a given dataset. However, just because 2-DNF is hard to learn in the *proper* learning model does not mean that learning 2-DNF is hard. Indeed, if we allow our learning algorithm to output a linear classifier instead of a 2-term DNF then 2-DNF is easy to learn [96]. Of course, for some functions it is very easy to predict challenge-response pairs without learning σ. For example, if f is the constant function — or any function highly correlated with the constant function — then it is easy to predict the value of $f(\sigma(C))$. However, any function that is highly correlated with a constant function is a poor choice for a human computable passwords scheme. We argue that any adversary that can win the password game can be converted into an adversary that properly learns σ provided that our function f has certain reasonable properties.

Definition 11. *We say that a function f is (δ_1, δ_2)—hard to predict if $\forall \sigma, \sigma' \in \mathbb{Z}_d^n$ s.t. σ is δ_1-balanced and σ' is not δ_1-correlated with σ we have*

$$\Pr_{C \sim X_k}\left[f(\sigma(C)) = f(\sigma'(C))\right] \leq \frac{1}{d} + \delta_2 .$$

Intuitively, Definition 11 says that if σ is approximately balanced (e.g., for each $i \in \mathbb{Z}_d$ the string σ contains $\approx n/d$ i's) and σ' is not δ_1-correlated with σ then $f(\sigma'(C))$

is not a good predictor of $f(\sigma(C))$. We note that if f is highly correlated with a constant function then $f(\sigma'(C))$ will always be a good predictor of $f(\sigma(C))$.

Corollary 1 says that any statistical adversary needs to see at least $\tilde{\Omega}\left(n^{r(f)/2}\right)$ example challenge response pairs before it can accurately guess the value of $f(\sigma(C))$ for a randomly chosen challenge $C \in X_k$. Corollary 1 follows easily from Theorems 8 and 6.

Theorem 8. *Let f be (δ_1, δ_2)—hard to predict, let $\sigma \sim \mathbb{Z}_d^n$ denote the secret mapping, let $\epsilon > 0$ be any constant and suppose that we are given labels $\ell_C \in \mathbb{Z}_d$ for every $C \in X_k$ s.t*

$$\Pr_{C \sim X_k}\left[f(\sigma(C)) = \ell_C\right] \geq \frac{1}{d} + \delta_2 + \epsilon\,.$$

There is a polynomial time algorithm (in $n, 1/\epsilon, 1/\delta_2$) that with high probability finds a mapping $\sigma' \in \mathbb{Z}_d^n$ such that σ' is δ_1-correlated with σ provided that σ is δ_1-balanced.

Corollary 1. *Let $\epsilon > 0$ be any constant and let $\mathcal{A}$ be any statistical adversary that outputs labels ℓ_C for each clause $C \in X_k$ after making at most $m = o\left(\frac{n^{r(f)/2}}{\log^{r(f)} n}\right)$ queries to 1-MSTAT$\left(n^{r(f)/2}\right)$. If f is (δ_1, δ_2)—hard to predict then with probability $1 - o(1)$ we have*

$$\Pr_{\substack{\sigma \sim \mathbb{Z}_d^n \\ C \sim X_k}}\left[f(\sigma(C)) = \ell_C\right] \leq \frac{1}{d} + \delta_2 + \epsilon\,.$$

We will briefly overview the proof of Theorem 8 here — see Appendix 8.4 for more details. We first randomly partition $[n]$ into n/τ parts $S_1, ..., S_{n/\tau}$ where $\tau = \tilde{O}(\log n)$. For each set S_i, we can check all of the mappings $\sigma_i'(S_i) \in \mathbb{Z}_d^\tau$ to find the one that is consistent with the most noisy labels ℓ_C for each $C \in S_i$. With high probability for each set S_i we have

$$\Pr\left[\ell_C = f(\sigma(C)) \mid C \subset S_i\right] \approx \Pr\left[\ell_C = f(\sigma(C))\right]\,.$$

Because f is (δ_1, δ_2)—hard to guess this means that for each i the strings $\sigma_i'(S_i) \in \mathbb{Z}_d^\tau$ and $\sigma(S_i) \in \mathbb{Z}_d^\tau$ are δ_2-correlated. We combine each of the σ_i' mappings to construct a mapping $\sigma' \in \mathbb{Z}_d^n$ that is δ_2-correlated with σ.

Theorem 9 further extends our argument. Any adversary $\mathcal{A}$ that can win our security game could be used in a blackbox manner to recover a mapping σ' that is highly correlated with the secret mapping σ. Theorem 9 implies that no statistical adversary can break the security of our human computable password schemes with $o\left(n^{s(f)}\right)$ examples.

Theorem 9. *Suppose that f is (δ_1, δ_2)—hard to predict, but that f is not* **UF−RCA** *(n, m, t, δ)−secure for $\delta > \left(\frac{1}{d} + \delta_2 + \epsilon\right)^t$. Then there is a probabilistic polynomial time algorithm (in n, m, $1/\delta_1$, $1/\delta_2$, $1/\epsilon$) that extracts a string $\sigma' \in \mathbb{Z}_d^n$ that is c-correlated with σ after seeing $\tilde{O}(m)$ examples, where $c > 0$ is a constant.*

To prove Theorem 9 we first show how to use the adversary $\mathcal{A}$ as a blackbox to generate (noisy) predictions ℓ_C for every clause $C \in X_k$. By Lemma 8 we can use these predictions to find a mapping σ' that is highly correlated with the secret mapping σ.

We use $\mathcal{A}_{C_1,\ldots,C_m} : (X_k)^t \to \mathbb{Z}_d^t$ to denote an adversary who sees examples $C_1, \ldots, C_m \in X_k$ and $f(C_1), \ldots, f(C_m)$. $\mathcal{A}_{C_1,\ldots,C_m}\left(C'_1, \ldots, C'_t\right) \in \mathbb{Z}_d^t$ denotes the adversaries prediction of $f\left(\sigma\left(C'_1\right)\right), \ldots, f\left(\sigma\left(C'_t\right)\right)$. Given a function $b : (X_k)^t \to \mathbb{Z}_d^t$, challenges $C'_1, \ldots, C'_t \in X_k$ and responses $f\left(\sigma\left(C'_1\right)\right), \ldots, f\left(\sigma\left(C'_t\right)\right)$ we use $\mathcal{P}_{b,i,C'_1,\ldots,C'_m} : X_k \times [t] \to \mathbb{Z}_d \cup \{\perp\}$ to predict the value of a clause $C \in X_k$

$$\mathcal{P}_{b,C'_1,\ldots,C'_t}(C, i) = \begin{cases} b\left(\hat{C}_1, \ldots, \hat{C}_t\right)[i], & \text{if } f\left(\sigma\left(\hat{C}_j\right)\right) = b\left(\hat{C}_1, \ldots, \hat{C}_t\right)[j] \ \ \forall j < i \\ \perp, & \text{otherwise} \end{cases}$$

where $\hat{C}_i = C$ and $\hat{C}_j = C'_j$ for $j \neq i$. We allow our predictor $\mathcal{P}_{b,C'_1,\ldots,C'_t}(C, i)$ to output $\perp$ when it is unsure. Informally, Claim 1 says that for $b = \mathcal{A}_{C_1,\ldots,C_m}$ our predictor $\mathcal{P}_{b,i,C'_1,\ldots,C'_m}$ is reasonably accurate whenever it is not unsure. The proof of Claim 1 can be found in Appendix 8.4. Briefly, Claim 1 follows because for $b = \mathcal{A}_{C_1,\ldots,C_m}$ we have

$$\Pr\left[\mathbf{Wins}\left(\mathcal{A}, n, m, t\right)\right] = \prod_{i=1}^{d} \Pr_{\substack{C \sim X_k \\ C_1,\ldots,C_m \sim X_k \\ C'_1,\ldots,C'_t \sim X_k}} \left[\mathcal{P}_{b,C'_1,\ldots,C'_t}(C, i) = f(\sigma(C)) \,\middle|\, \mathcal{P}_{b,C'_1,\ldots,C'_t}(C, i) \neq \perp\right].$$

Claim 1. *Let $\mathcal{A}$ be an adversary s.t $\Pr\left[\mathbf{Wins}\left(\mathcal{A}, n, m, t\right)\right] > \left(\frac{1}{d} + \delta + \epsilon\right)^t$ and let $b = \mathcal{A}_{C_1,\ldots,C_m}$ then*

$$\Pr_{\substack{i \sim [t], C \sim X_k \\ C_1,\ldots,C_m \sim X_k \\ C'_1,\ldots,C'_t \sim X_k}} \left[\mathcal{P}_{b,C'_1,\ldots,C'_t}(C, i) = f(\sigma(C)) \,\middle|\, \mathcal{P}_{b,C'_1,\ldots,C'_t}(C, i) \neq \perp\right] \geq \left(\frac{1}{d} + \delta + \epsilon\right).$$

Now we can select a random index $i_C \sim [t]$ for each clause $C \in X_k$, and set $\ell_C = \mathcal{P}_{b,C'_1,\ldots,C'_t}(C, i_C)$ whenever $\mathcal{P}_{b,C'_1,\ldots,C'_t}(C, i_C) \neq \perp$. The remaining challenge is that we need to label for all of the clauses $C \in X_k$ before we can apply Theorem 8. To ensure that all clauses are labeled we construct multiple independent predictors. Notice that for each clause $C \in X_k$ the probability that $\mathcal{P}_{b,C'_1,\ldots,C'_t}(C, i_C) \neq \perp$ is at least $1/t$ (the probability that $i_C = 1$).

3.5.2 Gaussian Elimination

Most known algorithmic techniques can be modeled within the statistical query framework. Gaussian Elimination is a notable exception. As an example consider the function $f(x_1, \ldots, x_7) = x_1 + \ldots + x_7 \mod 10$ (in this example $r(f) = 7$ and $g(f) = 0$). Our previous results imply that any statistical algorithm would need to see at least $m = \tilde{\Omega}\left(n^{7/2}\right)$ challenge response pairs $(C, f(\sigma(C)))$ to recover σ. However, it is trivial to recover σ from $O(n)$ random challenge response pairs using Gaussian Elimination. In general, consider the following attacker shown in algorithm 3.1, which uses Gaussian Elimination. Algorithm 3.1 relies on the subroutine **TryExtract** $(C, f(\sigma(C)) S, \alpha)$, which attempts to extract a linear constraint from $(C, f(\sigma(C)))$ under the assumption that $\sigma(S) = \alpha$. We assume **TryExtract** $(C, f(\sigma(C)) S, \alpha)$ returns $\emptyset$ if it cannot extract a linear constraint.

Algorithm 3.1 GaussianAttack

Input: Clauses $C_1, \ldots, C_m \sim X_k$, and labels $f(\sigma(C_1)), \ldots, f(\sigma(C_m))$.
for all $S \in X_{g(f)}$, $\alpha \in \mathbb{Z}_d^{g(f)}$ **do**
  LC $\leftarrow \emptyset$ ▷ **LC** is the set of linear constraints extracted
  for all $C \in \{C_1, \ldots, C_m\}$ **do**
    LC $\leftarrow$ **LC** $\bigcup$ **TryExtract** $(C, f(\sigma(C)), S, \alpha)$
    if $|\mathbf{LC}| \geq n$ **then**
      $\sigma' \leftarrow$ **LinearSolve** (LC)
      if $\forall i \in [m].\ f(\sigma'(C_i)) = f(\sigma(C_i)) \in C$ **then return** σ'

Fact 1 says that an attacker needs at least $m = \tilde{\Omega}\left(n^{1+g(f)}\right)$ challenge-response pairs to recover σ using Gaussian Elimination. This is because the probability that **TryExtract** $(C, f(\sigma(C)) S, \alpha)$ extracts a linear constraint is at most $O\left(\left(\frac{|S|}{n}\right)^{-g(f)}\right)$, which is $O\left(n^{-g(f)}\right)$ for $|S|$ constant. The adversary needs $O(n)$ linearly independent

constraints to run Gaussian Elimination. If the adversary can see at most $\tilde{O}\left(n^{s(f)}\right)$ examples neither approach (Statistical Algorithms or Gaussian Elimination) can be used to recover σ.

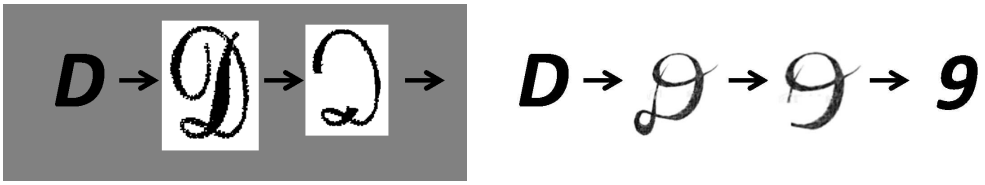
Fact 1. *Algorithm 3.1 needs to see at least* $m = \tilde{\Omega}\left(n^{1+g(f)}\right)$ *challenge-response pairs to recover* σ.

Remark 2 explores the tradeoff between the adversary's running time and the number of challenge-response pairs that an adversary would need to see to recover σ using Gaussian elimination. In particular the adversary can recover σ from $\tilde{O}\left(n^{1+g(f)/2}\right)$ challenge-response pairs if he is willing to increase his running time by a factor of $d^{\sqrt{n}}$. In practice, this attack may be reasonable for $n \leq 100$ and $d = 10$, which means that it may be beneficial to look for candidate human computable functions f that maximize $\min\{r(f)/2, 1 + g(f)/2\}$ instead of $s(f)$ whenever $n \leq 100$.

Remark 2. *If the adversary correctly guesses value of* $\sigma(S)$ *for* $|S| = n^{\epsilon}$ *then he may be able to extract a linear constraint from a random example with probability* $\Omega(1/n^{(1-\epsilon)g(f)})$. *The adversary would only need* $\tilde{O}\left(n^{1+(1-\epsilon)g(f)}\right)$ *examples to solve for* σ, *but his running time would be proportional to* $d^{\epsilon n}$ *— the expected number of guesses before he is correct.*

3.6 Candidate Secure Human Computable Functions

For all of our candidate human computable functions $f : \mathbb{Z}_d^k \to \mathbb{Z}_d$ we fix $d = 10$ because most humans are used to performing arithmetic operations on digits. A good human computable function should balance security and usability. A secure human computable function should have $r(f)$ and $g(f)$ large. This makes it challenging to simultaneously achieve usability because usable human computable function should only require the user to perform a few simple operations to evaluate f. We present two candidate human computable functions and analyze their security parameters. We consider the usability of our human computable password schemes by (1) discussing ways that the secret mapping could be memorized easily, (2) analyzing the extra effort that a user needs to spend rehearsing to remember the secret mapping σ, and (3) estimating the time it would take a human user to compute a password. Algorithms 3.2 and 3.3 illustrate the authentication process. To protect users from offline attacks in the event of a server breach, passwords should be stored using a slow cryptographic hash function $\mathbf{H}$ like BCRYPT [122]. Servers could also use GOTCHAs (Chapter 6) or HOSPs [51] for additional protection.



(a) $M_{D,2}$ (b) $M_{D,9}$

Figure 3.1: Mnemonics to help memorize the secret mapping σ

Mnemonics to help memorize σ In practice, we envision that the user memorizes a mapping from n objects (e.g., images) to digits. For example, if $n = 26$ and $d = 10$ then the user might memorize a mapping from characters to digits. To memorize the mapping $\sigma(D) = 2$ we might show a visually inclined user an animation of the letter D transforming into a 2 (see Figure 3.1a). If instead $\sigma(D) = 9$ then we would show the user a different animation (see Figure 3.1b). One nice feature of this approach is that we only need to generate nd illustrations to help our users memorize any mapping (e.g., to help users memorize any mapping from characters to digits we would need just 260 such illustrations — 10 for each character).

Algorithm 3.2 CreateChallenges

Input: n, m, base d, random bits b, images $I_1, ..., I_n$, and mnemonic helpers $M_{i,j}$ for $i \in [n]$, $j \in \{0, \ldots, d-1\}$.

▷ Generate and Memorize Secret Mapping

for $i = 1 \rightarrow n$ **do**

$\sigma(i) \sim \{0, \ldots, d-1\}$ %`Using random bits b`

$M_i \leftarrow M_{i,\sigma(i)}$

(User) Using M_i memorizes the association $(I_i, \sigma(i))$ for $i \in [n]$.

▷ Generate Challenges

for $i = 1 \rightarrow m$ **do**

for $j = 1 \rightarrow t$ **do**

$C_j^i \sim X_k$

$\vec{C}_i \leftarrow \left\langle C_1^i, \ldots, C_t^i \right\rangle$ ▷ **H** is a strong cryptographic hash function

(User) Computes $\langle q_1, \ldots, q_t \rangle = f\left(\sigma\left(\vec{C}_i\right)\right)$

(Server i**)** Stores $h_i = \mathbf{H}\left(\vec{C}_i, \langle q_1, \ldots, q_t \rangle\right)$

return $\vec{C}_1, \ldots, \vec{C}_m$

Algorithm 3.3 Authenticate

Input: Security parameter t. Account $i \in [m]$. Challenges $\vec{C}_1, \ldots, \vec{C}_m$.
$\left\langle C_1^i, \ldots, C_t^i \right\rangle \leftarrow \vec{C}_i$ ▷ Display Single Digit Challenges
for $j = 1 \rightarrow t$ **do**
(Semi-Trusted Computer) Displays C_j^i to the user.
(User) Computes $q_j \leftarrow f\left(\sigma\left(C_i^j\right)\right)$.
(Semi-Trusted Computer) Sends $\langle q_1, \ldots, q_t \rangle$ to the server for account i.
(Server) Verifies that $\mathbf{H}\left(\vec{C}_i, \langle q_1, \ldots, q_t \rangle\right) = h_i$

3.6.1 Candidate Scheme 1

Our first candidate human computable password scheme uses the function

$$f_1\left(x_0, x_1, x_2, x_3, x_4, \ldots, x_{13}\right) = x_{13} + x_{12} + x_{(x_{11}+x_{10} \mod 10)} \mod 10\,.$$

Claim 2 and Theorems 6 and 9 provide strong evidence that an adversary will need to see $\tilde{\Omega}\left(n^{1.5}\right)$ example challenge-response pairs before he can recover the secret mapping σ or begin to forge the user's passwords. A formal proof of Claim 2 can be found in the appendix. We first observe that to influence the value of $f_1(x_0, \ldots, x_{13})$ we must fix the values of x_{12}, x_{13} and at least one x_i for $i \in \mathbb{Z}_{10}$. Similarly, we must fix the values of x_{10} and x_{11} to make the resulting function linear. Therefore, $r(f_1) = 3$ and $g(f_1) = 2$.

Claim 2. *$r(f_1) = 3$, $g(f_1) = 2$ and $s(f_1) = 3/2$.*

The proof of fact 2 can be found in the appendix.

Fact 2. *f_1 is $(0.01, 0.045)$—hard to predict.*

Remark 3. *Claim 2 and Theorem 6 imply that any statistical adversary needs to see $\tilde{\Omega}\left(n^{3/2}\right)$ example challenge-response pairs to recover σ in the human computable password scheme given by f_1. Fact 2 and Corollary 1 demonstrate that a statistical adversary needs at least $\tilde{\Omega}\left(n^{3/2}\right)$ example challenge response pairs before it can guess the response to a random challenge with probability > 0.145. Theorem 9 provides strong evidence that f_1 is $\mathbf{UF-RCA}\,(n, m, t, \delta)$ − secure for $m = \tilde{\Omega}\left(n^{3/2}\right)$ and $\delta > 0.145^t$.*

3.6.2 Candidate Scheme 2

Our second candidate human computable password scheme uses the function

$$f_2(x_0, x_1, x_2, x_3, x_4, \ldots, x_{13}) = x_{13} + x_{12} + x_{11} + x_{(x_{10} \mod 10)} \mod 10 .$$

Claim 3 and Theorem 6 provide strong evidence that an adversary will need to see $\tilde{\Omega}\left(n^{1.5}\right)$ example challenge-response pairs before he can recover the secret mapping σ or begin to forge the user's passwords. The proof of Claim 3 is very similar to the proof of Claim 2. To influence the value of $f_1(x_0, \ldots, x_{13})$ we must fix the values of x_{11}, x_{12}, x_{13} and at least one x_i for $i \in \mathbb{Z}_{10}$. Similarly, we must fix the value of x_{10} to make the resulting function linear. Therefore, $g(f_2) = 1$ and $r(f_2) = 3$.

Claim 3. *$r(f_2) = 4$, $g(f_2) = 1$ and $s(f_2) = 2$.*

Fact 3. *f_2 is $(0.01, 0.01)$—hard to predict.*

Remark 4. *Claim 2 and Theorem 6 imply that any statistical adversary needs to see $\tilde{\Omega}(n^2)$ example challenge response pairs to recover σ in the human computable password scheme given by f_2. Fact 2 and Corollary 1 demonstrate that a statistical adversary needs at least $\tilde{\Omega}(n^2)$ example challenge response pairs before it can guess the response to a random challenge with probability > 0.11. Theorem 9 provides strong evidence that f_1 is (n, m, t, δ) for $m = \tilde{\Omega}(n^2)$ and $\delta > 0.11^t$.*

3.6.3 Usability:

We analyze usability along two dimensions: (1) the extra effort required for the user to memorize and rehearse the secret mapping σ, and (2) the time that it takes the user to compute his password when he wants to login.

Memorizing and Rehearsing σ

We adopt the usability model from Chapter 2.4 to quantify the extra effort that a user would need to spend rehearsing the mapping σ (the results are summarized in Table 3.1).We quantify usability by calculating $\mathbb{E}[XR_{365}]$, the expected number of extra rehearsals that the user will be required to do to remember the secret mapping σ during the first year.

Review. Suppose that the user has m accounts $A_1, \ldots, A_m$. Recall that a visitation schedule for an account A_i is a sequence of real numbers $\tau_0^i < \tau_1^i < \ldots$, which represent the times when the account A_i is visited by the user. Recall that a rehearsal requirement $\left[t_i^{\hat{c}}, t_{i+1}^{\hat{c}}\right)$ for a cue-association pair $(\hat{c}, \hat{a})$ can be satisfied naturally if the user visits a site A_j that uses the cue $\hat{c}$ $\left(\hat{c} \in c_j\right)$ during the given time window. Here, c_j denote the set of cue-association pairs that the user must remember when logging into account A_j. In our case the user must remember the cue-association pairs $(i, \sigma(i))$ for each $i \in [n]$.

Example: Consider the human computable function f_1 from Section 3.6, and suppose that the user has to compute $f_1(\sigma(C_i))$ to authenticate at account A_j, where $C_i = (x_0, \ldots, x_{13})$. When the user computes f_1 he must rehearse the associations $(x_{10}, \sigma(x_{10}))$, $(x_{11}, \sigma(x_{11}))$, $(x_{12}, \sigma(x_{12}))$, $(x_{13}, \sigma(x_{13}))$ and $(x_i, \sigma(x_i))$ where $i = (\sigma(x_{10}) + \sigma(x_{11}) \mod 10)$. Thus $c_j \supset \{x_i, x_{10}, x_{11}, x_{12}, x_{13}\}$. When user authenticates he naturally rehearses each of these associations in c_j.

Evaluating Usability Given a sufficient rehearsal schedule and a visitation schedule, Theorem 1 predicts the value of XR_t, the total number of extra rehearsals that a user will need to do to remember all of the cue-association pairs required to reconstruct all of his passwords for t days. We use the formula from Theorem 1 to obtain the usability results in Table 2.3. To evaluate this formula we need to be given the rehearsal requirements, a visitation schedule (λ_i) for each account A_i and a set of public challenges $\vec{C}_i \in (X_{14})^{10}$ for each account A_i. The rehearsal requirements are given by the Expanding Rehearsal Assumption from Chapter 2.4 (we use the same association strength parameter $\sigma = 1$), and the visitation schedules for each user are given in Table 2.1. We assume that each password is 10 digits long and that the challenges $\vec{C}_i \in (X_{14})^{10}$ are chosen at random by algorithm 2.2. Notice that each time the user responds to a single digit challenge he rehearses the secret mapping at five locations (see section 3.6.3). Because the value of $\mathbb{E}[XR_{365}]$ depends on the particular password challenges that we generated for each account, we ran Algorithm 3.2 and computed the resulting value $\mathbb{E}[XR_{365}]$ one-hundred times. The values in Table 2.3 represent the mean value of $\mathbb{E}[XR_{365}]$.

Discussion One of the advantages of our human computable passwords schemes is that memorization is essentially a one time cost for our Very Active, Typical and Occasional users. That is once the user has memorized the mapping

	Our Scheme $\left(\sigma \in \mathbb{Z}_{10}^{n}\right)$			*Shared Cues*		
User	$n = 100$	$n = 50$	$n = 30$	SC-0	SC-1	SC-2
Very Active	0.396	0.001	≈ 0	≈ 0	3.93	7.54
Typical	2.14	0.039	≈ 0	≈ 0	10.89	19.89
Occasional	2.50	0.053	≈ 0	≈ 0	22.07	34.23
Infrequent	70.7	22.3	6.1	≈ 2.44	119.77	173.92

Table 3.1: $\mathbb{E}[XR_{365}]$: Extra Rehearsals over the first year to remember σ in our scheme. Compared with *Shared Cues* schemes SC-0,SC-1 and SC-2[33].

A	B	C	D
0	E	5	J
1	F	6	K
2	G	7	L
3	H	8	M
4	I	9	N

Table 3.2: Single-Digit Challenge Layout in Scheme 1

$\sigma : \{1, ..., n\} \rightarrow \mathbb{Z}_d$ he will get sufficient natural rehearsal to maintain this memory. With the exception of SC-0 (the least secure *Shared Cues* scheme), our schemes require the user to expend *less* extra effort rehearsing his secret mapping. Intuitively, this is because human computable password schemes give the user more opportunities to naturally rehearse σ during the authentication process. To compute $f_1(\sigma(\{1, \ldots, 14\}))$ the user would need to recall the values of $\sigma(11), \sigma(12), \sigma(13), \sigma(14)$ and $\sigma(1 + (\sigma(11) + \sigma(12) \mod 10))$. If the user has 10 digit passwords then he will naturally rehearse the value of σ at up to fifty different locations each time he computes one of his passwords. The disadvantage is that the user needs to spend extra time computing his password each time he authenticates.

Computation Time

To help the user compute the response to a single digit challenge C more quickly a semi-trusted computer could display the challenge in a more helpful manner. For example, the challenge $C = (E, F, G, H, I, J, K, L, M, N, A, B, C, D) \in X_{14}$ might be displayed to the user as in Table 3.2. Now to compute $f(\sigma(C))$ in scheme 1 the user would execute the following steps (1) Recall $\sigma(A)$ — the number associated

with the letter A, (2) Recall $\sigma(B)$, (3) Compute $i = \sigma(A) + \sigma(B) \mod 10$ — without loss of generality suppose that $i = 8$, (4) Find the letter at index i—M if $i = 8$, (5) Recall $\sigma(M)$ (6) Recall $\sigma(C)$ (8) Compute $j = \sigma(M) + \sigma(C) \mod 10$ (9) Recall $\sigma(D)$ (10) Return $j + \sigma(D) \mod 10$.

Notice that the computation at each step only relies on values from the last two steps so we do not require the user to keep more than 7 chunks of information in active memory [108]. Thus, in scheme 1 the user can compute his response to a single-digit challenge in 10 mental steps, and it would take 10ℓ steps to respond to a length-ℓ password challenge.

We timed ourselves to determine how long each scheme took one of the authors to evaluate. After the first author had memorized the secret mapping it took him $\hat{t} = 7.5$ seconds on average to respond to compute the response $f(\sigma(C))$ to a random challenge $C \in X_k$ in both schemes. Thus, our schemes are 7.5-human computable for at least some human users so it would take 75 seconds to compute a 10 digit password using this scheme[4].

3.6.4 Statistical Algorithms: Security Upper Bound

Theorem 10 demonstrates that our lower bound for statistical algorithms are asymptotically tight for both of our human computable password schemes. In particular, we demonstrate that $m = \tilde{O}\left(n^{r(f)/2}\right)$ queries to 1-MSTAT are sufficient for a statistical algorithm to recover σ.

Theorem 10. *For $f_i \in \{f_1, f_2\}$ there is a randomized algorithm that makes $O\left(n^{\max\{1, r(f_i)/2\}} \log^2 n\right)$ calls to the 1-MSTAT$\left(n^{\lceil r(f_i)/2 \rceil}\right)$ oracle and returns σ with probability $1 - o(1)$.*

For binary functions $f' : \{0,1\}^k \to \{0,1\}$, Feldman et al. [73] gave a randomized statistical algorithm to find $\sigma' \in \{0,1\}^n$ using just $O\left(n^{r(f)/2} \log^2 n\right)$ calls to the 1-MSTAT$\left(n^{\lceil r(f)/2 \rceil}\right)$ oracle. Their main technique is a discrete spectral iteration procedure to find the eigenvector (singular vector) with the largest eigenvalue (singular value) of a matrix M sampled from a distribution $M_{\sigma',p}$ over $\left|X_{\lfloor r(f)/2 \rfloor}\right| \times \left|X_{\lceil r(f)/2 \rceil}\right|$ matrices. With probability $1 - o(1)$ this eigenvector will encode the value $\sigma'(C)$ for each clause $C \in X_{r(f)/2}$. We show that the discrete spectral iteration algorithm of

[4] Admittedly, we may not be a representative sample for an average human user, but we would argue that this is at least a reasonable approximation of the average member of the computer science community.

Feldman et al. [73] can be extended to recover $\sigma \in \mathbb{Z}_{10}$ when $f \in \{f_1, f_2\}$ is one of our candidate human computable functions. See Appendix 8.4.1 for more details.

Discussion We note that Theorem 10 cannot be extended to arbitrary functions $f : \mathbb{Z}_d^k \to \mathbb{Z}_d$. Consider for example the unique function $f : \mathbb{Z}_{10}^6 \to \mathbb{Z}_{10}$ s.t. $f(x_1, \ldots, x_6) \equiv f'(x_1 \mod 2, \ldots, x_6 \mod 2) \mod 2$ and $f(x_1, \ldots, x_6) \equiv f''(x_1 \mod 5, \ldots, x_6 \mod 5) \mod 5$, where $f' : \mathbb{Z}_2^6 \to \mathbb{Z}_2$ and $f'' : \mathbb{Z}_5^6 \to \mathbb{Z}_5$. By the Chinese Remainder Theorem instead of picking a secret mapping $\sigma \in \mathbb{Z}_{10}^n$ we could equivalently pick the unique secret mappings $\sigma_1 \in \mathbb{Z}_2^n$ and $\sigma_2 \in \mathbb{Z}_5^n$ s.t $\sigma \equiv \sigma_1 \mod 2$ and $\sigma \equiv \sigma_2 \mod 5$. Now drawing challenge response pairs from the distributions Q_σ^f is equivalent to drawing challenge-response pairs from the distributions $Q_{\sigma_1}^{f'}$ and $Q_{\sigma_2}^{f''}$. Suppose that $f'(x_1, \ldots, x_6) = x_1x_2 + x_3 + x_4 + x_5 + x_6 \mod 2$, and $f''(x_1, \ldots, x_6) = x_1$. Then we have $r(f) = \min(r(f'), r(f'')) = r(f'') = 1$, but $r(f') = 4$. We can find σ_2 using $O\left(n \log^2 n\right)$ calls to 1-MSTAT(n), but to find σ we must first recover σ_1, which requires $\tilde{\Omega}\left(n^{r(f')/2}\right) = \tilde{\Omega}(n^2)$ calls to 1-MSTAT(n^2).

3.7 Discussion

3.7.1 Human Computable Passwords Challenge

Our security lower bounds are asymptotic (e.g., an adversary needs to see $m = \tilde{\Omega}\left(n^{s(f)}\right)$ challenge-response pairs to forge passwords), but in our context the constants are very important. To better understand the exact security bounds in our scheme we created several public challenges to break our candidate human computable password schemes under different parameters (see Table 8.1). The challenges can be found at`http://www.cs.cmu.edu/~jblocki/HumanComputablePasswordsChallenge/challenge.htm`. For each challenge we selected a random secret mapping $\sigma \in \mathbb{Z}_{10}^n$, and published (1) m single digit challenge-response pairs $(C_1, f(\sigma(C_1))), \ldots, (C_m, f(\sigma(C_m)))$, where each clause C_i is chosen uniformly at random from X_k, and (2) 20 length—10 password challenges $\vec{C}_1, \ldots, \vec{C}_{20} \in (X_k)^{10}$. The goal of each challenge is to correctly guess one of the secret passwords $p_i = f\left(\sigma\left(\vec{C}_i\right)\right)$ for some $i \in [20]$. More details can be found in Appendix 8.1.

3.7.2 Security Under Continuous Leakage

Consider the following scenario: the adversary infects the user's computer with malware which is never detected. Every time the user computes a password in response to a challenge the adversary observes the password in plaintext. One way to protect the user in this extreme scenario would be to generate multiple (e.g., 10^6) one-time passwords for each of the user's accounts. While usability concerns make this approach infeasible in a traditional password scheme (it would be far too difficult for the user to memorize a million one-time passwords for each of his accounts), it may be feasible to do this using a human computable password scheme. When we initially generate the secret mapping $\sigma \sim \mathbb{Z}_d^{100}$ we could also generate cryptographic hashes for multiple one-time passwords $\mathbf{H}\left(\vec{C}, f_3\left(\sigma\left(\vec{C}\right)\right)\right)$.

We conjecture that the following candidate human computable password scheme f_3 could be used to provide security even in this extreme scenario

$$f_3\left(x_0, x_1, x_2, x_3, x_4, \ldots, x_{31}\right) = \left(\sum_{i=21}^{31} x_i\right) + x_{\left(\sum_{i=10}^{20} x_i \mod 10\right)} \mod 10\,.$$

The drawback is that f_3 will take longer for a user to execute in his head. It requires the user to perform 23 additions modulo 10 compared with three in the previous schemes f_1 and f_2. The advantage is that the security parameters are quite strong (e.g., $g(f_3) = 11$, $r(f_3) = 12$ and $s(f_3) = 6$), which implies that a polynomial time adversary needs $m = \tilde{\Omega}\left(n^6\right)$ challenge response pairs to recover the secret mapping. If $n = 100$ then the adversary would need around 10^{12} challenge response pairs before he could break **UF-RCA** security. Even if the adversary runs in time proportional to $n^{\sqrt{n}}$ and uses the attack from remark 2 he would still need $\tilde{\Omega}\left(n^{1+5.5}\right)$ examples. If we make the reasonable assumption that a single user has at most 10^5 accounts and never authenticates to any single account more than 10^6 times over the course of his life then the adversary will never see enough examples to recover σ.

3.7.3 Open Questions

Can we precisely characterize the functions $f : \mathbb{Z}_d^k \to \mathbb{Z}_d$ for which we can efficiently recover σ after seeing $\tilde{O}\left(n^{r(f)/2}\right)$ challenge-response pairs? Feldman et al. [73] gave a statistical algorithm that recovers the secret mapping whenever

$d = 2$ after making $\tilde{O}\left(n^{r(f)/2}\right)$ queries to 1-MSTAT$\left(n^{r(f)/2}\right)$. While we show that the same algorithm can be used to recover σ after making $\tilde{O}\left(n^{r(f)/2}\right)$ queries to 1-MSTAT$\left(n^{r(f)/2}\right)$ in our candidate human computable password schemes with $d = 10$, we also showed that these results do not extend to all functions $f : \mathbb{Z}_d^k \to \mathbb{Z}_d$.

Improving Usability Is it possible to improve usability by designing a human computable function $f : \mathbb{Z}_{10}^k \to \mathbb{Z}_{100}$? This could potentially allow the user to generate a secure length t password after responding to only $t/2$ challenges. Our statistical dimension lower bounds also hold for functions $f : \mathbb{Z}_{d_1}^k \to \mathbb{Z}_{d_2}$. As before the challenge would be designing a function that is human computable and has strong security properties (e.g., $s(f)$ is large and f is (δ_1, δ_2)-hard to guess).

Chapter 4

Empirical Validation of User Model

4.1 Introduction

In this chapter we discuss our ongoing user study to quantify the effects of rehearsal and the use of mnemonic techniques on long term memory retention. We are conducting this study online using Amazon's Mechanical Turk framework. Our goal is to empirically evaluate the usability model presented in Chapter 2 as well as the *Shared Cues* password management scheme.

Specific Rehearsal Schedules: Recall that the usability model of Chapter 2 was based on a sufficient rehearsal assumption. The expanding rehearsal assumption from Chapter 2 said that a user can maintain a memory by rehearsing once during each of the time intervals $\left[2^{i\sigma}, 2^{(i+1)\sigma}\right)$, where $i \in \mathbb{N}$ is number of previous rehearsals and σ is a constant that measures association strength. This assumption implies that after the i'th rehearsal a user will be able to recall a memory for $2^{i\sigma}$ more days without any additional rehearsals. Wozniak and Gorzelanczyk [160] conducted an empirical study of undergraduate students who were learning vocabulary words for a foreign language. Their results indicated that σ_w varied slightly with each vocabulary word w (σ_w was smaller for difficult vocabulary words). This raises an important question. What specific rehearsal schedules work in our password context?

Advantages of Mnemonic Techniques: In Chapter 2 we simply used $\sigma = 1$ to measure the usability of a password management scheme, whether or not the password management scheme used mnemonic techniques to help users remember their passwords. This raises an important question. Does the use of mnemonic

techniques (e.g., method of loci, person-action-object stories) allow us to safely adopt a rehearsal schedule with longer intervals in between rehearsals? More formally, is $\sigma_{mnemonic} > \sigma_{non-mnemonic}$?

Interference: Suppose that our user is able to remember a person-action-object story by following the rehearsal schedule given by the expanding rehearsal assumption with association strength σ. Can our user memorize n person-action-object stories by following the same rehearsal schedule or does the user need to follow a more conservative schedule (smaller value of σ) when he is memorizing multiple person-action-object stories?

Study Overview Each participant in the study was asked to memorize several randomly selected actions (e.g., 'swallowing,' 'kicking') and several randomly selected objects (e.g., 'bike,' 'car'). Participants assigned to the mnemonic group were given specific instructions about how to memorize the actions following the *Shared Cues* password management scheme in Chapter 2.6. To help participants in the mnemonic group memorize one of their action(s) and object(s) each participant was shown two additional photos of a person and a scene and was asked to imagine the corresponding person-action-object story taking place inside the scene (e.g., the user might be shown a photos of Bill Gates and a beach and asked to imagine "Bill Gates swallowing a bike on the beach."). Other participants were assigned to the standard group and were simply instructed to memorize their actions and objects (e.g., by typing in their words several times). Participants were paid \$0.50 for completing the memorization phase. After participants memorized their words we periodically asked them to return to rehearse their words. During each rehearsal participants in the mnemonic group were shown the photos of the person and the scene as a cue to help them remember the associated action and object. Participants in the standard group were simply asked to recall their actions and objects. Each participant was assigned a specific rehearsal schedule (e.g., participants in the aggressive rehearsal group were reminded to rehearse on the following days: $1, 2, 4, 8, 16, 32, 64$). During each rehearsal participants were given three chances to remember all of their actions and objects (e.g., their password). To encourage participants to return we paid \$0.75 for each complete rehearsal. To incentivize participants to remember their words we required who forgot their words to re-complete the memorization phase before paying they were paid \$0.75. Participants who did not remember their words during a rehearsal were not asked

to return for future rehearsals.

Preliminary Results. While the user study is still ongoing, we present the results from completed rehearsals in Section 4.4. Our results provide strong empirical evidence that user can remember person-action-object stories by following a rehearsal schedule that satisfies the expanding rehearsal assumption. **Specific Rehearsal Schedules:** Our results demonstrate the benefit of having several early rehearsals. Participants who followed the heavierstart rehearsal schedule (a schedule with several rehearsals on day one) have been very successful at remembering their action-object pairs during rehearsals. Participants following the aggressive rehearsal schedule (the same schedule as heavierstart, but without the extra rehearsal on day 1) struggled to remember all of their action-object pairs during the first rehearsal on day one (25% of participants forgot at least one of their stories), but participants who survived this first rehearsal had much higher success rates during all of the ensuing rehearsals. **Mnemonic Advantage:** Our results strongly support the hypothesis that recall is significantly improved by asking users to follow specific mnemonic techniques to memorize their actions and objects. Participants in the mnemonic group consistently outperformed participants in standard text group during each rehearsal. **Interference:** While participants in other groups did well, no other group did as well as participants who only had to memorize one or two action-object pairs – even participants in groups with more frequent rehearsals. Participants who were asked to memorize only one or two action-object pairs at a time have perfectly remembered their stories during each rehearsal phase.

Organization. In Section 4.2 we discuss related work. We then overview the design of our user study in Section 4.3. While the study is still ongoing we do have some preliminary results from the study. We present these results in Section 4.4.

4.2 Related Work

Pimsleur[120] proposed a rehearsal schedule to help people memorize unfamiliar vocabulary words. His proposed schedule is precisely the schedule given by expanding rehearsal assumption with the association strength constant set to $\sigma = \log_2 5 \approx 2.3$ and the initial delay before the first rehearsal set to 5 seconds (e.g., he suggested rehearsing after 5 seconds, 25 seconds, 2 minutes, 10 minutes, 5

hours, 1 day, 5 days, 20 days). Pimsleur based his recommendations on previous empirical studies[159, pp. 726 ff]. The application SuperMemo[161] uses a similar rehearsal schedule to help users remember flashcards. Wozniak and Gorzelanczyk conducted an empirical study to test these rehearsal schedules[160]. In their study undergraduate students were asked to memorize and rehearse vocabulary words for a foreign language by following a rehearsal schedule very similar to the expanding rehearsal schedule[1]. While these prior studies provide strong empirical evidence for the expanding rehearsal assumption from Chapter 2 we stress that there are two key differences in our study: First, because we are asking the user to memorize secrets that will be used to form passwords our rehearsal schedule needs to be conservative enough that our user will consistently be able to remember his secrets during each rehearsal. In other studies the information participants were asked to memorize (e.g., vocabulary words for a foreign language) was not secret so if the participant forgot this information during a rehearsal they could simply look up the correct answer. However, in the password setting the secrets that the user memorizes should not be written down because they are sensitive so we will not always be able to refresh the user's memory if he forgets his secret. Second, in our password management scheme we are asking users to memorize secret person-action-object stories by following particular mnemonic techniques. Because these stories may be easier or harder to memorize than other information the ideal rehearsal schedule should be tailored to particular mnemonic techniques. Previous studies have demonstrated that cued recall is easier than pure recall (see for example [23]) and that we have a large capacity for visual memories[145]. However, we are not aware of any prior studies which compare cued recall and pure recall when participants are following a rehearsal schedule similar to the one suggested by the expanding rehearsal assumption.

Bonneau and Schechter conducted a user study in which participants were encouraged to slowly memorize a stronger password using spaced repetition[42]. Each time a participant returned to complete a distractor task he was asked to login by entering his password. During the first login the participant was shown four additional random characters and asked to type them in after his password. To encourage participants to memorize these four characters they would intentionally

[1]Wozniak and Gorzelanczyk tracked each students performance with each particular vocabulary word and used that information to estimate how difficult each word was. If a word was deemed 'difficult' then the length of the time interval before the next rehearsal would only increase by a small multiplicative constant (e.g., 1.5) and if the word was judged to be 'easy' then this time interval would increase by a larger multiplicative constant (e.g., 4).

wait a few seconds before displaying them to the user the next time he was asked to login to complete a distractor task. Once a participant was able to login several times in a row (without waiting for the characters to be displayed) they would encourage that participant to memorize four additional random characters in the same way. They found that 88% of participants were able to recall their entire password without any prompting three days after the study was completed. There are several key difference between their study and ours: First, in our study participants were asked to memorize their entire password at the start of the study. By contrast, Bonneau and Schechter encouraged participants to slowly memorize their passwords. Second, Bonneau and Schechter did not tell participants that their goal was to slowly memorize a strong 56 bit password. By contrast, in our study we explicitly told participants that their goal was to remember their words (without writing them down). Finally, participants in our study were given fewer chances to rehearse their passwords and were asked to remember their passwords over a longer duration of time (3 months vs 2 weeks). Bonneau and Schechter asked participants to login 90 times over a two week period. In our study participants were asked to rehearse *at most* 11 times over a period of up to 85 days.

4.3 Study Design

Our user study is being conducted online using Amazon's Mechanical Turk framework. It was approved by the Institutional Review Board (IRB) at Carnegie Mellon University under IRB protocol HS14-294: Sufficient Rehearsal Schedules and Mnemonic Techniques. After participants consented to participate in the research study we randomly assigned each participant to a particular study condition. Members in a particular condition were asked to memorize a particular number of action-object pairs (either 1,2 or 4) by using a particular memorization technique (e.g., mnemonic or standard) and following a particular rehearsal schedule (e.g., aggressive, conservative, heavystart). After we assigned participants to a study condition we asked each participant to complete the memorization phase. During the memorization phase each participant was given several randomly generated actions (e.g., swallowing) and several randomly generated objects (e.g., bike), and asked to memorize each word. Participants in mnemonic conditions were given specific instructions about how to memorize their words. We paid participants $0.50 for completing the memorization phase. After each participant completed the memorization phase we asked them to return periodically to rehearse their words. To encourage participants to return we paid participants $0.75 for each

rehearsal — whether or not they were able to remember the words. If a participant forgot the action and the object then we reminded the participant of the actions and objects that he had memorized and asked that user to complete the memorization phase again.

Below we provide examples of the instructions given to each participant during the memorization and rehearsal phases.

4.3.1 Recruitment Text

On the Mechanical Turk website, participants were recruited with the following text:

> Participate in a Carnegie Mellon University research study on memory. You will be asked to memorize and rehearse random words for a 50 cent payment. After you complete the memorization phase, we will periodically ask you to return to check if you still remember the words. If you forget the words then we will remind you of the words and ask you to complete the memorization phase again. You will be paid 75 cents upon the completion of each rehearsal.
>
> Because this is a memory study we ask that you do not write down the words that we ask you to memorize. You will be paid for each completed rehearsal phase — even if you forgot the words.

After each participant consented to participate in the research study they were assigned to the mnemonic group or to the standard group. We then asked each participant to complete the memorization phase of the study.

4.3.2 Memorization Phase

Mnemonic Group

We first describe the memorization phase for participants assigned to the mnemonic group. Participants in the mnemonic group were first given the following instructions.

Instructions

> This study is being conducted as part of a Carnegie Mellon University research project. It is important that you answer questions honestly and completely. Please take a minute to read the following instructions.
>
> The goal of this study is to quantify the effects of rehearsal and the use of mnemonic techniques on long term memory retention. In this study you will be asked to memorize and rehearse eight random words (four actions and four objects). During the first phase we will ask you to memorize the eight random words - you will be paid $0.50 upon completion of the memorization phase. After you complete the memorization phase we will periodically ask you to return via email to check if you still remember the words. If you forget the words, we will remind you of the words and ask you to complete the memorization phase again. You will be paid $0.75 upon the completion of each rehearsal.
>
> **Important:** Because this is a memory study we ask that you do not write down the words we ask you to memorize. You will be paid for each completed rehearsal phase - even if you forgot the words. You have been assigned to the mnemonic group, which means that we give you specific instructions about how to memorize the words. One of the purposes of this study is to determine how effective certain mnemonic techniques are during the memorization task. We ask that you follow the directions exactly - even if you would prefer to memorize the words in a different way.

After participants finished reading the instructions the memorization phase proceeded as follows:

Memorization Steps. Step 0) Initially, participants were shown a photo of a scene (e.g., Figure 4.1a). Participants were then asked to select a famous person or character (e.g., Darth Vader) and were shown a photograph of the famous person that they selected — see Figure 4.1b. We then generated a random action (e.g., bribing) and a random object (e.g., roach). See Section 4.3.6 for the lists of people, actions and objects used in the study.

Figures 4.1a and 4.1b illustrates Step 0. After we generated the random action and the random object we asked the participant to memorize their action and their

(a) Scene: Lily Pads on the Amazon River (b) Person: Darth Vader

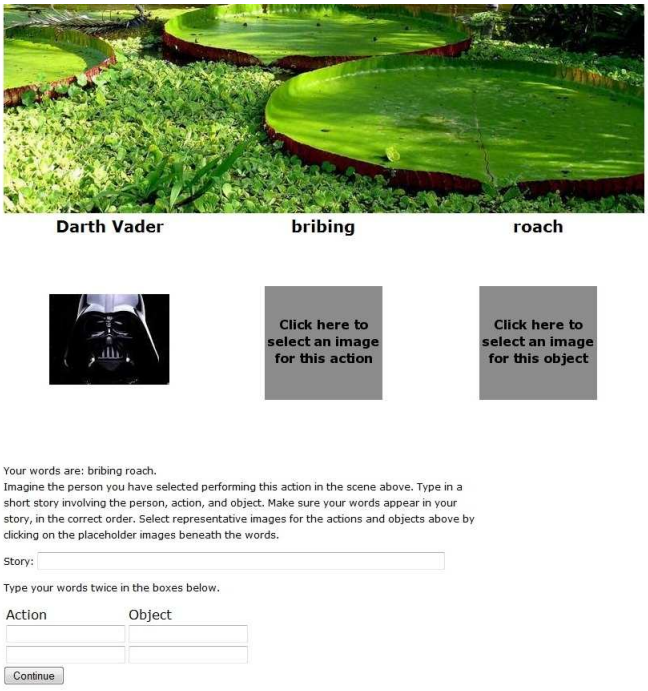
Figure 4.1: Memorization Step 0. Scene and Person.

object by completing Steps 1–3. Figure 4.2a illustrates these steps. Step 1) We asked participants to imagine the person they selected performing the action in the given scene (e.g., imagine Darth Vader bribing the roach on the lily pad). Step 2) We asked each participant to make up a story involving their person, action and object and enter it (e.g., "Darth Vader is bribing a roach")[2]. Step 3) Select a photograph of the action and a photograph of the object, and type in the action and the object two more times. Step 4) We asked most participants to repeat Steps 0 through 3 four times using a new scene (e.g., a baseball field or a hotel room underneath the sea), a new famous person/character and a new — randomly selected — action-object pair during each repetition. Thus, most participants memorized a total of eight words (four actions and four objects). Step 5) Finally, we asked each participant to complete a rehearsal phase (See Figure 4.2b).

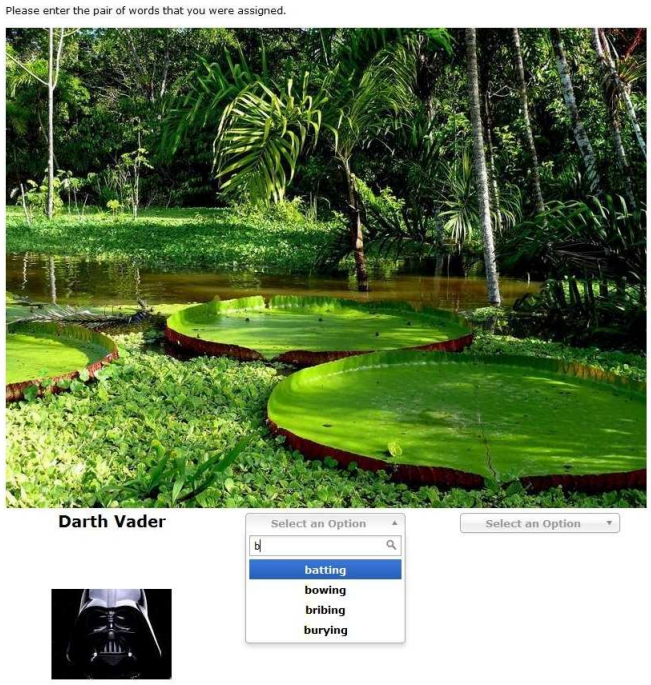
Standard Group

We next describe the memorization phase for participants assigned to the standard group. Participants in the standard group were first given the following instructions.

[2]We required participants to type in a story that contained all of their words in the correct order (Person-Action-Object)



(a) Memorization Steps 1–3. Darth Vader bribing a roach on the lily pad.



(b) Rehearsal Phase. Darth Vader and the photo of the lily pads on the Amazon River are a cue to aid memory recall.

Instructions

> This study is being conducted as part of a Carnegie Mellon University research project. It is important that you answer questions honestly and completely. Please take a minute to read the following instructions.
>
> The goal of this study is to quantify the effects of rehearsal and the use of mnemonic techniques on long term memory retention. In this study you will be asked to memorize and rehearse eight random words (four actions and four objects). During the first phase we will ask you to memorize the eight random words you will be paid $0.50 upon completion of the memorization phase. After you complete the memorization phase we will periodically ask you to return via email to check if you still remember the words. If you forget the words, we will remind you of the words and ask you to complete the memorization phase again. You will be paid $0.75 upon the completion of each rehearsal.
>
> **Important:** Because this is a memory study we ask that you do not write down the words we ask you to memorize. You will be paid for each completed rehearsal phase even if you forgot the words.

After participants finished reading the instructions the memorization phase proceeded as follows:

Memorization Steps. Step 0) We generated a random action and a random object, and displayed these words to the user. Step 1) We asked each participant to spend one minute memorizing his words. We suggested that participants imagine a person performing the action with the object. Step 2) We asked each participant to type in a story which includes the action and the object in the correct order. Step 3) We asked each participant to type in the both words two times — paying attention to the order. See Figure 4.2 for an example of Steps 0–3. Step 4) Most participants were asked to complete Steps 0 through 3 four times to memorize a total of eight words (four actions and four objects). Step 5) Finally, we asked each participant to complete a rehearsal phase.

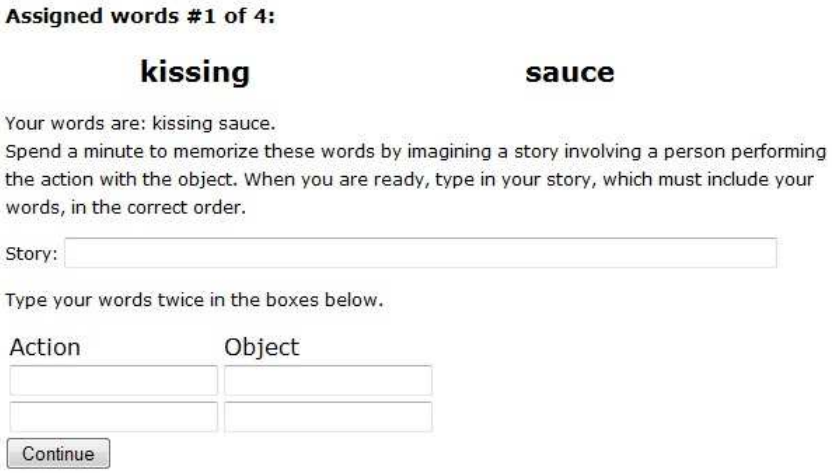


Figure 4.2: User Study: Non-Mnemonic Group Memorization Phase

4.3.3 Rehearsal Phase

Each participant was assigned a particular rehearsal schedule. The particular times that we ask the participant to return were given by the rehearsal schedule that participant was assigned to use (see Table 4.1). We e-mailed participants to remind them to return for each rehearsal:

> Dear Carnegie Mellon study participant: Please return to (url) to participate in the next part of the memory study. If you do not return promptly upon receiving this email, you might not be considered for future phases of the study. You will receive a $0.75 bonus payment for completing this task and it should take less than five minutes.
>
> Remember that you should not write down the words that were assigned to you. You will be paid for each completed rehearsal phase – even if you forgot the words.
>
> There is no need to return to Mechanical Turk and find the HIT to receive the bonus, this bonus and any future bonuses will be applied to this MTurk account automatically as you complete each phase. Please do not attempt to take the HIT again on MTurk as this will result in a rejection.
>
> If, for any reason, you do not want to complete the study, please reply to this email and let us know why, so we can improve our protocol

for future studies.

Thank you! The Carnegie Mellon University Study Team

We describe the rehearsal phase below:

Mnemonic Group

Each participant from the mnemonic group was shown the picture of a scene and the picture of the person that he chose while memorizing his first story during the memorization phase (see Figure 4.2b). We then asked each participant to recall the person action object story he made up and enter the associated action and the object. If the participant was correct then we moved on to the next story. If the participant was incorrect then we asked the participant to try again. After three incorrect guesses we asked the participant to repeat the memorization phase with the same actions and objects, and try again. Once the participant correctly entered all four action-object pairs the rehearsal is finished.

Standard Group

Each participant from the standard group was simply asked to recall the random actions and the random objects that he was given during the memorization phase. If the participant was incorrect then we asked the participant to try again. After three incorrect guesses we asked the participant to repeat the memorization phase, and try again. The rehearsal was finished when the participant enters in all of the actions and objects correctly.

4.3.4 Follow Up Survey

Some participants did not return to rehearse their stories during the rehearsal phase. We cannot tell whether or not these participants would have remembered their passwords if they had returned. Instead we can only report the fraction of participants who remembered their passwords among those who returned for each rehearsal during the study. There are several reasons why a participant may not have returned (e.g., too busy, did not get the follow up message in time, convinced s/he would not remember the password). If participants do not return because they are convinced that they would not remember the password then this could

be a source of bias (e.g., we would be selecting participants who are confident that they remember the story). Our hypothesis is that the primary reason that participants do not return is because they were too busy, because they did not get our follow up message in time or because they do not interested in interacting with us outside of the initial Mechanical Turk Hit, and not because they were convinced that they would not remember the story. In order to test our hypothesis we sent a follow up survey to all participants who did not return to complete a rehearsal phase. The purpose of this survey was to allow us to check for sources of biases. Participants were paid 25 cents for completing this survey. The survey is described below:

> You are receiving this message because you recently participated in a CUPS Memory Study at CMU. A while ago you received an e-mail to participate in a follow up test. We would like to ask you you to complete a quick survey to help us determine why participants were not able to return to complete this follow up study. The survey should take less than a minute to complete, and you will be paid 25 cents for completing the survey. The survey consists of one question. Which of the following reasons best describes why you were unable to return to take the follow up test?
>
> A I no longer wished to participate in the study.
>
> B I was too busy when I got the e-mail for the follow up test.
>
> C I did not see the e-mail for the follow up test until it was too late.
>
> D I was convinced that I would not be able to remember the words/stories that I memorized when I received the e-mail for the follow up test.
>
> E I generally do not participate in follow up studies on mechanical turk.

Discussion It is possible that some participants will choose not to participate in the follow up survey. However, in our case their decision not to participate is valuable information which supports our hypothesis.

4.3.5 Rehearsal Schedules

Each user was assigned one of the rehearsal schedules from Table 4.1. If the participant was assigned to the Aggressive rehearsal schedule then we would send

that participant a reminder to rehearse 1 day after the memorization phase. If that participant successfully completes the first rehearsal phase then we will send that participant another reminder to rehearse 2 days after the memorization phase, and the next reminder would come on day four, etc... The final rehearsal would take place on day 64.

We use the following syntactic pattern to denote a group of participants (Memorization Technique)_(Rehearsal Schedule)_(Number of action-object pairs memorized). For example, a participant in the group mnemonic_aggressive4 refers to a user who was asked to memorize four actions and four objects using the mnemonic techniques we suggested and to rehearse his person-action-object stories following the Aggressive rehearsal schedule from Table 4.1. Because most participants were asked to memorize four actions and four objects we will sometimes drop the number at the end unless the participants was only asked to memorize one or two action-object pairs.

Remark 5. *We use the label "Aggressive" to refer to a rehearsal schedule that we believe will be more challenging for each participant (e.g., the length of time between consecutive rehearsals grows at a faster rate). Similarly, we use the label "Conservative" to refer to a rehearsal schedule that we believe will be less challenging for each participant.*

Schedule	**Multiplier**	**Base**	**Rehearsal Times**
Aggressive	×2	1 Day	1, 2, 4, 8, 16, 32, 64
Conservative	×1.5	1 Day	1, 2.5, 5, 8, 13, 21,32,49,74
Very Conservative	×1.5	0.5 days	0.5, 1.25, 2.4, 4, 6.5, 10,16,24,37,56,85
Heavy Start	×2	1 Day	0.1, 0.5, 1, 2, 4, 8, 16, 32, 64
Heavier Start	×2	30 min	1 hr, 2 hr, 4 hr, 8 hr, 1 day, 2 , 4 , 8, 16 , 32 , 64

Table 4.1: Rehearsal Schedules

4.3.6 List of People, Actions and Objects from the User Study

Here are a list of the people, actions and objects we used in the study.

People: Bill Gates, Bill Clinton, George W Bush, Lebron James, Kobe Bryant, Brad Pitt, Darth Vader, Luke Skywalker, Frodo, Gandalf, Michael Jordan, Tiger Woods, Michael Phelps, Angelina Jolie, Albert Einstein, Oprah Winfrey, Nelson Mandela, Bart Simpson, Homer Simpson, Adolf Hitler, Steve Jobs, Mark Zuckerberg, Justin Timberlake, Jay Z, Beyonce, Kim Jong Un, Joe Biden, Barack Obama, Pope Francis, Rand Paul, Ron Paul, Ben Afleck, Hillary Clinton, Jimmy Fallon

Actions: gnawing, mowing, rowing, oiling, egging, waving, bowing, seizing, stewing, signing, searing, bribing, swallowing, sucking, saving, sipping, tazing, tattooing, drying, dueling, dodging, tugging, taping, nosing, hunting, numbing, inhaling, knifing, nipping, muddying, miming, marrying, mauling, mashing, mugging, moving, mopping, racing, riding, reeling, reaching, raking, lassoing, welding, aligning, leashing, elbowing, juicing, shining, sheering, judging, choking, chipping, coating, concealing, destroying, kissing, aiming, kicking, punching, canning, combing, gluing, cooking, giving, copying, vising, voting, fanning, fuming, firing, fishing, high fiving, batting, burying, plowing, puking, popping, tasting, pulling, climbing, weeping, swimming, stretching, following, paddling, howling, smelling, rolling, waking, jumping

Objects: saw, teacup, hen, ammo, arrow, owl, shoe, cow, hoof, boa, sauce, suit, snow, piranha, chainsaw, shark, tiger, snake, razor-blade, sumo, seal, sock, safe, soap, daisy, toad, dime, tire, dish, duck, dove, ant, onion, wiener, nail, navy, menu, mummy, hammer, mail, microphone, horse, rat, iron, ram, pin, roach, rib, lion, lime, leach, lock, leaf, cheese, jet, chain, chime, gyro, chili, jeep, goose, cat, wagon, igloo, couch, cake, coffee, cab, vase, foot, phone, waffle, fish, bus, patty, bunny, bomb, pill, bush, bike, beehive, puppy, kite, canoe, boar, apple, moon, moose, tepee, ditch, key, shoe, home, toe, nose, cheetah

4.4 Preliminary Results

We say that a participant *survived* through rehearsal i if that participant correctly remembered all of his stories in ≤ 3 attempts during rehearsals $j = 1, \ldots, i$, and we used **Survived** (i) to denote the number of participants who survived through rehearsal i (**Survived** (0)) denotes the number of initial participants). Table 4.2 shows how many participants survived each round. We stress that there are two reasons that a participant might not survive through rehearsal i: (1) the

	Initial	**Survived**(i)										
Condition / i	$(i = 0)$	1	2	3	4	5	6	7	9	9	10	11
mnemonic_heavierstart	73	38	27	26	24	22	22	22	22	22	20	18
mnemonic_heavystart	80	48	41	38	36	36	35	33	27	7		
text_heavystart	100	63	52	51	51	48	46	39	31	7		
mnemonic_aggressive_real	75	50	42	40	38	36	30					
mnemonic_aggressive_2	81	50	42	42	41	38	37	36	33			
mnemonic_aggressive_1	86	64	52	49	49	47	46	45	41			
mnemonic_veryconservative	83	62	52	51	51	49						

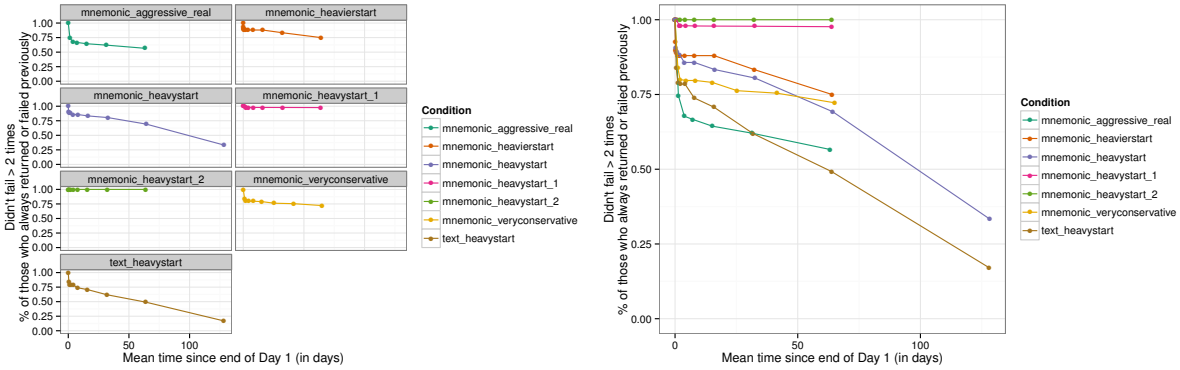
Table 4.2: Survived

participant failed to return to rehearse in a timely manner when we asked, or (2) the participant failed to remember all of his stories in ≤ 3 attempts. Because we used the mnemonic_heavystart and the text_heavystart conditions for our pilot study we have the results from more rehearsals under those conditions. Observe that for the first rehearsal we have the largest drop-off under the heavystart and heavierstart conditions. This may seem paradoxical because we would expect participants in these conditions to have a better chance of remembering their words because they had less time to forget them. Indeed, this is true for participants who returned to complete the first rehearsal. However, many participants in these conditions were not able to return in a timely manner because less time had elapsed.

Total Survival Rate Among Participants who Always Returned Figures 4.3a and 4.3b shows the total survival rate for each completed rehearsal under each study condition for participants who always returned when we asked. More specifically, Figures 4.3a and 4.3b plot the value of

$$\textbf{Survived}\,(i)\,/\left(\textbf{Survived}\,(t)+\sum_{j=1}^{t}\textbf{Failed}\,(j)\right),$$

where t denotes the total number of rehearsals and $\textbf{Failed}\,(j)$ denotes the number of participants who survived through rehearsals $1,\ldots,j-1$ and did not remember all of their words on rehearsal j in ≤ 3 attempts. We use $\textbf{Time}\,(i)$ to denote the time of the i'th rehearsal. Observe that each of the curves in these figures is monotonically decreasing. This is because a participant who did not survive round i will also be counted as a participant who did not survive round j for each

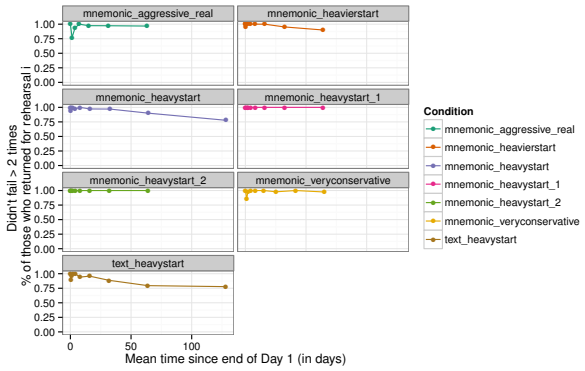


(a) Faceted (b) Together

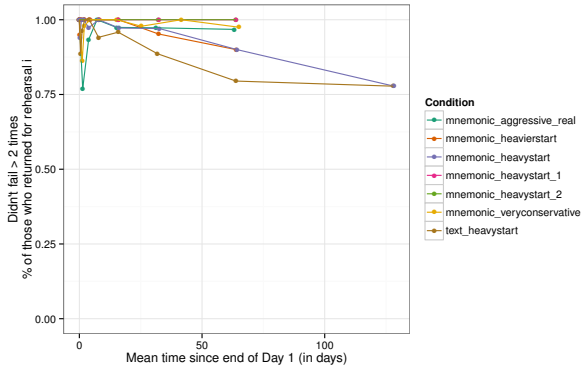
Figure 4.3: Total Survival with Failures Carried Forward $\left(\frac{\textbf{Survived}(i)}{\textbf{Survived}(t)+\sum_{j=1}^{t}\textbf{Failed}(j)}\right)$ vs **Time** (i).

$j > i$. While we include this data for completeness we emphasize that this view is overly pessimistic. For example, consider a participant who correctly remembered his stories during the first three rehearsals, but was not able to return for the fourth rehearsal (e.g., because he went on vacation). The results of this participant would be dropped. However, if the same participant had failed during round three then his results would be included because we would not have asked him to return for the fourth rehearsal while he was on vacation. Suppose that participants who return for rehearsal one succeed with probability 0.99 and that participants who return for rehearsal $i > 1$ succeed with probability 1. If participants always returned to rehearse when we asked them to then the survival rate after rehearsal i would be 99% for all $i > 0$. However, if participants are not able to return to complete each rehearsal phase independently with probability $p > 0$ then the total survival rate among participants who always returned will always tend to 0.

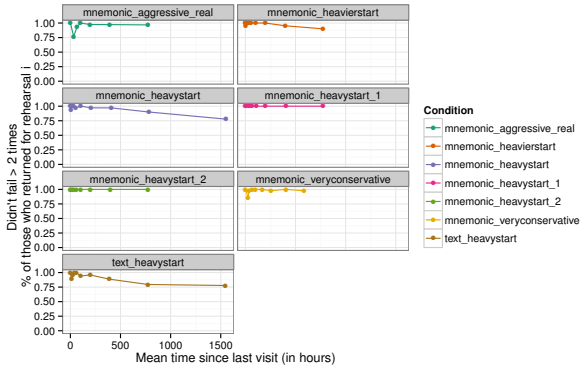
Conditional Survival Probability Figures 4.4a and 4.4b show the conditional probability of survival (e.g., % of those who participants who correctly remembered all of their stories in ≤ 3 attempts on the i'th rehearsal conditioned on the event that the participant survived rounds $j = 1, \ldots, i-1$ and returned for rehearsal i). More formally, these Figures plot the value of **Survived** (i) /**Returned** (i), where **Returned** (i) counts the number of participants who survived rounds $j = 1, \ldots, i-1$ and returned for rehearsal i. Figures 4.4c and 4.4d show the same data with a



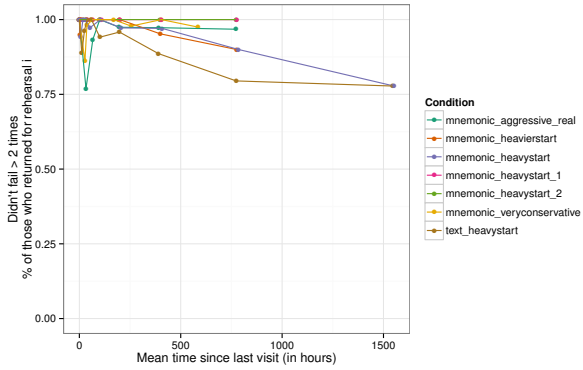
(a) Faceted. Mean Time Since Memorization.



(b) Together. Mean Time Since Memorization.



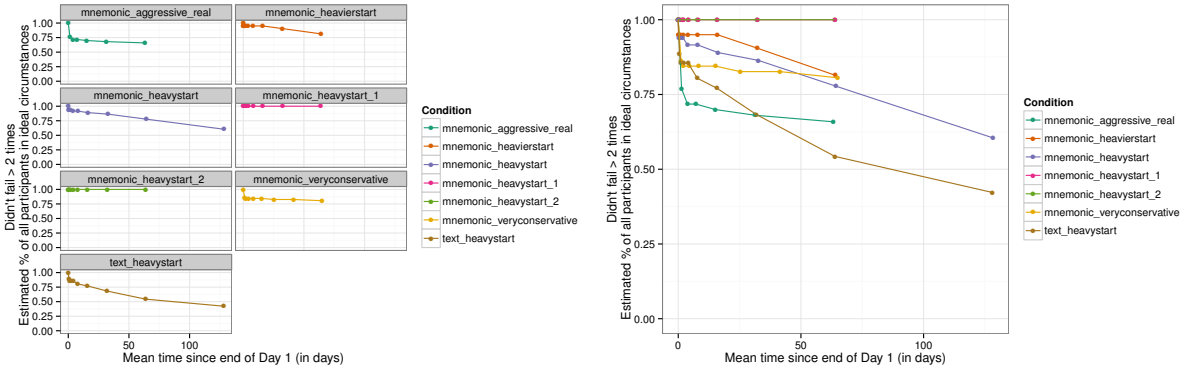
(c) Faceted. Mean Time Since Last Rehearsal.



(d) Together. Mean Time Since Last Rehearsal.

Figure 4.4: Conditional Survival: **Survived** (i) /**Returned** (i) vs **Time** (i).

different x-axis (e.g., mean time since last visit instead of mean time since first visit). Notice that these curves are not necessarily monotonic. For example, in the mnemonic_aggressive condition 25% of users failed to remember all four of their stories during the first rehearsal. However, every participant who survived to rehearsal three also survived to rehearsal four in the mnemonic_aggressive condition. This illustrates the advantage of having several immediate rehearsals. In the heavystart conditions the last rehearsal on day 64 was the most difficult for participants.



(a) Faceted. (b) Together.

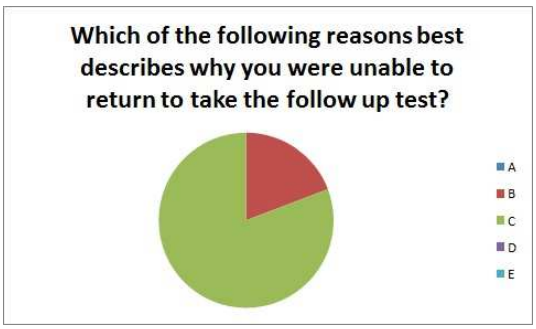
Figure 4.5: **EstimatedSurvival** (i) vs **Time** (i)

Estimated True Total Survival Rate We can use our conditional success probabilities to estimate what the true survival rate would have been under ideal circumstances (e.g., all participants are always available to return to rehearse when we asked them). Our results are shown in Figures 4.5a and 4.5b. We use the estimate

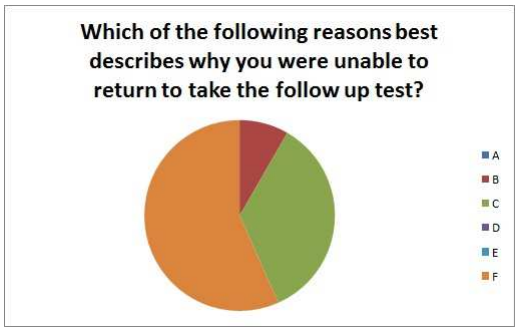
$$\textbf{EstimatedSurvival}\,(i) = \prod_{j=1}^{i} \frac{\textbf{Survived}\,(j)}{\textbf{Returned}(j)}\,,$$

where $\frac{\textbf{Survived}(j)}{\textbf{Returned}(j)}$ denotes our empirical estimate of the conditional probability that a participant will survive round j given that the participant survived all previous rounds and returned for rehearsal j.

Survey Results We surveyed 61 participants who did not return to complete their first rehearsal to ask them why they were not able to return. The results from our survey are presented in Figures 4.6a and 4.6b. The results from our survey strongly supports our hypothesis that the primary reason that participants do not return is because they were too busy, because they did not get our follow up message in time or because they do not interested in interacting with us outside of the initial Mechanical Turk Hit, and not because they were convinced that they would not remember the story.



(a) Participants who Completed the Survey



(b) All Invited Survey Participants

Figure 4.6: Survey: Which of the following reasons best describes why you were unable to return to take the follow up test?
A: I no longer wished to participate in the study.
B: I was too busy when I got the e-mail for the follow up test.
C: I did not see the e-mail for the follow up test until it was too late.
D: I was convinced that I would not be able to remember the words/stories that I memorized when I received the e-mail for the follow up test.
E: I generally do not participate in follow up studies on Mechanical Turk.
F: Participant did not respond to survey.

Fun We had several participants e-mail us to tell us how much fun they were having memorizing person-action-object stories. The results from our survey are also consistent with the hypothesis that memorizing person-action-object stories is fun (e.g., no participants said that they no longer wished to participate in the study).

4.4.1 Discussion

Mnemonic Advantage. Our results strongly support the hypothesis that $\sigma_{mnemonic} > \sigma_{non-mnemonic}$. Participants consistently did better in the mnemonic_heavystart condition than in the text_heavystart condition. For example, compare the estimated survival rates for mnemonic_heavystart and text_heavystart in Figure 4.5a or compare the conditional survival rates for mnemonic mnemonic_heavystart and text_heavystart in Figure 4.4a. Even the pessimistic total survival rates shown in Figure 4.3a support this hypothesis.

Benefit of Several Early Rehearsals. Participants did very well in conditions which involve several early rehearsals like the mnemonic_heavystart and the mnemonic_heavierstart conditions. In the mnemonic_veryconservative condition a few participants struggled during the first rehearsal, but have been perfect after that. In the mnemonic_aggressive condition participants also struggled most during the first rehearsal.

Interference. While a few participants struggled under the mnemonic_heavystart condition this was not the case when we only asked participants to memorize one or two person-action object stories. This is likely because participants had less energy to devote to memorizing the third and fourth story in the mnemonic_heavystart condition. Our results do not mean that users are incapable of remembering multiple stories. In fact, most participants in the mnemonic_heavystart were able to remember all four of their stories. However, our results do indicate that it may be prudent to either adopt a rehearsal schedule with many early rehearsals whenever the user is memorizing multiple stories at once, or space out the memorization process so that users are not memorizing multiple stories at once.

Chapter 5

Password Composition Policies: A Defense Against Online Attacks

5.1 Introduction

Imagine a web surfer, an online shopper, or a reviewer in a prominent CS and Economics conference who logs on for the first time to a server; so that she can sign up for some service, place a shopping order, or view a list of assigned papers. Such a user registers on the server by choosing a username and picking a password. Naturally, our user's first attempt at picking a password is her favorite combination '`123456`', which the server declines. She then has to pick a password that follows certain guidelines: of suitable length, involving lower- and upper-case letters, with numbers or special characters, etc. Such *password composition policies* defend against the "first line" of attack – guessing attacks by uninformed attackers (attackers with no previous knowledge of the user whose account they are trying to break into).

Password composition policies are a necessity because — without them — user-selected passwords are predictable. Indeed, many unrestricted users would select simple passwords like '`123456`', '`password`' and '`letmein`' [66]. Furthermore, this issue is of great importance to today's economy. Passwords are commonly used in electronic commerce to protect financial assets. In fact, the passwords themselves have financial value. Symantec reported that compromised passwords are sold for between $4 and $30 on the black market [79], and a 2004 Gartner case study [158] estimated that it cost a large firm over $17 per password-reset call. Nevertheless, existing password composition policies are typically not principled, and do not

necessarily result in less common passwords. For example, studies show that users respond to restrictions in predictable ways [97], or pick weaker passwords due to user-fatigue [56, 102].

In this chapter, we initiate the *algorithmic* study of password composition policies. Such policies restrict the space of passwords to a subset of allowed passwords, and force each user to pick a password in this subset. Thus, n users induce a distribution over passwords where for a password w, $\Pr[w] = \frac{1}{n}\left|\{i : i \text{ picks } w\}\right|$. By declaring different subsets of allowed passwords, different password composition policies induce different distributions. Our work formalizes and addresses the algorithmic problem a server administrator faces when designing a password composition policy; we ask:

> *In what settings can the information about the users' preferences over passwords allow us to design a password composition policy that is* guaranteed *to induce a password distribution as close to uniform as possible?*

We wish to stress at this point that in this chapter we do not take a cryptographic approach to the problem: we do not design a protocol aimed at amplifying a password's strength, nor do we rely on standard cryptographic assumptions or techniques in designing our password composition policies. Single-factor authentication does not defend against an attacker who learns about the most probable password from an external source. Furthermore, because password systems often allow users multiple attempts in entering their password, an attacker can make a small number of guesses with impunity. Therefore, we instead focus on the design and analysis of algorithms for optimizing the password composition policy's induced distribution over passwords, and in our theoretical results compare the performance of our algorithm to the optimal policy among exponentially many potential policies in the *worst case*.

5.1.1 Our Model

We study the algorithmic problem of optimizing password composition policies along multiple dimensions: the goal, the user model, and the policy structure.

Goal. We focus on designing a policy that maximizes the minimum-entropy of the resulting password distribution. Specifically, we assume the server deals with n users, each picking a password from some space of passwords $\mathcal{P}$ that respects the

server's password composition policy. These n passwords form a distribution over the domain of all allowed passwords and our goal is to minimize the probability of the most likely password. This is a natural goal (see Section 5.7), as opposed to maximizing the Shannon-entropy of the distribution, which for example is still high even if half the people choose the same password and the other half choose a password uniformly at random from $\mathcal{P}$. From a security standpoint, the minimum entropy represents the fraction of accounts that could be compromised in one guess. For example, an adversary would be able to crack 0.9% of RockYou passwords [92] with only one guess. Alternatively, should the attacker attempt to break into only one account, the minimum entropy represents the likelihood that the account is compromised on the first guess. We also consider a slightly stronger goal of minimizing the fraction of accounts that could be compromised using k guesses, that is, the overall probability of the k most likely passwords [45].

User model. We consider two models for how users select passwords when presented with a password composition policy.

In the *ranking model*, each user has an implicit ranking over passwords, from the most preferred to the least preferred. Given a password policy, each user selects the highest-ranking password among those allowed by the policy. There is a distribution over the space of rankings that determines the fraction of users with each possible ranking. Note that for any password composition policy, such a distribution over rankings induces a distribution over the most preferred *allowed* passwords.

In the *normalization model*, there is a distribution $\mathcal{D}$ over the space of all passwords. This distribution tells us the likelihood that an unrestricted user would select a given password. Given a password composition policy, $\mathcal{D}$ induces a new distribution over the allowed passwords (which can be obtained by normalizing the probabilities under $\mathcal{D}$ of the allowed passwords). When we ban a password the fraction of users that prefer each allowed password grows; the natural interpretation is that users who preferred an allowed password still use that password, but users who preferred a banned password are redistributed among the allowed passwords according to the induced distribution.

As we show, the normalization model is strictly more restrictive than the ranking model: any distribution in the normalization model can be simulated in the ranking model, but there exist hardness results for the ranking model that do not hold for the normalization model.

Policy structure. We consider the best policy that is restricted to manipulation of a

given set of *rules* — each rule is simply a predefined subset of potential passwords. These rules are given to us as part of the problem (see Section 5.7 for a discussion of this point). If we interpret a rule as a subset of banned passwords (e.g., passwords shorter than seven characters), its complement (e.g., passwords of at least seven characters) can be interpreted as a subset of allowed passwords. As such, when we take the union of rules we get either a set of banned passwords (*negative rules*) or allowed passwords (*positive rules*); this is our password composition policy. While the distinction between the two cases may at first seem a mere technicality, it is in fact quite significant due to the following observation. If we ban the union of rules then in order to ban a password that was picked by too many users, we may ban any rule that contains this password. In contrast, if we allow a union of rules then in order to ban this password we must not allow *any* rule that contains it. In other words, when our goal is to discard a password in the negative rules setting, we have multiple ways to do so. When our goal is to discard a password in the positive rules setting, we have only one way to do so — excluding all rules that allow this password. As we shall see, this seemingly small difference leads to a clear separation between the two scenarios in terms of the complexity of designing optimal policies.

We pay special attention to the case where each password has its own singleton rule. In this setting, a policy can be interpreted as a "blacklist" of banned passwords that do not necessarily share common characteristics. Note that when each password has its own singleton rule, it does not matter whether these rules are positive or negative.

5.1.2 Our Results

As we noted above, a password composition policy induces a distribution over most preferred passwords (in both user models). We study algorithms that *sample* these distributions — algorithms that repeatedly query random users and ask them to choose a password constrained by some policy, and then output the a good policy for the empirical sample of users. Our goal is therefore twofold: (i) to show that having sufficiently many samples (i.e., sufficiently many users queried) guarantees that w.h.p the best policy for the empirical sample is good for all users; and (ii) exhibit algorithms that find an optimal (or close-to-optimal) policy for a given sample. Clearly, we want our sample size to be "small". In particular, since the size of the space of all passwords $\mathcal{P}$ — which we denote by N — is typically very large (e.g., $\mathcal{P}$ can include all passwords that are no longer than 32 ASCII

Table 5.1: Summary of Complexity Results.

	Ranking Model		**Normalization Model**	
	Constant k	**Large** k	**Constant** k	**Large** k
Singleton rules	P	NP-Hard (Thm 13) APX-Hard w. UGC (Thm 14)	P	P (Thm 16)
Positive rules	P (Thm 12)	NP-Hard	P	NP-Hard (Thm 18)
Negative rules	$n^{1/3}$-approx is NP-hard (Thm 15)	NP-Hard	NP-Hard (Thm 17)	NP-Hard

characters), we wish to get a bound on the sample size that is independent of N.

For the ease of exposition, we discuss goal (ii) before goal (i). I.e., we first (Sections 5.3 and 5.4) study the problem in a simpler setting where the preferences of all users are given to us as input; and only then (Section 5.5) we introduce an algorithm that samples users' preferences. Also for the ease of exposition, we first discuss algorithms where $\mathcal{P}$ is a part of the input, so they are allowed to run in time polynomial in N. This is motivated by the fact that computational complexity of problems in this setting informs their study in the sampling setting — it is hopeless to design efficient sampling algorithms for problems that are computationally hard. (Efficient sampling algorithms are applicable only to computationally tractable problems.)

Table 5.1 summarizes our complexity results. The parameter k refers to our optimization target: minimizing the likelihood of the k most likely passwords. Some results are direct corollaries of others — using the fact that singleton rules are a special case of positive rules and the fact that the normalization model is a special case of the ranking model (see Section 5.2). Looking at the table one immediately notices a clear separation between negative rules and positive rules: optimization using the latter is much easier.

We therefore focus on positive rules in our attempt to design an efficient sampling algorithm. Our main result is the best one could hope for in this setting. We

design an algorithm that works in the more general ranking model, and finds a policy whose entropy is ϵ-close to optimal with probability $1 - \delta$, for any given $\epsilon, \delta > 0$. The required number of samples is polynomial in $1/\epsilon$, $\log(1/\delta)$, and the number of positive rules m. We can assume that m is small, because each rule corresponds to a subset of passwords that can be concisely described to users.

These results can be applied in a practical setting, and we show this through simulated sampling experiments using natural rules and a large dataset of real passwords. The experimental results provide evidence for the difficulty of the negative rules setting: we search all combinations of rules to find the optimal policy and then attempt to discover this policy by making decisions both randomly and with a heuristic. In the negative rules setting, neither approach succeeded at finding the optimal policy after hundreds of iterations at various sample sizes, and average-case performance did not improve with sample size. In the positive rules setting, the average-case performance of our efficient algorithm improved with sample size and, with a moderate sample size, found policies that were either optimal or very close to optimal.

5.1.3 Related Work

It has been repeatedly demonstrated that users tend to select easily guessable passwords [39, 66, 92] and NIST recommends that organizations "should also ensure that other trivial passwords cannot be set," to thwart potential attackers [133]. Unfortunately, this task is more difficult than it might appear at first. Policies were initially developed without empirical data to support them, since such data was not available to policy designers [50]. When hackers leaked the RockYou dataset to the Internet, both researchers (and attackers) suddenly had access to password data, leading to many insights into true passwords [156]. However, recent research analyzing leaked datasets from non-English speakers, notably Hebrew and Chinese-language websites, shows that trivial password choices can vary between contexts, making a simple blacklist approach ineffective [40]. This means that, depending on the context, a policy based on leaked password data might provide no security guarantee, and it has ethical issues as well.

To combat this issue, researchers have turned to a sampling approach. Bonneau 2012 added a system for sampling to the Yahoo! password infrastructure. This system allows one to gain empirical data about the frequency distribution of passwords without revealing the passwords themselves. Such approaches provide a way of gathering empirical data about passwords while maintaining the

anonymity of users. Our algorithms could be used in conjunction with such an infrastructure to optimize policies.

Komanduri et al. 2012 studied the effectiveness of several basic password composition policies by using Amazon's Mechanical Turk to conduct a large scale user study. They found that people often respond to restrictions in predictable ways (e.g., if the password needs to contain a capital letter users might tend to capitalize the first letter of a password) and provide very general recommendations for password composition policies. However, no theoretical model has been proposed for studying the password composition problem.

Schechter et al. 2010 suggest using a popularity oracle to prevent individual passwords that have been used too frequently from being selected by new users. They also proposed using the count-min sketch data structure [57] to build such a popularity oracle. Malone and Maher 2012 suggest a similar system using a Metropolis-Hastings scheme to force an approximately uniform distribution on passwords. Usability results on the effectiveness of dictionary checks [97] suggest that such policies would be very frustrating since the policy is hidden from users behind an oracle. In contrast, we seek to construct optimal policies from combinations of rules that are visible to the user and can be described in natural language.

This consideration of users is important to electronic commerce, even where security is concerned. Florencio and Herley 2010 studied the economic factors that drive institutions to adopt strict password composition policies and find that they often value the user experience over security. An e-mail provider like Yahoo! might adopt simple composition policies because a frustrated user could easily switch to Gmail, while universities are free to adopt strict policies because users cannot switch easily.

5.2 A Model of Password Composition Policies

We use $\mathcal{P}$ to denote the space of all possible passwords. $N = |\mathcal{P}|$ is used to denote the total number of passwords. We denote the number of users by n.

A password composition policy may be specified in terms of rules. A rule is a subset of passwords $R \subseteq \mathcal{P}$ (e.g., the set of all passwords with more than seven characters). We use $R_1, ..., R_m$ to denote a list of rules that may be active or inactive. We consider two schemes.

- *Positive Rules:* A password w is allowed if and only if it is allowed by some active positive rule. Formally, a password composition policy $\mathcal{A}_S = \bigcup_{i \in S} R_i$ is specified by a set $S \subseteq [m] = \{1, ..., m\}$ of active rules. In this setting rules should consist of sets of passwords which we expect to be strong (e.g., R_i might be the set of all passwords longer than 10 characters, or the set of all passwords that use both upper and lowercase letters, or the set of all passwords that do not include a dictionary word).

- *Negative Rules:* A password w is allowed if and only if it is not contained in any active negative rule. Formally, a solution $\mathcal{A}_S = \{w \in \mathcal{P} \mid w \notin \bigcup_{i \in S} R_i\}$ is given by a subset $S \subseteq [m]$ of active rules. A negative rule should consist of passwords that we expect to be weak (e.g., R_i might be the set of all passwords without an uppercase letter, or the set of all passwords shorter than 6 characters, or the set of all passwords that include a dictionary word).

We also consider the special case of *singleton rules*, where our rules are $\{w_1\}, \ldots, \{w_N\}$. Equivalently, we are allowed to ban or allow any individual password.

We use $\Pr[w \mid \mathcal{A}]$ to denote the probability of a password w given composition policy $\mathcal{A}$. For $w \notin \mathcal{A}$ we have $\Pr[w \mid \mathcal{A}] = 0$. Given a set $W \subseteq \mathcal{A}$ we will also use $\Pr[W \mid \mathcal{A}] = \sum_{w \in W} \Pr[w \mid \mathcal{A}]$. We use $p(k, \mathcal{A}) = \max_{W \subseteq \mathcal{A}: |W| = k} \Pr[W \mid \mathcal{A}]$ to denote the probability of the k most popular passwords. Intuitively, $p(k, \mathcal{A})$ represents the probability that an adversary can successfully guess a password using k attempts. To avoid cumbersome notation we sometimes use $p_1 = p(1, \mathcal{A})$ to denote the probability of the most popular password. Similarly, we use p_2 (resp., p_k) to denote the probability of the second (resp., k'th) most popular password.

We consider two user models that determine how users choose passwords under a given password composition policy.

- *The ranking model:* A ranking is simply a permutation of $\mathcal{P}$, which represents a user's password preferences. It can be represented using an ordered list $\ell_i = w_{1,i}, ..., w_{N,i}$; user i prefers password $w_{j,i}$ to $w_{j+1,i}$ for all j. The ranking ℓ_i naturally tells us which password i will pick under any composition policy $\mathcal{A}$. Specifically, i will use password $w_{\mathcal{A},i} = w_{j,i}$ where $j = \text{argmin}\{t : w_{t,i} \in \mathcal{A}\}$. Given a distribution $\mathcal{D}$ over rankings, we have

$$\Pr[w \mid \mathcal{A}] = \Pr_{\ell_i \sim \mathcal{D}}[w_{\mathcal{A},i} = w] .$$

- *The normalization model:* Let $\mathcal{D}$ be an initial distribution over $\mathcal{P}$, and let $\Pr[w] = \Pr_{x\sim\mathcal{D}}[w = x]$. If we select the composition policy $\mathcal{A}$ then the probabilities of all $w \in \mathcal{A}$ are simply re-normalized so that

$$\forall w \in \mathcal{P}, \mathcal{A} \subseteq \mathcal{P}, \Pr[w|\mathcal{A}] = \frac{\Pr[w]}{\Pr[\mathcal{A}]} .$$

Clearly it holds for both models that the probability of an allowed password monotonically increases as one bans more passwords. Formally, for all $w \in \mathcal{A}$ and $B \subseteq \mathcal{P}$ such that $w \notin B$ we have

$$\Pr[w|\mathcal{A}] \leq \Pr[w|\mathcal{A}\backslash B] . \tag{5.1}$$

Another important observation is that for our purposes the ranking model is more general than the normalization model. Indeed, we argue that a distribution $\mathcal{D}$ over passwords in the normalization model induces an equivalent distribution over rankings. To generate the most highly ranked password, draw a password w_1 from $\mathcal{D}$. Next, let $\mathcal{A}_1 = \mathcal{P} \setminus \{w_1\}$, and draw the next most preferred password w_2, where $w_2 = w$ with probability $\Pr[w|\mathcal{A}_1]$. In the following round we ban w_2 to obtain a policy $\mathcal{A}_2$, and so on, until all passwords have been banned.

Given $k \in \mathbb{N}$, our goal is to find $S \subseteq [m]$ such that $p(k, \mathcal{A}_S) \leq p(k, \mathcal{A}_{S'})$ for all $S' \subseteq [m]$. When $k = 1$ this goal is equivalent to maximizing the minimum entropy. If $p(k, \mathcal{A}_S) \leq c \cdot p(k, \mathcal{A}_{S'}) + \epsilon$ for all $S' \subseteq [m]$ then we say that S is a (c, ϵ)-approximation. To simplify notation we sometimes use c-approximation instead of $(c, 0)$-approximation.

5.3 Ranking Model: Complexity Results

In this section we consider the complexity of finding the optimal password composition policy in the more general ranking model when the organization is given complete information about users' preferences. Specifically, the organization is given the rankings $\ell_1, ..., \ell_n$ of every user.

Our first result is for the positive rules setting. Given positive rules $R_1, ..., R_m$ we show that $p(k, \mathcal{A}_S)$ can be computed efficiently for constant values of k (see Theorem 12). In fact, for the special case $k = 1$ we present a very simple algorithm that suffices. Both algorithms can be easily extended to the less general normalization model. Our algorithms are based on three simple ideas: (1) Reduced Preference

Lists — each preference list ℓ_i can be efficiently reduced to a short (length $\leq m$) preference list $\hat{\ell}_i$. (2) Guess and Check — start by guessing the 'structure' of the optimal solution and find the resulting solution. (3) Iterative Elimination — find the most popular password w and eliminate all positive rules that contain w. Our sampling algorithms are based on the same core ideas.

Unfortunately, the picture is different in the negative rules even when k is a constant. Given negative rules $R_1, ..., R_m$ we show that it is hard to even $n^{1/3}$-approximate $p(1, \mathcal{A}_S)$. Also, for non-constant values of k we show that it is hard to compute $p(k, \mathcal{A}_S)$ in the singleton rules setting, which immediately implies hardness in both the positive rules setting and in the negative rules setting. Given a stronger complexity assumption known as the Unique Games Conjecture [98] it is also hard to c_0-approximate $p(k, \mathcal{A}_S)$ in the singleton rules setting for some constant c_0. However, our hardness results do not rule out the possibility of a c-approximation for a larger constant c.

5.3.1 Positive Rules: Efficient Algorithm for Constant k

We first show that $p(k, \mathcal{A}_S)$ can be computed efficiently for constant values of k in the positive rules setting. In this section the organization is given positive rules $R_1, ..., R_m$ as well as preference lists $\ell_1, ..., \ell_n$. We assume that the organization can efficiently query the preference lists (e.g., given $S \subseteq [m]$ the organization can efficiently find $\ell_i(\mathcal{A}_S)$ — user i's preferred password given policy $\mathcal{A}_S$).

We elaborate on the key algorithmic ideas listed above. First, we can efficiently reduce each preference list ℓ_i to a list $\hat{\ell}_i$ of at most m passwords (Claim 4). While the reduced list $\hat{\ell}_i$ is much shorter than ℓ_i it is still sufficient to determine user i's preferred password given policy $\mathcal{A}_S$ for any $S \subseteq [m]$. We use $\hat{\mathcal{P}}$ to denote the reduced space of potential passwords.

Claim 4. *Algorithm 5.1 makes at most m queries to ℓ and m^2 membership queries and outputs a reduced preference list $\hat{\ell}$ over at most m passwords such that for every $S \subseteq [m]$ it holds that $\hat{\ell}(\mathcal{A}_S) = \ell(\mathcal{A}_S)$.*

Proof. Clearly, the algorithm's main loop iterates at most m times because for each i we eliminate at least one rule (e.g., $|S_{i+1}| < |S_i|$), so the bound on queries and the length of $\hat{\ell}$ are immediate. (Because we assume that we can query ℓ efficiently Algorithm 5.1 is also efficient.) By construction we have $\hat{\ell}(S_i) = \ell(S_i)$ for each S_i. Fix any $S \subseteq [m]$. Let S_i be such that $S \subseteq S_i$ yet $S \not\subseteq S_{i+1}$ and let w_i be the most

Algorithm 5.1 Reduce

Input:
Preference List: ℓ
Positive Rules: $R_1, ..., R_m$
Initialize: $i \leftarrow 0$, $S_0 \leftarrow [m]$, $\hat{\ell} \leftarrow$ empty ranking.
while $S_i \neq \emptyset$ **do**
  Let w be $\ell(\mathcal{A}_{S_i})$.
  $\hat{\ell} \leftarrow \langle \hat{\ell}, w \rangle$ ▷ 'Append' the current most preferred password to $\hat{\ell}$
  $S_{i+1} \leftarrow S_i \setminus \left\{ j \,\middle|\, w \in R_j \right\}$ ▷ Deactivate all rules that contain w
  $i \leftarrow i + 1$
return $\hat{\ell}$

preferred word in ℓ out of all words in $\bigcup_{j \in S_i} R_j$. If it is the case that $w_i \in \bigcup_{j \in S} R_j$, then w_i is the most preferred word in S too and we're done. Otherwise, $w_i \in \bigcup_{j \in S_i \setminus S} R_j$ which means that removing the set $\{j \in S_i \ : \ w_i \in R_j\}$ creates a set S_{i+1} s.t. $S \subseteq S_{i+1}$, contradiction. □

5.3.2 Special Case $k = 1$

For the special case $k = 1$ the simple algorithm IterativeElimination (Algorithm 5.2) suffices. The basic idea is very simple: iteratively eliminate the most popular password w by deactivating all positive rules that contain w. We repeat this process until no passwords remain. We claim that one of the solutions along the way was the optimal solution.

Algorithm 5.2 IterativeElimination

Input:
Preference Lists: $\ell_1, ..., \ell_n$
Positive Rules: $R_1, ..., R_m \subseteq \mathcal{P}$
Initialize: $S_0 \leftarrow [m]$, $i \leftarrow 0$
while $S_i \neq \emptyset$ **do**
  $w(S_i) \leftarrow \arg\max \{\Pr[w \mid \mathcal{A}_{S_i}] \mid w \in \mathcal{A}_{S_i}\}$ ▷ $w(S_i)$ is most popular allowed pwd
  $S_{i+1} \leftarrow S_i \setminus \left\{ j \,\middle|\, w(S_i) \in R_j \right\}$ ▷ Deactivate all rules that contain $w(S_i)$
  $i \leftarrow i + 1$
return S_{i^*} where $i^* \leftarrow \arg\min_i p(1, \mathcal{A}_{S_i})$

Theorem 11. *Algorithm 5.2 outputs a set of positive rules $S \subseteq [m]$ such that*

$$\forall S' \subseteq [m]\ ,\ \ p\left(1, \mathcal{A}_S\right) \leq p\left(1, \mathcal{A}_{S'}\right)\ .$$

Proof. Let T denote the optimal policy. Clearly if $T = [m]$ then our algorithm returns $S^* = T$ because that is the first set we try. Otherwise, $T \subsetneq [m]$. Let S be the last set our algorithm considers that has the property that $T \subseteq S$. Again, if $T = S$, our algorithm returns S. Let $w(T)$ be the most popular word in $\mathcal{A}_T$, and because of optimality $\Pr[w(T) \mid \mathcal{A}_T] \leq \Pr[w(S) \mid \mathcal{A}_S]$.

Now, because we modify S to not contain T in the next iteration, then the most popular word in S, $w(S)$ has to belong to some rule R_j where $j \in T$. Therefore $w(S) \in \bigcup_{j \in T} R_j$, and by the definition, the most popular word in $\mathcal{A}_T$ satisfies $\Pr[w(T) \mid \mathcal{A}_T] \geq \Pr[w(S) \mid \mathcal{A}_T]$.

But observe, because $w(S) \in \bigcup_{j \in T} R_j$, we must have that $w(S)$ is at least as popular in T. Indeed, if ℓ is a preference list where we disallowed $\mathcal{P} \setminus \bigcup_{j \in S} R_j$ and the most preferred word is $w(S)$, then as long as we disallow more words but keep allowing $w(S)$ the word $w(S)$ remains at the top of the list. Therefore, $\Pr[w(S) \mid \mathcal{A}_T] \geq \Pr[w(S) \mid \mathcal{A}_S]$. Combining together all inequalities we get $\Pr[w(T) \mid \mathcal{A}_T] = \Pr[w(S) \mid \mathcal{A}_S]$, which means our algorithm returns $S^* = S$. □

5.3.3 The General Case

We now present an algorithm "Guess and Check" to find the optimal password composition policy for any constant value of k. Our algorithm starts by guessing what the optimal solution looks like (e.g., what the k most popular passwords will be in the optimal solution and what the probability of the k'th most popular password is). There are at most $(mn)^{O(k)}$ potential solutions to brute-force try. As we show, for each candidate solution, it is easy to figure out which sets must be eliminated using the iterative elimination idea behind Algorithm 5.2.

Theorem 12. *Algorithm 5.3 runs in time polynomial in n^k, m^k and outputs a set of positive rules $S \subseteq [m]$ of positive rules such that*

$$p\left(k, \mathcal{A}_S\right) \leq p\left(k, \mathcal{A}_{S'}\right)$$

for every other set $S' \subseteq [m]$.

Algorithm 5.3 GuessAndCheck

Input:
Preference Lists: $\ell_1, ..., \ell_n$
Positive Rules: $R_1, ..., R_m \subseteq \mathcal{P}$
Integer k
Initialize: $Candidates \leftarrow \emptyset$ ▷ Candidate Solutions
for $i = 1 \rightarrow n$ **do**
  $\hat{\ell}_i \leftarrow Reduce\,(\ell_i, R_1, ..., R_m)$
$\hat{\mathcal{P}} \leftarrow \bigcup_{i=1}^{n} \hat{\ell}_i$. ▷ Reduced Password Space
for all (G, p) with $G \subseteq \hat{\mathcal{P}}$ s.t. $|G| = k$ and $p \in \{1/n, 2/n, ..., 1\}$ **do**
  $S_{G,p} \leftarrow [m]$
  while $S_{G,p} \neq \emptyset$ and $\exists w \in \left(\hat{\mathcal{P}} \setminus G\right) \cap \mathcal{A}_{S_{G,p}}$ s.t $\Pr\left[w \,\middle|\, \mathcal{A}_{S_{G,p}}\right] > p$ **do**
    $S_{G,p} \leftarrow S_{G,p} \setminus \{j \mid w \in R_j\}$ ▷ Ban w because it is inconsistent with guess
  if $\Pr\left[w \,\middle|\, \mathcal{A}_{S_{G,p}}\right] \leq p$ for all $w \in \left(\mathcal{A}_{S_{G,p}} \setminus G\right)$ **then**
    $Candidates \leftarrow Candidates \cup \{S_{G,p}\}$
return $\arg\min_{(G,p) \in Candidates} p\left(k, \mathcal{A}_{S_{G,p}}\right)$

Proof. It is evident that the running time of the algorithm is poly(n^k, m^k) since we only have $O((nm)^k)$ potential solutions to try.

Let $\mathcal{A}_{S^*}$ denote an optimal solution and let G^* denote the k most popular passwords in this solution. Suppose we start with the correct guess ($G = G^*$ and p is the probability of the k'th most popular password), then we claim that our algorithm must produce the optimal solution. In particular, we maintain the invariant that $\mathcal{A}_{S^*} \subseteq \mathcal{A}_{S_{G,p}}$ until we converge to the optimal solution. Clearly, this is true initially — before we have eliminated any passwords.

Suppose that the invariant holds and that our algorithm bans a password $w \in \mathcal{P} \setminus G$ by deactivating all rules in $S_{G,p}$ that contain w. Then by the definition of our algorithm we must have $\Pr\left[w \,\middle|\, \mathcal{A}_{S_{G,p}}\right] > p$. If $w \in \mathcal{A}_{S^*}$ then by Equation (5.1) we have

$$\Pr\left[w \,|\, \mathcal{A}_{S^*}\right] \geq \Pr\left[w \,\middle|\, \mathcal{A}_{S_{G,p}}\right] > p\,,$$

which contradicts the choice of G. Therefore $w \notin \mathcal{A}_{S^*}$, so all rules that contain it are deactivated in $\mathcal{A}_{S^*}$ and the invariant still holds. By definition Algorithm 5.3 terminates when every password $w \in A_{S_{G,p}} \setminus G$ has probability at most p. Because

our invariant still holds we can apply Equation (5.1) again to get

$$\Pr\left[G \,\middle|\, \mathcal{A}_{S_{G,p}}\right] \leq \Pr\left[G \,|\, \mathcal{A}_{S^*}\right] = p\left(k, \mathcal{A}_{S^*}\right) \ .$$

Hence, $A_{S_{G,p}}$ is an optimal solution. □

5.3.4 Singleton Rules: Hardness for Large k

Now we turn our attention to the problem of optimizing $p\left(k, \mathcal{A}_S\right)$ for large values of k. Theorem 13 says that unless $P = NP$ no polynomial time algorithm can compute $p\left(k, \mathcal{A}_S\right)$ even with singleton rules. If we are willing to make the Unique Games Conjecture (UGC) [98] then it is hard to even c_0-approximate $p\left(k, \mathcal{A}_S\right)$ for some constant c_0. These results immediately imply hardness in both the positive and negative rules setting because these settings are a generalization of the singleton rules setting.

Theorem 13. *Unless $P = NP$ there is no* poly(k, n, N)*-algorithm that gets as input an arbitrary set of n preference-lists $\ell_1, ..., \ell_n$ over $\mathcal{P}$ and an integer k, and outputs the optimal $p(k, \mathcal{A})$ in the singleton rules setting.*

Proof. We prove the theorem using a reduction from the Vertex-Cover problem. Given a graph G over g vertices and e edges and an integer t, we first define

$$\mathcal{P} = \{w_u \ : \ u \in V(G)\} \cup \{w_{u,v} \ : \ (u, v) \in E(G)\}$$

and observe that $|\mathcal{P}| = g + e$. We also construct the following $n = 2e$ preference-lists, where for every edge $(u, v) \in E(G)$ we have the two lists:

$$\ell_{u,v} = w_u, w_{u,v}, \ldots$$
$$\ell_{v,u} = w_v, w_{u,v}, \ldots$$

where the choice of passwords below position 2 is arbitrary, but both rankings must be identical from position 2 onwards. Finally, we set $k = g + e - t - 1$.

Given a policy $\mathcal{A} \subseteq \mathcal{P}$, we denote all banned words as $\mathcal{B} = \mathcal{P} \setminus \mathcal{A}$. We denote by $L_{\mathcal{B}}$ as the set of words that at least one user ranks first after banning all words in $\mathcal{B}$. Observe, $L_\emptyset = \{w_u \ : \ u \in V(G)\}$. Using this notation, we show this reduction indeed proves NP-hardness.

First, suppose G has a vertex cover C of size $\leq t$. Then by banning all passwords $\mathcal{B} = \{w_v \ : \ v \in C\}$ we now have $L_{\mathcal{B}} = \mathcal{P} \setminus \mathcal{B}$, because for every $(u, v) \in E(G)$ either w_u

or w_v are banned, so the word $w_{u,v}$ appears at the top of at least one of the two lists $\{\ell_{u,v}, \ell_{v,u}\}$. Therefore, the n preference-lists induce a distribution whose support contains $g + e - |\mathcal{B}| \geq g + e - t$ words, thus $p(g + e - t - 1, \mathcal{A}) < 1$.

Conversely, suppose all vertex covers of G are of size at least $t + 1$. Let $\mathcal{A}$ be any set of banned words. Clearly, if $|\mathcal{B}| \geq t + 1$ then the distribution induced by the n preferences-lists has support of size at most $g + e - t - 1$, which means that $p(g + e - t - 1, \mathcal{A}) = 1$. Otherwise, $|\mathcal{B}| \leq t$, and we denote the set of vertices $C = \{v : w_v \in \mathcal{B}\}$. Observe, since any vertex cover of G must contain $\geq t+1$ vertices, then there has to be at least $t + 1 - |C|$ edges that C does not cover (since we can always complete C to a vertex cover by adding one vertex from each uncovered edge). Therefore, there have to be at least $t + 1 - |C|$ words that do not appear at the top of any preference list. We conclude that the distribution induced by the n preference-lists has a support of size at most

$$|L_{\mathcal{B}}| = g - |C| + e - (t + 1 - |C|) \leq g + e - t - 1$$

thus $p(g + e - t - 1, \mathcal{A}) = 1$. □

From the same reduction described in Theorem 13 we get *UGC*-hardness of approximation. While there are sub-exponential time algorithms to solve the Unique Games problem [20], there are no known polynomial time algorithms. Many famous approximation hardness results are based on the Unique Games Conjecture (e.g., $2 - \epsilon$ hardness for vertex cover [99]). Our reduction relies on a result in [22], which says that vertex cover is hard to approximate up to a (say) 1.5-factor even on bounded degree graphs. Because we start with a bounded degree graph we can argue that each password in our reduction appears at the top of at most d preference-lists for some constant d.

Theorem 14. *There exists a constant $c > 1$ such that it is UGC-hard for a* poly(n, N, k)*-time algorithm to c-approximate the optimal $p(k, \mathcal{A})$ in the singleton rules setting and the rankings model.*

Proof of Theorem 14. We begin with a construction of a bounded degree graph which is hard approximate up to a (say) 1.5-factor. As shown in [22], for every constant d there exists a family of d-regular graphs for which it is *UGC*-hard to determine whether there exists a vertex cover of size t, or all vertex-covers have size at least $(2 - O(\log\log(d)/\log(d)) - \epsilon)\, t$. Fixing d to be a large enough constant such that this factor is > 1.5, we now reduce this family of instances to a password

problem using the exact same construction as in the proof of Theorem 13, with the exception that we set $k = g + e - (1.5 - \epsilon)t$.

Observe, for this family of instances, $e = O(g)$ so $|\mathcal{P}| = O(g)$, but also the size of the optimal vertex-cover has to be $\Theta(g)$ (at most g and at least g/d). Furthermore, each password appears at the top of at most d preference-lists. Therefore, by allowing $\mathcal{A}$ and banning $\mathcal{B} = \mathcal{P} \setminus \mathcal{A}$, we not only have a distribution whose support is of size $|L_{\mathcal{B}}|$, but it also holds that the probability of each word in $L_{\mathcal{B}}$ is $\Omega(1/|L_{\mathcal{B}}|)$.

Therefore, if the graph has a vertex-cover C of size t, then by banning all words $\mathcal{B} = \{w_u : u \in C\}$ we have that the n preference-lists induce a distribution over $|L_{\mathcal{B}}| \geq g + e - t$. Since we set $k = g + e - (1.5 - \epsilon)t$ we have that the set of most uncommon passwords contain at least $(0.5 - \epsilon)t = \Omega(|L_{\mathcal{B}}|)$ words, each with $\Omega(1/|L_{\mathcal{B}}|)$ probability, thus $p(k, \mathcal{A}) = 1 - \Omega(1)$. (And, in particular, for the optimal policy $\mathcal{A}^*$ we have $p(k, \mathcal{A}^*) = 1 - \Omega(1)$.)

In contrast, applying the same argument from the proof of Theorem 13, we have that if G has all vertex-covers of size $> (1.5 - \epsilon)t$ then $p(k, \mathcal{A}) = 1$. The $O(1)$-hardness of approximation follows. □

5.3.5 Negative Rules: Hardness of Approximation for $k = 1$

We next turn to negative rules, where we show that the problem is extremely difficult even for $k = 1$.

Theorem 15. *Let $\epsilon > 0$. Unless $P = NP$ there is no polynomial time algorithm (in N, n, m) that approximates $\min_{S \subseteq [m]} p(1, \mathcal{A}_S)$ to a factor of $n^{1/3-\epsilon}$ in the negative rules setting and the rankings model.*

Proof of Theorem 15. Fix $\epsilon > 0$. Our reduction is from the Max-Independent-Set problem, which is known to be hard to approximate up to a factor of $n^{1-\epsilon}$ [90]. We are given a graph G with g vertices and e edges, and we must determine whether the size of G's largest independent set is $g^{1-\epsilon}$ or g^{ϵ}.

Given a Max-Independent-Set instance, we denote $K = g^{\epsilon}$ and create the following password policy instance, which is composed out of the following set of

possible words:

$$\begin{aligned}\mathcal{P} \;=\; & \{A_1, ..., A_K\} \cup \{B_1, ..., B_g\} \\ & \cup \left(\bigcup_{\{u,v\} \in E(G)} \left(\{C^v_{u,1}, ..., C^v_{u,g}\} \cup \{C^u_{v,1}, ..., C^u_{v,g}\} \right) \right) \\ & \cup \left(\bigcup_{v \in V(G), 1 \le i < j \le K} \left(\{D_{v,i,j,1}, ..., D_{v,i,j,g}\} \cup \{D_{v,j,i,1}, ..., D_{v,j,i,g}\} \right) \right) \cup \{X\}\end{aligned}$$

We now describe the $n = g + ge + g^2\binom{K}{2} \le g^3 + g^{2+2\epsilon}$ users' preference-lists. We start with the g rankings specified in Table 5.2a. We continue with ge more rankings, where for each edge $(u, v) \in E(G)$ we add g more rankings, as detailed in Table 5.2b. Lastly, we add $g^2\binom{K}{2}$ more rankings, where for each triple (v, i, j) where v is a vertex of G and $i \neq j \in [K]$ we add g rankings, as detailed in Table 5.2c. (Observe, the tables detail the first few words in each list, then end with "..." mark, which indicates that from that point on the remaining words may appear in any order.)

Finally, we detail our rules. For every $i \in [K]$ and $u \in V(G)$ we have a rule which roughly corresponds to deciding that u is a member of the independent set:

$$R_{u,i} = \{A_i\} \cup \bigcup_{\{v:\ (u,v) \in E(G)\}} \{C^v_{u,1}, C^v_{u,2}, \ldots, C^v_{u,g}\} \cup \bigcup_{j \in [K], j \neq i} \{D_{u,i,j,1}, \ldots, D_{u,i,j,g}\}\,.$$

Our analysis now follows from a series of observations.

Observation 1: If we do not ban all of the passwords $A_1, ..., A_K$ then $p_1 \ge g/n$. Therefore, for every i, we must choose at least one of the rules $\{R_{u,i}\}$ to activate, or else we have that $p_1 \ge g/n$

Observation 2: If we ban $C^v_{u,1}, \ldots, C^v_{u,g}$ and $C^u_{v,1}, \ldots, C^u_{v,g}$ then we must have $p_1 \ge g/n$. Therefore, for any $i \neq j$ it must not be the case that we ban $R_{u,i}$ and $R_{v,j}$ where $(u, v) \in E(G)$, or else we have that $p_1 \ge g/n$.

Observation 3: If we ban $D_{v,i,j,1}, \ldots, D_{v,i,j,g}$, and $D_{v,j,i,1}, \ldots, D_{v,j,i,g}$ then $p_1 \ge g/n$. Therefore, for any $i \neq j$ it must not be the case that we ban $R_{u,i}$ and $R_{u,j}$, or else we have that $p_1 \ge g/n$.

These observations lead us to the following conclusion. If G contains an independent set $v_1, ..., v_K$ of size K, then activating the rules $\{R_{v_1,1}, R_{v_2,2}, \ldots, R_{v_K,K}\}$ leads

Table 5.2: Rankings used in the proof of Theorem 15.

ℓ_1	...	ℓ_g
A_1	...	A_1
A_2	...	A_2
...		
A_K	...	A_K
B_1	...	B_g
...		

(a) Type 1

$\ell_{u,v,1}$	...	$\ell_{u,v,g}$
$C^v_{u,1}$	...	$C^v_{u,g}$
$C^u_{v,1}$	...	$C^u_{v,g}$
X	...	X
...		

(b) Type 2

$\ell_{v,i,j,1}$	...	$\ell_{v,i,j,g}$
$D_{v,i,j,1}$	...	$D_{v,i,j,g}$
$D_{v,j,i,1}$	...	$D_{v,j,i,g}$
X	...	X
...		

(c) Type 3

to a setting where each truncated ranking begins with a unique word, so $p_1 = 1/n$. In contrast, if G does not have an independent set of size K, then $p_1 = g/n$. Since $n = O(g^3)$ we have an $\Omega(n^{1/3})$-hardness of approximation. Observe also that the number of total words is $N = K + g + 2eg + g^2K(K-1) + 1 = O(g^3) = O(n)$ so it is also hard to approximate the problem to a factor of $\Omega(N^{1/3})$. □

5.4 Normalization Model: Complexity Results

In this section we focus on complexity results for the normalization model. Here the structure of the input to our problem is a bit different: For each password $w \in \mathcal{P}$ we are given the probability $\Pr[w]$ that w is selected by a random user when $\mathcal{A} = \mathcal{P}$. Note that now we can give the distribution explicitly because it requires N numbers (whereas a distribution over rankings requires $N!$ numbers). This distribution induces a distribution over $\mathcal{P}$ for any password composition policy $\mathcal{A}$ by normalizing probabilities, as explained in Section 5.2.

Because the normalization model is a special case of the ranking model our algorithms for the ranking model can also be applied in the normalization model. The question is whether or not the hardness results carry over.

We first consider the singleton rules setting with large k, and show that that we can compute $\arg\min_{\mathcal{A}\subseteq\mathcal{P}} p(k, \mathcal{A})$ in polynomial time in N (Theorem 16). This result separates the normalization model from the ranking model (e.g., compare Theorems 16 and 13). However, it does not extend to the positive rules setting. In fact, we show that optimizing $p(k, \mathcal{A}_S)$ is NP-Hard when k is a parameter (Theorem 18).

With negative rules $R_1, ..., R_m$ we show that it is hard to c_0-approximate $\arg\max_{S\subseteq[m]} p(1, \mathcal{A}_S)$ (Theorem 17). However, we cannot rule out the possibility of an efficient c-approximation algorithm for some constant c in the normalization model (recall that Theorem 15 ruled out the possibility of a c-approximation algorithm in the ranking model for any c).

5.4.1 Singleton Rules: Efficient Algorithm for large k

We present SortAndOptimize — an efficient algorithm to optimize $p(k, \mathcal{A})$ in the singleton rules setting for *any* value of k. The key intuition behind our algorithm is that if $w_1 \in \mathcal{P}$ is the most likely password then w_1 will remain the most likely

allowed password unless we ban it — a property that does not hold in the rankings model. A formal proof of Theorem 16 can be found in Appendix 9.1.

Theorem 16. *For every k, Algorithm 5.4 computes* $\arg\min_{\mathcal{A}} p(k, \mathcal{A})$ *in the singleton rules setting of the normalized probabilities model, in time* $O(N \log(N))$.

Algorithm 5.4 SortAndOptimize

Input:
Password space $\mathcal{P}$ and a probability distribution over $\mathcal{P}$.
Integer k.
Sort the words in $\mathcal{P}$ from highest to lowest probability, $w_1, w_2, \ldots, w_N$.
return the set $\mathcal{A}_i = \{w_j \ : \ j \geq i\}$, where i minimizes the ratio

$$p(k, \mathcal{A}_i) = \frac{\sum_{i \leq j \leq i+k} \Pr[w_j]}{\sum_{j \geq i} \Pr[w_j]}$$

5.4.2 Negative Rules: Hardness for $k = 1$

We next prove an inapproximability result that is somewhat weaker than the one that we obtained for the more general ranking model.

Theorem 17. *There exists some constant $c_0 > 1$ such that unless NP = BPP no polynomial time algorithm (in n, N, m) can c_0-approximate* $\min_{S \subseteq [m]} p(1, \mathcal{A}_S)$ *in the negative rules setting and the normalization model.*

We will require the following construction; the proof is given in Appendix 9.1.

Lemma 4. *Fix m and s such that $m \geq s$. There exists a domain D of size $\Theta(s^2 \log(m))$ and a family of m sets, $F_1, F_2, \ldots, F_m \subseteq D$, such that each set in the family contains $\frac{|D|}{2s}$ elements, and for every $C \subseteq [m]$ of size $|C| \leq s$, we have that the size of the union $\left|\bigcup_{i \in C} F_i\right| \geq \frac{|D|}{2s}\frac{|C|}{4}$. This domain can be constructed in randomized* $\mathrm{poly}(s, m)$ *time.*

That is, each set in this family contains exactly the same fraction of the domain, and furthermore — any union of $|C| \leq s$ sets has the property that its cardinality is proportional to $\Omega(|C|)|F_i|$.

Proof of Theorem 17. We reduce from Set-Cover — one of the classic *NP*-Complete problems [94]. We are given sets $S_1, ..., S_m \subseteq U$, universe $U = \{1, ..., g\}$, and an integer $t \leq m$, and we are asked whether there is a set $C \subseteq [m]$ of size $\leq t$ such that $U = \bigcup_{i \in C} S_i$.

It is a known fact that there exist Set-Cover instances, with (g, m, t) all polynomially dependent of each other, that are hard to approximate to a factor of $c \ln n$ [17]. That is, on this particular family of instances, it is *NP*-hard to distinguish whether there exists a cover of size t or all covers have size $(1 - \epsilon)c \cdot t \ln n$.

We now describe the reduction. Given a (g, m, t)-Set Cover instance, we set $s = c \cdot t \ln g = \Theta(t \ln t)$ and construct a domain D and m sets $F_1, F_2, \ldots, F_m \subseteq D$ as in Lemma 4. We then create the following password-banning instance. First $\mathcal{P}$ is the union of D with additional disjoint g words denoted $w_1, ..., w_g$. Now, for each set S_i in the Set-Cover we add a rule R_i where $R_i = \{w_j\}_{j \in S_i} \cup F_i$. Finally, we set the words' probabilities as follows. Fixing some arbitrarily small $\delta > 0$, we set for every i the probability $\Pr[w_i] = \frac{1-\delta}{g}$, and for every $x \in D$ we set the probability $\Pr[x] = \frac{\delta}{|D|}$.

Without loss of generality we can assume that $|D| \geq 100g$ (because, for example, we can take $100g$ copies of the original D). Therefore, any policy that bans all of $\{w_1, w_2, \ldots w_g\}$ yet leaves a constant (say $> 1/10$) fraction of D has $p_1 \leq 10/|D|$, whereas any policy that keeps even one of the words in $\{w_1, w_2, \ldots, w_g\}$ has $p_1 \geq 1/(2g)$. Therefore, if the Set-Cover instance has a cover of size $\leq s = \Theta(t \ln g)$, then a c_0-approximation of the optimal banning-policy must find a cover for $\{w_1, w_2, \ldots, w_g\}$. We will assume from now on that our Set-Cover instance is such that it has a cover of size $\leq s$. (Indeed, if $s > t \log(t)$ then the instance is no longer *NP*-hard, since the greedy algorithm must return a cover of size $> t \log(t)$ which causes us to deduce that the optimal cover must have size $> t$.)

So now, suppose our Set-Cover instance has a cover of size t. Then the respective union of rules bans every password in $\{w_1, w_2, \ldots, w_g\}$ and no more than $\frac{t}{2s}|D|$ words of D (we get an upper bound by multiplying the size of each set by the number of sets). This leaves a collection of $\left(1 - \frac{t}{2s}\right)|D|$ equally likely words, so $p_1 = \left(1 - \frac{t}{2s}\right)^{-1} |D|^{-1} = (1 - O(1/\log(g)))^{-1}|D|^{-1} = (1 + o(1))|D|^{-1}$. In contrast, if all covers of our Set-Cover instance have size $s' \geq c \cdot t \ln(g)$ (where, because we assume some cover has size $\leq s$, we have $s' \leq s$,) then any collection of rules that bans all words in $\{w_1, w_2, \ldots, w_g\}$ must also ban at least $\frac{s'}{8s}|D|$ words out of D. This leaves at most $(1 - \Omega(1))|D|$ words in D and so $p_1 \geq (1 - \Omega(1))^{-1}|D|^{-1}$. Denoting the latter constant as c_0^{-1}, we have that any $c_0 - \epsilon$ approximation of the optimal banning-policy indicates the existence of a cover of cardinality $< c \cdot t \ln(g)$. □

5.4.3 Positive Rules: Hardness of Approximation for Large k

While we can show that it is possible to optimize $p(k, \mathcal{A})$ in the singleton rules setting our result does not extend to the more general positive rules setting. We are able to show that it is NP-Hard to compute $\arg\min_{S\subseteq[m]} p(k, \mathcal{A}_S)$. However, our reduction does not imply approximation hardness so we cannot rule out the existence of a PTAS.

Theorem 18. *Unless $P = NP$ there is no polynomial time algorithm (in N, m, n) which outputs $\arg\min_{S\subseteq[m]} p(k, \mathcal{A}_S)$ in the positive rules setting and the normalization model.*

The theorem's proof is relegated to Appendix 9.1.

5.5 Efficient Sampling Algorithms

In a sense, our complexity results are not "realistic", and in particular in the ranking model our positive algorithmic results assume access to each user's full preferences. Moreover, some algorithms are allowed to run in polynomial time in the number of passwords N, which can be huge. In this section we use our complexity results as guidelines in the design of practical sampling algorithms.

In more detail, we are given oracle access to rules $R_1, ..., R_m$ (e.g., we can ask whether or not a password $w \in R_i$) and we are allowed to sample from the distribution induced by the password composition policy $\mathcal{A}_S$ for any $S \subseteq [m]$. Less formally, a sample is equivalent to asking a random user what her favorite password is given the current policy.

We will work in the more general ranking model, so there is essentially only one positive result we can build on: Theorem 12, a polynomial time algorithm for constant k in the positive rules setting. When adapting this algorithm to the sampling setting, we cannot expect it to work perfectly due to the inherent uncertainty of this domain. Instead we expect the algorithm to find an ϵ-optimal password composition policy with probability at least $1 - \delta$, for any given ϵ and δ. Crucially, the number of samples must not depend on the number of passwords N, and must have a polynomial dependence on the other parameters.

Formally, we let $S^* \subseteq [m]$ denote the optimal collection of positive rules to activate (for all $S \subseteq [m]$, $p(1, \mathcal{A}_{S^*}) \leq p(1, \mathcal{A}_S)$). Our goal is to find a $(1, \epsilon)$-approximation $S \subseteq [m]$ to $p(1, \mathcal{A}_{S^*})$, that is, S such that $p(1, \mathcal{A}_S) \leq p(1, \mathcal{A}_{S^*}) + \epsilon$, with probability $1 - \delta$.

We first present Algorithm 5.5 that achieves our goal for $k = 1$; this algorithm is an adaptation of Algorithm 5.2.

Algorithm 5.5 SampleAndEliminate

Positive Rules: $R_1, ..., R_m$
Input: ϵ, δ
Initialize: $S_0 \leftarrow [m], i \leftarrow 0$
$s \leftarrow \frac{100}{\epsilon^2} \log\left(\frac{4m}{\epsilon\delta}\right)$
while $S_i \neq \emptyset$ **do**
  Sample: Draw samples $w_1, ..., w_s$ according to the distribution $\Pr[w \mid \mathcal{A}_{S_i}]$
  $W \leftarrow \{w_1, ..., w_s\}$
  $s_w \leftarrow \left|\left\{j \,\middle|\, w_j = w\right\}\right|$ for each $w \in W$.
  $w^* \leftarrow \arg\max \{s_w \mid w \in W\}$ ▷ w^* is the most frequently sampled password
  $\hat{p}_i \leftarrow \frac{s_{w^*}}{s}$ ▷ $\hat{p}_i$ is our estimation of $\Pr[w^* \mid \mathcal{A}_{S_i}]$
  if $\hat{p}_i \leq \epsilon/2$ **then return** S_i ▷ The current solution is already sufficiently good
  else
    $S_{i+1} \leftarrow S_i - \{j \mid w^* \in S_j\}$ ▷ Deactivate all rules that contain w^*
    $i \leftarrow i + 1$
return S_{i^*} where $i^* = \arg\max\left\{\hat{p}_j \,\middle|\, j \leq m\right\}$.

Theorem 19. *Algorithm 5.5 runs in polynomial time in $m, 1/\epsilon, 1/\delta$, requires $O(m \log(m/\delta)/\epsilon^2)$ samples and returns a $(1, \epsilon)$-approximation $S \subseteq \{1, ..., m\}$ of $p(1, \mathcal{A}_{S^*})$ with probability at least $1 - \delta$.*

Proof. Let

$$BAD_i = \left\{\exists w \in \mathcal{A}_{S_i} \,\middle|\, \left|\frac{s_w}{s} - \Pr[w \mid \mathcal{A}_{S_i}]\right| \geq \epsilon/2\right\},$$

denote the event that our probability estimates are off during iteration i. Claim 5 bounds the probability of any bad event. The proof of Claim 5 can be found in the appendix. The proof involves bucketing the passwords based on their probability, applying Chernoff Bounds to upper bound the probability of a bad estimate for our passwords in each bucket, and repeatedly applying union bounds.

Claim 5. $\Pr[\exists i, BAD_i] \leq \delta$.

For the rest of the analysis we assume that no bad event occurs. Let $p^* = \min_{S \subseteq [m]} p(1, \mathcal{A}_S)$ and suppose that $A_{S^*} \subseteq A_{S_i}$. Clearly, this is true when $i = 0$. If $\hat{p}_i \geq \epsilon/2 + p^*$ then $\Pr[w^* \mid \mathcal{A}_{S^*}] \geq \Pr[w^* \mid \mathcal{A}_{S_i}] > p^*$ so that $w^* \notin A_{S^*}$. Hence, $A_{S^*} \subseteq A_{S_{i+1}}$

and the property is maintained for at least one more iteration. If instead $\hat{p}_i < \epsilon/2 + p^*$ then we have $\hat{p}_{i^*} \leq \hat{p}_i \leq p^* + \epsilon/2$ so for each $w \in \mathcal{A}_{S_{i^*}}$ we have $\Pr\left[w \,\middle|\, \mathcal{A}_{S_{i^*}}\right] \leq p^* + \epsilon$. We conclude that the solution S_{i^*} is a $(1, \epsilon)$-approximation. □

We next explain how to extend Algorithm 5.3 to $(1, \epsilon)$-approximate the optimal $p(k, \mathcal{A}_S)$ for any constant k.

Theorem 20. *There is an algorithm which runs in polynomial time (in $m, 1/\epsilon, \delta$), takes a polynomial number of samples, and returns a $(1, \epsilon)$-approximation $S \subseteq [m]$ of $p(k, \mathcal{A}_{S^*})$ with probability at least $1 - \delta$.*

sketch. To extend Algorithm 5.3 to $(1, \epsilon)$-approximate $p(k, \mathcal{A}_S)$ for constant k we need one more idea. We cannot simply obtain a reduced password space $\hat{P}$ by reducing preference lists because we can only sample from our distribution. Notice that for any $S \subseteq [m]$ such that $i \in S$ we have $\Pr[w \,|\, \mathcal{A}_S] \leq \Pr[w \,|\, \mathcal{A}_{\{i\}}]$ so to obtain a $(1, \epsilon)$-approximation it is sufficient to limit our attention to passwords in the following set

$$\hat{P} = \left\{ w \,\middle|\, \exists i, \Pr\left[w \,\middle|\, \mathcal{A}_{\{i\}} \geq \frac{\epsilon}{k}\right] \right\} .$$

We can obtain a superset of $\hat{P}$ by sampling. For each positive rule R_i we draw s independent samples from the distribution $\mathcal{A}_{\{i\}}$ and set

$$T_i = \left\{ w \,\middle|\, \frac{s_w}{s} > \frac{\epsilon}{2k} \right\} .$$

Intuitively, a password w is included in T_i if and only if our estimated probability is sufficiently large. Let $T = \bigcup_i T_i$. For a sufficiently large sample size $s = O(poly(m, k, 1/\epsilon, 1/\delta))$ we can apply Chernoff Bounds to argue that with probability $1 - \delta$ (1) $|T|$ is small, i.e., $O(poly(m, k, 1/\epsilon, 1/\delta))$, and (2) $T \supset \hat{P}$. □

5.6 Experiments

To demonstrate how our ideas could apply in a real-world scenario, we simulated runs of Algorithm 5.5 by sampling with replacement from the RockYou leaked password set [92]. The set contains over 32 million passwords with a frequency distribution similar to that of many other password sets [39]. Note that all results presented here are limited by the dataset and assume the normalization model. Working in the normalization model is crucial because we cannot ask the RockYou

users for their preferred password under a specific policy; an initial distribution over $\mathcal{P}$ — which is available to us — is sufficient though, because it induces a distribution for any policy $\mathcal{A}$.

We selected 21 positive rules that mirror commonly used password composition rules that are used in practice, and looked at sample sizes s of 100, 500, 1000, 5000, and 10000. The rules included length requirements, character class requirements, combinations of requirements, a dictionary check, etc. (See Table 5.3 in Section 5.6.1 for a complete listing of the rules we selected.) For each run with a particular value of s, the algorithm returns a policy $\mathcal{A}_S$ for which we can measure $p(1, \mathcal{A}_S)$ in the original dataset and compare with the optimal $p(1, \mathcal{A}_{S^*})$, determined from running Algorithm 5.2 on the original dataset. We performed 500 runs for each of the five values of s.

To gain an understanding of how policies based on negative rules perform, we took the complement of the 21 positive rules selected above to get 21 negative rules. We then determined the optimal negative rules policy by calculating $S^* = \arg\min_{S \subseteq [m]} p(1, \mathcal{A}_S)$ via brute-force. This was required because we have no equivalent to Algorithm 5.2 for negative rules. With this baseline in hand, we designed two naïve algorithms, similar in spirit to Algorithm 5.5. There are multiple ways to discard a password in the negative rules setting, and one algorithm makes this decision randomly while the other bans the smallest subset as determined from the current sample. Again, 500 runs were performed for each $s \in \{100, 500, 1000, 10000, 50000\}$.

5.6.1 Experiment Rules

We selected rules based on common types of rules used in constructing password composition policies, e.g., the policies recommended by NIST [50]. The rules we selected are shown in Table 5.3. Positive and negative forms of each rule are shown. In the positive rules setting, a password is allowed if it matches any positive rule. In the negative rules setting, a password is banned if it matches any negative rule.

The dictionary check used the cracking dictionary from openwall.com. This dictionary is used by one of the most well-known password crackers, John the Ripper [63]. Since this dictionary contains all alphabetic strings up to size 3, it was pruned to only include entries of 4 characters or more for the "contains a dictionary word" dictionary check.

Notice that for some groups of rules, e.g., length rules, digit rules, etc., the

subsets defined by these rules are subsets or supersets of each other. For example, if the positive rule "8 characters or more" is in a policy, adding the "10 characters or more" rule yields the same policy. We did this to prevent the selection of overly complex policies, e.g., "8 characters" OR "11 characters" OR "12 characters" OR "14 characters." However, we also selected a couple of "combination rules" to make policies more interesting.

5.6.2 Baselines

We examined several baselines for comparison with our algorithm. Table 5.4 shows these baselines, the probability of the most frequent password in the resulting policy, and the optimal policy as a union or intersection of rules (for clarity, the complement of the union of negative rules is shown as the intersection of positive rules).

As shown in Table 5.4 from the means across policies, randomly selecting a policy from the power set of rules can be worse than having no policy. The "one rule maximum" baseline was selected because, if decided based on sampling, only m distributions need be sampled. Our efficient algorithm requires the same amount of sampling, but can find the optimal policy over $S \subseteq [m]$ rather than $S \in \{1, ..., m\}$. Also of interest is the optimal policy with negative rules, which is over 3x better than the optimal policy with positive rules. However, as shown in the following section, the performance of our sampling algorithms with negative rules was far worse than in the positive rules setting.

5.6.3 Performance

In the positive rules setting (see Table 5.5), the algorithm performed extremely well even at moderate sample sizes. The average policy selected with $s = 500$ was almost 10x better than having no policy. At $s = 1000$, the optimal policy was found 10% of the time (50 out of 500 times).

In the negative rules setting (see Table 5.6), however, neither algorithm found the optimal policy. The "Ban Smallest" heuristic, when faced with a choice between multiple subsets that contain the most likely password, decides to ban the smallest available subset, disrupting the space the least. This might seem like an intuitively good choice but, in fact, it fails to find a better policy than the empty set at large sample sizes. The randomized algorithm does better (it cannot actually do

worse) but still has much worse average case performance than using our efficient algorithm with positive rules.

5.7 Discussion

We conclude this chapter by discussing some key points.

Usability. In this chapter our goal was to optimize the *security* of a password composition policy. However, many users find it difficult to comply with all of the requirements of a complicated password composition policy. Can we quantify the usability costs of a password composition policy? Can we characterize the trade-off between security and usability in password composition policies? Can we find the optimal password composition policy subject to usability constraints?
Where do the rules comes from? Throughout the paper we have assumed that the rules (whether positive or negative) are given as part of the input; it is not up to us to find these rules. Our experiments indicate that a collection of intuitive and practical rules can already give very good results on real data. However, the question of deciding which rules should be added to our collection is outside the scope of this paper. Much like the problem of feature selection, it is an interesting problem with real-life implications, which we suspect will be very difficult in practice.

Alternate policy goals. Our goal [45] has been to minimize $p(k, \mathcal{A}_S)$. Intuitively, $p(k, \mathcal{A}_S)$ represents the probability that an adversary with no background knowledge can successfully guess the password of a randomly selected user in k tries. A small value of k optimizes security guarantees against an online guessing attack in which the adversary is locked out after k failed attempts to login. A much larger value of k (e.g., 2^{32}) is necessary to optimize security against an adversary who has obtained the cryptographic hash of a password and is able to mount a brute-force dictionary attack [136]. However, the optimal solutions for $p(1, \mathcal{A}_S)$ and $p(2^{32}, \mathcal{A}_S)$ might be completely different. One stronger goal that we might hope to achieve is to optimize both goals simultaneously. More formally, can we find a policy $S \subseteq [m]$ such that for every $S' \subseteq [m]$ and every $k \leq N$ we have $p(k, \mathcal{A}_S) \leq c \cdot p(k, \mathcal{A}_{S'})$ for some constant c? Unfortunately, the answer is no. For any constant c this universal approximation goal is impossible to satisfy in the ranking model — see Theorem 32 in the appendix.

Other natural goals include α-work factor [121] and a refinement called α-

guesswork [39] (e.g., maximize the total number of guesses needed to compromise a fraction α of the accounts). While α-guesswork is an useful metric to analyze the security of 70 million Yahoo passwords [39], it may not be a desirable optimization goal for the organization because it might allow the adversary to crack up to $(\alpha - \epsilon)$-fraction of the accounts with relatively few guesses.

Another interesting direction is to account for an adversary with basic background information about the user (e.g., e-mail address, username, birthday). It may not always be realistic to assume that the adversary has no background knowledge because the adversary can often easily obtain some background knowledge about a user by searching for publicly available information on the internet. One approach might be to design a rule R to specify different passwords for different users (e.g., the set of passwords that contain the username or birthday of the user).

Open Questions. While we were able to prove several hardness results about finding the optimal password composition policy in the negative rules setting, it is possible that these hardness results could be circumvented by making mild (hopefully realistic) assumptions about the underlying password distribution or the rules $R_1, ..., R_m$. Are there efficient algorithms to optimize $p(k, \mathcal{A}_S)$ in the negative rules setting given realistic assumptions? It is also possible that mild realistic assumptions could be used to circumvent the impossibility result of Theorem 32, and design a universal approximation algorithm.

There are also several interesting technical questions that remain open:

1. Normalization model with negative rules: Can we efficiently c-approximate $p(1, \mathcal{A}_{S^*})$ for any constant c? Is there a sub-exponential algorithm (in m) to compute $p(1, \mathcal{A}_{S^*})$?

2. Ranking model with positive rules: Can we efficiently c-approximate $p(k, \mathcal{A}_{S^*})$ for some constant c when k is a parameter?

The future. There is a real need for a principled approach to optimizing password composition policies. We have taken a first step in this direction by providing an intuitive theoretical model and showing that it leads to algorithms that perform well on real data. We can only hope that our work will spark a fundamentally new interaction between theory and practice in passwords research.

Positive Rule	**Negative Rule**	**Details**
8 characters or more	Less than 8 characters	Length rules
9 characters or more	Less than 9 characters	
10 characters or more	Less than 10 characters	
11 characters or more	Less than 11 characters	
12 characters or more	Less than 12 characters	
13 characters or more	Less than 13 characters	
14 characters or more	Less than 14 characters	
15 characters or more	Less than 15 characters	
16 characters or more	Less than 16 characters	
1 digit or more	Less than 1 digit	Character class rules
1 symbol or more	Less than 1 symbol	
1 lowercase or more	Less than 1 lowercase	
1 uppercase or more	Less than 1 uppercase	
2 digits or more	Less than 2 digits	
2 symbols or more	Less than 2 symbols	
2 lowercase or more	Less than 2 lowercase	
2 uppercase or more	Less than 2 uppercase	
In a dictionary	Not in a dictionary	Dictionary checks
Contains a dictionary word	Does not contain a dictionary word	
8 characters or more AND 1 uppercase or more	Less than 8 characters OR less than 1 uppercase	Combination Rules
8 characters or more AND 1 uppercase or more AND 1 digit or more	Less than 8 characters OR less than 1 uppercase OR less than 1 digit	

Table 5.3: Rules Used in Sampling Experiments

Baseline	$p\,(1, \mathcal{A}_S)$	S
Mean across negative rules policies	1.3×10^{-2}	
Mean across positive rules policies	1.0×10^{-2}	
All passwords allowed (no policy)	9.2×10^{-3}	
One positive rule ($S \in \{1, ..., m\}$)	6.8×10^{-4}	8 chars, 1 upper, 1 digit
Optimal policy with positive rules	4.4×10^{-4}	14 chars OR 2 symbols OR 8 chars, 1 upper, 1 digit
Optimal policy with negative rules	1.4×10^{-4}	10 chars AND 2 digits AND 1 symbol AND 1 lowercase AND not in dictionary

Table 5.4: Baseline probabilities for the RockYou dataset

Sample Size	mean $p\,(1, \mathcal{A}_S)$	min $p\,(1, \mathcal{A}_S)$	**% Optimal**
100	6.8×10^{-3}	1.2×10^{-3}	
500	9.7×10^{-4}	$\mathbf{4.4\times10^{-4}}$	2%
1000	9.5×10^{-4}	$\mathbf{4.4\times10^{-4}}$	10%
5000	6.0×10^{-4}	$\mathbf{4.4\times10^{-4}}$	14%
10000	5.7×10^{-4}	$\mathbf{4.4\times10^{-4}}$	19%

Table 5.5: Performance of Sampling Algorithms with Positive Rules

	Random Decision		**Ban Smallest**	
Sample Size	mean $p\,(1, \mathcal{A}_S)$	min $p\,(1, \mathcal{A}_S)$	mean $p\,(1, \mathcal{A}_S)$	min $p\,(1, \mathcal{A}_S)$
100	6.8×10^{-3}	1.2×10^{-3}	7.2×10^{-3}	2.3×10^{-3}
500	4.4×10^{-3}	6.3×10^{-4}	9.0×10^{-3}	2.3×10^{-3}
1000	4.3×10^{-3}	4.5×10^{-4}	8.6×10^{-3}	2.3×10^{-3}
5000	6.3×10^{-3}	4.5×10^{-4}	9.2×10^{-3}	9.2×10^{-3}
10000	7.2×10^{-3}	4.5×10^{-4}	9.2×10^{-3}	9.2×10^{-3}

Table 5.6: Performance of Sampling Algortihms with Negative Rules

Chapter 6

GOTCHAs: A Defense Against Offline Attacks

6.1 Introduction

Any adversary who has obtained the cryptographic hash of a user's password can mount an automated brute-force attack to crack the password by comparing the cryptographic hash of the user's password with the cryptographic hashes of likely password guesses. This attack is called an offline dictionary attack, and there are many password crackers that an adversary could use [63]. Offline dictionary attacks against passwords are — unfortunately — powerful and commonplace [87]. Adversaries have been able to compromise servers at large companies (e.g., Zappos, LinkedIn, Sony, Gawker [5, 9, 10, 11, 13, 28]) resulting in the release of millions of cryptographic password hashes[1]. It has been repeatedly demonstrated that users tend to select easily guessable passwords [39, 66, 92], and password crackers are able to quickly break many of these passwords[136]. Offline attacks are becoming increasingly dangerous as computing hardware improves — a modern GPU can evaluate a cryptographic hash function like SHA2 about 250 million times per second [165] — and as more and more training data — leaked passwords from prior breaches — becomes available [87]. Symantec reported that compromised passwords have significant economic value to an adversary (e.g., compromised passwords are sold on black market for between $4 and $30) [79].

HOSPs (Human-Only Solvable Puzzles) were suggested by Canetti, Halevi

[1] In a few of these cases [5, 10] the passwords were stored in the clear.

and Steiner as a way of defending against offline dictionary attacks [51]. The basic idea is to change the authentication protocol so that human interaction is *required* to verify a password guess. The authentication protocol begins with the user entering his password. In response the server randomly generates a challenge — using the password as a source of randomness — for the user to solve. Finally, the server appends the user's response to the user's password, and verifies that the hash matches the record on the server. To crack the user's password offline the adversary must simultaneously guess the user's password and the answer to the corresponding puzzle. The challenge should be easy for a human to solve consistently so that a legitimate user can authenticate. To mitigate the threat of an offline dictionary attack the HOSP should be difficult for a computer to solve — even if it has all of the random bits used to generate the challenge.

The basic HOSP construction proposed by Canetti et al. [51] was to to fill a hard drive with regular CAPTCHAs (e.g., distorted text) by storing the puzzles without the answers. This solution only provides limited protection against an adversary because the number of unique puzzles that can be generated is bounded by the size of the hard drive (e.g., the adversary could pay people to solve all of the puzzles on the hard drive). See Appendix 10.2 for more discussion. Finding a usable HOSP construction which does not rely on a very large dataset of pregenerated CAPTCHAs is an open problem. Several candidate HOSPs were experimentally tested [59] (they are called POSHs in the second paper), but the usability results were underwhelming.

Contributions In this chapter we introduce a simple modification of HOSPs that we call GOTCHAs (Generating panOptic Turing Tests to Tell Computers and Humans Apart). We use the adjective Panoptic to refer to a world without privacy — there are no hidden random inputs to the puzzle generation protocol. The basic goal of GOTCHAs is similar to the goal of HOSPs — defending against offline dictionary attacks. GOTCHAs differ from HOSPs in two ways (1) Unlike a HOSP a GOTCHA may require human interaction during the *generation* of the challenge. (2) We relax the requirement that a user needs to be able to answer all challenges easily and consistently. If the user can remember his password during the authentication protocol then he will only ever see one challenge. We only require that the user must be able to answer this challenge consistently. If the user enters the wrong password during authentication then he may see new challenges. We do not require that the user must be able to solve these challenges consistently because authentication will fail in either case. We do require that it is difficult

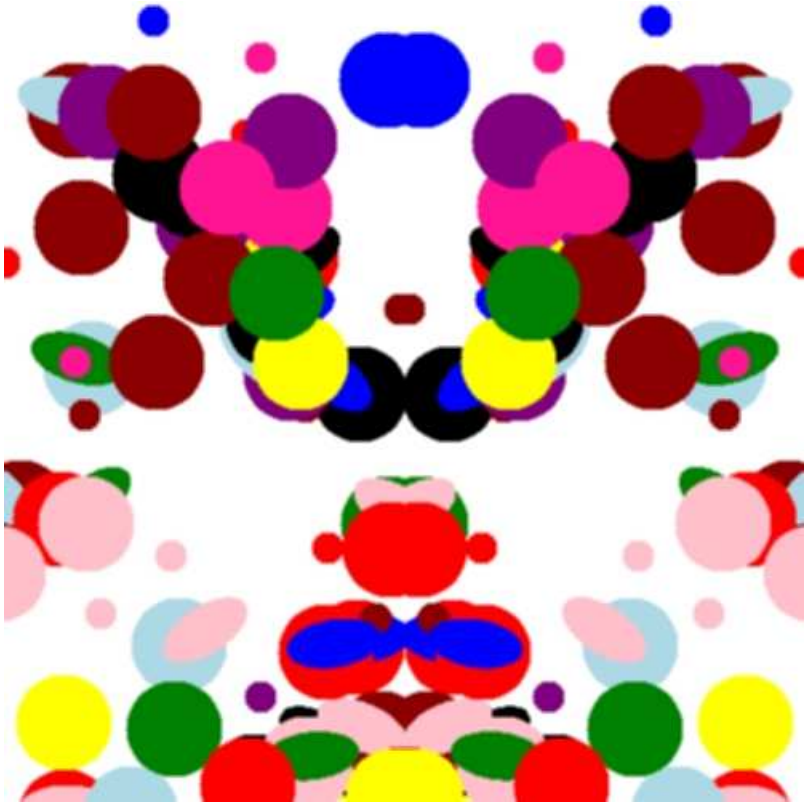


Figure 6.1: Randomly Generated Inkblot Image—An evil clown?

for a computer to distinguish between the "correct" challenge and an "incorrect" challenge. Our main theorem demonstrates that GOTCHAs like HOSPs can be used to defend against offline dictionary attacks. The goal of these relaxations is to enable the design of usable GOTCHAs.

We introduce a candidate GOTCHA construction based on Inkblot images. While the images are generated randomly by a computer, the human mind can easily imagine semantically meaningful objects in each image. To generate a challenge the computer first generates ten inkblot images (e.g., figure 6.1). The user then provides labels for each image (e.g., evil clown, big frog). During authentication the challenge is to match each inkblot image with the corresponding label. We empirically evaluate the usability of our inkblot matching GOTCHA construction by conducting a user study on Amazon's Mechanical Turk. Finally, we challenge the AI community to break our GOTCHA construction.

Organization The rest of this chapter is organized as follows: We next discuss related work in section 6.1.1. We formally define GOTCHAs in section 6.2 and formalize the properties that a GOTCHA should satisfy. We present our candidate GOTCHA construction in section 6.3, and in section 6.3.1 we demonstrate how our GOTCHA could be integrated into an authentication protocol. We present the results from our user study in section 6.3.2, and in section 6.3.3 we challenge the AI and security communities to break our GOTCHA construction. In section 6.4 we prove that GOTCHAs like HOSPs can also be used to design a password storage system which mitigates the threat of offline attacks. We conclude by discussing future directions and challenges in section 6.5.

6.1.1 Related Work

Inkblots [148] have been proposed as an alternative way to generate and remember passwords. Stubblefield and Simon proposed showing the user ten randomly generated inkblot images, and having the user make up a word or a phrase to describe each image. These phrases were then used to build a 20 character password (e.g., users were instructed to take the first and last letter of each phrase). Usability results were moderately good, but users sometimes had trouble remembering their association. Because the Inkblots are publicly available there is also a security concern that Inkblot passwords could be guessable if different users consistently picked similar phrases to describe the same Inkblot.

We stress that our use of Inkblot images is different in two ways: (1) Usability: We do not require users to recall the word or phrase associated with each Inkblot. Instead we require user's to recognize the word or phrase associated with each Inkblot so that they can match each phrase with the appropriate Inkblot image. Recognition is widely accepted to be easier than the task of recall [23, 155]. (2) Security: We do not need to assume that it would be difficult for other humans to match the phrases with each Inkblot. We only assume that it is difficult for a computer to perform this matching automatically.

CAPTCHAs — formally introduced by Von Ahn et al. [152] — have gained widespread adoption on the internet to prevent bots from automatically registering for accounts. A CAPTCHA is a program that generates a puzzle — which should be easy for a human to solve and difficult for a computer to solve — as well as a solution. Many popular forms of CAPTCHAs (e.g., reCAPTCHA [153]) generate

garbled text, which is easy[2] for a human to read, but difficult for a computer to decipher. Other versions of CAPTCHAs rely on the natural human capacity for audio [131] or image recognition [67].

CAPTCHAs have been used to defend against online password guessing attacks — users are sometimes required to solve a CAPTCHA before signing into their account. An alternative approach is to lock out a user after several incorrect guesses, but this can lead to denial of service attacks [60]. However, if the adversary has access to the cryptographic hash of the user's password, then he can circumvent all of these requirements and execute an automatic dictionary attack to crack the password offline. By contrast HOSPs — proposed by Canetti et al.[51] — were proposed to defend against offline attacks. HOSPs are in some ways similar to CAPTCHAs (Completely Automated Turing Tests to Tell Computers and Humans Apart) [152]. CAPTCHAs are widely used on the internet to fight spam by preventing bots from automatically registering for accounts. In this setting a CAPTCHA is sent to the user as a challenge, while the secret solution is used to grade the user's answer. The implicit assumption is that the answer and the random bits used to generate the puzzle remain hidden — otherwise a spam bot could simply regenerate the puzzle and the answer. While this assumption may be reasonable in the spam bot setting, it does not hold in our offline password attack setting in which the server has already been breached. A HOSP is different from a CAPTCHA in several key ways: (1) The challenge must remain difficult for a computer to solve even if the random bits used to generate the puzzle are made public. (2) There is no single correct answer to a HOSP. It is okay if different people give different responses to a challenge as long as people can respond to the challenges easily, and each user can consistently answer the challenges.

The only HOSP construction proposed in [51] involved stuffing a hard drive with unsolved CAPTCHAs. The problem of finding a HOSP construction that does not rely on a dataset of unsolved CAPTCHAs was left as an open problem [51]. Several other candidate HOSP constructions have been experimentally evaluated in subsequent work [59] (they are called POSHs in the second paper), but the usability results for every scheme that did not rely on a large dataset on unsolved CAPTCHAs were underwhelming.

GOTCHAs are very similar to HOSPs. The basic application — defending against offline dictionary attacks — is the same as are the key tools: exploiting the power of interaction during authentication, exploiting hard artificial intelligence problems. While the authentication with HOSPs is interactive, the initial

[2]Admitedly some people would dispute the use of the label 'easy.'

generation of the puzzle is not. By contrast, our GOTCHA construction requires human interaction during the initial generation of the puzzle. This simple relaxation allows for the construction of new solutions. In the HOSP paper humans are simply modeled as a puzzle solving oracle, and the adversary is assumed to have a limited number of queries to a human oracle. We introduce a more intricate model of the human agent with the goal of designing more usable constructions.

Password Storage Password storage is an incredibly challenging problem. Adversaries have been able to compromise servers at many large companies (e.g., Zappos, LinkedIn, Sony, Gawker [5, 9, 10, 11, 13, 28]). For example, hackers were able to obtain 32 million plaintext passwords from RockYou using a simple SQL injection attack [5]. While it is considered an extremely poor security practice to store passwords in the clear [141], the practice is still fairly common [5, 10, 41]. Many other companies [13, 41] have used cryptographic hashes to store their passwords, but failed to adopt the practice of salting (e.g., instead of storing the cryptographic hash of the password $\mathbf{H}(pw)$ the server stores $(\mathbf{H}(pw, r), r)$ for a random string r [16]) to defend against rainbow table attacks. Rainbow tables, which consist of precomputed hashes, are often used by an adversary to significantly speed up a password cracking attack because the same table can be reused to attack each user when the passwords are unsalted [117].

Cryptographic hash functions like SHA1, SHA2 and MD5 — designed for fast hardware computation — are popular choices for password hashing. Unfortunately, this allows an adversary to try up to 250 million guesses per second on a modern GPU [165]. The BCRYPT [122] hash function was designed specifically with passwords in mind — BCRYPT was intentionally designed to be slow to compute (e.g., to limit the power of an adversary's offline attack). The BCRYPT hash function takes a parameter which allows the programmer to specify how costly the hash computation should be. The downside to this approach is that it also increases costs for the company that stores the passwords (e.g., if we want it to cost the adversary $1,000 for every million guesses then it will also cost the company at least $1,000 for every million login attempts).

Users are often advised (or required) to follow strict guidelines when selecting their password (e.g., use a mix of upper/lower case letters, include numbers and change the password frequently) [133]. However, empirical studies show that user's are are often frustrated by restricting policies and commonly forget their

passwords [34, 75, 102][3]. Furthermore, the cost of these restrictive policies can be quite high. For example, a Gartner case study [158] estimated that it cost over $17 per password-reset call. Florencio and Herley [76] studied the economic factors that institutions consider before adopting password policies and found that they often value usability over security.

6.2 Definitions

In this section we seek to establish a theoretical basis for GOTCHAs. Several of the ideas behind our definitions are borrowed from theoretical definitions of CAPTCHAs [152] and HOSPs [51]. Like CAPTCHAs and HOSPs, GOTCHAs are based on the assumption that some AI problem is hard for a computer to solve, but easy for a person to solve. Ultimately, these assumptions are almost certainly false (e.g., because the human brain can solve a GOTCHA it is reasonable to believe that there exists a computer program to solve the problems). However, it may still be reasonable to assume that these problems cannot be solved by applying *known* ideas. By providing a formal definition of GOTCHAs we can determine whether or not a new *idea* can be used to break a candidate GOTCHA construction.

We use $c \in C$ to denote the space of challenges that might be generated. We use $\mathcal{H}$ to denote the set of human users and $H(c, \sigma_t)$ to denote the response that a human $H \in \mathcal{H}$ gives to the challenge $c \in C$ at time t. Here, σ_t denotes the state of the human's brain at time t. σ_t is supposed to encode our user's existing knowledge (e.g., vocabulary, experiences) as well as the user's mental state at time t (e.g., what is the user thinking about at time t). Because σ_t changes over time (e.g., new experiences) we use $H(c) = \{H(c, \sigma_t) \mid t \in \mathbb{N}\}$ to denote the *set* of all answers a human might give to a challenge c. We use $\mathcal{A}$ to denote the range of possible responses (answers) that a human might give to the challenges.

Definition 12. *Given a metric $d : \mathcal{A} \times \mathcal{A} \to \mathbb{R}$, we say that a human H can* consistently solve *a challenge $c \in C$ with accuracy α if $\forall t \in \mathbb{N}$*

$$d(H(c, \sigma_0), H(c, \sigma_t)) \leq \alpha ,$$

where σ_0 denotes the state of the human's brain when he initially answers the challenge. If $|H(c)| = 1$ then we simply say that the human can consistently solve the challenge.

[3] In fact the resulting passwords are sometimes more vulnerable to an offline attack! [34, 102]

Notation: When we have a group of challenges $\langle c_1, \ldots, c_k \rangle$ we will sometimes write $H(\langle c_1, \ldots, c_k \rangle, \sigma_t) = \langle H(c_1, \sigma_t), \ldots, H(c_k, \sigma_t) \rangle$ for notational convenience. We use $y \sim \mathcal{D}$ to denote a random sample from the distribution $\mathcal{D}$, and we use $r \sim \{0,1\}^n$ to denote a element drawn from the set $\{0,1\}^n$ uniformly at random. We stress that while H denotes a human user in this chapter, **H** still denotes a cryptographic hash that would be evaluated by a computer as in the rest of this thesis.

One of the requirements of a HOSP puzzle system [51] is that the human H must be able to *consistently* answer *any* challenge that is generated (e.g., $\forall c \in C$, H can consistently solve c). These requirements seem to rule out promising ideas for HOSP constructions like Inkblots[59]. In this construction the challenge is a randomly generated inkblot image I, and the response $H(I, \sigma_0)$ is word or phrase describing what the user initially sees in the inkblot image (e.g., evil clown, soldier, big lady with a ponytail). User studies have shown that $H(I, \sigma_0)$ does not always match $H(I, \sigma_t)$ — the phrase describing what the user sees at time t [59]. In a few cases the errors may be correctable (e.g., capitalization, plural/singular form of a word), but oftentimes the phrase was completely different — especially if a long time passed in between trials[4]. By contrast, our GOTCHA construction does not require the user to remember the phrases associated with each Inkblot. Instead we rely on a much weaker assumption — the user can consistently recognize his solutions. We say that a human can recognize his solutions to a set of challenges if he can consistently solve a matching challenge (definition 13) in which he is asked to match each of his solutions with the corresponding challenge.

Definition 13. *Given an integer k, and a permutation $\pi : [k] \rightarrow [k]$, a* matching challenge $\hat{c}_\pi = (\vec{c}, \vec{a}) \in C$ *of size k is given by a k-tuple of challenges $\vec{c} = \langle c_{\pi(1)}, \ldots, c_{\pi(k)} \rangle \in C^k$ and solutions $\vec{a} = H(\langle c_1, \ldots, c_k \rangle, \sigma_0)$. The response to a matching challenge is a permutation $\pi' = H(\vec{c}_\pi, \sigma_t)$.*

For permutations $\pi : [k] \rightarrow [k]$ we use the distance metric

$$d_k(\pi_1, \pi_2) = |\{i \mid \pi_1(i) \neq \pi_2(i) \wedge 1 \leq i \leq k\}| \ .$$

$d_k(\pi_1, \pi_2)$ simply counts the number of entries where the permutations don't match. We say that a human can consistently recognize his solution to a matching

[4]We would add the requirement that the human must be able to consistently answer the challenges without spending time memorizing and rehearsing his response to the challenge. Otherwise we could just as easily force the user to remember a random string to append on to his password.

challenge $\hat{c}_\pi$ with accuracy α if $\forall t.d_k\left(H\left(\hat{c}_\pi, \sigma_t\right), \pi\right) \leq \alpha$. We use $\{\pi' \mid d_k(\pi, \pi') \leq \alpha\}$ to denote the set of permutations π' that are α-close to π.

The puzzle generation process for a GOTCHA involves interaction between the human and a computer: (1) The computer generates a set of k challenges. (2) The human solves these challenges. (3) The computer uses the solutions to produce a final challenge[5]. Formally,

Definition 14. *A puzzle-system is a pair (G_1, G_2), where G_1 is a randomized challenge generator that takes as input 1^k (with k security parameter) and a pair of random bit strings $r_1, r_2 \in \{0,1\}^*$ and outputs k challenges $\langle c_1, \ldots, c_k\rangle \leftarrow G_1\left(1^k, r_1, r_2\right)$. G_2 is a randomized challenge generator that takes as input 1^k (security parameter), a random bit string $r_1 \in \{0,1\}^*$, and proposed answers $\vec{a} = \langle a_1, ..., a_k\rangle$ to the challenges $G_1\left(1^k, r_1, r_2\right)$ and outputs a challenge*
*$\hat{c} \leftarrow G_2\left(1^k, r_1, \vec{a}\right)$. We say that the puzzle-system is (α, β)-*usable *if*

$$\Pr_{H \sim \mathcal{H}}\left[\textbf{Accurate}\left(H, \hat{c}, \alpha\right)\right] \geq \beta\ ,$$

whenever $\vec{a} = H\left(G_1\left(1^k, r_1, r_2\right), \sigma_0\right)$, where **Accurate** $(H, \hat{c}, \alpha)$ *denotes the event that the human H can consistently solve $\hat{c}$ with accuracy α.*

In our authentication setting the random string r_1 is extracted from the user's password using a strong pseudorandom function **Extract**. To provide a concrete example of a puzzle-system, G_1 could be a program that generates a set of inkblot challenges $\langle I_1, \ldots, I_k\rangle$ using random bits r_1, selects a random permutation $\pi : [k] \rightarrow [k]$ using random bits r_2, and returns $\langle I_{\pi(1)}, \ldots, I_{\pi(k)}\rangle$. The human's response to an Inkblot — $H\left(I_j, \sigma_0\right)$ — is whatever he/she imagines when he sees the inkblot I_j for the first time (e.g., some people might imagine an evil clown when they look at figure 6.1). Finally, G_2 might generate Inkblots $\vec{c} = \langle I_1, \ldots, I_k\rangle$ using random bits r_1, and return the matching challenge $\hat{c}_\pi = (\vec{c}, \vec{a})$. In this case the matching challenge is for the user to match his labels with the appropriate Inkblot images to recover the permutation π. Observe that the final challenge — $\hat{c}_\pi$ — can only be generated after a round of *interaction* between the computer and a human. By contrast, the challenges in a HOSP must be generated automatically by a computer. Also notice that if G_2 is executed with a different random bit string r_1' then we do not require the resulting challenge to be consistently recognizable (e.g., if the user enters in

[5]We note that a HOSP puzzle system (G) [51] can be modeled as a GOTCHA puzzle system (G_1, G_2) where G_1 does nothing and G_2 simply runs G to generate the final challenge $\hat{c}$ directly.

the wrong password then authentication will fail regardless of how he solves the resulting challenge). For example, if the user enters the wrong password the user might be asked to match his labels $\langle \ell_{\pi(1)}, ..., \ell_{\pi(k)} \rangle = H\left(\langle I_{\pi(1)}, \ldots, I_{\pi(k)} \rangle, \sigma_0\right)$ with Inkblots $\langle I'_1, \ldots, I'_k \rangle$ that he has never seen.

An adversary could attack a puzzle system by either (1) attempting to distinguish between the correct puzzle, and puzzles that might be meaningless to the human, or (2) by solving the matching challenge directly.

We say that an algorithm A can distinguish distributions $\mathcal{D}_1$ and $\mathcal{D}_2$ with advantage ϵ if

$$\left| \Pr_{x \sim \mathcal{D}_1} [A(x) = 1] - \Pr_{y \sim \mathcal{D}_2} [A(y) = 1] \right| \geq \epsilon \, .$$

Our formal definition of a GOTCHA is found in definition 15. Intuitively, definition 15 says that (1) The underlying puzzle-system should be usable — so that legitimate users can authenticate. (2) It should be difficult for the adversary to distinguish between the correct matching challenge (e.g., the one that the user will see when he types in the correct password), and an incorrect matching challenge (e.g., if the user enters the wrong password he will be asked to match his labels with different Inkblot images), and (3) It should be difficult for the adversary to distinguish between the user's matching, and a random matching drawn from a distribution R with sufficiently high minimum entropy.

Definition 15. *A puzzle-system (G_1, G_2) is an $(\alpha, \beta, \epsilon, \delta, \mu)$-GOTCHA if (1) (G_1, G_2) is* (α, β)-usable *(2) Given a human $H \in \mathcal{H}$ no probabilistic polynomial time algorithm can distinguish between distributions*

$$\mathcal{D}_1 = \left\{ \begin{matrix} H\left(G_1\left(1^k, r_1, r_2\right), \sigma_0\right), \\ G_2\left(1^k, r_1, H\left(G_1\left(1^k, r_1, r_2\right), \sigma_0\right)\right) \end{matrix} \;\middle|\; r_1, r_2 \sim \{0,1\}^n \right\}$$

and

$$\mathcal{D}_2 = \left\{ \begin{matrix} H\left(G_1\left(1^k, r_1, r_2\right), \sigma_0\right), \\ G_2\left(1^k, r_3, H\left(G_1\left(1^k, r_1, r_2\right), \sigma_0\right)\right) \end{matrix} \;\middle|\; r_1, r_2, r_3 \sim \{0,1\}^n \right\}$$

with advantage greater than ϵ, and (3) Given a human $H \in \mathcal{H}$, there is a distribution $R(c)$ with $\mu(m)$ bits of minimum entropy such that no probabilistic polynomial time algorithm can distinguish between distributions

$$\mathcal{D}_3 = \left\{ \begin{matrix} H\left(G_1\left(1^k, r_1, r_2\right), \sigma_0\right) \\ G_2\left(1^k, r_1, H\left(G_1\left(1^k, r_1, r_2\right), \sigma_0\right)\right), \\ H\left(G_2\left(1^k, r_1, H\left(G_1\left(1^k, r_1, r_2\right), \sigma_0\right)\right), \sigma_0\right) \end{matrix} \;\middle|\; r_1, r_2 \sim \{0,1\}^n \right\}$$

and

$$\mathcal{D}_4 = \left\{ \begin{matrix} H\left(G_1\left(1^k, r_1, r_2\right), \sigma_0\right) \\ G_2\left(1^k, r_1, H\left(G_1\left(1^k, r_1, r_2\right), \sigma_0\right)\right), \\ R(G_2(1^m, r_1, \langle a_1, \ldots, a_m \rangle), \sigma_0) \end{matrix} \;\middle|\; r_1, r_2 \sim \{0,1\}^n \right\}$$

with advantage greater then δ.

6.2.1 Password Storage and Offline Attacks

To protect users in the event of a server breach organizations are advised to store salted password hashes — using a cryptographic hash function ($\mathbf{H} : \{0,1\}^* \to \{0,1\}^n$) and a random bit string ($s \in \{0,1\}^*$) [133]. For example, if a user (u) chose the password (pw) the server would store the tuple $(u, s, \mathbf{H}(s, pw))$. Any adversary who has obtained $(u, s, \mathbf{H}(s, pw))$ (e.g., through a server breach) may mount a — fully automated — offline dictionary attack using powerful password crackers like John the Ripper [63]. To verify a guess pw' the adversary simply computes $\mathbf{H}(s, pw')$ and checks to see if this hash matches $\mathbf{H}(s, pw)$.

We assume that an adversary **Adv** who breaches the server can obtain the code for h, as well as the code for any GOTCHAs used in the authentication protocol. Given the code for h and the salt value s the adversary can construct a function

$$\textbf{VerifyHash}(pw') = \begin{cases} 1 & \text{if } \mathbf{H}(s, pw) = \mathbf{H}(s, pw') \\ 0 & \text{otherwise.} \end{cases}.$$

We also allow the adversary to have black box access to a GOTCHA solver (e.g., a human). We use c_H to denote the cost of querying a human and $c_{\mathbf{H}}$ to denote the cost of querying the function **VerifyHash**[6], and we use n_H (resp. $n_{\mathbf{H}}$) to denote the number of queries to the human (resp. **VerifyHash**). Queries to the human GOTCHA solver are much more expensive than queries to the cryptographic hash function ($c_H \gg c_{\mathbf{H}}$) [110]. For technical reasons we limit our analysis to conservative adversaries.

Definition 16. *We say that an adversary* **Adv** *is* conservative *if (1)* **Adv** *uses the cryptographic hash function* **H** *in a black box manner (e.g., the hash function* **H** *and the stored hash value are only used to construct a subroutine* **VerifyHash** *which is then used as a black box by* **Adv** *), (2) The pseudorandom function* **Extract** *is used as a black box, and (3) The adversary only queries a human about challenges generated using a password guess.*

[6]The value of $c_{\mathbf{H}}$ may vary widely depending on the particular cryptographic hash function — it is inexpensive to evaluate SHA1, but BCRYPT [122] may be very expensive to evaluate.

It is reasonable to believe that our adversary is conservative. All existing password crackers (e.g., [63]) use the hash function as a black box, and it is difficult to imagine that the adversary would benefit by querying a human solver about Inkblots that are unrelated to the password.

We use $D \subseteq \{0,1\}^*$ to denote a dictionary of likely guesses that the adversary would like to try,

$$\mathbf{Cost}\,(\mathbf{Adv}, D) = (n_{\mathbf{H}} c_{\mathbf{H}} + n_H c_H)$$

to denote the cost of the queries that the adversary makes to check each guess in D, and $\mathbf{Succeed}\,(\mathbf{Adv}, D, pw)$ to denote the event that the adversary makes a query to **VerifyHash** that returns 1 (e.g., the adversary successfully finds the user's password pw). The adversary might use a computer program to try to solve some of the GOTCHAs — to save cost by not querying a human. However, in this case the adversary might fail to crack the password because the GOTCHA solver found the wrong solution to one of the challenges.

Definition 17. *An adversary* **Adv** *is* (C, γ, D)*-successful if* $\mathbf{Cost}\,(\mathbf{Adv}, D) \leq C$*, and*

$$\Pr_{pw \sim D}\left[\mathbf{Succeed}\,(\mathbf{Adv}, D, pw)\right] \geq \gamma\ .$$

Our attack model is slightly different from the attack model in [51]. They assume that the adversary may ask a limited number of queries to a human challenge solution oracle. Instead we adopt an economic model similar to [30], and assume that the adversary is instead limited by a budget C, which may be used to either evaluate the cryptographic hash function $\mathbf{H}$ or query a human H.

6.3 Inkblot Construction

Our candidate GOTCHA construction is based on Inkblots images. We use algorithm 6.1 to generate inkblot images. Algorithm 6.1 takes as input random bits r_1 and a security parameter k — which specifies the number of Inkblots to output. Algorithm 6.1 makes use of the randomized subroutine

DrawRandomEllipsePairs $(I, t, width, height)$ which draws t pairs of ellipses on the image I with the specified width and height. The first ellipse in each pair is drawn at a random (x, y) coordinate on the left half of the image with a randomly selected color and angle α of rotation, and the second ellipse is mirrored on the right half

Algorithm 6.1 GenerateInkblotImages

Input: Security Parameter 1^k, Random bit string $r_1 \in \{0,1\}^*$.
for $j = 1, \ldots, k$ **do**
  $I_j \leftarrow$ new Blank Image ▷ The following operations only use the random bit string r_1 as a source of randomness
  DrawRandomEllipsePairs $\left(I_j, 150, 60, 60\right)$
  DrawRandomEllipsePairs $\left(I_j, 70, 20, 20\right)$
  DrawRandomEllipsePairs $\left(I_j, 150, 60, 20\right)$
return $\langle I_1, \ldots, I_k \rangle$ ▷ Inkblot Images

of the image. Figure 6.1 is an example of an Inkblot image generated by algorithm 6.1.

Our candidate GOTCHA is given by the pair (G_1, G_2) — algorithms 6.2 and 6.3. G_1 runs algorithm 6.1 to generate k Inkblot images, and then returns these images in permuted order — using a function **GenerateRandomPermutation** (k, r), which generates a random permutation $\pi : [k] \rightarrow [k]$ using random bits r. G_2 also runs algorithm 6.1 to generate k Inkblot images, and then outputs a matching challenge.

Algorithm 6.2 G_1

Input: Security Parameter 1^k, Random bit strings $r_1, r_2 \in \{0,1\}^*$.
$\langle I_1, \ldots, I_k \rangle \leftarrow$ **GenerateInkblotImages** (k, r_1)
$\pi \leftarrow$ **GenerateRandomPermutation** (k, r_2)
return $\langle I_{\pi(1)}, \ldots, I_{\pi(k)} \rangle$

After the Inkblots $\langle I_{\pi(1)}, \ldots, I_{\pi(k)} \rangle$ have been generated, the human user is queried to provide labels $\ell_{\pi(1)}, \ldots, \ell_{\pi(k)}$ where

$$\langle \ell_{\pi(1)}, \ldots, \ell_{\pi(k)} \rangle = H\left(\langle I_{\pi(1)}, \ldots, I_{\pi(k)} \rangle, \sigma_0\right) .$$

In our authentication setting the server would store the labels $\ell_{\pi(1)}, \ldots, \ell_{\pi(k)}$ in permuted order. The final challenge — generated by algorithm 6.3 — is to match the Inkblot images $I_1, \ldots, I_k$ with the user generated labels $\ell_1, ..., \ell_k$ to recover the permutation π.

Observation: Notice that if the random bits provided as input to **GenerateInkblotImages** and **GenerateMatchingChallenge** match that the user

Algorithm 6.3 GenerateMatchingChallenge G_2

Input: Security Parameter 1^k, Random bits $r_1 \in \{0,1\}^*$ and labels $\vec{a} = \langle \ell_{\pi(1)}, \ldots, \ell_{\pi(k)} \rangle$.
$\langle I_1, \ldots, I_k \rangle \leftarrow$ **GenerateInkblotImages** $\left(1^k, r_1\right)$
return $\hat{c}_\pi = (\vec{c}, \vec{a})$ ▷ Matching Challenge

will see the same Inkblot images in the final matching challenge. However, if the random bits do not match (e.g., because the user typed the wrong password in our authentication protocol) then the user will see different Inkblot images. The labels $\ell_1, \ldots, \ell_k$ will be the same in both cases.

6.3.1 GOTCHA Authentication

To illustrate how our GOTCHAs can be used to defend against offline attacks we present the following authentication protocols: **Create Account** (protocol 6.3.1) and **Authenticate** (protocol 6.3.2). Communication in both protocols should take place over a secure channel. Both protocols involve several rounds of interaction between the user and the server. To create a new account the user sends his username/password to the server, the server responds by generating k Inkblot images $I_1, \ldots, I_k$, and the user provides a response $\langle \ell_1, \ldots, \ell_k \rangle = H(\langle I_1, \ldots, I_k \rangle, \sigma_0)$ based on his mental state at the time — the server stores these labels in permuted order $\ell_{\pi(1)}, \ldots, \ell_{\pi(k)}$[7]. To authenticate later the user will have to match these labels with the corresponding inkblot images to recover the permutation π.

In section 6.4 we argue that the adversary who wishes to mount a cost effective offline attack needs to obtain constant feedback from a human. Following [51] we assume that the function **Extract** : $\{0,1\}^* \rightarrow \{0,1\}^n$ is a strong randomness extractor, which can be used to extract random strings from the user's password. Recall that **H** : $\{0,1\}^* \rightarrow \{0,1\}^*$ denotes a cryptographic hash function.

Our protocol could be updated to allow the user to reject challenges he found confusing during account creation in protocol 6.3.1. In this case the server would simply note that the first GOTCHA was confusing and generate a new GOTCHA.

[7]For a general GOTCHA, protocol 6.3.1 would need to have an extra round of communication. The server would send the user the final challenge generated by G_2 and the user would respond with $H(G_2(,), \sigma_0)$. Protocol 6.3.1 takes advantage of the fact that $\pi = H(G_2(,), \sigma_0)$ is already known.

Protocol 6.3.1: Create Account

Security Parameters: k, n.
(User): Select username (u) and password (pw) and send (u, pw) to the server.
(Server): Sends Inkblots $\langle I_1, \ldots, I_k \rangle$ to the user where:
$r' \sim \{0,1\}^n$, $r_1 \leftarrow \textbf{Extract}\,(pw, r')$, $r_2 \sim \{0,1\}^n$ and
$\langle I_1, \ldots, I_k \rangle \leftarrow \textbf{GenerateInkblotImages}\left(1^k, r_1\right)$
(User): Sends responses $\langle \ell_1, ..., \ell_k \rangle$ back to the server where:
$\langle \ell_1, \ldots, \ell_k \rangle \leftarrow H\left(\langle I_1, \ldots, I_k \rangle, \sigma_0\right)$.
(Server): Store the tuple t where t is computed as follows:
Salt: $s \sim \{0,1\}^n$
$\pi \leftarrow \textbf{GenerateRandomPermutation}\,(k, r_2)$.
$h_{pw} \leftarrow \textbf{H}\,(u, s, pw, \pi(1), ..., \pi(k))$
$t \leftarrow \left(u, r', s, h_{pw}, \ell_{\pi(1)}, \ldots, \ell_{\pi(k)}\right)$

Once our user has created an account he can login by following protocol 6.3.2.

Claim 6 says that a legitimate user can successfully authenticate if our Inkblot construction satisfies the usability requirements of a GOTCHA. The proof of claim 6 can be found in appendix 10.1.

Claim 6. *If (G_1, G_2) is a $(\alpha, \beta, \epsilon, \delta, \mu)$-GOTCHA then at least β-fraction of humans can successfully authenticate using protocol 6.3.2 after creating an account using protocol 6.3.1.*

One way to improve usability of our authentication protocol is to increase the neighborhood of acceptably close matchings by increasing α. The disadvantage is that the running time for the server in protocol 6.3.2 increases with the size of α. Claim 7 bounds the time needed to enumerate over all close permutations. The proof of claim 7 can be found in appendix 10.1.

Claim 7. *For all permutations $\pi : [k] \to [k]$ and $\alpha \geq 0$*

$$|\{\pi' \mid d_k(\pi, \pi') \leq \alpha\}| \leq 1 + \sum_{i=2}^{\alpha} \binom{k}{i} i! \,.$$

For example, if the user matches $k = 10$ Inkblots and we want to accept matchings that are off by at most $\alpha = 5$ entries then the server would need to enumerate

Protocol 6.3.2: Authenticate

Security Parameters: k, n.
Usability Parameter: α
(User): Send username (u) and password (pw') — pw' may or may not be correct.
(Server): Sends challenge $\hat{c}$ to the user where $\hat{c}$ is computed as follows:
Find $t = \left(u, r', s, h_{pw}, \ell_{\pi(1)}, \ldots, \ell_{\pi(k)}\right)$
$r'_1 \leftarrow \textbf{Extract}\left(pw', r'\right)$
$\langle I'_1, ..., I'_k \rangle \leftarrow \textbf{GenerateInkblotImages}\left(r'_1, k\right)$
$\hat{c}_\pi \leftarrow \left(\langle I_1, ..., I_k \rangle, \langle \ell_{\pi(1)}, \ldots, \ell_{\pi(k)} \rangle\right)$
(User): Solves $\hat{c}_\pi$ and sends the answer $\pi' = H\left(\hat{c}, \sigma_t\right)$.
(Server):
for all π_0 s.t $d_k\left(\pi_0, \pi'\right) \leq \alpha$ **do**
$h_{pw,0} \leftarrow \mathbf{H}\left(u, s, pw', \pi_0(1), ..., \pi_0(k)\right)$
if $h_{pw,0} = h_{pw}$ **then**
Authenticate
Deny

over at most $36,091$ permutations[8]. Organizations are already advised to use password hash functions like BCRYPT [122] which intentionally designed to be slower than standard cryptographic hash functions — often by a factor of millions. Instead of making the hash function a million times slower to evaluate the server might instead make the hash function a thousand times slower to evaluate and use these extra computation cycles to enumerate over close permutations. The organization's trade-off is between: security, usability and the resources that it needs to invest during the authentication process.

We observe that an adversary mounting an online attack would be naturally rate limited because he would need to solve a GOTCHA for each new guess. Protocol 6.3.2 could also be supplemented with a k-strikes policy — in which a user is locked out for several hours after k incorrect login attempts — if desired.

[8] A more precise calculation reveals that there are exactly $13,264$ permutations s.t. $d_{10}\left(\pi', \pi\right) \leq 5$ and a random permutation π' would only be accepted with probability 3.66×10^{-3}

6.3.2 User Study

To test our candidate GOTCHA construction we conducted an online user study[9]. We recruited participants through Amazon's Mechanical Turk to participate in our study. The study was conducted in two phases. In phase 1 we generated ten random Inkblot images for each participant, and asked each participant to provide labels for their Inkblot images. Participants were advised to use creative titles (e.g., evil clown, frog, lady with poofy dress) because they would not need to remember the exact titles that they used. Participants were paid $1 for completing this first phase. A total of 70 users completed phase 1.

After our participants completed the first phase we waited ten days before asking our participants to return and complete phase 2. During phase 2 we showed each participant the Inkblot images they saw in phase 1 (in a random order) as well as the titles that they created during phase 1 (in alphabetical order). Participants were asked to match the labels with the appropriate image. The purpose of the longer waiting time was to make sure that participants had time to forget their images and their labels. See figure 6.3 for an example of phase 2. Participants were paid an additional $1 for completing phase 2 of the user study. At the beginning of the user study we let participants know that they would be paid during phase 2 even if their answers were not correct. We adopted this policy to discourage cheating (e.g., using screen captures from phase 1 to match the images and the labels) and avoid positively biasing our results.

We measured the time it took each participant to complete phase 1. Our results are summarized in Table 6.1. It is quite likely that some participants left their computer in the middle of the study and returned later to complete the study (e.g., one user took 57.5 minutes to complete the study). While we could not measure time away from the computer, we believe that it is likely that at least 9 of our participants left the computer. Restricting our attention to the other 61 participants who took at most 20 minutes we get an adjusted average completion time of 6.2 minutes.

Fifty-eight of our participants returned to complete phase 2 by taking our matching test. It took these participants 4.5 minutes on average to complete the matching test. Seventeen of our participants correctly matched all ten of their labels, and 69% of participants matched at least 5 out of ten labels correctly. Our results are summarized in Table 6.2.

[9]Our study protocol was approved for exemption by the Institutional Review Board (IRB) at Carnegie Mellon University (IRB Protocol Number: HS13-219).

	Phase 1	Phase 2
Average	9.3	4.5
StdDev	9.6	3
Max	57.5	18.5
Min	1.4	1.6
Average ≤ 20	6.2	N/A

Table 6.1: Completion Times

α-accurate	# participants	$\frac{\text{\# participants}}{58}$	$\frac{\lvert\{\pi' \mid d_{10}(\pi,\pi') \leq \alpha\}\rvert}{10!}$
$\alpha = 0$	17	0.29	2.76×10^{-7}
$\alpha = 2$	22	0.38	1.27×10^{-5}
$\alpha = 3$	26	0.45	7.88×10^{-5}
$\alpha = 4$	34	0.59	6.00×10^{-4}
$\alpha = 5$	40	0.69	3.66×10^{-3}

Table 6.2: Usability Results: Fraction of Participants who would have authenticated with accuracy parameter α

Discussion Our user study provides evidence that our construction is at least $(0, 0.29)$-usable or $(5, 0.69)$-usable. While this means that our Inkblot Matching GOTCHA could be used by a significant fraction of the population to protect their passwords during authentication it also means that the use of our GOTCHA would have to be voluntary so that users who have difficulty won't get locked out of their accounts. Another approach would be to construct different GOTCHAs and allow users to choose which GOTCHA to use during authentication.

Study Incentives: There is evidence that the lack of monetary incentives to perform well on our matching test may have negatively influenced the results (e.g., some participants may have rushed through phase 1 of the study because their payment in round 2 was independent of their ability to match their labels correctly). For example, none of our 18 fastest participants during phase 1 matched all of their labels correctly, and — excluding participants we believe left their computer during phase 1 (e.g., took longer than 20 minutes) — on average participants who

failed to match at least five labels correctly took 2 minutes less time to complete phase 1 than participants who did.

Time: We imagine that some web services may be reluctant to adopt GOTCHAs out of fear driving away customers who don't want to spend time labeling Inkblot images [76]. However, we believe that for many high security applications (e.g., online banking) the extra security benefits of GOTCHAs will outweigh the costs — GOTCHAs might even help a bank keep its customers by providing extra assurance that users' passwords are secure. We are looking at modifying our Inkblot generation algorithm to produce Inkblots which require less "mental effort" to label. In particular could techniques like Perlin Noise [119] be used to generate Inkblots that can be labeled more quickly and matched more accurately?

Accuracy: We believe that the usability of our Inkblot Matching GOTCHA construction can still be improved. One simple way to improve the usability of our GOTCHA construction would be to allow the user to reject Inkblot images that were confusing. We also believe that usability could be improved by providing users with specific strategies for creating their labels (e.g., we found that simple labels like "a voodoo mask" were often mismatched, while more elaborate stories like "A happy guy on the ground, protecting himself from ticklers" were rarely mismatched).

6.3.3 An Open Challenge to the AI Community

We envision a rich interaction between the security community and the artificial intelligence community. To facilitate this interaction we present an open challenge to break our GOTCHA scheme.

Challenge Setup We chose several random passwords $(pw_1, ..., pw_4) \sim \{0, 10^7\}$ and $pw_5 \sim \{0, 10^8\}$. We used a function **GenerateInkblots** $(pw_i, 10)$ to generate ten inkblots $I^i_1, ..., I^i_{10}$ for each password, and we had a human label each inkblot image $\langle \ell^i_1, \ldots, \ell^i_{10} \rangle \leftarrow H\left(\langle I^i_1, \ldots, I^i_{10} \rangle, \sigma_0\right)$. We selected a random permutation $\pi_i : [10] \rightarrow [10]$ for each account, and generated the tuple

$$T_i = \left(s_i, h\left(pw_i, s_i, \pi_i(1), ..., \pi_i(10)\right), \ell^i_{\pi_i(1)}, ..., \ell^i_{\pi_i(10)}\right),$$

where s_i is a randomly selected salt value and h is a cryptographic hash function. We are releasing the source code that we used to generate the Inkblots and evaluate

the hash function **H** along with the tuples $T_1, ..., T_5$ — see `http://www.cs.cmu.edu/~jblocki/GOTCHA-Challenge.html`.

Challenge: Recover each password pw_i.

Approaches One way to accomplish this goal would be to enumerate over every possible password guess pw'_i and evaluate $\mathbf{H}\left(pw'_i, s_i, \pi(1), ..., \pi(10)\right)$ for every possible permutation $\pi : [10] \to [10]$. However, the goal of this challenge is to see if AI techniques can be applied to attack our GOTCHA construction. We intentionally selected our passwords from a smaller space to make the challenge more tractable for AI based attacks, but to discourage participants from trying to brute force over all password/permutation pairs we used BCRYPT (Level 15)[10] — an expensive hash function — to encrypt the passwords. Our implementation allows the Inkblot images to be generated very quickly from a password guess pw' so an AI program that can use the labels in the password file to distinguish between the correct Inkblots returned by **GenerateInkblots** $(pw_i, 10)$ and incorrect Inkblots returned by **GenerateInkblots** $\left(pw'_i, 10\right)$ would be able to quickly dismiss incorrect guesses. Similarly, an AI program which generates a small set of likely permutations for each password guess could allow an attacker to quickly dismiss incorrect guesses.

6.4 Analysis: Cost of Offline Attacks

In this section we argue that our password scheme (protocols 6.3.2 and 6.3.1) significantly mitigates the threat of offline attacks. An informal interpretation of our main technical result — Theorem 21 — is that either (1) the adversary's offline attack is prohibitively expensive (2) there is a good chance that adversary's offline attack will fail, or (3) the underlying GOTCHA construction can be broken. Observe that the security guarantees are still meaningful even if the security parameters ϵ and δ are not negligibly small.

Theorem 21. *Suppose that our user selects his password uniformly at random from a set D (e.g., $pw \xleftarrow{\$} D$) and creates his account using protocol 6.3.1. If algorithms 6.2 and 6.3*

[10]The level parameter specifies the computation complexity of hashing. The amount of work necessary to evaluate the BCRYPT hash function increases exponentially with the level so in our case the work increases by a factor of 2^{15}.

are an (ϵ, δ, μ)*-GOTCHA then no conservative offline adversary is* $\left(C, \gamma + \epsilon + \delta + \frac{n_H}{|D|}, D\right)$*-successful for* $C < \gamma |D| 2^{\mu(k)} c_{\mathbf{H}} + n_H c_H$

Proof of Theorem 21. (Sketch) We use a hybrid argument. An adversary who breaches the server is able to recover the tuple $t = \left(u, r', s, \mathbf{H}\left(u, s, pw, \pi(1), \ldots, \pi(k)\right), \ell_{\pi(1)}, \ldots, \ell_{\pi(k)}\right)$ as well as the code for the cryptographic hash function **H** and the code for our GOTCHA — (G_1, G_2).

1. World 0: W_0 denotes the real world in which the adversary has recovered the tuple

   $$t_0 = \left(u, r', s, \mathbf{H}\left(u, s, pw, \pi(1), \ldots, \pi(k)\right), \ell_{\pi(1)}, \ldots, \ell_{\pi(k)}\right)$$

   as well as the code for the cryptographic hash function **H** and the code for our GOTCHA — (G_1, G_2). Because the adversary **Adv** is conservative it constructs the function

   $$\mathbf{VerifyHash}\left(pw', \pi'\right) = \begin{cases} 1 & \text{if } pw' = pw \text{ and } \pi' = \pi \\ 0 & \text{otherwise.} \end{cases},$$

   and uses **VerifyHash** as a blackbox. We say that **Adv** queries a human H about password pw' if it queries H for $H\left(\mathbf{GenerateInkblotImages}\left(1^k, \mathbf{Extract}\left(pw', r'\right)\right)\right)$, and we let $D' \subseteq D$ denote the set of passwords for which the adversary queries a human.

2. World 1: W_1 denotes a hypothetical world that is similar to W_0 except that **VerifyHash** function the adversary uses as a blackbox is replaced with the following incorrect version

   $$\mathbf{VerifyHash}^1\left(pw', \pi'\right) = \begin{cases} 1 & \text{if } pw' \notin D', pw' = pw \text{ and } \pi' = \pi \\ 0 & \text{otherwise.} \end{cases},$$

   where $D' \subseteq D$ is a subset of passwords which denotes the set of passwords for which the adversary makes queries to a human in the real world.

3. World 2: W_2 denotes a hypothetical world that is similar to W_1 except that $\mathbf{VerifyHash}^1$ function the adversary uses as a blackbox is replaced with the

following incorrect version

$$\mathbf{VerifyHash}^2(pw', \pi') = \begin{cases} 1 & \text{if } \pi' = R\left(G_2\left(1^k, \mathbf{Extract}(pw', r'), \ell_1, \ldots, \ell_k\right)\right), \\ & pw' \notin D' \text{ and } pw' = pw \\ 0 & \text{otherwise.} \end{cases},$$

where R is a distribution with minimum entropy $\mu(k)$ as in definition 15.

4. World 3: W_3 denotes a hypothetical real world which is similar to world 2, except that the labels $\ell_{\pi(1)}, \ldots, \ell_{\pi(k)}$ are replaced with the labels $\ell'_{\pi'(1)}, \ldots, \ell'_{\pi'(k)}$, where $\pi' : [k] \rightarrow [k]$ is a new random permutation, and the labels ℓ'_i are for a completely unrelated set of Inkblot challenges

$$\ell'_1, \ldots, \ell_k \leftarrow H\left(G_1\left(1^k, x_1, x_2\right)\right),$$

where $x_1, x_2 \in \{0,1\}^n$ are freshly chosen random value.

In world 3 it is easy to bound the adversary's probability of success. No adversary is (C, γ, D)-successful for $C < \gamma|D|2^{\mu(k)}c_{\mathbf{H}}$, because the fake Inkblot labels are not correlated with the actual Inblots that were generated with the real password. Our particular adversary cannot be (C, γ, D)-successful for $C < \gamma|D|2^{\mu(k)}c_{\mathbf{H}} + |D'|c_H$. In world 2 the adversary might improve his chances of success by looking at the Inblot labels, but by definition of $(\alpha, \beta, \epsilon, \delta, \mu)$-GOTCHA his chances change by at most δ. In world 1 the adversary might further improve his chances of success, but by definition of $(\alpha, \beta, \epsilon, \delta, \mu)$-GOTCHA his chances improve by at most ϵ. Finally, in world 0 the adversary improves his chances by at most $|D'|/|D|$ by querying the human about passwords in D'. □

6.5 Discussion

We conclude by discussing some key directions for future work.

Improved Inkblots One way to improve our GOTCHA construction would be to improve the Inkblot generation algorithm. One idea is to use random walks to generate Inkblots[129] instead of adding colored ellipses with random sizes, locations and orientations (e.g., Figure 6.1). Figure 6.4a is an example of an Inkblot

produced with random walks. The hope is that users will find it easier to label these Inkblot images. Another potential improvement would be to have the user identify and highlight several specific objects in his Inkblot image(s) during account creation (see Figure 6.4b). When the user authenticates he would be asked to click on each of these objects (e.g., click on the "Bunny Ears"). One advantage is that we may be able to provide equivalent security guarantees by having the user specify two or three specific objects in the Inkblot images instead of requiring the user to label and match ten different Inkblot images. While the author of this thesis has personally found these random walk Inkblots much easier to label than the Inkblots described earlier, we have not yet conducted a user study to empirically evaluate these potential improvements.

Other GOTCHA Constructions Because GOTCHAs allow for human feedback during puzzle generation — unlike HOSPs [51] — our definition potentially opens up a much wider space of potential GOTCHA constructions. One idea might be to have a user rate/rank random items (e.g., movies, activities, foods). By allowing human feedback we could allow the user to dismiss potentially confusing items (e.g., movies he hasn't seen, foods about which he has no strong opinion). There is some evidence that this approach could provide security (e.g., Narayanan and Shmatikov showed that a Netflix user can often be uniquely identified from a few movie ratings [114].).

Obfuscating CAPTCHAs If it were possible to efficiently obfuscate programs then it would be easy to construct GOTCHAs from CAPTCHAs (e.g., just obfuscate a program that returns the CAPTCHA without the answer). Recently, Garg et al. showed how to obfuscate arbitrary programs [80] using multilinear maps[11]. Unfortunately, their obfuscator is not yet efficient enough for practical use. However, it may be still be possible to find an efficient way to obfuscate our particular CAPTCHA program.

[11]While Barak et al. [24] showed that there is no general program obfuscator, their impossibility result was for a stronger notion of obfuscation called blackbox obfuscation, which requires that any adversary with access to an obfuscated program can be simulated with only blackbox access to the same program. Garg et al. [80] used a weaker notion of obfuscation known as "indistinguishability obfuscation," which (loosely) only guarantees that the adversary cannot distinguish between the obfuscations of two circuits which compute the same function.

Exploiting The Power of Interaction Can interaction be exploited and used to improve security or usability in human-authentication? While interaction is an incredibly powerful tool in computer security (e.g., nonces [128], zero-knowledge proofs [85], secure multiparty computation [163]) and in complexity theory[12], human authentication typically does not exploit interaction with the human (e.g., the user simply enters his password). We view the idea behind HOSPs and GOTCHAs — exploiting interaction to mitigate the threat of offline attacks — as a positive step in this direction. Could interaction be exploited to reduce memory burden on the user by allowing a user to reuse the same secret to authenticate to multiple different servers? The human-authentication protocol of Hopper et al. [91] — based on the noisy parity problem — could be used by a human to repeatedly authenticate over an insecure channel. Unfortunately, the protocol is slow and tedious for a human to execute, and it can be broken if the adversary is able to ask adaptive parity queries [103].

[12] A polynomial time verifier can verify **PSPACE**-complete languages by interacting with a powerful prover [138], by contrast the same verifier can only check proofs of **NP**-Complete languages without interaction.

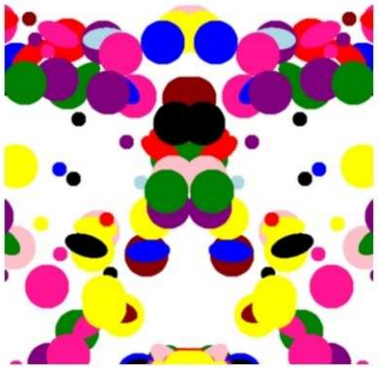
Using Your Imagination Enter a title for the Image Below *

Aunt Martha wants to squeeze your cheeks

Figure 6.2: Phase 1

10. ~~wreck it ralph~~

Which title did you use for the Image Above? *
fruit with flies around it



1. ~~Bane from Batman~~
2. ~~Large Dog with Giant Ears~~
3. ~~Old Cow Guy~~
4. ~~Unhappy little guy in the center~~
5. ~~cartoon guy that looks a bit like a panda~~
6. ~~eyes crossed~~
7. ~~fruit with flies around it~~
8. **guy on a hang glider**
9. ~~guy with head and eyes in the center of his steroid body~~
10. ~~wreck it ralph~~

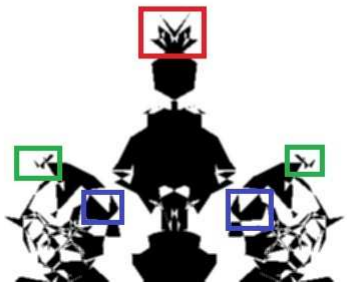
Which title did you use for the Image Above? *
guy on a hang glider

Figure 6.3: Phase 2

(a) Example Random Walk Inkblot.

Two Bicyclists Pedal Away from a Tall bunny in a Suit

Bunny Ears
Bicyclists Eyes
Bicyclists Butt



(b) Example Inkblot with Labels.

Figure 6.4: Random Walk Inkblots

Chapter 7

Appendix: Naturally Rehearsing Passwords

7.1 Missing Proofs

Before we prove Lemma 1 and Theorem 1 we first formally define a Poisson arrival process (Definition 18) and state a few basic facts about a Poisson arrival process.

Definition 18. *Given* $0 \leq t_1 \leq t_2$ *we use* $\mathbf{Visits}_i(t_1, t_2) = \left|\left\{j \,\middle|\, \tau_j^i \in (t_1, t_2)\right\}\right|$ *to denote the number of times the user visits account* A_i *during the interval* (t_1, t_2)*. We say that* $\mathbf{Visits}_i(t_1, t_2)$ *represents a Poisson arrival process with parameter* λ_i *if*

$$\Pr\left[\mathbf{Visits}_i(t_1, t_2) = k\right] = e^{\lambda_i(t_1 - t_2)} \frac{\left(\lambda_i(t_2 - t_1)\right)^k}{k!},$$

and the random variables $\mathbf{Visits}_i(t_1, t_2)$ *and* $\mathbf{Visits}_i(t_3, t_4)$ *are independent whenever* $0 \leq t_1 \leq t_2 \leq t_3 \leq t_4$*.*

Fact 4 says that $1/\lambda_i$ represents the average inter-visitation time for account A_i whose visitation schedule follows a Poisson arrival process with parameter λ_i.

Fact 4. *If* $\mathbf{Visits}_i(t_1, t_2)$ *represents a Poisson arrival process with parameter* λ_i *then* $1/\lambda_i$ *represents the average inter-visitation time for account* A_i*. More formally, for all* $j > 0$ *we have* $\mathbb{E}\left[\tau_j^i - \tau_{j-1}^i\right] = \frac{1}{\lambda_i}$.

Fact 5 says that the sum of two Poisson arrival processes with parameters λ_1 and λ_2 is itself a Poisson arrival process with parameter $(\lambda_1 + \lambda_2)$.

Fact 5. *If* $\mathbf{Visits}_i(t_1, t_2)$ *represents a Poisson arrival process with parameter* λ_i *and* $\mathbf{Visits}_j(t_1, t_2)$ *represents an independent Poisson arrival process with parameter* λ_j *then* $\mathbf{Visits}_{i,j}(t_1, t_2) \doteq \mathbf{Visits}_i(t_1, t_2) + \mathbf{Visits}_j(t_1, t_2)$ *is a Poisson arrival process with parameter* $\left(\lambda_i + \lambda_j\right)$*.*

Reminder of Lemma 1. *Let* $S_{\hat{c}} = \{i \mid \hat{c} \in c_i\}$ *and let* $\lambda_{\hat{c}} = \sum_{i \in S_{\hat{c}}} \lambda_i$ *then the probability that the cue* $\hat{c}$ *is not naturally rehearsed during time interval* $[a, b]$ *is* $\exp\left(-\lambda_{\hat{c}}(b - a)\right)$*.*

Proof of Lemma 1. Let $N(t_1, t_2) = \left|\left\{\tau_k^i \,\middle|\, i \in S_{\hat{c}} \wedge t_1 \leq \tau_k^i \leq t_2\right\}\right|$ denote the number of times the cue $\hat{c}$ is rehearsed during the interval $[t_1, t_2]$. Notice that the rehearsal requirement $[a, b]$ is naturally satisfied if and only if $N(a, b) > 0$. By Fact 5, $N(t_1, t_2) = \sum_{i \in S_{\hat{c}}} \mathbf{Visits}_i(t_1, t_2)$ describes a Poisson arrival process with parameter $\lambda_{\hat{c}} = \sum_{i \in S_{\hat{c}}} \lambda_i$ so we can apply the definition of a Poisson arrival process to get

$$\Pr\left[N\left(a,b\right)=0\right]=\exp\left(-\lambda_{\hat{c}}\left(b-a\right)\right)\ .$$

□

Reminder of Theorem 1. *Let* $i_{\hat{c}}* = \left(\arg\max_x t_x^{\hat{c}} < t\right) - 1$ *then*

$$\mathbb{E}\left[XR_t\right]=\sum_{\hat{c}\in C}\sum_{i=0}^{i_{\hat{c}}*}\exp\left(-\left(\sum_{j:\hat{c}\in c_j}\lambda_j\right)\left(t_{i+1}^{\hat{c}}-t_i^{\hat{c}}\right)\right)$$

Proof of Theorem 1. Let $S_{\hat{c}}=\{i\,|\,\hat{c}\in c_i\}$ and let $V_{a,b}\left(\hat{c}\right)$ be the indicator for the event that $\exists i\in S_{\hat{c}}, k\in\mathbb{N}.\tau_k^i\in[a,b]$ (e.g., cue $\hat{c}$ is rehearsed naturally during the time interval $[a,b]$). Then by linearity of expectation

$$\mathbb{E}\left[XR_{t,\hat{c}}\right]=\sum_{i=0}^{i_{\hat{c}}*}\left(1-\mathbb{E}\left[V_{t_i,t_{i+1}}\left(\hat{c}\right)\right]\right)\ ,$$

where

$$\mathbb{E}\left[1-V_{t_i,t_{i+1}}\left(\hat{c}\right)\right]=\sum_{i=0}^{i_{\hat{c}}*}\exp\left(-\left(\sum_{j:\hat{c}\in c_j}\lambda_j\right)\left(t_{i+1}^{\hat{c}}-t_i^{\hat{c}}\right)\right)\ ,$$

by Lemma 1. The result follows immediately from linearity of expectation. □

Reminder of Theorem 4. *It is NP-Hard to approximate* **Min-Rehearsal** *within a constant factor.*

Proof of Theorem 4. Let $\gamma>0$ be any constant. We prove that it is NP-Hard to even γ-approximate **Min-Rehearsal**. The reduction is from set cover.

Set Cover Instance: Sets $S_1,...,S_n$ and universe $U=\bigcup_i S_i$. A set cover is a set $S\subseteq\{1,...,n\}$ such that $\bigcup_{i\in S}S_i=U$.
Question: Is there a set cover of size k?

Given a set cover instance, we set $C=U$ create public cues $c_1,...,c_m\subseteq C$ for each account by setting $c_i=S_i$. We set the following visitation schedule

$$\lambda_i=\frac{\ln\left(\gamma\left|U\right|\left(\max_{\hat{c}\in C}i_{\hat{c}}^*\right)\right)}{\min_{j,\hat{c}}\left(t_{j+1}^{\hat{c}}-t_j^{\hat{c}}\right)}\ ,$$

for $i = 1, \ldots, k$ and $\lambda_{k+1}, ..., \lambda_n = 0$. There are two cases: (1) There is a set cover $S = \{x_1, ..., x_k\} \subseteq \{1, ..., n\}$ of size k. If we assign $\pi(i) = x_i$ for each $i \leq k$ then for each base cue $\hat{c} \in U$ we have

$$\lambda_{\hat{c}} = \sum_{i:\hat{c} \in S_i} \lambda_i \geq \lambda_1 \, .$$

Applying Theorem 1 we get

$$\begin{aligned}
\mathbb{E}\left[XR_t\right] &= = \sum_{\hat{c} \in C} \sum_{i=0}^{i_{\hat{c}}^*} \exp\left(- \left(t_{i+1}^{\hat{c}} - t_i^{\hat{c}} \right) \sum_{i:\hat{c} \in S_{\pi(i)}} \lambda_i \right) \\
&\leq |C| \left(\max_{\hat{c} \in C} i_{\hat{c}}^* \right) \exp\left(- \left(t_{i+1}^{\hat{c}} - t_i^{\hat{c}} \right) \frac{\ln\left(\gamma \left| U \right| \left(\max_{\hat{c} \in C} i_{\hat{c}}^* \right) \right)}{\left(\min_{j,\hat{c}} \left(t_{i+1}^{\hat{c}} - t_i^{\hat{c}} \right) \right)} \right) \\
&\leq |U| \left(\max_{\hat{c} \in C} i_{\hat{c}}^* \right) \exp\left(- \ln\left(\gamma \left| U \right| \left(\max_{\hat{c} \in C} i_{\hat{c}}^* \right) \right) \right) \\
&\leq |U| \left(\max_{\hat{c} \in C} i_{\hat{c}}^* \right) \frac{1}{\gamma \left| U \right| \left(\max_{\hat{c} \in C} i_{\hat{c}}^* \right)} \\
&= \frac{1}{\gamma} \, .
\end{aligned}$$

(2) If there is no set cover of size k. Given a mapping π we let $S_\pi = \{i \mid \exists j \leq k . \pi(j) = i\}$ be the set of all public cues visited with frequency at least λ_1. Because $|S_\pi| = k$, S_π cannot be a set cover and there exists some $\hat{c}_j \in C$ which is never visited so no rehearsal requirements are satisfied naturally.

$$\mathbb{E}\left[XR_t\right] = \sum_{\hat{c} \in C} \sum_{i=0}^{i_{\hat{c}}^*} \exp\left(- \left(t_{i+1}^{\hat{c}} - t_i^{\hat{c}} \right) \sum_{i:\hat{c} \in S_{\pi(i)}} \lambda_i \right) \geq \sum_{i=0}^{i_{\hat{c}_j}^*} 1 \geq 1 \, .$$

□

Reminder of Theorem 2. *Let $\{c_1, \ldots, c_m\}$ be a (n, ℓ, γ)-sharing set of m public cues produced by the password management scheme $\mathcal{G}_m$. If each $a_i \in \mathcal{AS}$ is chosen uniformly at random then $\mathcal{G}_m$ satisfies (q, δ, m, s, r, h)-security for $\delta \leq \frac{q}{|\mathcal{AS}|^{\ell - \gamma r}}$ and any h.*

Proof of Theorem 2. Recall that S (resp. S') denotes the set of accounts that the adversary selected for plaintext recovery attacks. Let $\left(k, p'_k\right)$ denote the adversary's final answer. We can assume that $k \notin S$ because the adversary cannot win by

outputting a password he obtained earlier in the game during a plaintext recovery attack. We define

$$U_k = c_k - \left\{ \hat{c} \,\middle|\, \exists j \in S.\, \hat{c} \in c_j \right\} ,$$

to be the set of all uncompromised base cues in c_k. Observe that

$$\begin{aligned} |U_k| &\geq |c_k| - \sum_{j \in S} \left| c_k \bigcap c_j \right| \\ &\geq \ell - \sum_{j \in S} \gamma \\ &\geq \ell - r\gamma , \end{aligned}$$

by definition 5 of a (n, ℓ, γ)-sharing family of public cues.

For each, $\hat{c} \in U_k$ the corresponding association $\hat{a}$ was chosen uniformly at random from $\mathcal{AS}$. We can upper bound $B_{\mathcal{A}}$ — the bad event that the adversary $\mathcal{A}$ guesses (k, p_k) in at most q attempts.

$$\Pr[B_{\mathcal{A}}] \leq \frac{q}{|\mathcal{AS}|^{|U_k|}} \leq \frac{q}{|\mathcal{AS}|^{\ell - r\gamma}} .$$

□

Reminder of Theorem 3. *Suppose that $\mathcal{S} = \{S_1, ..., S_m\}$ is a (n, ℓ, γ)-sharing set family of size m then $m \leq \binom{n}{\gamma+1} \Big/ \binom{\ell}{\gamma+1}$.*

Proof of Theorem 3. Let $S \in \mathcal{S}$ be given, and let $T \subseteq S$ be subset of size $|T| = \gamma + 1$. By definition of (n, ℓ, γ)-sharing we cannot have $T \subseteq S'$ for any other set $S' \in \mathcal{S} - S$. In total there are $\binom{n}{\gamma+1}$ subsets of $[n]$ of size $\gamma + 1$ and each $S \in \mathcal{S}$ contains $\binom{\ell}{\gamma+1}$ of them. The result follows from the pigeonhole principle. □

7.2 Varying the Association Strength Constant

In Tables 2.2 and 2.3 we used the same association strength constant for each scheme $\sigma = 1$ — though we expect that σ will be higher for schemes like *Shared Cues* that use strong mnemonic techniques. We explore the effect of σ on $\mathbb{E}[XR_{t,c}]$ under various values of the natural rehearsal rate λ. Table 7.1 shows the values $\mathbb{E}[XR_{t,c}]$ under the expanding rehearsal assumption for $\sigma \in \{0.1.0.5, 1, 2\}$. We

λ (visits/days)	2	1	$\frac{1}{3}$	$\frac{1}{7}$	$\frac{1}{31}$
$\sigma = 0.1$	0.686669	2.42166	5.7746	7.43555	8.61931
$\sigma = 0.5$	0.216598	0.827594	2.75627	4.73269	7.54973
$\sigma = 1$	0.153986	0.521866	1.56788	2.61413	4.65353
$\sigma = 2$	0.135671	0.386195	0.984956	1.5334	2.57117

Table 7.1: Expanding Rehearsal Assumption: $\mathbb{E}[XR_{365,c}]$ vs. λ_c and σ

λ (visits/days)	2	1	$\frac{1}{3}$	$\frac{1}{7}$	$\frac{1}{31}$
$\sigma = 1$	49.5327	134.644	262.25	317.277	354.382
$\sigma = 3$	0.3024	6.074	44.8813	79.4756	110.747
$\sigma = 7$	0.0000	0.0483297	5.13951	19.4976	42.2872
$\sigma = 31$	0.000	0.0000	0.0004	0.1432	4.4146

Table 7.2: Constant Rehearsal Assumption: $\mathbb{E}[XR_{365,c}]$ vs. λ_c and σ

consider the following natural rehearsal rates: $\lambda = 1$ (e.g., naturally rehearsed daily), $\lambda = 3$, $\lambda = 7$ (e.g., naturally rehearsed weekly), $\lambda = 31$ (e.g., naturally rehearsed monthly).

Table 7.2 shows the values $\mathbb{E}[XR_{t,c}]$ under the constant rehearsal assumption for $\sigma \in \{1, 3, 7, 31\}$ (e.g., if $\sigma = 7$ then the cue must be rehearsed every week).

7.3 Baseline Password Management Schemes

In this section we formalize our baseline password management schemes: *Reuse Weak* (Algorithm 7.1), *Reuse Strong* (Algorithm 7.2),*Lifehacker* (Algorithm 7.3) and *Strong Random and Independent* (Algorithm 7.4). The first three schemes (*Reuse Weak*,*Reuse Strong*,*Lifehacker*) are easy to use, but only satisfy weak security guarantees. *Strong Random and Independent* provides very strong security guarantees, but is highly difficult to use.

Vague instructions and strategies do not constitute a password management scheme because it is unclear what the resulting distribution over $\mathcal{P}$ looks like. When given such vague instructions (e.g., "pick a random sentence and use the

first letter of each word") people tend to behave predictably (e.g., picking a popular phrase from a movie or book). For example, when people are required to add special symbols to their passwords they tend to use a small set of random symbols and add them in predictable places (e.g., end of the password) [101]. Most password advice provides only vague instructions. However, many of these vague strategies can be tweaked to yield formal password management schemes. *Reuse Weak*, *Reuse Strong*, and *Lifehacker* are formalizations of popular password management strategies.

Each of these password management schemes ignores the visitation schedule $\lambda_1, ..., \lambda_m$. None of the schemes use cues explicitly. However, the user always has an implicitly cue when he tries to login. For example, the implicit cue in *Reuse Weak* might be "that word that I always use as my password." We use four implicit cues for *Reuse Strong* to represent the use of four separate words (chunks [108]). These implicit cues are shared across all accounts — a user rehearses the implicit association(s) when he logs into any of his accounts.

Algorithm 7.1 *Reuse Weak* $\mathcal{G}_m$

Input: Background knowledge $k \in \mathcal{K}$ about the user. Random bits b, $\lambda_1, ..., \lambda_m$.

Random Word: $w \xleftarrow{\$} D_{20,000}$. ▷ Select w uniformly at random from a dictionary of 20,000 words.

for $i = 1 \rightarrow m$ **do**

$p_i \leftarrow w$

$c_i \leftarrow \{\text{'}word\text{'}\}$

return $(p_1, c_1), ..., (p_m, c_m)$

User: Memorizes and rehearses the cue-association pairs $(\text{'}word\text{'}, p_i)$ for each account A_i by following the rehearsal schedule (e.g., CR or ER).

Lifehacker uses a derivation rule to get a different password for each account. There is no explicit cue to help the user remember the derivation rule, but the implicit cue (e.g., "that derivation rule I *always* use when I make passwords") is shared across every account — the user rehearses the derivation rule every time he logs into one of his accounts. There are four base cues — three for the words, one for the derivation rule.

Strong Random and Independent also uses implicit cues (e.g., the account name A_i), which are *not* shared across accounts so the only way to naturally rehearse the association (A_i, p_i) is to visit account A_i.

Algorithm 7.2 *Reuse Strong $\mathcal{G}_m$*

Input: Background knowledge $k \in \mathcal{K}$ about the user. Random bits $b, \lambda_1, ..., \lambda_m$.
for $i = 1 \to 4$ **do**
  Random Word: $w_i \xleftarrow{\$} D_{20,000}$.
for $i = 1 \to m$ **do**
  $p_i \leftarrow w_1 w_2 w_3 w_4$
  $c_i \leftarrow \{(\text{'}Word\text{'}, j) \mid j \in [4]\}$
return $(p_1, c_1), ..., (p_m, c_m)$
User: Memorizes and rehearses the cue-association pairs $\left((\text{'}Word\text{'}, j), w_j\right)$ for each $j \in [4]$ by following the rehearsal schedule (e.g., CR or ER).

Algorithm 7.3 *Lifehacker $\mathcal{G}_m$*

Input: Background knowledge $k \in \mathcal{K}$ about the user. Random bits $b, \lambda_1, ..., \lambda_m$.
for $i = 1 \to 3$ **do**
  Random Word: $w_i \xleftarrow{\$} D_{20,000}$.
Derivation Rule: $d \xleftarrow{\$} DerivRules$. ▷ *DerivRules* is a set of 50 simple derivation rules to map the name of a site A_i to a string $d(A_i)$ (e.g., use the first three consonants of A_i).
for $i = 1 \to m$ **do**
  $p_i \leftarrow w_1 w_2 w_3 d(A_i)$
  $c_i \leftarrow \{(\text{'}Word\text{'}, j) \mid j \in [3]\} \cup \{\text{'}Rule\text{'}\}$
return $(p_1, c_1), ..., (p_m, c_m)$
User: Memorizes and rehearses the cue-association pairs $\left((\text{'}Word\text{'}, j), w_j\right)$ for each $j \in [3]$ and $(\text{'}Rule\text{'}, d)$ by following the rehearsal schedule (e.g., CR or ER).

Algorithm 7.4 *Strong Random and Independent $\mathcal{G}_m$*

Input: Background knowledge $k \in \mathcal{K}$ about the user. Random bits $b, \lambda_1, ..., \lambda_m$.
for $i = 1 \to m$ **do**
  for $j = 1 \to 4$ **do**
    Random Word: $w_j^i \xleftarrow{\$} D_{20,000}$.
  $p_i \leftarrow w_1^i w_2^i w_3^i w_4^i$
  $c_i \leftarrow \{(A_i, j) \mid j \in [4]\}$
return $(p_1, c_1), ..., (p_m, c_m)$
User: Memorizes and rehearses the association $\left((A_i, j), w_j^i\right)$ for each account A_i and $j \in [4]$ by following the rehearsal schedule (e.g., CR or ER).

7.3.1 Security Of Baseline Password Management Schemes

Reuse Weak is not $(q_{\$1}, \delta, m, s, 0, 1)$-secure for any $\delta < 1$ — an adversary who is only willing to spend \$1 on password cracking will still be able to crack the user's passwords! While *Reuse Weak* does provide some security guarantees against online attacks they are not very strong. For example, *Reuse Weak* is not even $(q_{\$1}, .01, 100, 3, 0, 0)$-secure because an adversary who executes an online attack can succeed in breaking into at least one of the user's 100 accounts with probability at least .01 — even if all accounts implement a 3-strike limit. If the adversary recovers any of the user's passwords ($r > 0$) then all security guarantees break down.

Reuse Strong is slightly more secure. It satisfies $(q_{\$10^6}, 3.222 \times 10^{-7}, m, s, 0, m)$-security meaning that with high probability the adversary who has not been able to recover any of the user's passwords will not even be able to mount a successful offline attack against against the user. However, *Reuse Strong* is not $(q, \delta, m, s, 1, 0)$-secure — if the adversary is able to recover just one password p_i for any account A_i then the adversary will be able to compromise all of the user's accounts.

Lifehacker is supposed to limit the damage of a recovery attack by using a derived string at the end of each password. However, in our security model the adversary knows that the user used *Lifehacker* to generate his passwords. The original article [4] instructs users to pick a simple derivation rule (e.g., "use the first three consonants in the site name"). Because this instruction is vague we assume that there are a set of 50 derivation rules and that one is selected at random. If the adversary sees a password $p_i = w_1 w_2 w_3 d(A_i)$ for account A_i then he can immediately infer the base password $b = w_1 w_2 w_3$, and the adversary needs at most 50 guesses to discover one of the user's passwords[1] — so if $(m-1)s \geq 50$ then *Lifehacker* is not $(q, \delta, m, s, 1, 0)$-secure for any values of δ, q. *Lifehacker* is $(q_{\$10^6}, 1.29 \times 10^{-4}, m, s, 0, m)$-secure — it defends against offline and online attacks in the absence of recovery attacks.

Strong Random and Independent is highly secure! It satisfies $(q_{\$10^6}, 3.222 \times 10^{-7}, m, s, \alpha, m)$-security for any $\alpha \leq m$. This means that even after the adversary learns many of the user's passwords he will fail to crack any other password with high probability. Unfortunately, *Strong Random and Independent*is very difficult to use.

[1]In fact the adversary most likely needs far fewer guesses. He can immediately eliminate any derivation rule $\hat{d}$ s.t. $\hat{d}(A_i) \neq d(A_i)$. Most likely this will include almost all derivation rules besides the correct one.

7.3.2 Usability of Baseline Schemes

Usability results for *Lifehacker* and *Strong Random and Independent* can be found in Table 2.2 of the paper. We evaluate usability using the formula from Theorem 1. We present our results for the Very Active, Typical, Occasional and Infrequent users under both sufficient rehearsal assumptions CR and ER — with association strength $\sigma = 1$. The usability results for *ReuseStrong* are identical to *Lifehacker*, because they have the same number of cues and each cue is rehearsed anytime the user visits any account A_i. Similarly, the usability results for *ReuseWeak* are better by a factor of 4 (e.g., because there is only one cue-association pair to rehearse and the natural rehearsal rates are identical).

7.3.3 Sources of Randomness

Popular password advice tends to be informal — the user is instructed to select a character/number/digit/word, but is not told how to do this. Certainly one reason why people do not select random passwords is because they worry about forgetting their password [102]. However, even if the user is told to select a the character uniformly at random it is still impossible to make any formal security guarantees without understanding the entropy of a humanly generated random sequence. We have difficulty consciously generating a random sequence of numbers even when they are not trying to construct a memorable sequence [154] [111] [74].

This does not rule out the possibility that human generated random sequence could provide a weak source of entropy [88] — which could be used to extract a truly random sequence with computer assistance [65, 137]. We envision a computer program being used to generate random words from a dictionary or random stories (e.g., Person-Action-Object stories) for the user to memorize. The source of randomness could come from the computer itself or it could be extracted from a human source (e.g., a user randomly typing on the keyboard).

7.4 Other Measures of Password Strength

In this section we discuss other security metrics (e.g., entropy, minimum entropy, password strength meters, α-guesswork) and their relationship to our security model.

Our security model is fundamentally different from metrics like guessing entropy (e.g., How many guesses does an adversary need to guess all of passwords in a dataset [107]?) and partial guessing entropy (e.g., How many guesses does the adversary need to crack α-fraction of the passwords in a dataset [39, 121]? How many passwords can the adversary break with β guesses per account [45]?), which take the perspective of a system administrator who is trying to protect many users with password protected accounts on his server. For example, a system administrator who wants to evaluate the security effects of a a new password composition policy may be interested in knowing what fraction of user accounts are vulnerable to offline attacks. By contrast, our security model takes the perspective of the user who has many different password protected accounts. This user wants to evaluate the security of various password management schemes that he could choose to adopt.

Our threat model is also strictly stronger than the threat models behind metrics like α-guesswork because we consider targeted adversary attacks from an adversary who may have already compromised some of the user's accounts.

Password strength meters can provide useful feedback to a user (e.g., they rule out some insecure password management schemes). However, password strength meters are insufficient for our setting for several reasons: (1) They fail to rule out some weak passwords, and (2) They cannot take correlations between a user's passwords (e.g., Is the user reusing the same password?) into account. (3) They do not model the adversaries background knowledge about the user (e.g., Does the adversary know the user's birth date or favorite hobbies?). Entropy is bad measure of security for the same reasons. While minimum entropy fixes some of these problems, minimum entropy still does not address problem 2 — minimum entropy does not deal with correlated user passwords.

7.4.1 Password Strength Meters

Password strength meters use simple heuristics (e.g., length, character set) to estimate the entropy of a password. A password strength meter can provide useful feedback to the user by warning the user when he picks passwords that are easy to guess. However, password strength meters can also give users a false sense of confidence (e.g., 'mmmmmmmmmmmmmmmmmmmmmmmmmmm' is clearly predictable, but is ranked 'Best' by some meters [2] — see Figure 7.1 [2]). A password like *Mm1!Mm1!Mm1!Mm1!Mm1!Mm1!* would be rated as very secure by almost any password strength meter because it is long, it uses upper case and

lower case letters and it includes a special symbol (!). However, the password is based on a very simple repeated pattern and has low entropy (e.g., it could be compressed easily). A password strength meter cannot guarantee that a password is secure because (1) It does not know whether or not the user has already used this password (or a very similar password) somewhere else (2) It does not know if the user is basing his password on personal knowledge (e.g., wife's birthday) (3) It does not know what background knowledge the adversary might have about the user (e.g., does the adversary know the user's wife's birthday).

Figure 7.1: mmmmmmmmmmmmmmmmmmmmmmmmmmmm: sounds delicious, but is it really a strong password?

7.4.2 Entropy

Entropy [139] can be used to measure the average number of guesses an adversary would need to guess a password chosen at random from a distribution D over passwords

$$H(D) = \sum_x \Pr[x \mid D] \log_2 \left(\frac{1}{\Pr[x \mid D]} \right) .$$

While entropy has been a commonly used information theoretic measure of password strength [101, 109], it is not always a good indicator of password strength [107]. For example, consider the following distributions over binary passwords D_1 and D_2:

$$D_1(n) = \begin{cases} 1^{n-1} & \text{with probability } 1/2, \\ x \in \{0,1\}^{2n-2} & \text{with probability } 2^{-2n+1}. \end{cases}$$

$$D_2(n) = x \in \{0,1\}^n \quad \text{with probability } 2^{-n} .$$

While there is no difference in the entropy of both generators

$$H(D_1(n)) = \frac{1}{2}\log_2\left(\frac{1}{1/2}\right) + \sum_x 2^{-2n+1}\log_2\left(2^{2n-1}\right) = \frac{1}{2} + \frac{2n-1}{2} = n = H(D_2(n)) \ ,$$

D_1 and D_2 are by no means equivalent from a security standpoint! After just one guess an adversary can successfully recover the password generated by D_1 with probability $\geq \frac{1}{2}$! By contrast an adversary would need at least 2^{n-1} guesses to recover the password generated by D_2 with probability $\geq \frac{1}{2}$.

7.4.3 Minimum Entropy

If we instead consider the minimum entropy

$$H_{\min}(G) = \min_x \log_2\left(\frac{1}{\Pr[x|\,G]}\right) \ ,$$

of both generators we get a different story.

$$H_{\min}(D_1(n)) = \log_2\left(\frac{1}{1/2}\right) = 1 \ll H_{\min}(D_2(n)) = \log_2(2^n) = n \ .$$

High minimum entropy guarantees with high probability any adversary will fail to guess the password even after many guesses. However, even minimum entropy is not a great measure of security when the user is managing multiple passwords because it does not consider correlations between passwords. Suppose for example that each user needs two passwords (x_1, x_2) and again consider two password distributions D_1 and D_2 redefined below:

$$D_1(n) = (x, x) \ \text{ with probability } 2^{-2n} \text{ for each } x \in \{0,1\}^{2n} \ .$$

$$D_2(n) = (x_1, x_2) \ \text{ with probability } 2^{-2n} \text{ for each } (x_1, x_2) \in \{0,1\}^n \times \{0,1\}^n \ .$$

The min-entropy of both generators is the same ($2n$). However, D_1 provides no security guarantees against a recovery attack — any adversary who knows x_2 can immediately guess x_1. However, when the passwords are chosen from D_2 an adversary who knows x_2 has no advantage in guessing x_1.

Benefit (B)	BCRYPT	MD5	SHA1
q_B	$B\left(5.155 \times 10^4\right)$	$B\left(9.1 \times 10^9\right)$	$B \times 10^{10}$

Table 7.3: Upper Bound: q_B for BCRYPT, MD5 and SHA1

7.5 Economics

In this section we discuss how the parameter q_B - our upper bound on the total number of adversary guesses - could be selected. Our upper bound is based on the economic cost of guessing. Guessing is not free! The basic premise is that the adversary will not try more than $q_{\$B}$ guesses to break into an account if his maximum benefit from the attack is $\$B$. The cost of guessing is influenced by several factors including the cost of renting or buying computing equipment (e.g., Cray, GPUs), the cost of electricity to run the computers and the complexity of the cryptographic hash function used to encrypt the password. The value of $q_{\$B}$ depends greatly on the specific choice of the cryptographic hash function. Table 7.3 shows the values of $q_{\$B}$ we computed for the BCRYPT, SHA1 and MD5 hash functions.

7.5.1 Password Storage

There are many cryptographic hash functions that a company might use (e.g., MD5, SHA1, SHA2, BCRYPT) to store passwords. Some hash functions like BCRYPT [122] were designed specifically with passwords in mind — BCRYPT was intentionally designed to be slow to compute (e.g., to limit the power of an adversary's offline attack). The BCRYPT hash function takes a parameter which allows the programmer to specify how slow the hash computation should be — we used L12 in our experiments. By contrast, MD5, SHA1 and SHA2 were designed for fast hardware computation. Unfortunately, SHA1 and MD5 are more commonly used to hash passwords [13]. In economic terms, hash functions like BCRYPT increase the adversary's cost of guessing. We use $F_{\mathbf{H}}$ to denote number of times that the hash function $\mathbf{H}$ can be computed in one hour on a 1 GHz processor. We estimated $F_{\mathbf{H}}$ experimentally on a Dell Optiplex 960 computer for BCRYPT, MD5 and SHA1 (Table 7.4) — as expected the value of $F_{\mathbf{H}}$ is much lower for BCRYPT than SHA1 and MD5.

The rainbow table attack can be used to significantly speed up password crack-

ing attempts after the adversary performs some precomputation [117]. Rainbow table attacks can be prevented by a practice known as password salting (e.g., instead of storing the cryptographic hash of the password $\mathbf{H}(p)$ the a server stores $(\mathbf{H}(p, r), r)$ for a random string r) [16].

Note: , In reality, many companies do not salt their passwords [9, 13] (in fact some do not even hash them [5]). In this paper, we assume that passwords are stored properly (e.g., salted and hashed), and we use optimistic estimates for $q_{\$B}$ based on the BCRYPT hash function. To justify these decisions we observe that a user could easily ensure that his passwords are salted and encrypted with a slow hash function f (e.g., BCRYPT [122]) by using $f(U, A_i, p_i)$ as his password for account i - where U is the username and A_i is the name of account i. Because the function f is not a secret, its code could be stored locally on any machine being used or publicly on the cloud.

7.5.2 Attack Cost and Benefit

Suppose that company A_i is hacked, and that the usernames and password hashes are stolen by an adversary. We will assume that company A has been following good password storage practices (e.g., company A_i hashes all of their passwords with a strong cryptographic hash function, and company A_i salts all of their password hashes). The adversary can purchase any computing equipment he desires (e.g., Cray supercomputer, GPUs, etc) and run any password cracker he wants for as long as he wants. The adversary's primary limitation is money. It costs money to buy all of this equipment, and it costs money to run the equipment. If the adversary dedicates equipment to run a password cracker for several years then the equipment may be obsolete by the time he is finished (depreciation). We define C_g to be the amortized cost per guesses for the adversary.

7.5.3 Cost of Guessing

Included in the amortized guessing cost are: the price of electricity and the cost of equipment. We estimate C_g by assuming that the adversary rents computing time on Amazon's cloud EC2 [1]. This allows us to easily account for factors like energy costs, equipment failure and equipment depreciation. Amazon measures rented computing power in ECUs [1] — "One EC2 Compute Unit (ECU) provides the equivalent CPU capacity of a 1.0-1.2 GHz 2007 Opteron or 2007 Xeon processor."

Hash Function (**H**)	$F_{\mathbf{H}}$	C_q
SHA1	$\sim 576 \times 10^6$ guesses per hour	$\$1 \times 10^{-10}$
MD5	$\sim 561 \times 10^6$ guesses per hour	$\$1.1 \times 10^{-10}$
BCRYPT (L12)	$\sim 31 \times 10^3$ guesses per hour	$\$1.94 \times 10^{-5}$

Table 7.4: Guessing Costs

We use C_GHz to denote the cost of renting a 1 GHz processor for 1 hour on Amazon. We have

$$C_g = \frac{C_{GHz}}{F_{\mathbf{H}}} .$$

Using the Cluster GPU Instance rental option the adversary could rent 33.5 ECU compute units for \$2.10 per hour (C_{GHz} = \$.06).

Our results are presented in Table 7.4.

7.5.4 Benefit

The benefit B_j of cracking an account A_j is dependent on both the type of account (e.g., banking, e-mail, commerce, social network, leisure) and the adversary's background knowledge about the user (e.g., Does the user reuse passwords? Is the user rich? Is the user a celebrity?).

Password reuse has a tremendous impact on B. An adversary who cracked a user's ESPN account would likely get little benefit — unless the user reused the password elsewhere. For most non-celebrities, B_j can be upper bounded by the total amount of money that the user has in all of his financial accounts. In fact, this may be a significant overestimate — even if the user reuses passwords — because banks are usually successful in reversing large fraudulent transfers [77]. Indeed, most cracked passwords sell for between \$4 and \$17 on the black market [79]. An adversary might also benefit by exploiting the user's social connections (e.g., tricking the user's friends to wire money). Some user's passwords may also be valuable because have access to valuable information (e.g., celebrity gossip, trade secrets).

Most users should be able to safely assume that no adversary will spend more than \$1,000,000 to crack their account even if they reuse passwords. Table 7.5 shows the value of $q_{\$1,000,000}$ for various hash functions.

Hash Function	$q_{\$1,000,000}$
SHA1	10^{16}
MD5	9.1×10^{15}
BCRYPT (L12)	5.2×10^{10}

Table 7.5: $q_{\$1,000,000}$

7.6 Associative Memory and Sufficient Rehearsal Assumptions

The expanding rehearsal assumption makes empirical predictions about long term memory retention (e.g., a user who follows a rehearsal schedule for a cue-association pair will retain that memory for many years). Empirical studies of human memory are often limited in duration due to practical constraints.

The most relevant long term memory study was conducted by Wozniak and Gorzelanczyk [160]. They supervised a group of 7 people who learned 35,000 Polish-English word pairs over 18 months. Their goal was to optimize the intervals between rehearsal of each word pair. They ended up with the following recursive formula

$$I(EF, R) = I(EF, R-1) \times OF(EF, R) \ ,$$

where $I(EF, R)$ denotes the time interval before the R'th rehearsal, EF denotes the easiness factor of the particular word pair, and $OF(EF, R)$ is a coefficient matrix which specifies how quickly the intervals grow [2]. The intervals are very similar to those generated by the expanding rehearsal assumption. Our association strength parameter σ is similar to the easiness factor EF. However, in the expanding rehearsal assumption $OF(EF, R)$ would be a constant that does not vary with R.

Squire tested very long term memory retention by conducting a series of studies over 30 years [144]. To conduct his studies Squire selected a TV show that was canceled after one season, and quizzed participants about the show. It was not surprising that participants in the early studies — conducted right after the show was canceled — had the best performance on the quizzes. However, after a couple of years performance dropped to a stable asymptote [144]. The fact that participants were able to remember some details about the show after 30 years

[2]SuperMemo, a popular commercial memory program `http://www.supermemo.com/`, also uses a similar rehearsal schedule.

suggests that it is possible to maintain a cue-association pair in memory without satisfying all of the rehearsal requirements given by our pessimistic constant rehearsal assumption.

7.6.1 Squared Rehearsal Assumption

Anderson and Schooler demonstrated that the availability of a memory is correlated with recency and the pattern of previous exposures (rehearsals) to the item [18]. Eventually, the following equation was proposed

$$A_i(t) = \sum_{j=1}^{n} \frac{1}{\sqrt{t - t_j}}$$

where $A_i(t)$ denotes the availability of item i in memory at time t and $t_1, \ldots t_n < t$ denote the previous exposures to item i [151]. In this model the rehearsal schedule $R(\vec{c}, j) = j^2$ is sufficient to maintain high availability. To see this consider an arbitrary time t and let k be the integer such that $(k^2 < t \leq (k+1)^2)$. Because $t_k = k^2 < t$ at least k previous rehearsals have occurred by time t so

$$A_i(t) = \sum_{j=1}^{k} \frac{1}{\sqrt{t - t_j}} = \sum_{j=1}^{k} \frac{1}{\sqrt{t - j^2}} = \sum_{j=1}^{k} \frac{1}{\sqrt{(k+1)^2}} \geq \frac{k}{k+1}\,.$$

Squared Rehearsal Assumption (SQ): The rehearsal schedule given by $R(\hat{c}, i) = i^2\sigma$ is sufficient to maintain the association $(\hat{c}, \hat{a})$.

While SQ is certainly not equivalent to ER it is worth noting that our general conclusions are the same under both memory assumptions. The rehearsal intervals grow with time under both memory assumptions yielding similar usability predictions — compare Tables 2.2,2.3 and 7.6. The usability predictions are still that (1) *Strong Random and Independent* —though highly secure — requires any user with infrequently visited accounts to spend a lot of extra time rehearsing passwords, (2) *Lifehacker* requires little effort — but it is highly insecure, (3) SC-0, which is almost as good as *Lifehacker* from a usability standpoint, provides the user with some provable security guarantees, and (4) SC-1 and SC-2 are reasonably easy to use (except for the Infrequent user) and provide strong provable security guarantees — though not as strong as *Strong Random and Independent*.

Schedule/Scheme	B+D	SC-0	SC-1	SC-2	SRI
Very Active	≈ 0	≈ 0	2.77	5.88	794.7
Typical	≈ 0	≈ 0	7.086	12.74	882.8
Occasional	≈ 0	≈ 0	8.86	16.03	719.02
Infrequent	.188	2.08	71.42	125.24	1176.4

Table 7.6: $\mathbb{E}[XR_{365}]$: Extra Rehearsals over the first year under the Squared Rehearsal Assumption — $\sigma = 1$.
B+D: *Lifehacker*
SRI: *Strong Random and Independent*

While the expanding rehearsal assumption yields fewer rehearsal requirements over the first year, the usability results for *Lifehacker* and *Shared Cues* are even stronger because the intervals *initially* grow faster. The usability results are worse for *Strong Random and Independent* because many of the cues are naturally rehearsed with frequency $\lambda = 1/365$ — in this case most rehearsal requirement will require an extra rehearsal[3].

7.7 (n, ℓ, γ)-sharing Set Families

Our notion of (n, ℓ, γ)-sharing set families (definition 5) is equivalent to Nisan and Wigderson's definition of a (k, m)-design [115]. Nisan and Wigderson provided several constructions of (k, m)-designs. For example, one of their constructions implies that their is a (n, ℓ, γ)-sharing set family of size $m = \ell^t$ for $n = \ell^c \lceil \frac{\ell}{c} \rceil$ and $\gamma = \lceil \frac{t}{c} \rceil$, whenever ℓ is a prime power. While this construction is useful for building pseudorandom bit generators, it is not especially helpful in the password context because ℓ should be a small constant. For example, if we set $L = 4$ and the user needs to create $m = 64$ accounts then we would need to set $t = \log_L m = 3$. This a $(128, 4, 1)$-sharing set family of size $m = 64$ or a $(32, 4, 2)$-sharing set family of size $m = 64$. By contrast, we can construct a $(43, 4, 1)$-sharing set family of size $m > 64$ as well as a $(23, 4, 1)$-sharing set family of size $m > 64$ (Observe that $23 = 3+5+7+8$ and $3 \times 5 \times 7 > 64$ so we can apply the Chinese Remainder Theorem construction from Chapter 2.6). In Section 7.7.1 we show that the Chinese Remainder Theorem

[3]The usability results for our occasional user are better than the very active user because the occasional user has fewer sites that a visited with frequency $\lambda = 1/365$.

(n, ℓ, γ)-sharing	Lower Bound (m)	Upper Bound (Thm 3)	Comment
$(n, \ell, \ell - 1)$	$\binom{n}{\ell}$	$\binom{n}{\ell}$	Claim 8
$(9, 4, 3)$	126	126	Greedy Construction (Alg 7.5)
$(16, 4, 1)$	16	20	Greedy Construction (Alg 7.5)
$(20, 6, 2)$	40	57	Greedy Construction (Alg 7.5)
$(25, 6, 2)$	77	153	Greedy Construction (Alg 7.5)
$(18, 6, 3)$	88	204	Greedy Construction (Alg 7.5)
$(19, 6, 3)$	118	258	Greedy Construction (Alg 7.5)
$(30, 9, 3)$	36	217	Greedy Construction (Alg 7.5)
$(40, 8, 2)$	52	176	Greedy Construction (Alg 7.5)
$(43, 4, 1)$	110	150	Theorem 22

Table 7.7: (n, ℓ, γ)-sharing set family constructions

construction from Section 2.6 can be improved slightly. In Section 7.7.2 we show that our construction of (n, ℓ, γ)-sharing may have applications to the construction of highly parallelizable pseudorandom generators. Section 7.7.2 is based on work of Beideman and Blocki [25].

7.7.1 Improved Constructions

In this section we discuss additional (n, ℓ, γ)-sharing set family constructions. Theorem 22 demonstrates how our Chinese Remainder Theorem construction can be improved slightly. For example, we can get a $(43, 4, 1)$-sharing set family of size $m = 110$ with the additional optimizations from Theorem 22 — compared with $m = 90$ without the optimizations. We also use a greedy algorithm to construct (n, ℓ, γ)-sharing set families for smaller values of n. Our results our summarized in Table 7.7 — we also include the theoretical upper bound from Theorem 3 for comparison.

Theorem 22. *Suppose that $n_1 < \ldots < n_\ell$ are pairwise co-prime and that for each $1 \leq i \leq \ell$ there is a (n_i, ℓ, γ)-sharing set family of size m_i. Then there is a $\left(\sum_{i=1}^{\ell} n_i, \ell, \gamma\right)$-sharing set family of size $m = \prod_{i=1}^{\gamma+1} n_i + \sum_{i=1}^{\ell} m_i$.*

Algorithm 7.5 Greedy Construction

Input: n, ℓ, γ
All Subsets: $\mathcal{S}' \leftarrow \{S \subseteq [n] \mid |S| = \ell\}$
Candidates: $\mathcal{S} \leftarrow \emptyset$
for all $S \in \mathcal{S}'$ **do**
  $okToAdd \leftarrow True$
  for all $T \in \mathcal{S}$ **do**
    if $|T \bigcap S| > \gamma$ **then**
      $okToAdd \leftarrow False$
  if $okToAdd$ **then**
    $\mathcal{S} \leftarrow \mathcal{S} \cup \{S\}$
return $\mathcal{S}$

Proof. We can use Algorithm 2.1 to construct a (n, ℓ, γ)-sharing set family $\mathcal{S}_0$ of size $m' = \prod_{i=1}^{\gamma} n_i$. Let $T_1 = \{k \,|\, k < n_1\}$ and for each $i > 1$ let $T_i = \left\{k + \sum_{j=1}^{i-1} n_j \,\middle|\, k < n_i\right\}$. By construction of $\mathcal{S}_0$ it follows that for each $S \in \mathcal{S}_0$ and each $1 \le i \le \ell$ we have $|S \bigcap T_i| = 1$. By assumption, for each $i \ge 1$ there is a (n, ℓ, γ)-sharing family of subsets of T_i of size m_i — denoted $\mathcal{S}_i$. For each pair $S' \in \mathcal{S}_i$, and $S \in \mathcal{S}_0$ we have

$$\left|S \bigcap S'\right| \le \left|S \bigcap T_i\right| \le 1 \,,$$

and for each pair $S' \in \mathcal{S}_i$, and $S \in \mathcal{S}_i$ $(S \ne S')$

$$\left|S \bigcap S'\right| \le \gamma \,,$$

because $\mathcal{S}_i$ is (n_i, ℓ, γ)-sharing. Finally, for each pair $S' \in \mathcal{S}_i$, and $S \in \mathcal{S}_j$ $(j \ne i)$ we have

$$\left|S \bigcap S'\right| \le \left|S \bigcap T_i\right| \le 0 \,.$$

Therefore,

$$\mathcal{S} = \bigcup_{i=0}^{\ell} \mathcal{S}_i \,,$$

is a $\left(\sum_{i=1}^{\ell} n_i, \ell, \gamma\right)$-sharing set family of size $m = \prod_{i=1}^{\gamma} n_i + \sum_{i=1}^{\ell} m_i$. □

Claim 8. *For any $0 < \ell \le n$ there is a $(n, \ell, \ell - 1)$-sharing set family of size $m = \binom{n}{\ell}$, and there is no $(n, \ell, \ell - 1)$-sharing set family of size $m' > m$.*

Proof. It is easy to verify that

$$\mathcal{S} = \{S \subseteq [n] \mid |S| = \ell\} \ ,$$

the set of all subsets of size ℓ, is a $(n, \ell, \ell - 1)$-sharing set family of size $m = \binom{n}{\ell}$. Optimality follows immediately by setting $\gamma = \ell - 1$ in Theorem 3. □

7.7.2 Applications to Pseudorandom Number Generators

Applications to Pseudorandom Number Generation A pseudorandom number generator is a function $\mathbf{G} : \{0,1\}^n \to \{0,1\}^m$ which takes a uniformly random seed $x \sim \{0,1\}^n$ of length n, and outputs a string $\mathbf{G}(x) \in \{0,1\}^m$ ($m \gg n$) which "looks random." Nisan and Wigderson used a $\left(n, \ell = O\left(\sqrt{n}\right), \gamma = \log m\right)$-sharing set family $\mathcal{S} = \{S_1, \ldots, S_m\}$ of size m to construct pseudorandom number generators [115]. In particular, they define the pseudorandom number generator $\mathbf{NW}_{P,\mathcal{S}}(x) = P(x_{|S_1}) \ldots P(x_{|S_m})$, where $x_{|S_i} \in \{0,1\}^\ell$ denotes the bits of $x \in \{0,1\}^\ell$ at the indices specified by S_i and $P : \{0,1\}^\ell \to \{0,1\}$ is a predicate. If the predicate $P : \{0,1\}^\ell \to \{0,1\}$ is "hard" for circuits of size $H_\ell(P)$ to predict [4] then no circuit of size $H_\ell(P) - O(m2^\gamma)$ will be able to distinguish $\mathbf{NW}_{P,\mathcal{S}}(x)$ from a truly random binary string of length m, when the seed $x \sim \{0,1\}^n$ is chosen uniformly at random. In this context, n is the length of the random seed, m is the number of random bits extracted and the pseudorandom number generator fools circuits of size $H_\ell(P) - O(m2^\gamma)$. Thus, we would like to find (n, ℓ, γ)-sharing set families where n is small, m is large (e.g., we can extract many pseudorandom bits from a small seed) and γ is small (e.g., so that the pseudorandom bits look random to a large circuit). Nisan and Wigderson gave an explicit construction of an (ℓ^2, ℓ, γ)-sharing set family of size $\ell^{\gamma+1}$.

Applications to Randomness Extractors Trevisan used the pseudorandom number generator of Nisan and Wigderson to construct a randomness extractor [149]. A (k, ϵ) randomness extractor is a function $\mathbf{Ext} : \{0,1\}^{\hat{\ell}} \times \{0,1\}^n \to \{0,1\}^m$ that takes a string $x_1 \sim D$, where D is a distribution over $\{0,1\}^{\hat{\ell}}$ with minimum entropy k, along with a n additional uniformly random bits $x_2 \sim \{0,1\}^n$ and extracts an m-bit string $y \in \{0,1\}^m$ that is almost uniformly random (e.g., distribution over $y \in \{0,1\}^m$ is ϵ-close to the uniform distribution U_m over $\{0,1\}^m$). Trevisan used the string x_1 to select a random predicate $P : \{0,1\}^\ell \to \{0,1\}$, and then extracted m bits

[4] Nisan and Wigderson observe that a random predicate P will satisfy this property with high probability[115].

by running $\mathbf{NW}_{P,\mathcal{S}}(x_2)$. Raz et al [126] observed that the pseudorandom number generator Nisan and Wigderson could be built using a *weak* (n, ℓ, γ)-sharing set family of size m, and showed how to construct *weak* (n, ℓ, γ)-sharing set family of size m for *any* value of m as long as $n \geq \lceil \frac{\ell}{\gamma} \rceil \ell$. However, their construction was not explicit (Informally, we say that a construction is explicit if their is a fast parallel algorithm to output the i'th set.). Hartman and Raz[89] showed how to use the Nisan-Wigderson construction to obtain an explicit construction of weak (n, ℓ, γ)-sharing set families. While their construction requires less space than our construction, our construction can be computed faster on a parallel machine.

Advantages of Explicit Constructions One nice property of the Nisan Wigderson Pseudorandom number generator is that it is highly parallelizable. For each $j \in [m]$ we can compute the j'th bit $\mathbf{NW}_{P,\mathcal{S}}(x)[j] = P\left(x_{|S_j}\right)$ independently as long as we can quickly find the set $S_j \in \mathcal{S}$. Observe that we would need space at least $O(m\ell \log n)$ to store the set family $\mathcal{S} = \{S_1, \ldots, S_m\}$, which could be a problem especially when m is very large. However, if the set family has an explicit construction (e.g., there is a small circuit C s.t. $C(i) = S_i$ for all $i \in [m]$) then we can simply compute $\mathbf{NW}_{P,\mathcal{S}}(x)[j] = P\left(x_{|C(j)}\right)$.

Preliminaries

Before we formally define a pseudorandom number generator we first define a pseudorandom distribution X over $\{0,1\}^m$. Informally, Definition 19 say that a distribution is pseudorandom if the distribution that 'appears' random to any 'small enough' circuit. Given a circuit C we use

$$\mathbf{Adv}_C(X) = \left| \Pr_{x \in X}[C(x) = 1] - Pr_{x \in U_m}[C(x) = 1] \right|$$

to denote the advantage of C at predicting whether x was drawn from the distribution X or from U_m, where U_m is the uniform distribution over $\{0,1\}^m$. The distribution X 'appears' random to a circuit C if $\mathbf{Adv}_C(X)$ is small.

Definition 19. *A distribution X over $\{0,1\}^m$ is said to be (s, ϵ)-pseudorandom if, given any circuit C (taking m inputs) of size at most s, $\mathbf{Adv}_C(X) \leq \epsilon$.*

Given a distribution X over $\{0,1\}^n$ and a function $G : \{0,1\}^n \to \{0,1\}^m$ we use $G(X)$ to denote the distribution over $\{0,1\}^m$ induced by G. Informally, a function $G : \{0,1\}^n \to \{0,1\}^m$ is pseudorandom if it induces a pseudorandom distribution.

Definition 20. *Let $\{G_n\}_{n\in N}$ be a family of functions such that $G_n : \{0,1\}^n \rightarrow \{0,1\}^m$. We say the family is a (s,ϵ)-pseudorandom number generator if G is computable in time $2^{O(n)}$, and $G(U_n)$ considered as a distribution is (s,ϵ)-pseudorandom.*

Nisan and Wigderson [115] show how to construct a pseudorandom number generator $G : \{0,1\}^n \rightarrow \{0,1\}^m$ using any (n,ℓ,γ)-sharing set family of size m. Their construction assumes the existence of a predicate $f : \{0,1\}^\ell \rightarrow \{0,1\}$ that is hard for 'small' circuits to predict.

Definition 21. *Let $f : \{0,1\}^\ell \rightarrow \{0,1\}$ be a boolean function. We say that f is (s,ϵ)-hard if for any circuit C of size s, $\left|\Pr_{x\sim\{0,1\}^\ell}[C(x) = f(x)] - \frac{1}{2}\right| \leq \epsilon$.*

Observe that a random function will fool all small circuits with high probability[5]. Following, Nisan and Wigderson we use $H(f)$ to denote the hardness of a function f.

Definition 22. *Let $f : \{0,1\}^* \rightarrow \{0,1\}$ be a boolean function and let f_ℓ be the restriction of f to strings of length ℓ. The hardness of f at ℓ, $H_f(\ell)$ is defined to be the maximum integer h_ℓ such that f_ℓ is $(1/h_\ell, h_\ell) - hard$.*

Raz et al [126] showed that the Nisan-Wigderson pseudorandom number generator works even if the family of sets $S_1, ..., S_m$ only satisfies the weaker condition from definition 23. Observe that any (n,ℓ,γ)-sharing set family is also a *weak* (n,ℓ,γ)-sharing set family, but the converse is not necessarily true. We also note that as m increases the requirement $\sum_{j<i} 2^{|S_i \cap S_j|} \leq 2^\gamma(m-1)$ becomes increasingly lax. This allows us to construct arbitrarily large weak (n,ℓ,γ)-sharing families.

Definition 23. *A family of sets $S_1, ..., S_m \subset [n]$ is a weak (n,ℓ,γ)-sharing set family if (1) $\forall i \in [m]$. $|S_i| = \ell$, and (2) $\forall i \in [m].\sum_{j<i} 2^{|S_i \cap S_j|} \leq 2^\gamma(m-1)$.*

Informally, we say that a set family $S_1, \ldots, S_m$ is *explicitly constructible* if their is a fast parallel algorithm $\mathcal{A}(i)$ to compute the i'th set S_i. We use $\mathbf{Depth}(A)$ to denote the running time of $\mathcal{A}$ when executed in parallel processes $\mathbf{Work}(\mathcal{A})$ to denote the total number of steps executed in all processes. Similarly, $\mathbf{Space}(\mathcal{A})$ denotes the total space requirement of $\mathcal{A}$. Our notion of explicit constructions (Definition 24) is similar to the notion used by Hartman and Raz[89] except that we also consider the parallel running time of $\mathcal{A}$.

[5]The argument is straightforward. Fix any circuit C. A random function $f : \{0,1\}^\ell \rightarrow \{0,1\}$ will satisfy $\mathbf{Adv}_C(f(U_\ell)) \leq \epsilon$ with very high probability by Chernoff bounds. We can then apply union bounds to argue that a random f will satisfy $\max_{C\in\mathcal{C}} \mathbf{Adv}_C(X) \leq \epsilon$ for any sufficiently small class $\mathcal{C}$ of circuits.

Definition 24. *We say that a set family $S_1, \ldots, S_m \subseteq [n]$, where the size of each set is ℓ, is (t_1, t_2, t_3)-explicitly constructible if there exists an algorithm $\mathcal{A}$ s.t. for all $1 \leq i \leq m$ we have $\mathcal{A}(i) = S_i$, $\textbf{Work}(\mathcal{A}) \leq \mathcal{A}$, $\textbf{Depth}(A) \leq t_2$ and $\textbf{Space}(\mathcal{A}) \leq t_3$.*

Definition 25. *We use $m(n, \ell, \gamma)$ to denote the maximum value m such that there exists a (n, ℓ, γ)-sharing set family.*

The (n, ℓ, γ)-sharing set family construction of Chapter 2.6 relies on the Chinese Remainder Theorem. To analyze their construction we will be interested in finding a large set $S = \{t_1, \ldots, t_\ell\}$ of integers such that S has size ℓ, the numbers in S are pairwise coprime, $\sum_{i=1}^{\ell} t_i \leq n$ and each $t_i \geq \frac{n}{2\ell}$. We will rely on recent results on prime density.

Definition 26. *$\pi(t)$ indicates the number of prime numbers less than or equal to t. $\pi\pi(t)$ indicates the maximum $|S|$ such that $S \subseteq \left\{\lceil \frac{t}{2} \rceil, ..., t\right\}$ and $\forall i \neq j \in S.\textbf{GCD}(i, j) = 1$.*

We are particularly interested in lower bounding the value $\pi\pi(x)$. Clearly, $\pi\pi(x) \geq \pi(x) - \pi(x/2)$. As it turns out this lower bound is nearly tight (see Theorem 25). We can bound $\pi(x) - \pi(x/2)$ using Ramanujan primes.

Definition 27. *[124] The t'th Ramanujan Prime is the smallest integer R_t s.t. $\pi(x) - \pi(x/2) \geq t$ for all $x \geq R_t$.*

Allowing n to equal at least ℓR_ℓ guarantees that $\left\{\frac{n}{2\ell}, \frac{n}{\ell}\right\}$ contains at least ℓ primes which will satisfy the conditions of the Chinese Remainder Theorem construction. Sondow's bounds on Ramanujan primes (see Theorem 23) allow us to express this bound on n as an elementary function.

Constructions

Nisan and Wigderson [115] gave an explicit construction of (ℓ^2, ℓ, γ)-sharing set families of size $m = \ell^{\gamma+1}$ for any prime power ℓ. Given a polynomial $p(x)$ with coefficients in $\textbf{GF}(\ell)$, the finite field of size ℓ, they define the set $S_p = \{(x, p(x)) \mid x \in \textbf{GF}(\ell)\}$. The family $\mathcal{S} = \left\{S_p \,\middle|\, p \text{ has degree} \leq \gamma\right\}$ is (ℓ^2, ℓ, γ)-sharing and has size $m = |\mathcal{S}| = \ell^{\gamma+1}$. Given pairwise coprime numbers $n_1 < \ldots < n_\ell$ we provided an explicit construction of $\left(\sum_{i=1}^{\ell} n_i, \ell, \gamma\right)$-sharing families in Chapter 2. Briefly, given an integer $i \geq 0$ we defined the set $S_i = \{1 + \sum_{k=1}^{j-1} n_k + \left(i \mod n_j\right) : j \in [\ell]\}$, and showed that

the family $\mathcal{S} = \left\{S_i \middle| 0 \leq i < \prod_{j=1}^{\gamma+1} n_j\right\}$ is an $\left(\sum_{i=1}^{\ell} n_i, \ell, \gamma\right)$-sharing set family of size $\prod_{j=1}^{\gamma+1} n_i$.

The proof of Theorem 24 is based on Theorem 22. We take advantage of Sondow's results on prime density [142] to compare our construction to the construction of Nisan and Wigderson.

Theorem 23. *[142] For all $t \geq 1$ the following bound holds $2t \ln t < R_t < 4t \ln 4t$.*

Theorem 24. *$\forall n \geq 4\ell^2 \ln 4\ell$, $m(n, \ell, \gamma) \geq (2\ell \ln 2\ell)^{\gamma+1}$. Furthermore, this set family is (t_1, t_2, t_3)-explicitly constructible with $t_1 = O(\ell (\log m)(\log \ell))$,$t_2 = O(\log \log m)$ and $t_3 = O(\ell (\log m)(\log \ell))$.*

Proof. Theorem 23 due to Sondow [142] shows that there will always be at least ℓ primes $p_1, \ldots, p_\ell$ between $2\ell \ln 2\ell$ and $4\ell \ln 4\ell$. We have $\sum_{i=1}^{\ell} p_i \leq \ell(4\ell \ln 4\ell) \leq n$. Note that $\prod_{i=1}^{\gamma+1} p_i \geq (2\ell \ln 2\ell)^{\gamma+1}$. It follows from Theorem 22 that $m(n, \ell, \gamma) \geq (2\ell \ln 2\ell)^{\gamma+1}$.
To construct this set family our algorithm $\mathcal{A}$ will store the following values (1) the primes $p_1, \ldots, p_\ell$, (2) the precomputed values $2^k \mod p_j$ for each $0 \leq k \leq \min\left\{p_j - 1, \lfloor \log m \rfloor\right\}$ and $1 \leq j \leq \ell$, and (3) the precomputed values kp_j for each $1 \leq k \leq \log \log \log m$ and each $1 \leq j \leq \ell$. We need $O(\ell \log \ell)$ bits of space to store each of the primes, $O(\ell (\log \ell)(\log m))$ bits of space to store each of the the values $2^k \mod p_j$ and $O(\ell (\log \ell)(\log \log \log m))$ bits of space to store each of the values kp_j. Given an index $i_0 = i \leq m$ our algorithm $\mathcal{A}$ computes each of the elements of S_i in parallel — to compute $S_i[j]$ it suffices to compute the value $i_0 \mod p_j$. We will focus on the computation of $i_0 \mod p_j$. For each $k \leq \lfloor \log m \rfloor$ we look up the precomputed values $2^k \mod p_j$ and compute $y_k = i_0[k]2^k \mod p_j$ (here, $i_0[k]$ denotes the k'th bit of i_0 when i_0 is viewed as a binary string). We then compute the value $i_1 = \sum_{k=1}^{\lfloor \log m \rfloor} y_k$ (observe that $i_1 \equiv i_0 \mod p_j$). This can be done by a depth $O(\log \log m)$ circuit by using two tricks: divide-and-conquer and carry-save addition. We group $y_1, \ldots, y_{\lfloor \log m \rfloor}$ into triples (e.g., $(y_1, y_2, y_3), \ldots,$) for each triple we compute the partial sum **ps** (the bits of **ps** are defined as $\mathbf{ps}[t] = y_1[t] \oplus y_2[t] \oplus y_3[t]$ for each index t) and the shift-carry **sc** ($\mathbf{sc}[t+1] = (y_1[t] \wedge y_2[t]) \vee (y_1[t] \wedge y_3[t]) \vee (y_2[t] \wedge y_3[t])$). Observe that $y_1 + y_2 + y_3 = \mathbf{ps} + \mathbf{sc}$. We went from $\log m$ values that we needed to add to $\frac{2}{3} \log m$ values that we needed to add — after $O(\log \log m)$ rounds we will have the value $i_1 \equiv i_0 \mod p_j$. While i_1 may not be the final answer (e.g., we could have $i_1 > p_j$) we have $i_1 \leq p_j \log m \ll i_0 = O(m)$ so we are making progress. Now we can recursively compute a value i_2 s.t. $i_2 \equiv i_1 \mod p_j$ and

$i_2 \leq p_j \log(1 + \log\log m)$ — this time we are only adding $O(\log\log m)$ numbers so we need $O(\log\log\log m)$ rounds of computation to find i_2. Now, we can search for the biggest value kp_j s.t. $i_2 \geq kp_j$ (note that there are only $\log\log\log m$ values to check). Our final answer is simply $i_2 - kp_j = i_0 \mod p_j$. □

Note that our construction only requires relatively prime numbers. So the results from Theorem 24 could be improved by including non-prime values. However, Theorem 25 implies that these improvements will not be particularly significant.

Theorem 25. $\forall n \in \mathbb{Z}^+.\ \pi\pi(n) \leq \pi(n) - \pi(\frac{n}{2}) + \pi(\sqrt{n})$.

Proof. Let $S \subseteq \left\{\lceil \frac{n}{2} \rceil, \ldots, n\right\}$ be a set of coprime numbers of maximum size. Observe that each prime number $p \in [n]$ is a factor of at most one number in S. Without loss of generality we can assume that each of the primes between n and $\frac{n}{2}$ are contained in S (if $p \notin S$ then, because S is of maximum size, we must have some $t = pq \in S$, but in this case we can simply replace t with p). The number of primes between n and $\frac{n}{2}$ is $\pi(n) - \pi(\frac{n}{2})$, and all of these integers are relatively prime to each other and to every other number in the range $[n]$. All other numbers in S must have at least two prime factors, and at least one of them must be less than or equal to $\sqrt{n}$. Since each prime factor less than or equal to $\sqrt{n}$ can be used at most once, for the members of S to remain pairwise relatively prime, at most $\pi(\sqrt{n})$ non-primes can be included in the set, each containing a single prime factor less that $\sqrt{n}$. □

Comparing Constructions. To compare our construction from Chapter 2 with the construction of Nisan and Wigderson [115] we set $n = 4\ell'^2 \ln 4\ell'$ and we set $\ell = \sqrt{4\ell'^2 \ln 4\ell'}$. The construction of Nisan and Wigderson gives use $m(n, \ell, \gamma) \geq \ell^{\gamma+1} = \left(2\ell' \sqrt{\ln 4\ell'}\right)^{\gamma+1}$, while our construction gives us $m(n, \ell', \gamma) \geq (2\ell' \ln 2\ell')^{\gamma+1} > \left(2\ell' \sqrt{\ln 4\ell'}\right)^{\gamma+1}$. However, $\ell' < \ell$ so our construction has a smaller ℓ.

Constructing Weak (n, ℓ, γ)-sharing set families. We now show that our explicit Chinese Remainder Theorem set family construction can be also be used to construct weak (n, ℓ, γ)-sharing set families of arbitrary size m. Our main results are stated in Theorem 26.

Theorem 26. *For all m there is an explicitly constructible weak $(4\ell^2 \ln 4\ell, \ell, \gamma)$-sharing set family of size m as long as $2^\gamma \geq \left(1 + \frac{1}{-1+\ln 2\ell}\right)$. Furthermore, this set family is*

(t_1, t_2, t_3)*-explicitly constructible with* $t_1 = O(\ell(\log m)(\log \ell))$,$t_2 = O(\log\log m)$ *and* $t_3 = O(\ell(\log m)(\log \ell))$.

Proof. Let m be given. We use the explicit construction from Chapter 2.6. By Theorem 23 we can find ℓ primes such that $2\ell \ln 2\ell < p_1 < \ldots < p_\ell < 4\ell \ln 4\ell$. In particular, we let $S_i = \left\{1 + \sum_{k=1}^{j-1} p_k + \left(i \mod p_j\right) \;\middle|\; j \in [\ell]\right\}$. Now for $i \in [m]$ we have

$$\begin{aligned}
\sum_{j<i} 2^{|S_i \cap S_j|} &= \sum_{k=0}^{\infty} 2^k \left|\left\{j \;\middle|\; j < i \wedge \left|S_i \cap S_j\right| = k\right\}\right| \leq \sum_{k=0}^{\infty} 2^k \left|\left\{j \;\middle|\; j < i \wedge \left|S_i \cap S_j\right| \geq k\right\}\right| \\
&\leq \sum_{k=0}^{\infty} 2^k \binom{\ell}{k} \frac{i-1}{\prod_{j=1}^{k} p_i} \leq \sum_{k=0}^{\infty} 2^k \binom{\ell}{k} \frac{i-1}{(2\ell \ln 2\ell)^k} \\
&\leq \sum_{k=0}^{\infty} \frac{i-1}{(\ln 2\ell)^k} \leq (i-1)\left(\frac{\ln 2\ell}{-1 + \ln 2\ell}\right) \leq (m-1)\, 2^\gamma \,,
\end{aligned}$$

where the second inequality follows from the Chinese Remainer Theorem. We already showed that this set family is explicitly constructible in the proof of Theorem 24. □

Raz et al gave a randomized construction of weak $\left(\left\lceil \frac{\ell}{\gamma} \right\rceil \cdot \ell, \ell, \gamma\right)$-sharing set families for any $m, \gamma > 0$. While they showed that their construction could be derandomized, their construction is not explicit (e.g., the construction of i'th subset S_i is dependent on the sets $S_1, \ldots, S_{i-1}$). Theorem 26 shows that our construction is competitive with the construction of Raz et al [126] though the value of n is slightly larger. Hartman and Raz[89] later showed how to use the Nisan-Wigderson construction to get an explicit construction weak $(\ell^2, \ell, O(1))$-sharing set families with $t_1 = t_2 = O(poly(\ell, \log m))$ and $t_3 = O(\log m)$. Their construction requires less space than ours $O(\log m)$ vs $O(\ell(\log m)(\log \ell))$ space, but our construction will run faster on a parallel computer $O(\log\log m)$ vs $O(poly(\ell, \log m))$ time. The construction of Hartman and Raz[89] could be optimized to run in parallel time $t_2 = O(\log\log m)$ by precomputing a few strategic values (e.g., an $\ell \times \ell$ multiplication table and an $\ell \times \log m$ exponentiation table), but then we would require $O(\ell^2 \log \ell)$ space to store all of the precomputed values.

Parallel Pseudorandom Number Generators. Nisan and Wigderson proved that if $\gamma = \log m$, $\mathcal{S}$ is a (n, ℓ, γ)-sharing set family and $H_f(\ell) \geq 2m^2$ that their construction $\mathbf{NW}_{f,\mathcal{S}}$ is a $\left(m^2, \frac{1}{m}\right)$ pseudorandom number generator. In particular, Theorem

27 implies that if D is a circuit of size $|D| \leq m^2$ that distinguishes $\mathbf{NW}_{f,\mathcal{S}}(U_n)$ from U_m with advantage $\mathbf{ADV}_D\left(\mathbf{NW}_{f,\mathcal{S}}(U_n)\right) \geq \frac{1}{m}$ then there exists a circuit C of size $|C| \leq 2m^2$ which predicts $f(x)$ with advantage $\mathbf{ADV}_C(f(U_\ell)) \geq \frac{1}{2m^2}$. This contradicts the definition of $H_f(\ell)$. Raz et al [126] observed that it suffices for $\mathcal{S}$ to be a weak (n, ℓ, γ)-sharing set family. If we let $\mathcal{S}_m$ denote the explicitly constructible weak $(4\ell^2 \ln 4\ell, \ell, \gamma)$-sharing set family of size m then for any $m > 0$ $\mathbf{NW}_{f,\mathcal{S}_m}$ is a $\left(m^2, \frac{1}{m}\right)$ pseudorandom number generator with seed length $4\ell^2 \ln 4\ell$ assuming that $H_f(\ell) \geq 2m^2$. Because $\mathcal{S}_m$ is explicitly constructible we can compute each bit $\mathbf{NW}_{f,\mathcal{S}_m}(x)[i] = f(x_{|S_i})$ independently.

Theorem 27. *[115, 126] Let $f : \{0,1\}^\ell \to \{0,1\}$ be a boolean function and $\mathcal{S} = \{S_1, ..., S_m\}$ be an weak (n, ℓ, γ)-sharing set family. Suppose $D : \{0,1\}^m \to \{0,1\}$ is such that $\mathbf{ADV}_D\left(\mathbf{NW}_{f,\mathcal{S}}(U_n)\right) > \epsilon$, then there exists a circuit C of size $|C| \leq |D| + O\left(\max_{j\in[m]} \sum_{i<j} 2^{|S_i \cap S_j|} m\right)$ such that $\left|\Pr_{x\sim\{0,1\}^\ell}[C(x) = f(x)] - \frac{1}{2}\right| \geq \frac{\epsilon}{m}$*

7.7.3 Upper Bounds

Our main result in this section is Theorem 28. We prove that $m(n, \ell, \gamma) = c_1$ whenever $\ell = \frac{n}{c_1}$ and $\gamma = c_2 n$ provided that c_2 is sufficiently small. We previously showed that $m(n, \ell, \gamma) \leq \frac{\binom{n}{\gamma+1}}{\binom{\ell}{\gamma+1}}$. We note that this bound is far from tight whenever ℓ is large. For example, if $c_1 = 2$ and $c_2 = \frac{1}{10}$ then this upper bound $\binom{n}{\frac{n}{10}+10} \Big/ \binom{\frac{n}{2}}{\frac{n}{10}+10}$ grows exponentially with n. By contrast, Theorem 28 implies that $m(n, n/2, n/10) = 2$.

Theorem 28. $\forall\ 0 < c_2 < 1, n, c_1 \in \mathbb{N}$ *such that* $c_1 | n$. $m(n, \frac{n}{c_1}, c_2 n) = c_1$ *iff* $c_2 < \frac{2}{c_1^3 + c_1^2}$.

Before we prove Theorem 28 we first prove an easier result. Theorem 29 upper bounds $\lim_{n\to\infty} m(n, \ell, \gamma)$ when ℓ is in a constant ratio to n and γ is small. Theorem 29 holds because the k'th set S_k must use $cn - (k-1)\gamma$ new elements (elements that are not in $\bigcup_{i=1}^{k-1} S_i$).

Theorem 29. $\forall\ \gamma_c, 0 < c < 1$ *such that* $cn \in \mathbb{N}$. $m(n, cn, \gamma_c) \to \lfloor \frac{1}{c} \rfloor$ *as* $n \to \infty$.

Proof. Let $\ell = cn$ and let $\tau \in \mathbb{N}$ be an integer such that $\tau > \lfloor \frac{1}{c} \rfloor$. The first set will contain ℓ elements. The second set can share at most γ of them, so the second set must contain at least $\ell - \gamma$ previously unused elements. Therefore the union of the

first two sets must contain at least $2\ell - \gamma$ elements. In a similar manner, the *kth* set must contain at least $\ell - (k-1)\gamma$ new elements, therefore,

$$k\ell - \frac{(k-1)k\gamma}{2} \leq \left| \bigcup_{i=1}^{k} S_i \right| \leq n \,. \tag{7.1}$$

Assume for contradiction that $\limsup_{n\to\infty} m(n, cn, \gamma_c) = \tau$.Then we have

$$\begin{aligned} \lim_{n\to\infty} \left(n - \tau\ell + \frac{(k-1)k\gamma}{2} \right) &= \lim_{n\to\infty} \left(n - \tau cn + \frac{(k-1)k\gamma}{2} \right) \\ &= \lim_{n\to\infty} \left(n\left(1 - c\tau\right) \right) \\ &= -\infty \,. \end{aligned}$$

This contradicts equation 7.1. □

The proof of Theorem 28 is a bit longer. *Proof of Theorem 28.* Suppose that for some valid n, c_1, c_2 there is an (n, ℓ, γ)–sharing set family of size $c_1 + 1$. By equation 7.1, the number of elements used by such a set family must be at least:

$$(c_1 + 1)\ell - \frac{c_1(c_1+1)\gamma}{2} \leq n \tag{7.2}$$

Taking advantage of the fact that $\ell = \frac{n}{c_1}$ and $\gamma = c_2 n$, the inequality can be simplified:

$$\begin{aligned} n + \ell - \frac{c_1(c_1+1)\gamma}{2} &\leq n \\ \ell &\leq \frac{c_1(c_1+1)\gamma}{2} \\ \frac{n}{c_1} &\leq \frac{c_1(c_1+1)c_2 n}{2} \\ 2n &\leq (c_1^3 + c_1^2)c_2 n \\ \frac{2}{c_1^3 + c_1^2} &\leq c_2 \,. \end{aligned}$$

Thus, all set families of size $c_1 + 1$ or greater must have $c_2 \geq \frac{2}{c_1^3+c_1^2}$, and $c_2 < \frac{2}{c_1^3+c_1^2}$ guarantees the set family will have a size of at most c_1.

Since $c_1\ell = n$, it is possible to make a family of size c_1 for any value of c_2 by simply choosing sets that share no elements. Therefore, the size of the largest possible set family for any n, ℓ, γ meeting the specified conditions is c_1 if $c_2 < \frac{2}{c_1^3+c_1^2}$.

If $c_2 \geq \frac{2}{c_1^3+c_1^2}$, there will always exist a set family of size $\geq c_1 + 1$. To create such a family, choose $c_1 + 1$ sets such that each of them shares γ elements with each of the others. This will be possible as long as:

$$\begin{aligned} c_1\gamma &\leq \ell \\ nc_1c_2 &\leq \frac{n}{c_1} \\ c_1^2c_2 &\leq 1 \\ \frac{2c_1^2}{c_1^3+c_1^2} &\leq 1\,. \end{aligned}$$

Since this final inequality is true for all possible values of c_1, it will such a set family can always be created, and its size will be, as shown earlier, n when $c_2 = \frac{2}{c_1^3+c_1^2}$. Since increasing c_2 will not eliminate any possible set families, no n, ℓ, γ satisfying the conditions with $c_2 \geq \frac{2}{c_1^3+c_1^2}$ will have a maximum family size $< c_1 + 1$. Therefore, the size of the largest possible set family for a valid n, ℓ, γ will be c_1 iff $c_2 < \frac{2}{c_1^3+c_1^2}$. □

We now show that the upper bound from Theorem 28 is nearly tight. In particular, when $\gamma = c_2 n$ for a slightly larger constant c_2 then $m(n, \ell, \gamma)$ is exponentially large. Theorem 30 lower bounds the values of c_2 for which $m(n, \ell, \gamma)$ is exponentially large. In particular, we demonstrate the existence of an (n, ℓ, γ)–sharing set family of exponential size by showing that the probability of obtaining such a set family through random selection is non-zero. Our proof uses the following randomized construction of an (n, ℓ, γ)–sharing set family. Independently choose random integers r_i^j each in the range $0 \leq r_i < c_1$ for $i \in \{0, \ldots, \ell - 1\}$ and $j \in [m]$. Let $S_j = \bigcup_{i=0}^{\ell-1} \{ic_1 + r_i^j\}$. We use standard concentration bounds due to Chernoff [53] to show that $\left|S_j \bigcap S_j\right| \leq \gamma$ with high probability, and then we union bounds to argue that the entire set family is (n, ℓ, γ)–sharing with non-zero probability.

Theorem 30. *$\forall\, c_2 > 0, n, c_1 \in \mathbb{N}$ such that $c_1 | n$. $m(n, \frac{n}{c_1}, c_2 n) > exp(O(n))$ if $c_2 > \frac{1}{c_1^2} + \epsilon$.*

The proof of Theorem 30 is based on standard concentration bounds due to Chernoff. We use the specific form from Theorem 31. We demonstrate the existence of an (n, ℓ, γ)–sharing set family of exponential size by showing that the probability of obtaining such a set family through random selection is non-zero.

Theorem 31. *[53] Let $X_1, \ldots, X_n \in [0,1]$ be a sequence of independent random variables. Let $S = \sum_{i=1}^{n} x_i$, and let $\mu = \mathbb{E}[S]$. Then for all $\delta \geq 0$*

$$\Pr[S \geq \mu + \delta n] \leq e^{-2n\delta^2} .$$

Proof of Theorem 30. We create an (n, ℓ, γ)–sharing set family by creating sets in the following manner: Independently choose random integers r_i^j each in the range $0 \leq r_i < c_1$ for $j \in [m]$ and $i \in \{0, \ldots, \ell - 1\}$. Let $S_j = \bigcup_{i=0}^{\ell-1} \left\{ ic_1 + r_i^j \right\}$. Given two such sets, S_j, S_k let

$$x_i = \begin{cases} 1 & : r_i^j = r_i^k \\ 0 & : r_i^j \neq r_i^k \end{cases}$$

Then the number of elements shared by S_j and S_k is

$$S_j \cap S_k = \sum_{i=0}^{\ell-1} x_i .$$

Let $\mu = \mathbb{E}\left[S_j \cap S_k \right] = \frac{n}{c_1^2}$ denote the expected number of shared elements. The probability that two such sets share more than γ elements, given $c_2 = \frac{1}{c_1^2} + \epsilon$ is

$$\begin{aligned} Pr[\left|S_j \cap S_k\right| > \gamma] &= Pr[\sum_{i=0}^{\ell-1} x_i > c_2 n] \\ &= Pr[\sum x_i > \frac{n}{c_1^2} + n\epsilon] \\ &\leq Pr[\sum x_i \geq \mu + \epsilon n] \\ &\leq e^{-2n\epsilon^2} \end{aligned}$$

with the last step by Theorem 31. Thus the probability that two randomly selected sets share more than γ elements is at most $e^{-2n\epsilon^2}$.

An (n, ℓ, γ)–sharing set family of size m will contain $\binom{m}{2}$ pairs of sets. The probability that the family is valid, with none of the sets sharing more than γ

elements is

$$\begin{aligned} Pr[\exists j \neq k : \left|S_j \cap S_k\right| > \gamma] &\leq \binom{m}{2} Pr[\left|S_j \cap S_k\right| > \gamma] \\ &\leq \binom{m}{2} e^{-2n\epsilon^2} \\ &\leq m^2 e^{-2n\epsilon^2} \end{aligned}$$

by the union bound. For $m < e^{n\epsilon^2}$, this probability will be less than 1, meaning there is a non-zero chance of forming a valid set family of size m by random selection and therefore such a family must exist. □

We previously observed that $m(n, \gamma+1, \gamma) = \binom{n}{\gamma+1}$ whenever $n \geq \gamma + 1$. In general $m(n, \ell, \gamma) \geq m(n, \ell+1, \gamma)$ whenever $\ell \geq \gamma + 1$.

Claim 9. *For all $n \geq \gamma$ we have $m(n, \ell, \gamma) \geq m(n, \ell+1, \gamma)$ whenever $\ell \geq \gamma + 1$.*

Proof. Suppose that $\ell \geq \gamma + 1$ and we have an $(n, \ell+1, \gamma)$-sharing set family $S_1, \ldots, S_m \subseteq [n]$ of size m. We can form a (n, ℓ, γ)-sharing set family $S'_1, \ldots, S'_m \subseteq [n]$ by picking some element $s_i \in S_i$ setting $S'_i = S_i - \{s_i\}$ for each $i \in [m]$. Observe that this argument does not apply whenever $\ell = \gamma$ because then we might have $S'_i = S'_j$ for $i \neq j$. □

Claim 9 implies that whenever $n/2 \geq \gamma + 1$ we have

$$\max_{\ell \geq \gamma} m(n, \ell, \gamma) = m(n, \gamma+1, \gamma) = \binom{n}{\gamma+1},$$

and whenever $\gamma \geq n/2$ we have $\max_{\ell \geq \gamma} m(n, \ell, \gamma) = m(n, \gamma, \gamma) = \binom{n}{\gamma}$. Clearly, the inequality $m(n, \ell, \gamma) \geq m(n, \ell, \gamma+1)$ also holds. Both of these inequalities also hold for weak (n, ℓ, γ)-sharing set families.

7.7.4 Open Questions

We conclude with some open questions.

We have shown that our explicit construction of (n, ℓ, γ)-sharing set families can be used with the weaker requirements of Raz et al [126] to create weak (n, ℓ, γ)-sharing set families of arbitrarily large size. Our analysis uses a number of potentially loose bounds, however, so it is possible that a better analysis of our

construction for weak set families could improve our requirements on the parameters. Also of interest is whether there is another explicit construction that would perform better than the Blocki et al construction. The explicit construction of Hartman and Raz[89] runs in sequential time $poly(\log m, \ell)$ and space $O(\log m)$. Our construction runs in parallel time $O(\log \log m)$, but requires more space than the construction of Hartman and Raz. Future, work could explore the space-depth trade-off in explicit constructions of weak (n, ℓ, γ)-sharing set families.

We have shown that the value $m(n, n/c_1, nc_2)$ is constant whenever $c_2 \leq \frac{2}{c_1^3+c_1^2}$. Furthermore, we showed that whenever $c_2 > \frac{1}{c_1^2}$, $m(n, n/c_1, nc_2)$ grows exponentially. How does $m(n, n/c_1, nc_2)$ grow whenever $c_2 \in \left[\frac{2}{c_1^3+c_1^2}, \frac{1}{c_1^2}\right]$?

We have shown that $\pi\pi(n)$ never exceeds $\pi(n) - \pi(\frac{n}{2}) + \pi(\sqrt{n})$. We hypothesize that $\pi\pi(n) = \pi(n) - \pi(\frac{n}{2}) + \pi(\sqrt{n})$ for all $n \geq 55$. A simple method to select a maximally-sized set of relatively prime integers is to take the square of each prime between $\sqrt{\frac{n}{2}}$ and $\sqrt{n}$, and the product of the j'th prime less than $\sqrt{\frac{n}{2}}$ and the k'th prime greater than $\sqrt{n}$, for j from 1 to $\pi(\sqrt{n})$ and $k = j$ unless this would make the product less than $\frac{n}{2}$ in which case k is chosen to be the minimum value greater than the previous k so that the product is great than $\frac{n}{2}$. With the aid of a computer we have shown this equation true for all n from 1 to 100,000, except for 51, 52, 53, and 54.

Chapter 8

Appendix: Human Computable Passwords

8.1 Human Computable Passwords Challenge

	Scheme 1		Scheme 2	
n	m	Winner	m	Winner
100 digits	1000	N/A	500	N/A
	500	N/A	300	N/A
	300	N/A	200	N/A
50 digits	500	N/A	300	N/A
	300	N/A	150	N/A
	150	N/A	100	N/A
30 digits	300	N/A	150	N/A
	100	N/A	100	N/A
	50	N/A	50	N/A

Table 8.1: Human Computable Password Challenges
n — Secret Length
m—# Challenge-Response Pairs

While we provided asymptotic security bounds for our human computable password schemes in our context it is particularly important to understand the constant factors. In our context, we can assume that $n \leq 100$ so it would be feasible for the adversary to execute an attack that takes time proportional to $10^{\sqrt{n}} \leq 10^{10}$. We conjecture that in practice scheme 2 is slightly weaker than scheme 1 when $n \leq 100$ despite the fact that $s(f_1) < s(f_2)$ because of the attack described in remark 2. This attack requires $\tilde{O}\left(n^{1+g(f)/2}\right)$ examples, and the running time $O\left(d^{\sqrt{n}} poly(n)\right)$ may be feasible for $n \leq 100$. To better understand the exact security bounds we created several public challenges for researchers to break our human computable password schemes under different parameters (see Table 8.1). The challenges can be found at `http://www.cs.cmu.edu/~jblocki/HumanComputablePasswordsChallenge/challenge.htm`. These challenges were presented during the rump session at ASIACRYPT 2013 [29]. For each challenge we selected a random secret mapping $\sigma \in \mathbb{Z}_{10}^n$, and published (1) m single digit challenge-response pairs $(C_1, f(\sigma(C_1))), \ldots, (C_m, f(\sigma(C_m)))$, where each clause C_i is chosen uniformly at random from X_k, and (2) 20 length—10 password challenges

$\vec{C}_1, \ldots, \vec{C}_{20} \in (X_k)^{10}$. The goal is to guess one of the secret passwords $p_i = f\left(\sigma\left(\vec{C}_i\right)\right)$ for some $i \in [20]$.

8.2 Statistical Dimension

Our statistical dimension lower bounds closely mirror the lower bounds from [73] for binary predicates. In particular Lemmas 5,6,3 ,7 and 8 are similar to Lemmas 2, 4,5, 6 and 7 from [73] respectively. The high level proof strategy is also very similar. Because we are working with planted solutions $\sigma \in \mathbb{Z}_d^n$, instead of $\sigma \in \{\pm 1\}^n$, we need to use different Fourier basis functions. We use the basis functions χ_α where for $\alpha \in \mathbb{Z}_d^n$ is

$$\chi_\alpha(x) = \exp\left(\frac{-2\pi\sqrt{-1}(x \cdot \alpha)}{d}\right).$$

While the Fourier coefficients $\hat{b}_\alpha$ of a function $b : \mathbb{Z}_d^k \to \mathbb{R}$ might include complex numbers, Parseval's identity still applies: $\sum_{\alpha \in \mathbb{Z}_d^k} \left|\hat{b}_\alpha\right|^2 = \mathbb{E}_{x \sim \mathbb{Z}_d^k}\left[b(x)^2\right]$. We first consider the following search problem: find σ' that is ϵ-correlated with σ given m randomly chosen challenge clauses from the distribution $Q_\sigma^{f,j}$ for $j \in \mathbb{Z}_d$. Remark 6 explains how to generalize our results to the problem we are interested in: find σ' that is ϵ-correlated with σ given m randomly chosen challenge-response pairs from the distribution Q_σ^f. In this section we let U_k denote the uniform distribution over X_k.

Definition 28. *[73] Given a clause $C \in X_k$ and $S \subseteq [k]$ of size ℓ, we let $C_{|S} \in X_\ell$ denote the clause of variables of C at the positions with indices in S (e.g., if $C = (1, \ldots, k)$ and $S = \{1, 5, k-2\}$ then $C_{|S} = (1, 5, k-2) \in X_3$). Given a function $h : X_k \to \mathbb{R}$ and a clause $C_\ell \in X_\ell$ we define*

$$h_\ell(C_\ell) = \frac{|X_\ell|}{|X_k|} \sum_{S \subseteq [k], |S| = \ell, C \in X_k, C_{|S} = C_\ell} h(C) .$$

We first show that $\Delta(\sigma, h)$ can be expressed in terms of the Fourier coefficients of $\hat{Q}$ as well as the functions h_ℓ. In particular, we define the degree ℓ function $b_\ell : \mathbb{Z}_d^n \to \mathbb{C}$ as follows

$$b_\ell(\sigma) \doteq \frac{1}{|X_\ell|} \sum_{\alpha \in \mathbb{Z}_d^k : H(\alpha) = \ell} \hat{Q}_\alpha \sum_{C_\ell \in X_\ell} \chi_\alpha(\sigma(C_\ell)) h_\ell(C_\ell) .$$

Notice that if Q has distributional complexity r and $\ell \leq r$ then $b_\ell(\sigma) = 0$ because $\hat{Q}_\alpha = 0$ for all $\alpha \in \mathbb{Z}_d^k$ s.t. $1 \leq H(\alpha) \leq r$. This means that first r terms of the sum in Lemma 5 will be zero.

Lemma 5. *For every $\sigma \in \mathbb{Z}_d^k$ and $h : X_k \to \mathbb{R}$*

$$\Delta(\sigma, h) = \sum_{\ell=1}^{k} \frac{1}{|X_\ell|} b_\ell(\sigma) \ .$$

Proof.

$$\begin{aligned}
\mathbb{E}_{Q_\sigma}[h] &= \sum_{C \in X_k} h(C) \cdot Q_\sigma(C) \\
&= \frac{1}{|X_k|} \sum_{C \in X_k} h(C) \cdot Q(\sigma(C)) \\
&= \frac{1}{|X_k|} \sum_{C \in X_k} h(C) \sum_{\alpha \in \mathbb{Z}_d^k} \hat{Q}_\alpha \chi_\alpha(\sigma(C)) \\
&= \frac{1}{|X_k|} \sum_{\alpha \in \mathbb{Z}_d^k} \hat{Q}_\alpha \sum_{C \in X_k} h(C) \chi_\alpha(\sigma(C)) \\
&= \frac{1}{|X_k|} \sum_{\ell=0}^{k} \sum_{\alpha \in \mathbb{Z}_d^k : H(\alpha) = \ell} \hat{Q}_\alpha \sum_{C \in X_k} h(C) \chi_\alpha(\sigma(C))
\end{aligned}$$

Observe that whenever $\alpha = \vec{0}$ we have

$$\hat{Q}_\alpha = \mathbb{E}_{x \sim \mathbb{Z}_d^k}[Q(x) \chi_\alpha(x)] = \mathbb{E}_{x \sim \mathbb{Z}_d^k}[Q(x)] = \sum_{x \in \mathbb{Z}_d^k} \frac{Q(x)}{d^k} = 1 \ .$$

Therefore, for $\ell = 0$ in the above sum we have

$$\begin{aligned}
\frac{1}{|X_k|} \hat{Q}_\alpha \sum_{C \in X_k} h(C) \chi_\alpha(\sigma(C)) &= \frac{1}{|X_k|} \hat{Q}_{\vec{0}} \sum_{C \in X_k} h(C) \\
&= \frac{1}{|X_k|} \sum_{C \in X_k} h(C) \\
&= \mathbb{E}_{U_k}[h] \ .
\end{aligned}$$

Therefore,

$$\begin{aligned}
\mathbb{E}_{Q_\sigma}[h] - \mathbb{E}_{U_k}[h] &= \frac{1}{|X_k|}\sum_{\ell=1}^{k}\sum_{\alpha\in\{0,\ldots,d-1\}^k:H(\alpha)=\ell}\hat{Q}_\alpha\sum_{S\subseteq[k],|S|=\ell}\sum_{C_\ell\in X_\ell}\sum_{C\in X_k,C_{|S}=C_\ell}h(C)\,\chi_\alpha(\sigma(C)) \\
&= \frac{1}{|X_k|}\left(\sum_{\ell=1}^{k}\sum_{\alpha\in\{0,\ldots,d-1\}^k:H(\alpha)=\ell}\hat{Q}_\alpha\sum_{C_\ell\in X_\ell}\chi_\alpha(\sigma(C_\ell))\sum_{S\subseteq[k],|S|=\ell}\sum_{C\in X_k,C_{|S}=C_\ell}h(C)\right) \\
&= \frac{1}{|X_k|}\left(\sum_{\ell=1}^{k}\sum_{\alpha\in\mathbb{Z}_d^k:H(\alpha)=\ell}\hat{Q}_\alpha\sum_{C_\ell\in X_\ell}\chi_\alpha(\sigma(C_\ell))\cdot h_\ell(C_\ell)\frac{|X_k|}{|X_\ell|}\right) \\
&= \left(\sum_{\ell=1}^{k}\frac{1}{|X_\ell|}\sum_{\alpha\in\mathbb{Z}_d^k:H(\alpha)=\ell}\hat{Q}_\alpha\sum_{C_\ell\in X_\ell}\chi_\alpha(\sigma(C_\ell))\,h_\ell(C_\ell)\right) \\
&= \left(\sum_{\ell=1}^{k}\frac{1}{|X_\ell|}b_\ell(\sigma)\right)
\end{aligned}$$

□

The following lemma is similar to Lemma 4 from [73]. Lemma 6 is based on the general hypercontractivity theorem [116, Chapter 10] and applies to more general (non-boolean) functions.

Lemma 6. *[116, Theorem 10.23] If $b : \mathbb{Z}_d^n \to \mathbb{R}$ has degree at most ℓ then for any $t \geq \left(\sqrt{2e/d}\right)^\ell$,*

$$\Pr_{x\sim\mathbb{Z}_d^n}[|b(x)| \geq t\|b\|_2] \leq \frac{1}{d^\ell}\exp\left(-\frac{\ell}{2ed}t^{2/\ell}\right),$$

where $\|b\|_2 = \sqrt{\mathbb{E}_{x\sim\mathbb{Z}_d^n}\left[b(x)^2\right]}$

Lemma 3 and its proof are almost identical to Lemma 5 in [73]. We simply replace their concentration bounds with the concentration bounds in Lemma 6. We include the proof for completeness.

Reminder of Lemma 3. *Let $b : \mathbb{Z}_d^n \to \mathbb{R}$ be any function with degree at most ℓ, and let $\mathcal{S} \subseteq \mathbb{Z}_d^n$ be a set of assignments for which $d' = d^n / |\mathcal{S}| \geq e^\ell$. Then $\mathbb{E}_{\sigma\sim\mathcal{S}}[|b(\sigma)|] \leq \frac{2(\ln d'/c_0)^{\ell/2}}{d^\ell}||b||_2$, where $c_0 = \ell\left(\frac{1}{2ed}\right)$ and $||b||_2 = \sqrt{\mathbb{E}_{x\sim\mathbb{Z}_d^n}\left[b(x)^2\right]}$.*

Proof of Lemma 3. The set $\mathcal{S}$ contains $1/d'$ fraction of points in $\mathbb{Z}_d^n$. Therefore,

$$\Pr_{x\sim\mathcal{S}}\left[|b(x)| \geq t||b||_2\right] \leq \frac{d'}{d^\ell}\exp\left(-\frac{\ell}{2ed}t^{2/\ell}\right),$$

for any $t \geq \left(\sqrt{2e/d}\right)^\ell$. For any random variable Y and value $a \in \mathbb{R}$,

$$\mathbb{E}[Y] \leq a + \int_a^\infty \Pr[Y \geq t]dt..$$

We set $Y = |b(\sigma)|/||b||_2$ and $a = \left(\frac{\ln d'}{d^2 c_0}\right)^{\ell/2}$ Assuming that $a > \left(\sqrt{2e/d}\right)^\ell$ We get

$$\begin{aligned}
\frac{\mathbb{E}_{\sigma\sim\mathcal{S}}[|b(\sigma)|]}{||b||_2} &\leq (\ln d'/c_0)^{\ell/2} + \int_{(\ln d'/c_0)^{\ell/2}}^\infty \frac{d'}{d^\ell}\cdot e^{-c_0 t^{2/\ell}}dt \\
&= \frac{(\ln d'/c_0)^{\ell/2}}{d^\ell} + \frac{\ell\cdot d'}{2d^\ell\cdot c_0^{\ell/2}}\cdot\int_{\ln d'}^\infty e^{-z}z^{\ell/2-1}dz \\
&= \frac{(\ln d'/c_0)^{\ell/2}}{d^\ell} + \frac{\ell\cdot d'}{2d^\ell\cdot c_0^{\ell/2}}\cdot\left(-e^{-z}z^{\ell/2-1}\right)\Big|_{\ln d'}^\infty + (\ell/2-1)\int_{\ln d'}^\infty e^{-z}z^{\ell/2-2}dz \\
&= \ldots \leq \frac{(\ln d'/c_0)^{\ell/2}}{d^\ell} + \frac{\ell\cdot d'}{2d^\ell\cdot c_0^{\ell/2}}\sum_{\ell'=1/2}^{\lceil\ell/2\rceil-1}\left(-\frac{\lceil\ell/2\rceil!}{\ell'!}e^{-z}z^{\ell'}\right)\Big|_{\ln d'}^\infty \\
&= \frac{(\ln d'/c_0)^{\ell/2}}{d^\ell} + \frac{1}{2d^\ell\cdot c_0^{\ell/2}}\sum_{\ell'=0}^{\lceil\ell/2\rceil-1}\frac{\lceil\ell/2\rceil!}{\ell'!}(\ln d')^{\ell'} \leq \frac{2(\ln d'/c_0)^{\ell/2}}{d^\ell},
\end{aligned}$$

where we used the condition $d' \geq e^\ell$ to obtain the last inequality.

□

Lemma 7. *Let $\mathcal{S} \subseteq \{0,\ldots,d-1\}^n$ be a set of assignments for which $d' = d^n/|\mathcal{S}|$. Then*

$$\mathbb{E}_{\sigma\sim\mathcal{S}}\left[\left|\frac{1}{|X_\ell|}b_\ell(\sigma)\right|\right] \leq \frac{4(\ln d'/c_0)^{\ell/2}}{d^\ell}\frac{||h_\ell(\sigma)||_2}{\sqrt{|X_\ell|}}$$

Proof. For simplicity of notation we set $b = b_\ell$. By Parseval's identity we have

$$\begin{aligned}
\mathbb{E}_{\sigma\sim\mathbb{Z}_d^n}\left[b(\sigma)\overline{b(\sigma)}\right] &= \mathbb{E}_{\sigma\sim\mathbb{Z}_d^n}\left[|b(\sigma)|^2\right] \\
&= \mathbb{E}_{\sigma\sim\mathbb{Z}_d^n}\left[\left|\sum_{\substack{\alpha\in\mathbb{Z}_d^k \\ H(\alpha)=\ell}}\sum_{C_\ell\in X_\ell}\hat{Q}_\alpha\chi_\alpha(\sigma(C_\ell))h_\ell(C_\ell)\right|^2\right] \\
&= \mathbb{E}_{\sigma\sim\mathbb{Z}_d^n}\left[\sum_{\substack{\alpha\in\mathbb{Z}_d^k \\ H(\alpha)=\ell}}\sum_{C_\ell\in X_\ell}\left|\hat{Q}_\alpha\right|^2 h_\ell(C_\ell)\right] \\
&= |X_\ell|\sum_{\substack{\alpha\in\mathbb{Z}_d^k \\ H(\alpha)=\ell}}\left|\hat{Q}_\alpha\right|^2\frac{1}{|X_\ell|}\sum_{C_\ell\in X_\ell}h_\ell(C_\ell) \\
&= |X_\ell|\sum_{\substack{\alpha\in\mathbb{Z}_d^k \\ H(\alpha)=\ell}}\left|\hat{Q}_\alpha\right|^2\|h_\ell\|_2^2 \\
&\leq |X_\ell|\,\|h_\ell\|_2^2 .
\end{aligned}$$

Before we can apply Lemma 3 we must address a technicality. The range of $b = b_\ell$ might include complex numbers, but Lemma 3 only applies to functions b with range $\mathbb{R}$. For $c, d \in \mathbb{R}$ we adopt the notation $Im\left(c + d\sqrt{-1}\right) = d$ and $Re\left(c + d\sqrt{-1}\right) = c$. We observe that

$$\begin{aligned}
\mathbb{E}_{\sigma\sim\mathbb{Z}_d^n}\left[b(\sigma)\overline{b(\sigma)}\right] &= \mathbb{E}_{\sigma\sim\mathbb{Z}_d^n}\left[Re(b(\sigma))^2 + Im(b(\sigma))^2\right] \\
&= \|Re(b)\|^2 + \|Im(b)\|^2 .
\end{aligned}$$

We first observe that $Re(b)$ and $Im(b)$ are both degree ℓ functions because we can write

$$Re(b(\sigma)) = \frac{1}{|X_\ell|}\sum_{\alpha\in\mathbb{Z}_d^k : H(\alpha)=\ell}\sum_{C_\ell\in X_\ell} Re\left(\hat{Q}_\alpha\chi_\alpha(\sigma(C_\ell))\,h_\ell(C_\ell)\right)$$

and

$$Im(b(\sigma)) == \frac{1}{|X_\ell|}\sum_{\alpha\in\mathbb{Z}_d^k : H(\alpha)=\ell}\sum_{C_\ell\in X_\ell} Im\left(\hat{Q}_\alpha\chi_\alpha(\sigma(C_\ell))\,h_\ell(C_\ell)\right) .$$

Now we can apply Lemma 3 to get

$$\begin{aligned}\mathbf{E}_{\sigma\sim\mathcal{S}}\left[\left|Re\left(b\left(\sigma\right)\right)\right|\right] &\leq \frac{2\left(\ln d'/c_0\right)^{\ell/2}}{d^{\ell}}\|Re(b)\|_2 \\ &\leq \frac{2\left(\ln d'/c_0\right)^{\ell/2}}{d^{\ell}}\sqrt{|X_\ell|}\|h_\ell\left(\sigma\right)\|_2 .\end{aligned}$$

A symmetric argument can be used to bound $\mathbf{E}_{\sigma\sim\mathcal{S}}\left[Im\left(b\left(\sigma\right)\right)\right]$. Now because

$$\left|b\left(\sigma\right)\right| \leq \left|Re\left(b\left(\sigma\right)\right)\right| + \left|Im\left(b\left(\sigma\right)\right)\right| ,$$

it follows that

$$\begin{aligned}\mathbf{E}_{\sigma\sim\mathcal{S}}\left[\left|\frac{1}{|X_\ell|}b\left(\sigma\right)\right|\right] &\leq 2\left(\frac{1}{|X_\ell|}\right)\left(\frac{2\left(\ln d'/c_0\right)^{\ell/2}}{d^{\ell}}\right)\|h_\ell\left(\sigma\right)\|_2\sqrt{|X_\ell|} \\ &\leq \frac{4\left(\ln d'/c_0\right)^{\ell/2}}{d^{\ell}}\frac{\|h_\ell\left(\sigma\right)\|_2}{\sqrt{|X_\ell|}} .\end{aligned}$$

□

We will use Fact 6 to prove Lemma 8. The proof of Fact 6 is found in [73, Lemma 7]. We include it here for completeness.

Fact 6. *[73] If $h : X_\ell \to \mathbb{R}$ satisfies $\|h\|_2^2 = 1$ then $\|h_\ell\|_2^2 \leq 1$.*

Proof. First notice that for any C_ℓ,

$$\left|\{C \in X_k \,\middle|\, \exists S \subseteq [k], \text{ s.t. } |S| = \ell \wedge C_{|S} = C_\ell\}\right| = \frac{|X_k|}{|X_\ell|} .$$

By applying the definition of h_ℓ along with the Cauchy-Schwartz inequality

$$\begin{aligned}\|h_\ell\|_2^2 &= \mathbb{E}_{C_\ell\sim U_\ell}\left[h_\ell\left(C_\ell\right)^2\right] \\ &= \left(\frac{|X_\ell|}{|X_k|}\right)^2\mathbb{E}_{C_\ell\sim U_\ell}\left[\left(\sum_{S\subseteq[k],|S|=\ell,C\in X_k,C_{|S}=C_\ell} h\left(C\right)\right)^2\right] \\ &\leq \left(\frac{|X_\ell|}{|X_k|}\right)^2\mathbb{E}_{C_\ell\sim U_\ell}\left[\frac{|X_k|}{|X_\ell|}\left(\sum_{S\subseteq[k],|S|=\ell,C\in X_k,C_{|S}=C_\ell} h\left(C\right)^2\right)\right] \\ &\leq \left(\frac{|X_\ell|}{|X_k|}\right)\mathbb{E}_{C_\ell\sim U_\ell}\left[\left(\sum_{S\subseteq[k],|S|=\ell,C\in X_k,C_{|S}=C_\ell} h\left(C\right)^2\right)\right] \\ &= \mathbb{E}_{C\sim U_k}\left[h\left(C\right)^2\right] = \|h\|_2^2 = 1 .\end{aligned}$$

□

Lemma 8. *Let Q be a clause distribution with distributional complexity $r = r(Q)$, let $\mathcal{D}' \subseteq \{Q_\sigma\}_{\sigma\in\{0,\ldots,d-1\}^n}$ be a set of distributions over clauses and $d' = d^n/|\mathcal{D}'|$. Then $\kappa_2(\mathcal{D}', U_k) = O_k\left((\ln d'/n)^{r/2}\right)$*

Proof. Let $\mathcal{S} = \{\sigma | Q_\sigma \in \mathcal{D}'\}$ and let $h : X_k \to \mathbb{R}$ be any function such that $\mathbb{E}_{U_k}[h^2] = 1$. Using Lemma 5 and the definition of r,

$$\begin{aligned} |\Delta(\sigma, h)| &= \left|\sum_{\ell=r}^{k} \frac{1}{|X_\ell|} b_\ell(\sigma)\right| \\ &\leq \sum_{\ell=r}^{k} \left|\frac{1}{|X_\ell|} b_\ell(\sigma)\right| . \end{aligned}$$

We apply Lemma 7 and Fact 6 to get

$$\begin{aligned} \mathbb{E}_{\sigma\sim\mathcal{S}}[|\Delta(\sigma, h)|] &\leq \sum_{\ell=r}^{k} \left(\frac{4(\ln d'/c_0)^{\ell/2}}{d^\ell} \frac{\|h_\ell(\sigma)\|_2}{\sqrt{|X_\ell|}}\right) \\ &\leq \sum_{\ell=r}^{k} \left(\frac{4(\ln d'/c_0)^{\ell/2}}{d^\ell \sqrt{|X_\ell|}}\right) \\ &\leq O_k\left(\frac{(\ln d')^{\ell/2}}{d^\ell n^{r/2}}\right) . \end{aligned}$$

□

Remark 6. *Recall that Q^f_σ denotes the uniform distribution over pairs $(C, i) \in X_k \times \mathbb{Z}_d$ that satisfy $f(\sigma(C)) = i$. If we let U'_k denote the uniform distribution over $X_k \times \mathbb{Z}_d$ then for any function $h : X_k \times \mathbb{Z}_d \to \mathbb{R}$ we can apply Lemma 8 to write*

$$\begin{aligned} \mathbb{E}_{(C,j)\sim Q_\sigma}[h(C, j)] - \mathbb{E}_{(C,j)\sim U_k}[h(C, j)] &= \sum_{i=1}^{d} \Pr_{C\sim U_k}[f(\sigma(C)) = i]\left(\mathbb{E}_{C\sim Q^{f,i}_\sigma}\left[h^i(C)\right] - \mathbb{E}_{C\sim U_k}\left[h^i(C)\right]\right) \\ &\leq \sum_{i=1}^{d} \max_j \left\{\mathbb{E}_{C\sim Q^{f,j}_\sigma}\left[h^j(C)\right] - \mathbb{E}_{C\sim U_k}\left[h^j(C)\right]\right\} \\ &\leq O_k\left(\frac{(\ln d')^{\ell/2}}{d^\ell n^{r/2}}\right) , \end{aligned}$$

where $h^i(C) = h(C, i)$.

Reminder of Theorem 7. *Let $\mathcal{Z}_{Q,\epsilon}$ denote the problem of finding for every $\sigma \in \mathbb{Z}_d^n$, an assignment $\tau \in \mathbb{Z}_d^k$ that is ϵ-correlated with σ given access to distribution Q_σ^f over $X_k \times \mathbb{Z}_d$. Then there exists a constant $c_Q > 0$ such that for any $\epsilon > 1/\sqrt{n}$ and $q \geq n$,*

$$\mathrm{SDN}\left(\mathcal{Z}_{Q,\epsilon}, \frac{c_Q (\log q)^{r/2}}{n^{r/2}}, 2e^{-n\cdot\epsilon^2/2}\right) \geq q \ ,$$

where $r = r(f)$ is the distributional complexity of f.

Proof of Theorem 7.

We use the uniform distribution U_k' over $\mathbb{Z}_d^{k+1}$ as our reference distribution and we use $\mathcal{D}_U = \mathcal{D} = \{Q_\sigma^f\}_{\sigma\in\mathbb{Z}_d^n}$ to denote the set of distributions for all possible assignments. First note that, by Chernoff bounds, for any solution $\tau \in \mathbb{Z}_d^n$ the fraction of assignments $\sigma \in \mathbb{Z}_d^n$ such that τ and σ are ϵ-correlated (e.g., $H(\sigma,\tau) \leq \frac{n(d-1)}{d} - \epsilon \cdot n$) is at most $e^{-2n\cdot\epsilon^2}$. In other words $|\mathcal{D}_\tau| \geq \left(1 - e^{-2n\cdot\epsilon^2}\right)|\mathcal{D}|$, where $\mathcal{D}_\tau = \mathcal{D}\backslash\mathcal{Z}_{Q,\epsilon}^{-1}(\tau)$. Let $\mathcal{D}' \subseteq \mathcal{D}_\tau$ be a set of distributions of size $|\mathcal{D}_\tau|/q$ and $\mathcal{S} = \{\sigma \,\big|\, Q_\sigma^f \in \mathcal{D}'\}$. Then for $d' = d^n/|\mathcal{D}'| = q \cdot d^n/|\mathcal{D}_\tau|$, by Lemma 8 and remark 6, we get

$$\begin{aligned} \kappa_2(\mathcal{D}', U_k') &= O_k\left(\frac{(\ln d')^{r/2}}{n^{r/2}}\right) && (8.1)\\ &= O_k\left(\frac{(\ln q)^{r/2}}{n^{r/2}}\right) , && (8.2)\end{aligned}$$

where the last line follows by Sterling's Approximation

$$q = d'|\mathcal{D}_\tau|/d^n = d'|\mathcal{D}_\tau|/d^n \approx d'c'\sqrt{\frac{d}{n}}$$

for a constant c'. The claim now follows from the definition of SDN. □

The proof of Theorem 6 follows from Theorem 7 and the following result of Feldman et al. [73].

Reminder of Theorem 5 [73, Theorems 10 and 12]. *Let X be a domain and $\mathcal{Z}$ be a search problem over a set of solutions $\mathcal{F}$ and a class of distributions $\mathcal{D}$ over X. For $\kappa > 0$ and $\eta \in (0,1)$, let $d' = \mathrm{SDN}(\mathcal{Z},\kappa,\eta)$. Let D be the reference distribution and $\mathcal{D}_D$ be a set of distributions for which the value d' is achieved. Any randomized statistical algorithm that, given access to a VSTAT$\left(\frac{1}{3\kappa^2}\right)$ (resp. 1-MSTAT(L)) for a distribution chosen randomly and uniformly from $\mathcal{D}_D$, succeeds with probability $\Lambda > \eta$ over the choice of distribution*

and internal randomness requires at least $\frac{\Lambda-\eta}{1-\eta}d'$ *(resp.* $\Omega\left(\frac{1}{L}\min\left\{\frac{d'(\Lambda-\eta)}{1-\eta},\frac{(\Lambda-\eta)^2}{\kappa^2}\right\}\right)$*) calls to the oracle.*

Reminder of Theorem 6. *Let* $\sigma \in \mathbb{Z}_d^n$ *denote a secret mapping chosen uniformly at random and let* $\mathcal{Z}_{Q^f}$ *be a planted constrained satisfiability problem with distribution* Q_σ^f *over* $X_k \times \mathbb{Z}_d$*, where* f *has distributional complexity* $r = r(f)$*. Any randomized statistical algorithm that finds an assignment* τ *such that* τ *is* $\left(\sqrt{\frac{-2\ln(\eta/2)}{n}}\right)$*-correlated with* σ *with probability at least* $\Lambda > \eta$ *over the choice of* σ *and the internal randomness of the algorithm needs at least* m *calls to the 1-MSTAT*(L) *oracle (resp. VSTAT*$\left(\frac{n^r}{2(\log n)^{2r}}\right)$*)with* $m \cdot L \geq c_1\left(\frac{n}{\log n}\right)^r$ *(resp.* $m \geq n^{c_1 \log n}$*) for a constant* $c_1 = \Omega_{k,1/(\Lambda-\eta)}(1)$*. In particular if we set* $L = \left(\frac{n}{\log n}\right)^{r/2}$ *then our algorithms needs at least* $m \geq c_1\left(\frac{n}{\log n}\right)^{r/2}$ *calls to 1-MSTAT*(L)*.*

Proof of Theorem 6. We set $\epsilon = \sqrt{\frac{-2\ln(\eta/2)}{n}}$ and observe that $2e^{-n\epsilon^2/2} = \eta$, and we set $q = n^{\log n}$ in Theorem 7. Notice that we used $\mathcal{D}_U = \{Q_\sigma^f\}_{\sigma\in\mathbb{Z}_d^n}$ in the proof of Theorem 7. Now we apply Theorem 5 to get the desired lower bound

$$m = \Omega\left(\min\left\{\frac{n^{\Omega_k(\log n)}(\Lambda-\eta)}{1-\eta}, \left(\frac{n^{r/2}}{c_Q(\log(n^{\log n}))^{r/2}}\right)^2(\Lambda-\eta)^2\right\}/L\right) = \Omega\left(\left(\frac{n^r}{\log^{2r} n}\right)/L\right),$$

for the 1-MSTAT (L) oracle. For the VSTAT$\left(\frac{n^r}{2(\log n)^{2r}}\right)$ oracle we get $m = \frac{n^{\Omega_k(\log n)}(\Lambda-\eta)}{1-\eta}$.

□

8.3 Security Proofs

Reminder of Theorem 8. *Let* f *be* (δ_1, δ_2)*—hard to predict, let* $\sigma \sim \mathbb{Z}_d^n$ *denote the secret mapping, let* $\epsilon > 0$ *be any constant and suppose that we are given labels* $\ell_C \in \mathbb{Z}_d$ *for every* $C \in X_k$ *s.t*

$$\Pr_{C\sim X_k}[f(\sigma(C)) = \ell_C] \geq \frac{1}{d} + \delta_2 + \epsilon\,.$$

There is a polynomial time algorithm (in $n, 1/\epsilon, 1/\delta_2$*) that with high probability finds a mapping* $\sigma' \in \mathbb{Z}_d^n$ *such that* σ' *is* δ_1*-correlated with* σ *provided that* σ *is* δ_1*-balanced.*

Proof of Theorem 8. Let $\sigma \in \mathbb{Z}_d^n$ be given such that σ is δ_1-balanced (e.g., $\frac{d-1}{d} - \delta_1 \leq \max_{i \in \mathbb{Z}_d} \left(\frac{H(\sigma, \vec{i})}{n} \right) \leq \frac{d-1}{d} + \delta_1$.

We set $\tau = \frac{2}{\epsilon^2} \ln(T)$ and select clauses $C_1^j, \ldots C_\tau^j$ at random for $j \in [y]$. Then by Chernoff bounds

$$\begin{aligned} \Pr\left[\left| \left\{ i \,\middle|\, \ell_{C_i^j} = f\left(\sigma\left(C_i^j\right)\right) \right\} \right| \leq \frac{\tau}{d} + \tau\delta_2 \right] &\leq \exp\left(2\frac{2}{\epsilon^2} \ln(T)\epsilon^2\right) \\ &= \exp(4\ln(T)) \\ &= \frac{1}{T^4} . \end{aligned}$$

Let $T = n$, set $S^j = \bigcup_{i=1}^{\tau} C_i^j$, and define BAD^j to be the event that

$$\frac{\left| \left\{ C \in X_k \,\middle|\, C \subseteq S^j \wedge \ell_C = f(\sigma(C)) \right\} \right|}{|\{C \in X_k \,|\, C \subseteq S^j\}|} \leq \frac{1}{d} + \delta_2 .$$

By the union bound

$$\begin{aligned} \Pr\left[BAD^i\right] &\leq \binom{\frac{2k\ln T}{\epsilon^2}}{k} \frac{1}{T^4} \\ &\leq \frac{2^k k^k \ln^k T}{\delta^{2k} T^4} \\ &\leq \frac{1}{n^3} . \end{aligned}$$

If we set $y = O(n \log n)$ then with high probability $\bigcup_{j=1}^{y} S^j = [n]$. By applying the union bound again we have

$$\Pr\left[\exists j \in [y].BAD^j\right] \leq \frac{1}{n^{1.5}} .$$

Now for each $j \in [y]$ we can enumerate over all $n^{O((2k\ln d)/\delta_2^2)}$ mappings $\sigma^j \in \mathbb{Z}_d^{|S^j|}$ to find one that satisfies

$$\frac{\left| \left\{ C \in X_k \,\middle|\, C \subseteq S^j \wedge \ell_C = f\left(\sigma^j(C)\right) \right\} \right|}{|\{C \in X_k \,|\, C \subseteq S^j\}|} \geq \frac{1}{d} + \delta_2 ,$$

in polynomial time. Because f is (δ_1, δ_2)-hard to predict $\sigma^j\left(S_j\right)$ must be δ_1-correlated with $\sigma\left(S_j\right)$ (e.g., $\frac{H\left(\sigma\left(S^j\right),\sigma^j\left(S^j\right)\right)}{\left|S^j\right|} \leq \frac{d-1}{d} - \delta_1$). Now we can find σ' s.t. $H\left(\sigma, \sigma'\right) \leq \frac{d-1}{d} - \delta_1$ by combining the σ^i's. □

Reminder of Claim 1. *Let $\mathcal{A}$ be an adversary s.t* $\Pr\left[\textbf{Wins}\left(\mathcal{A}, n, m, t\right)\right] > \left(\frac{1}{d} + \delta + \epsilon\right)^t$ *and let $b = \mathcal{A}_{C_1,\ldots,C_m}$ then*

$$\Pr_{\substack{i\sim[t], C\sim X_k \\ C_1,\ldots,C_m\sim X_k \\ C'_1,\ldots,C'_t\sim X_k}}\left[\mathcal{P}_{b,C'_1,\ldots,C'_t}\left(C, i\right) = f\left(\sigma\left(C\right)\right) \,\middle|\, \mathcal{P}_{b,C'_1,\ldots,C'_t}\left(C, i\right) \neq \perp\right] \geq \left(\frac{1}{d} + \delta + \epsilon\right).$$

Proof of Claim 1. We draw examples $\left(C_1, f\left(\sigma\left(C_1\right)\right)\right), \ldots, \left(C_m, f\left(\sigma\left(C_m\right)\right)\right)$ to construct $b = \mathcal{A}_{C_1,\ldots,C_m}$. Given a random length-$t$ password challenge $\left(C'_1, \ldots, C'_t\right) \in \left(X_k\right)^t$ we let

$$p_j = \Pr_{C,C_1,\ldots,C_m,C'_1,\ldots,C'_t\sim X_k}\left[\mathcal{P}_{b,j,C'_1,\ldots,C'_t}\left(C\right) = f\left(\sigma\left(C\right)\right) \,\middle|\, \mathcal{P}_{b,j,C'_1,\ldots,C'_t}\left(C\right) \neq \perp\right]$$

denote the probability that the adversary correctly guesses the response to the j'th challenge conditioned on the event that the adversary correctly guesses all of the earlier challenges. Observe that

$$\Pr_{C,C_1,\ldots,C_m,C'_1,\ldots,C'_{t-1}\sim X_k, i\sim[t]}\left[\mathcal{P}_{b,i,C'_1,\ldots,C'_t}\left(C, i\right) = f\left(\sigma\left(C\right)\right)\right] = \sum_{i=1}^{t} p_i/t\,,$$

so it suffices to show that $\sum_{i=1}^{t} p_i/t \geq \frac{1}{d} + \delta + \epsilon$. We obtain the following constraint

$$\begin{aligned}
\prod_{i=1}^{t} p_i &= \prod_{i=1}^{t} \Pr_{C,C_1,\ldots,C_m,C'_1,\ldots,C'_t\sim X_k}\left[\mathcal{P}_{b,j,C'_1,\ldots,C'_t}\left(C\right) = f\left(\sigma\left(C\right)\right) \,\middle|\, \mathcal{P}_{b,j,C'_1,\ldots,C'_t}\left(C\right) \neq \perp\right] \\
&= \prod_{i=1}^{t} \Pr_{C_1,\ldots,C_m,C'_1,\ldots,C'_t\sim X_k}\left[\mathcal{A}_{C_1,\ldots,C_m}\left(C'_1, \ldots, C'_t\right)[i] = f\left(\sigma\left(C'_i\right)\right) \,\middle|\right. \\
&\qquad \left. \forall j < i.\ \mathcal{A}_{C_1,\ldots,C_m}\left(C'_1, \ldots, C'_t\right)[j] = f\left(\sigma\left(C'_j\right)\right)\right] \\
&= \Pr_{C_1,\ldots,C_m,C'_1,\ldots,C'_t\sim X_k}\left[\mathcal{A}_{C_1,\ldots,C_m}\left(C'_1, \ldots, C'_t\right) = \left(f\left(\sigma\left(C'_1\right)\right), \ldots, f\left(\sigma\left(C'_t\right)\right)\right)\right] \\
&\geq \left(\frac{1}{d} + \delta + \epsilon\right)^t.
\end{aligned}$$

If we minimize $\sum_{i=1}^{t} p_i/t$ subject to the constraint $\prod_{i=1}^{t} p_i \geq \left(\frac{1}{d} + \delta + \epsilon\right)^t$ then we obtain the desired upper bound $\sum_{i=1}^{t} p_i/t \geq \frac{1}{d} + \delta + \epsilon$. □

Reminder of Theorem 9. *Suppose that f is (δ_1, δ_2)—hard to predict, but that f is not* $\mathbf{UF} - \mathbf{RCA}(n, m, t, \delta)$ *– secure for $\delta > \left(\frac{1}{d} + \delta_2 + \epsilon\right)^t$. Then there is a probabilistic polynomial time algorithm (in n, m, $1/\delta_1$, $1/\delta_2$, $1/\epsilon$) that extracts a string $\sigma' \in \mathbb{Z}_d^n$ that is c-correlated with σ after seeing $\tilde{O}(m)$ examples, where $c > 0$ is a constant.*

Proof of Theorem 9. (sketch) We first partition X_k into $T = O\left(\frac{\log|X_k|}{\epsilon}\right)$ sets $S_1, \ldots, S_T$ of equal size. We let U_i^j denote the set of unlabeled clauses from S_j at time i. Initially, $U_0^j = S_j$. During step i we draw mT labeled examples $\left(C_1^{i,j}, f\left(\sigma\left(C_1^{i,j}\right)\right)\right), \ldots, \left(C_m^{i,j}, f\left(\sigma\left(C_m^{i,j}\right)\right)\right) \sim Q_\sigma^f$ and t labeled examples $\left(\hat{C}_1^{i,j}, f\left(\sigma\left(\hat{C}_1^{i,j}\right)\right)\right), \ldots, \left(\hat{C}_t^{i,j}, f\left(\sigma\left(\hat{C}_t^{i,j}\right)\right)\right) \sim Q_\sigma^f$. For each clause $C \in U_i$ we select $k_C \sim [t]$ uniformly at random. We set

$$U_{i+1}^j = \left\{ C \in U_i^j \,\middle|\, \mathcal{P}_{b,k_C,\hat{C}_1^{i,j},\ldots,\hat{C}_t^{i,j}}(C) = \perp \right\},$$

and we set

$$\ell_C = \mathcal{P}_{b,k_C,\hat{C}_1^{i,j},\ldots,\hat{C}_t^{i,j}}(C)$$

for all $C \in U_i^j \backslash U_{i+1}^j$. Here, $b = \mathcal{A}_{C_1^i,\ldots,C_m^i}$.

We first argue that $O\left(\frac{\ln|X_k|}{\ln\left(\frac{t}{t-1}\right)}\right)$ rounds suffice to label every clause $C \in X_k$. Notice that $\forall C \in X_k$ we have

$$\Pr_{\substack{j\sim[t] \\ C_1^{i,j},\ldots,C_m^{i,j}\sim X_k \\ \hat{C}_1^{i,j},\ldots,\hat{C}_t^{i,j}\sim X_k}} \left[\mathcal{P}_{b,j,\hat{C}_1^{i,j},\ldots,\hat{C}_t^{i,j}}(C) \neq \perp\right] \geq \Pr_{j\sim[t]}[j = 1] = \frac{1}{t}.$$

The probability that a clause C hasn't been labeled after i rounds is at most $\left(1 - \frac{1}{t}\right)^i$. By union bounds the probability that any clause is unlabeled is at most $|X_k|\left(1 - \frac{1}{t}\right)^i$ so after $i = O\left(\frac{\ln|X_k|}{\ln\left(\frac{t}{t-1}\right)}\right)$ rounds we will have $U_i^j = \emptyset$ for all $j \in [T]$.
We now argue that with high probability we label at least $\frac{1}{d} + \delta + \epsilon/2$ clauses correctly. Formally, let

$$x_i = \left|\left\{C \in S_i \,|\, \ell_C = f(\sigma(C))\right\}\right|,$$

denote the number of clauses in S_i labeled correctly. Notice that the random variables are independent, and by Claim 1 we have

$$\mathbb{E}\left[\sum_{i=1}^{T} \frac{x_i}{|X_k|}\right] \geq \frac{1}{d} + \delta + \epsilon \,.$$

Now by Chernoff Bounds it follows that with probability $1 - o(1)$

$$\sum_{i=1}^{T} \frac{x_i}{|X_k|} \geq \frac{1}{d} + \delta + \epsilon/2 \,,$$

so we can apply Lemma 8 to obtain the desired result.

□

8.4 Proofs of Claims and Facts

Reminder of Claim 2. *$r(f_1) = 3$, $g(f_1) = 2$ and $s(f_1) = 3/2$.*

Proof of Claim 2. We first observe that if we fix the values of $x_{10}, x_{11} \in \mathbb{Z}_{10}$ then $f'(x_0, \ldots, x_9, x_{12}, x_{13}) = f_1(x_0, \ldots, x_{13})$ is a linear function. Thus, $g(f_1) = 2$. We also note that for any $\alpha \in \mathbb{Z}_{10}^{14}$ s.t. $H(\alpha) < 3$ and $i, t \in \mathbb{Z}_{10}$ that

$$\Pr_{x \sim \mathbb{Z}_{10}^{14}}[f_1(x) = t \,|\, \alpha \cdot x \equiv i \mod 10] = \Pr_{x \sim \mathbb{Z}_{10}^{14}}[f_1(x) = t] = \frac{1}{10} \,.$$

Therefore,

$$\begin{aligned}
\hat{Q}_\alpha^{f_1,t} &= \mathbb{E}_{x \sim \mathbb{Z}_{10}^k}\left[Q^{f_1,t}(x)\,\chi_\alpha(x)\right] \\
&= \sum_{i=0}^{9} \Pr[\alpha \cdot x \equiv i \mod 10]\, \mathbb{E}_{x \sim \mathbb{Z}_{10}^k}\left[Q^{f_1,t}(x)\,\chi_\alpha(x) \,\middle|\, \alpha \cdot x \equiv i \mod 10\right] \\
&= \sum_{i=0}^{9} \exp\left(\frac{-2\pi i\sqrt{-1}}{10}\right) \Pr[\alpha \cdot x \equiv i \mod 10]\, \mathbb{E}_{x \sim \mathbb{Z}_{10}^k}\left[Q^{f_1,t}(x) \,\middle|\, \alpha \cdot x \equiv i \mod 10\right] \\
&= \frac{1}{10}\sum_{i=0}^{9} \exp\left(\frac{-2\pi i\sqrt{-1}}{10}\right) \mathbb{E}_{x \sim \mathbb{Z}_{10}^k}\left[Q^{f_1,t}(x) \,\middle|\, \alpha \cdot x \equiv i \mod 10\right] \\
&= 0 \,,
\end{aligned}$$

which implies that $r(f_1) \geq 3$. □

Reminder of Claim 3. *$r(f_2) = 4$, $g(f_2) = 1$ and $s(f_2) = 2$.*

Proof of Claim 3. We first observe that if we fix the values of $x_{10} \in \mathbb{Z}_{10}$ then $f'(x_0, \ldots, x_9, x_{11}, x_{12}, x_{13}) = f_2(x_0, \ldots, x_{13})$ is a linear function. Thus, $g(f_1) = 1$. We also note that for any $\alpha \in \mathbb{Z}_{10}^{14}$ s.t. $H(\alpha) < 4$ and $i, t \in \mathbb{Z}_{10}$ that

$$\Pr_{x \sim \mathbb{Z}_{10}^{14}} \left[f_2(x) = t \,\middle|\, \alpha \cdot x \equiv i \mod 10 \right] = \Pr_{x \sim \mathbb{Z}_{10}^{14}} \left[f_2(x) = t \right] = \frac{1}{10} .$$

Therefore,

$$\begin{aligned}
\hat{Q}_{\alpha}^{f_1,t} &= \mathbb{E}_{x \sim \mathbb{Z}_{10}^{k}} \left[Q^{f_1,t}(x) \chi_{\alpha}(x) \right] \\
&= \sum_{i=0}^{9} \Pr \left[\alpha \cdot x \equiv i \mod 10 \right] \mathbb{E}_{x \sim \mathbb{Z}_{10}^{k}} \left[Q^{f_1,t}(x) \chi_{\alpha}(x) \,\middle|\, \alpha \cdot x \equiv i \mod 10 \right] \\
&= \sum_{i=0}^{9} \exp\left(\frac{-2\pi i \sqrt{-1}}{10} \right) \Pr \left[\alpha \cdot x \equiv i \mod 10 \right] \mathbb{E}_{x \sim \mathbb{Z}_{10}^{k}} \left[Q^{f_1,t}(x) \,\middle|\, \alpha \cdot x \equiv i \mod 10 \right] \\
&= \frac{1}{10} \sum_{i=0}^{9} \exp\left(\frac{-2\pi i \sqrt{-1}}{10} \right) \mathbb{E}_{x \sim \mathbb{Z}_{10}^{k}} \left[Q^{f_1,t}(x) \,\middle|\, \alpha \cdot x \equiv i \mod 10 \right] \\
&= 0 ,
\end{aligned}$$

which implies that $r(f_2) \geq 4$. □

Reminder of Fact 2. *f_1 is $(0.01, 0.045)$—hard to predict.*

Proof of Fact 2. Let $\sigma, \sigma' \in \mathbb{Z}_{10}^{n}$ be given. We assume that σ is $\left(\frac{1}{100}\right)$-balanced and that σ and σ' are not $\left(\frac{1}{100}\right)$-correlated. This means that $\frac{9}{100} \leq \max_{i \in \mathbb{Z}_{10}} \left(n - \frac{H(\sigma, \vec{i})}{n} \right) \leq \frac{11}{100}$ and $\left(1 - \frac{H(\sigma,\sigma')}{n}\right) < \frac{11}{100}$. It suffices to show that

$$\Pr_{C=(x_1,\ldots,x_{14}) \sim X_{14}} \left[f(\sigma(C)) = f(\sigma'(C)) \right] \leq \frac{145}{1000} .$$

For $j \in \mathbb{Z}_{10}$ we let $p_j = \frac{n - H(\sigma_j, \sigma')}{n}$ where $\sigma_j(i) = (\sigma(i) + j \mod 10)$. In particular, $p_0 = \frac{n - H(\sigma,\sigma')}{n}$ denotes the probability that $\sigma(i) = \sigma'(i)$ for a random index $i \sim [n]$. By assumption $p_0 \leq \frac{1}{10} + \frac{1}{100}$. We have

$$
\begin{aligned}
&\Pr_{C=(x_1,\ldots,x_{14})\sim X_{14}}\left[f_1\left(\sigma\left(C\right)\right)=f_1\left(\sigma'\left(C\right)\right)\right]\\
=&\left(\sum_{\substack{i,j\in\mathbb{Z}_{10}\\ i+j\equiv 0 \mod 10}} p_ip_j\right)\Pr_{C=(x_1,\ldots,x_{14})\sim X_{14}}\left[f_1\left(\sigma\left(C\right)\right)=f_1\left(\sigma'\left(C\right)\right)\mid\right.\\
&\qquad\left.\sigma\left(x_{11}\right)+\sigma\left(x_{12}\right)\equiv\sigma'\left(x_{11}\right)+\sigma'\left(x_{12}\right)\mod 10\right]\\
&+\left(1-\sum_{\substack{i,j\in\mathbb{Z}_{10}\\ i+j\equiv 0 \mod 10}} p_ip_j\right)\Pr_{C=(x_1,\ldots,x_{14})\sim X_{14}}\left[f_1\left(\sigma\left(C\right)\right)=f_1\left(\sigma'\left(C\right)\right)\mid\right.\\
&\qquad\left.\sigma\left(x_{11}\right)+\sigma\left(x_{12}\right)\not\equiv\sigma'\left(x_{11}\right)+\sigma'\left(x_{12}\right)\mod 10\right]\\
=&\left(\sum_{\substack{i,j\in\mathbb{Z}_{10}\\ i+j\equiv 0 \mod 10}} p_ip_j\right)\Pr_{(x_1,x_2,x_3)\sim X_3}\left[\sum_{i=1}^{3}\left(\sigma\left(x_i\right)-\sigma'\left(x_i\right)\right)\equiv 0 \mod 10\right]\\
&+\left(1-\left(\sum_{\substack{i,j\in\mathbb{Z}_{10}\\ i+j\equiv 0 \mod 10}} p_ip_j\right)\right)\Pr_{\left(x,y,x_{13},x_{14}\right)\sim X_{14}}\left[\sigma\left(x\right)+\sigma\left(x_{13}\right)+\sigma\left(x_{14}\right)\equiv\right.\\
&\qquad\left.\sigma'\left(y\right)+\sigma'\left(x_{13}\right)+\sigma'\left(x_{14}\right)\mod 10\right]\\
=&\left(\sum_{\substack{i,j\in\mathbb{Z}_{10}\\ i+j\equiv 0 \mod 10}} p_ip_j\right)\Pr_{(x_1,x_2,x_3)\sim X_3}\left[\sum_{i,j,k\in\mathbb{Z}_{10}:i+j+k\equiv 0 \mod 10} p_ip_jp_k\right]\\
&+\left(1-\left(\sum_{\substack{i,j\in\mathbb{Z}_{10}\\ i+j\equiv 0 \mod 10}} p_ip_j\right)\right)\Pr_{\left(x,y,x_{13},x_{14}\right)\sim X_{14}}\left[\sigma\left(x\right)+\sigma\left(x_{13}\right)+\sigma\left(x_{14}\right)\equiv\right.\\
&\qquad\left.\sigma'\left(y\right)+\sigma'\left(x_{13}\right)+\sigma'\left(x_{14}\right)\mod 10\right]\\
\leq&\left(\sum_{\substack{i,j\in\mathbb{Z}_{10}\\ i+j\equiv 0 \mod 10}} p_ip_j\right)\left(\sum_{\substack{i,j,k\in\mathbb{Z}_{10}\\ i+j+k\equiv 0 \mod 10}} p_ip_jp_k\right)\\
&+\left(1-\left(\sum_{\substack{i,j\in\mathbb{Z}_{10}\\ i+j\equiv 0 \mod 10}} p_ip_j\right)\right)\max_{j\in\mathbb{Z}_{10}}\Pr_{x\sim[n]}\left[\sigma\left(x\right)\equiv j \mod 10\right]\\
\leq&\left(\sum_{\substack{i,j\in\mathbb{Z}_{10}\\ i+j\equiv 0 \mod 10}} p_ip_j\right)\left(\sum_{\substack{i,j,k\in\mathbb{Z}_{10}\\ i+j+k\equiv 0 \mod 10}} p_ip_jp_k\right)+\left(1-\left(\sum_{\substack{i,j\in\mathbb{Z}_{10}\\ i+j\equiv 0 \mod 10}} p_ip_j\right)\right)\left(\frac{11}{100}\right)
\end{aligned}
$$

Maximizing

$$\left(\sum_{\substack{i,j\in\mathbb{Z}_{10}\\ i+j\equiv 0 \mod 10}} p_i p_j\right)\left(\sum_{\substack{i,j,k\in\mathbb{Z}_{10}\\ i+j+k\equiv 0 \mod 10}} p_i p_j p_k\right) + \left(1 - \left(\sum_{\substack{i,j\in\mathbb{Z}_{10}\\ i+j\equiv 0 \mod 10}} p_i p_j\right)\right)\left(\frac{11}{100}\right),$$

subject to the constraint that $p_0 \leq \frac{1}{10} + \frac{1}{100}$, we obtain the desired upper bound

$$\left(\sum_{\substack{i,j\in\mathbb{Z}_{10}\\ i+j\equiv 0 \mod 10}} p_i p_j\right)\left(\sum_{\substack{i,j,k\in\mathbb{Z}_{10}\\ i+j+k\equiv 0 \mod 10}} p_i p_j p_k\right) + \left(1 - \left(\sum_{\substack{i,j\in\mathbb{Z}_{10}\\ i+j\equiv 0 \mod 10}} p_i p_j\right)\right)\left(\frac{11}{100}\right) \leq \frac{145}{1000}.$$

□

Reminder of Fact 3. *f_2 is* $(0.01, 0.01)$*—hard to predict.*

Proof of Fact 3. Let $\sigma, \sigma' \in \mathbb{Z}_{10}^n$ be given. We assume that σ is $\left(\frac{1}{100}\right)$-balanced and that σ and σ' are not $\left(\frac{1}{100}\right)$-correlated. This means that $\frac{9}{100} \leq \max_{i\in\mathbb{Z}_{10}}\left(\frac{H\left(\sigma,\vec{i}\right)}{n}\right) \leq \frac{11}{100}$ and $\left(1 - \frac{H(\sigma,\sigma')}{n}\right) < \frac{11}{100}$. It suffices to show that

$$\Pr_{C=(x_1,\ldots,x_{14})\sim X_{14}}\left[f_2\left(\sigma\left(C\right)\right) = f_2\left(\sigma'\left(C\right)\right)\right] \leq \frac{11}{100}.$$

For $j \in \mathbb{Z}_{10}$ we let $p_j = \frac{n - H\left(\sigma_j,\sigma'\right)}{n}$ where $\sigma_j\left(i\right) = \left(\sigma\left(i\right) + j \mod 10\right)$. In particular, $p_0 = \frac{n-H(\sigma,\sigma')}{n}$ denotes the probability that $\sigma(i) = \sigma'(i)$ for a random index $i \sim [n]$. By assumption $p_0 \leq \frac{1}{10} + \frac{1}{100}$. We have

$$
\begin{aligned}
& \Pr_{C=(x_1,\ldots,x_{14})\sim X_{14}}\left[f_2\left(\sigma\left(C\right)\right)=f_2\left(\sigma'\left(C\right)\right)\right] \\
= & \Pr_{C=(x_1,\ldots,x_{14})\sim X_{14}}\left[\sigma\left(x_{11}\right)=\sigma'\left(x_{11}\right)\right]\Pr_{C=(x_1,\ldots,x_{14})\sim X_{14}}\left[f_2\left(\sigma\left(C\right)\right)=f_2\left(\sigma'\left(C\right)\right)\,\middle|\,\sigma\left(x_{11}\right)=\sigma'\left(x_{11}\right)\right] \\
& +\Pr_{C=(x_1,\ldots,x_{14})\sim X_{14}}\left[\sigma\left(x_{11}\right)\neq\sigma'\left(x_{11}\right)\right]\Pr_{C=(x_1,\ldots,x_{14})\sim X_{14}}\left[f_2\left(\sigma\left(C\right)\right)=f_2\left(\sigma'\left(C\right)\right)\,\middle|\,\sigma\left(x_{11}\right)\neq\sigma'\left(x_{11}\right)\right] \\
= & p_0\Pr_{C=(x_1,\ldots,x_{14})\sim X_{14}}\left[f_2\left(\sigma\left(C\right)\right)=f_2\left(\sigma'\left(C\right)\right)\,\middle|\,\sigma\left(x_{11}\right)=\sigma\left(x_{11}\right)\right] \\
& +\left(1-p_0\right)\Pr_{C=(x_1,\ldots,x_{14})\sim X_{14}}\left[f_2\left(\sigma\left(C\right)\right)=f_2\left(\sigma'\left(C\right)\right)\,\middle|\,\sigma\left(x_{11}\right)\neq\sigma\left(x_{11}\right)\right] \\
= & p_0\Pr_{C=(x_{12},x_{13},x_{14})\sim X_3}\left[\sigma\left(x_{12}\right)+\ldots+\sigma\left(x_{14}\right)\equiv\sigma'\left(x_{12}\right)+\ldots+\sigma'\left(x_{14}\right)\mod 10\right] \\
& +\left(1-p_0\right)\Pr_{C=(x_1,\ldots,x_{14})\sim X_{14}}\left[f_2\left(\sigma\left(C\right)\right)=f_2\left(\sigma'\left(C\right)\right)\,\middle|\,\sigma\left(x_{11}\right)\neq\sigma\left(x_{11}\right)\right] \\
= & p_0\sum_{\substack{i,j,k\in\mathbb{Z}_{10}\\ i+j+k\equiv 0 \mod 10}} p_ip_jp_k \\
& +\left(1-p_0\right)\Pr_{C=(x_1,\ldots,x_{14})\sim X_{14}}\left[f_2\left(\sigma\left(C\right)\right)=f_2\left(\sigma'\left(C\right)\right)\,\middle|\,\sigma\left(x_{11}\right)\neq\sigma\left(x_{11}\right)\right] \\
= & p_0\sum_{\substack{i,j,k\in\mathbb{Z}_{10}\\ i+j+k\equiv 0 \mod 10}} p_ip_jp_k \\
& +\left(1-p_0\right)\Pr_{\left(x,y,x_{12},x_{13},x_{14}\right)\sim X_5}\left[\sigma(x)+\sigma(x_{12})+\sigma\left(x_{13}\right)+\sigma\left(x_{14}\right)\equiv\right. \\
& \qquad \left.\sigma'(y)+\sigma'(x_{12})+\sigma'\left(x_{13}\right)+\sigma'\left(x_{14}\right)\mod 10\right] \\
\leq & p_0\sum_{\substack{i,j,k\in\mathbb{Z}_{10}\\ i+j+k\equiv 0 \mod 10}} p_ip_jp_k+\left(1-p_{10}\right)\max_{j\in\mathbb{Z}_{10}}\Pr_{x\sim[n]}\left[\sigma(x)\equiv j\mod 10\right] \\
\leq & p_0\sum_{\substack{i,j,k\in\mathbb{Z}_{10}\\ i+j+k\equiv 0 \mod 10}} p_ip_jp_k+\left(1-p_{10}\right)\left(\frac{11}{10}\right).
\end{aligned}
$$

Maximizing

$$
p_0\sum_{\substack{i,j,k\in\mathbb{Z}_{10}\\ i+j+k\equiv 0 \mod 10}} p_ip_jp_k+\left(1-p_0\right)\left(\frac{11}{10}\right),
$$

subject to the constraint that $p_0\leq\frac{1}{10}+\frac{1}{100}$, we obtain the desired upper bound

$$p_0 \sum_{\substack{i,j,k \in \mathbb{Z}_{10} \\ i+j+k \equiv 0 \mod 10}} p_i p_j p_k + (1-p_0)\left(\frac{11}{10}\right) \leq \frac{11}{100} .$$

□

8.4.1 Security Upper Bounds

Background The proof of Theorem 10 relies on the discrete spectral iteration algorithm of [73]. We begin by providing a brief overview of their algorithm. In their setting the secret mapping σ is defined over the binary alphabet $\mathbb{Z}_2^n$. Let $k_1 = \lceil \frac{r(f)}{2} \rceil, k_2 = \lfloor \frac{r(f)}{2} \rfloor$ and let $\delta \in [0,2] \backslash \{1\}$. They use σ to define a distribution over $\left|X_{k_1}\right| \times \left|X_{k_2}\right|$ matrices $M_{\sigma,\delta,p} = \hat{M}\left(Q_{\sigma,\delta,p}\right) - Jp$, where J denotes the all ones matrix. For $(C_1) \in X_{k_1}, (C_2) \in X_{k_2}$ such that $C_1 \bigcap C_2 = \emptyset$ we have

$$\hat{M}\left(Q_{\sigma,\delta,p}\right)[(C_1),(C_2)] = \begin{cases} 1, & \text{with probability } (p(2-\delta)) \text{ if } \sum_{j \in C_1 \cup C_2} \sigma(j) \equiv 0 \mod 2 \\ 1, & \text{with probability } (p\delta) \text{ if } \sum_{j \in C_1 \cup C_2} \sigma(j) \not\equiv 0 \mod 2 \\ 0, & \text{otherwise} \end{cases} .$$

Given a vector $x \in\in \{\pm 1\}^{|X_{k_2}|}$ (resp. $y \in\in \{\pm 1\}^{|X_{k_1}|}$) $M_{\sigma,p}x$ defines a distribution over vectors in $\mathbb{R}^{|X_{k_1}|}$ (resp. $M_{\sigma,p}^T y$ defines a distribution over vectors in $\mathbb{R}^{|X_{k_1}|}$).

If $r(f)$ is even then the the largest eigenvalue of $\mathbb{E}\left[M_{\sigma,\delta,p}\right]$ has a corresponding eigenvector $x^* \in \{\pm 1\}^{X_{r(f)/2}}$, where for $C_i \in X_{r(f)/2}$ we have $x^*[C_i] = 1$ if $\sum_{j \in C_i} \sigma(j) \equiv 1 \mod 2$; otherwise $x^*[C_i] = -1$ (if $r(f)$ is odd then we consider the top singular value instead). Feldman et al. [73] use discrete spectral iteration to find x^*. Given x^* it is easy to find σ using Gaussian Elimination.
The discrete spectral iteration algorithm of Feldman et al. [73] starts with a random vector $x^0 \in \{0,1\}^{|X_{k_2}|}$. They then sample $x^{i+1} \sim M_{\sigma,p}x^i$ followed by a normalization step to ensure that $x^{i+1} \in \{0,1\}^{|X_{k_2}|}$. When $r(f)$ is odd, power iteration has two steps: draw a sample $y^i \sim M_{\sigma,\delta,p}x^i$ and sample from the distribution $x^{i+1} = M_{\sigma,\delta,p}^T y^i$. They showed that $O\left(\log\left|X_{r(f)}\right|\right)$ iterations suffice to recover σ whenever $p = \frac{K \log\left|X_{r(f)}\right|}{(\delta-1)^2\sqrt{\left|X_{r(f)}\right|}}$, and that for a vector $x \in \{0,1\}^{|X_{k_2}|}$ (resp. $y \in \{\pm 1\}^{|X_{k_1}|}$) it is possible to sample from $M_{\sigma,\delta,p}x$ (resp. $M_{\sigma,\delta,p}^T y$) using $O(1/p)$ queries to 1-MSTAT$\left(\left|X_{k_1}\right|\right)$.

Our Reduction The proof of Theorem 10 uses a reduction to the algorithm of Feldman et al. [73].

Reminder of Theorem 10. *For $f_i \in \{f_1, f_2\}$ there is a randomized algorithm that makes $O\left(n^{\max\{1, r(f_i)/2\}} \log^2 n\right)$ calls to the 1-MSTAT$\left(n^{\lceil r(f_i)/2 \rceil}\right)$ oracle and returns σ with probability $1 - o(1)$.*

Proof of Theorem 10. (sketch) Given a mapping $\sigma \in \mathbb{Z}_d^n$ and a number $i \in \mathbb{Z}_d$ we define a mapping $\sigma_i \in \mathbb{Z}_2^n$ where

$$\sigma_i(j) = \begin{cases} 1, & \textbf{if } \sigma(j) = i \\ 0, & \text{otherwise} \end{cases}.$$

Clearly, to recover σ it is sufficient to recover σ_i for each $i \in \mathbb{Z}_d$. Therefore, to prove Theorem 10 it suffices to show that given $x \in \{\pm 1\}^{|X_{k_2}|}$ (resp. $y \in \{\pm 1\}^{|X_{k_1}|}$) we can sample from the distribution $M_{\sigma_i,\delta,p}x$ (resp. $M^T_{\sigma_i,\delta,p}y$) using $O(1/p)$ queries to 1-MSTAT$\left(\left|X_{\lceil r(f)/2 \rceil}\right|\right)$ for each $i \in \{0, \ldots, d-1\}$, where 1-MSTAT uses the distribution Q^f_σ. In general, this will not possible for arbitrary functions f. However, Lemma 9 shows that for our candidate human computable functions f_1, f_2 we can sample from the distributions $M_{\sigma_i,\delta,p}x$ (resp. $M^T_{\sigma_i,\delta,p}y$). The proof of lemma 9 is similar to the proof of [73, Lemma 10]. □

Lemma 9. *Given vectors $\vec{x} \in \{\pm 1\}^{|X_{k_1}|}, \vec{y} \in \{\pm 1\}^{|X_{k_2}|}$ we can sample from $M_{\sigma,\delta,p}x$ and $M^T_{\sigma,\delta,p}y$ using $O\left(n^{r(f_i)/2} \log^2 n\right)$ calls to the 1-MSTAT$\left(n^{\lceil r(f_i)/2 \rceil}\right)$ oracle for $f_i \in \{f_1, f_2\}$.*

The proof of Lemma 9 relies on Facts 7 and 8.

Fact 7. *For each $j, t \in \mathbb{Z}_{10}$ we have*

$$\Pr_{(x_0,\ldots,x_{13}) \sim \mathbb{Z}_{10}^{14}} [x_{12} + x_{13} + x_t \equiv j \mid f_1(\sigma(x_0, \ldots, x_{13})) \equiv j \mod 10] = \frac{\left(\frac{9}{10}\left(\frac{1}{10}\right) + \frac{1}{10}\right)\left(\frac{1}{10}\right)}{\left(\frac{1}{10}\right)} = \frac{19}{100},$$

and

$$\Pr_{(x_0,\ldots,x_{13}) \sim \mathbb{Z}_{10}^{14}} [x_{12} + x_{13} + x_t \equiv j \mid f_1(\sigma(x_0, \ldots, x_{13})) \not\equiv j \mod 10] = \frac{\left(\frac{9}{10}\left(\frac{1}{10}\right) + \frac{1}{10}(0)\right)\left(\frac{1}{10}\right)}{\left(\frac{1}{10}\right)} = \frac{9}{100}.$$

Fact 8. *For each $j, t \in \mathbb{Z}_{10}$ we have*

$$\Pr_{(x_0,\ldots,x_{13})\sim\mathbb{Z}_{10}^{14}}\left[x_{11}+x_{12}+x_{13}+x_t \equiv j \,\middle|\, f_2\left(\sigma\left(x_0,\ldots,x_{13}\right)\right) \equiv j \mod 10\right] = \frac{\left(\frac{9}{10}\left(\frac{1}{10}\right)+\frac{1}{10}\right)\left(\frac{1}{10}\right)}{\left(\frac{1}{10}\right)} = \frac{19}{100},$$

and

$$\Pr_{(x_0,\ldots,x_{13})\sim\mathbb{Z}_{10}^{14}}\left[x_{11}+x_{12}+x_{13}+x_t \equiv j \,\middle|\, f_2\left(\sigma\left(x_0,\ldots,x_{13}\right)\right) \not\equiv j \mod 10\right] = \frac{\left(\frac{9}{10}\left(\frac{1}{10}\right)+\frac{1}{10}\left(0\right)\right)\left(\frac{1}{10}\right)}{\left(\frac{1}{10}\right)} = \frac{9}{100}.$$

Proof of Lemma 9. Let $x_j^i \in \{0,1\}$ denote a random variable that is 1 if and only if $x_j = i$. For f_1 we define the function $h^{i,+} : X_{14} \times \mathbb{Z}_{10} \to X_{k_1} \cup \{\perp\}$ as follows

$$h^{i,+}\left(x_0,\ldots,x_{13},f_1\left(\sigma\left(x_0,\ldots,x_{13}\right)\right)\right) = \begin{cases}\left(x_0,x_{12},x_{13}\right) & \text{if } f_1\left(\sigma\left(x_0,\ldots,x_{13}\right)\right) \equiv 3i \mod 10\\ \perp & \text{otherwise.}\end{cases}.$$

For f_2 we simply change the condition to $f_2\left(\sigma\left(x_0,\ldots,x_{13}\right)\right) \equiv 4i \mod 10$ for $h^{i,+}$.

Given a vector $x \in \{\pm 1\}^{\left|X_{k_1}\right|}$ we query our 1-MSTAT$\left(\left|X_{k_1}\right|+1\right)$ oracle $\lceil 10/p \rceil$ times with the function $h^{i,+}$ to sample from $M_{\sigma,\delta,p}x$. Let $q_1,\ldots,q_{\lceil 10/p\rceil}$ denote the responses. We observe that for $C \in X_{k_2}$ we have

$$M_{\sigma,\delta,p}x[C] \sim \sum_{\substack{i\in\lceil 10/p\rceil\\ q_i=C}} x\left[q_i\right] - p\sum_{C'\in X_{k_1}} x[C'],$$

for some $\delta \neq 1$ because by fact 7 it follows that

$$\begin{aligned}&\Pr_{(x_0,\ldots,x_{13})\sim\mathbb{Z}_{10}^{14}}\left[x_{12}^i+x_{13}^i+x_0^i \equiv 1 \mod 2 \,\middle|\, f_1\left(\sigma\left(x_0,\ldots,x_{13}\right)\right) \equiv 3i \mod 10\right]\\ \neq\ &\Pr_{(x_0,\ldots,x_{13})\sim\mathbb{Z}_{10}^{14}}\left[x_{12}^i+x_{13}^i+x_0^i \equiv 1 \mod 2 \,\middle|\, f_1\left(\sigma\left(x_0,\ldots,x_{13}\right)\right) \equiv 3i \mod 10\right].\end{aligned}$$

Similarly, by fact 8 it follows that

$$\begin{aligned}&\Pr_{(x_0,\ldots,x_{13})\sim\mathbb{Z}_{10}^{14}}\left[x_{11}^i+x_{12}^i+x_{13}^i+x_0^i \equiv 1 \mod 2 \,\middle|\, f_1\left(\sigma\left(x_0,\ldots,x_{13}\right)\right) \equiv 4i \mod 10\right]\\ \neq\ &\Pr_{(x_0,\ldots,x_{13})\sim\mathbb{Z}_{10}^{14}}\left[x_{11}^i+x_{12}^i+x_{13}^i+x_0^i \equiv 1 \mod 2 \,\middle|\, f_1\left(\sigma\left(x_0,\ldots,x_{13}\right)\right) \equiv 4i \mod 10\right].\end{aligned}$$

□

Chapter 9

Appendix: Password Composition Policies

9.1 Optimizing Password Composition Policies: Missing Proofs

Reminder of Theorem 16. *For every k, Algorithm 5.4 computes* $\arg\min_{\mathcal{A}} p(k, \mathcal{A})$ *in the singleton rules setting of the normalized probabilities model, in time* $O(N \log(N))$.

Proof of Theorem 16. Let $\mathcal{A}^*$ denote the optimal solution, denote its most k popular passwords as $w_{i_1}, \ldots, w_{i_k}$, and denote also P^* as the total probability mass of the words in $\mathcal{A}^*$ according to the initial distribution: $P^* = \sum_{w \in \mathcal{A}^*} \Pr[w]$. Therefore, $p(k, \mathcal{A}^*) = \sum_{j=1}^{k} \Pr[w_{i_j}]/P^*$.

Clearly, all words w_j s.t. $j > i_k$ belong to $\mathcal{A}^*$ – otherwise, we could add such a word and decrease the probability of the top k words. Similarly, all words w_j s.t $j < i_1$ must not belong to $\mathcal{A}^*$, otherwise they would belong to the set of most popular k words. We now claim that $w_{i_1}, \ldots, w_{i_k}$ are k consecutive words.

Suppose that there was some word w' between some w_{i_j} and $w_{i_{j+1}}$. Then $\mathcal{A}^*$ clearly banned it, otherwise it would be one of the most popular k words. We claim that the policy $\mathcal{A}'$ where we ban w_{i_1} and allow w' instead satisfies $p(k, \mathcal{A}') \leq p(k, \mathcal{A}^*)$.

We denote $p_1 = \Pr[w_{i_1}]$, $q = \sum_{j=2}^{k} \Pr[w_{i_j}]$ and $p' = \Pr[w']$, and we know $p_1 \geq p'$. Then $p(k, \mathcal{A}^*) = (p_1 + q)/P^*$, whereas

$$p(k, \mathcal{A}') = \frac{p' + q}{P^* - p_1 + p'} .$$

Our goal is to show $p(k, \mathcal{A}') \leq p(k, \mathcal{A}^*)$, which holds iff

$$(p' + q)P^* \leq (p_1 + q)(P^* - (p_1 - p'))$$

By some algebraic manipulations, this holds iff

$$(p_1 - p')P^* \geq (p_1 - p')(p_1 + q)$$

which clearly holds because $p_1 - p'$ is a non-negative quantity, and $p_1 + q = \sum_{j=1}^{k} \Pr[w_{i_j}] \leq \sum_{w \in \mathcal{A}^*} \Pr[w]$.

As for the running time of the algorithm, it is obvious that sorting requires $O(N \log N)$ time. Finding the minimum requires only $O(N)$ time: if we denote $a_i = \sum_{i \leq j \leq i+k} \Pr[w_j]$ and $b_i = \sum_{i \leq j} \Pr[w_j]$, then based on a_i and b_i it is easy to compute a_{i+1} and b_{i+1} in $O(1)$ time. □

Reminder of Lemma 4. *Fix m and s such that $m \geq s$. There exists a domain D of size $\Theta(s^2 \log(m))$ and a family of m sets, $F_1, F_2, \ldots, F_m \subseteq D$, such that each set in the family contains $\frac{|D|}{2s}$ elements, and for every $C \subseteq [m]$ of size $|C| \leq s$, we have that the size of the union $\left|\bigcup_{i \in C} F_i\right| \geq \frac{|D|}{2s}\frac{|C|}{4}$. This domain can be constructed in randomized $\mathrm{poly}(s, m)$ time.*

Proof of Lemma 4. Given m and s, we first pick a random function $\phi : [m] \to [2s]$. Fixing a subset $C \subseteq [m]$ of size $|C| \leq s$, we claim that $|\phi(C)| > |C|/2$ w.p. at least $1 - (0.825)^{|C|}$. Indeed,

$$\begin{aligned}
\Pr\left[|\phi(C)| \leq |C|/2\right] &\leq \Pr\left[\exists T \subseteq [2s] \text{ s.t. } |T| = |C|/2 \text{ and } \forall i \in C, \phi(i) \in T\right] \\
&\leq \binom{2s}{|C|/2} \Pr\left[\forall i \in C, \phi(i) \in T\right] \leq \left(\frac{4se}{|C|}\right)^{|C|/2} \left(\frac{|C|/2}{2s}\right)^{|C|} \\
&= e^{|C|/2} \left(\frac{|C|}{4s}\right)^{|C|/2} = \left(\sqrt{e/4}\right)^{|C|} < (0.825)^{|C|} .
\end{aligned}$$

So assuming $|C| \geq 8$ we have that C is mapped to at least $|C|/2$ distinct images by ϕ w.p.$> 3/4$. Also, if $|C| \leq 7$ then probability of even two elements getting mapped to the same image is at most $\binom{7}{2}\frac{1}{2s} < 0.25$ for $s > 42$.

We now construct D by taking d independently chosen such ϕ-mappings, which we denote as $\phi_1, \phi_2, \ldots, \phi_d$, and so $D = [2s] \times [d]$. We construct the family $F_i = \{(\phi_1(i), 1), (\phi_2(i), 2), \ldots, (\phi_d(i), d)\}$ for every $i \in [m]$. Clearly, for every i it holds that $|F_i| = d = |D|/2s$. Supposed for the sake of contradiction that there exists some $C \subseteq [m]$ of size $\leq s$ such that $\left|\bigcup_{i \in C} F_i\right| \leq \frac{|C|}{4}|F_i|$. By construction, we have that

$$\left|\bigcup_{i \in C} F_i\right| = \sum_{j=1}^{d} \left|\{(\phi_j(C), j)\}\right| = \sum_{j=1}^{d} |\phi_j(C)|$$

so by the Markov inequality we have that at least $d/2$ functions where the cardinality of the image of C is less than $|C|/2$. Let $X_{C,j}$ be the indicator random variable of ϕ_j mapping the set C to no more than $|C|/2$ distinct elements, the Hoeffding bound gives that

$$\Pr\left[\exists C \text{ of size} \leq s \text{ s.t. } \textstyle\sum_j X_{C,j} > d/2\right] \leq \sum_{s' < s} \binom{m}{s'} \Pr[\tfrac{1}{d} \sum_j X_{C,j} > 0.5] \leq m^{O(s)} e^{-d/10}$$

Setting $d = \Theta(s \log m)$ gives that w.p. $\geq 1/2$ no such C exists. □

Reminder of Theorem 18. *Unless $P = NP$ there is no polynomial time algorithm (in N, m, n) which outputs $\arg\min_{S \subseteq [m]} p(k, \mathcal{A}_S)$ in the positive rules setting and the normalization model.*

Proof of Theorem 18. Our reduction is from set cover.

Set Cover Instance: Sets $S_1, \ldots, S_m$, Universe $U = \{1, \ldots, n\}$ and integer k.
Question: Is there a set cover of size $k - 1$?

Now we define $W_1, \ldots, W_n$ to be n disjoint sets of passwords

$$W_i = \left\{ w_{i,\ell} \,\middle|\, 1 \leq \ell \leq n^5 m^5 \right\} .$$

We also define special passwords t_j ($j \leq m$) and τ_j ($j \leq k$) which are not contained in any W_i.

We define the following positive password rules:

$$R_i = \{t_i\} \cup \left\{ \tau_j \,\middle|\, 1 \leq j \leq k \right\} + \bigcup_{j : j \in S_i} W_j .$$

We assign probabilities as follows:

$$\Pr\left[w_{i,\ell}\right] = \left(1 - \frac{1}{n^3}\right) \frac{1}{m^5 n^6} ,$$

for each $i \leq n$ and $\ell \leq m^5 n^5$. Observe that

$$\Pr\left[\bigcup_{i \leq m} W_i\right] = \left(1 - \frac{1}{n^3}\right) ,$$

so that almost all of the probability mass is concentrated inside the sets W_i and the probability mass is uniformly distributed. We also set

$$\Pr\left[\tau_j\right] = \frac{1 - x}{n^3 k} ,$$

and

$$\Pr\left[t_j\right] = \frac{x}{n^3 m} ,$$

where $0 \leq x \leq 1$ will be defined later. First notice that

$$\sum_{j \leq k} \tau_j + \sum_{j \leq m} t_j = k\left(\frac{1 - x}{n^3 k}\right) + m\left(\frac{x}{n^3 m}\right) = \frac{1}{n^3} ,$$

so our probability distribution is well defined. Suppose that there is a set cover $C \subseteq [m]$ s.t. $|C| \leq k-1 \wedge \bigcup_{i \in C} S_i = U$, and consider the solution $\mathcal{A}_C$. We cover all W_i's and use at most $k-1$ t's. Hence,

$$p(k, \mathcal{A}_C) \leq ((k-1)\Pr[t] + \Pr[\tau]) \left(\frac{n^3}{n^3 - 1} \right) .$$

Suppose that there is no set cover of size k. For every set of k or more rules S we have at least k t's in our solution so

$$p(k, \mathcal{A}_S) \geq k \Pr[t] .$$

For every set of rules S that does not cover all the W_i's we have at most $\left(1 - \frac{1}{n}\right)\left(1 - \frac{1}{n^3}\right)$-fraction of the total probability mass so

$$p(k, \mathcal{A}_S) \geq \frac{((k-1)\Pr[\tau] + \Pr[t])}{\left(1 - \frac{1}{n}\right)\left(1 - \frac{1}{n^3}\right)} .$$

It suffices to select x s.t.

$$((k-1)\Pr[t] + \Pr[\tau]) \left(\frac{n^3}{n^3 - 1} \right) < \min \left\{ \frac{((k-1)\Pr[\tau] + \Pr[t])}{\left(1 - \frac{1}{n}\right)\left(1 - \frac{1}{n^3}\right)}, k \Pr[t] \right\} ,$$

or —after some algebraic manipulation — equivalently,

$$a = \frac{\left(\frac{n^3}{n^3-1}\right)}{\left(1 - \frac{1}{n^3-1}\right)} \Pr[\tau] < \Pr[t] < b = \Pr[\tau] \frac{(k-2) + \frac{1}{n-1}}{(k-2) - \frac{1}{n-1}} .$$

Observe that $a \leq \Pr[t] \leq b$ so it suffices to set x s.t. $\Pr[t] = \frac{a+b}{2}$. We can solve for x to get

$$x = \frac{m\left(-3 + 2n + 2n^3 - 2n^4 + k(n-1)^2(1 + n + n^2)\right)}{m(-3 + 2n + 2n^3 - 2n^4) + k^2(2 - 2n - n^3 + n^4) + k\left(-2 + 4n + n^3 - 2n^4 + m(n-1)^2(1 + n + n^2)\right)} .$$

□

Reminder of Claim 5. $\Pr[\exists i, BAD_i] \leq \delta$.

Proof of Claim 5. By the union bound it suffices to show that

$$\Pr[BAD_i] \leq \frac{\delta}{m} .$$

Our first step is to divide the passwords $w \in \mathcal{P}$ into buckets B_j based on their probability. For $j > 0$ we define

$$B_j = \left\{ w \,\middle|\, \frac{\epsilon}{2^j} \leq \Pr[w \,|\, \mathcal{A}_{S_i}] \leq \frac{\epsilon}{2^{j-1}} \right\} ,$$

and for $j = 0$ we set

$$B_0 = \{ w \,|\, \epsilon \leq \Pr[w \,|\, \mathcal{A}_{S_i}] \} .$$

Observe that

$$\mathcal{P} = \bigcup_{j=0}^{\infty} B_j .$$

Let $w \in B_j$ be given ($j > 0$) then by the Chernoff Bounds:

$$\Pr[s_w > s \Pr[w \,|\, \mathcal{A}_{S_i}] + s\epsilon/2] \leq \exp\left(-2^{j-1} \log\left(\frac{4m}{\delta\epsilon}\right)\right) \leq \frac{4^{-2^{j-1}}\delta\epsilon}{m} .$$

Notice that the bucket B_j contains at most $\left|B_j\right| = 2^j/\epsilon$ passwords.

$$\Pr\left[\exists w \in B_j, s_w > s \Pr[w \,|\, \mathcal{A}_{S_i}] + s\epsilon/2\right] \leq \frac{4^{-2^{j-1}}\delta\epsilon \left|B_j\right|}{m} \leq \frac{\delta}{2^{j+1}m} .$$

Now if we union bound across all $j > 0$ we get

$$\Pr\left[\exists w \in \bigcup_{j=1}^{\infty} B_j, s_w > s \Pr[w \,|\, \mathcal{A}_{S_i}] + s\epsilon/2\right] \leq \sum_{j=1}^{\infty} \frac{\delta}{2^{j+1}m} = \frac{\delta}{2m} .$$

Finally, we consider the passwords in B_0. By Chernoff Bounds for each $w \in B_0$ we have

$$\Pr\left[\left|s_w - s \Pr[w \,|\, \mathcal{A}_{S_i}]\right| > s\epsilon/2\right] \leq \frac{\delta\epsilon}{2m} ,$$

by applying the union bound $|B_0| \leq 1/\epsilon$ we get

$$\Pr\left[\exists w \in B_0 \left|s_w - s \Pr[w \,|\, \mathcal{A}_{S_i}]\right| > s\epsilon/2\right] \leq \frac{\delta}{2m} .$$

Combining our inequalities we obtain the desired result:

$$\Pr[BAD_i] \leq \Pr\left[\exists w \in \bigcup_{j=0}^{\infty} B_j, s_w > s \Pr[w \,|\, \mathcal{A}_{S_i}] + s\epsilon/2\right] \leq \frac{\delta}{m} .$$

□

9.2 Impossibility of constant-factor universal approximation

In this section we consider the following goal: given a constant c find a password composition policy $\mathcal{A}$ such that

$$p(k, \mathcal{A}) \leq c \cdot p(k, \mathcal{A}') \ ,$$

for any other policy $\mathcal{A}'$ and *every* value of $k \leq N$. Such a policy — if it exists — would provide a nearly optimal defense against both online attacks and dictionary attacks simultaneously [136]. Unfortunately, Theorem 32 rules out the possibility of a constant universal approximation in the rankings model. Our impossibility result holds even in the singleton rules setting. We show that it is possible to construct a distribution $\mathcal{D}$ over rankings for which no universal approximation exists.

We construct our distribution $\mathcal{D}$ (algorithm 9.1) over rankings by merging two distributions $\mathcal{D}_1$ and $\mathcal{D}_2$ over preference lists.

Intuition: Passwords sampled from D_2 are highly secure, but passwords sampled from D_1 are highly insecure. To make improve the security of D_1 it is necessary to ban all passwords in W, but this reduces the security of D_2 significantly.

We make two claims (1) We must ban *all* but a small subset of passwords if we want to even *approximately* optimize $p(1, \mathcal{A})$. (2) We must keep a larger subset of passwords to even *approximately* optimize $p(k, \mathcal{A})$ for large values of k.

Theorem 32. *For all constants $c > 0$ there exists distribution $\mathcal{D}$ over rankings such that $\forall \mathcal{A} \subseteq \mathcal{P}, \exists \mathcal{A}', k \in \mathbb{N}$, such that*

$$p(k, \mathcal{A}) > c \cdot p(k, \mathcal{A}') \ .$$

Proof. (sketch) Let $\mathcal{P} = W \cup X$ where $W = \bigcup_{i=1}^{r} W_i$ — $W_i = \{w_{i,1}, \ldots, w_{i,t}\}$ — and $X = \{x_1, \ldots, x_L\}$ are two disjoint sets of passwords, where the parameters are set as follows $q = \frac{1}{2c}$, $t = L = \log N$ and $r = \frac{N-L}{t}$. Our distribution over preference lists is given by algorithm 9.1.

There are two cases to consider:

Case 1: $\exists x \in W - \mathcal{A}$ then it is easy to see that

$$p(1, \mathcal{A}) \geq \frac{q}{t} = \frac{2c}{t} = \frac{2c}{L} \geq 2c \times p(1, X) \ .$$

Algorithm 9.1 Sample $\mathcal{D}$

Input:
Parameters L, r, q, t
Random Number $u \in [0, 1]$.
Random Permutation π_i of W_i for each $i \in \{1, ..., r\}$
Random Permutation pi_X over X
Random Permutation $\pi_{\mathcal{P}}$ of $\mathcal{P}$
Initialize: $\ell \leftarrow$ empty ranking
if $u \leq q$ **then** ▷ Select from $\mathcal{D}_1$
  for $i = 1 \rightarrow r$ **do**
    $\ell \leftarrow \langle \ell, \pi_{10^r} \rangle$ ▷ Append random permutation of W_i
  $\ell \leftarrow \langle \ell, \pi_X \rangle$ ▷ Append random permutation of X
else ▷ Select from $\mathcal{D}_2$
  $\ell \leftarrow \pi_{\mathcal{P}}$
return ℓ

Case 2: Suppose that $\forall x \in W$ we have $x \notin \mathcal{A}$ and consider $k = L$ with the solution $\mathcal{P}$ — don't ban any passwords. For the solution $\mathcal{P}$ we have

$$p_i = \frac{q}{t} + \frac{1-q}{|X| + |W|} ,$$

for $i \leq t$ (e.g., for the t the passwords in W_1), and

$$p_i = \frac{1-q}{|X| + |W|} ,$$

for $i > t$.

$$\begin{aligned} c \times p(k, \mathcal{P}) &= c \sum_{i=1}^{t} \left(\frac{q}{t} + \frac{1-q}{|X| + |W|} \right) + c \sum_{i=t+1}^{k} \frac{1-q}{|X| + |W|} \\ &= c \left(q + (1-q) \frac{L}{L + 10^r} \right) \\ &= \frac{1}{2} + \left(c - \frac{1}{2} \right) \frac{L}{L + 10^r} \\ &< 1 = p(k, \mathcal{A}) . \end{aligned}$$

□

Chapter 10

Appendix: GOTCHA Password Hackers

10.1 Missing Proofs

Reminder of Claim 6. *If (G_1, G_2) is a $(\alpha, \beta, \epsilon, \delta, \mu)$-GOTCHA then at least β-fraction of humans can successfully authenticate using protocol 6.3.2 after creating an account using protocol 6.3.1.*

Proof of Claim 6. A legitimate user $H \in \mathcal{H}$ will use the same passwords in protocols 6.3.1 and 6.3.2. Hence,

$$r'_1 = \textbf{Extract}\left(pw', r'\right) = \textbf{Extract}\left(pw, r'\right) = r_1 \ ,$$

and the final matching challenge $\hat{c}_\pi$ is the same one that would be generated by $G_2\left(1^k, r_1, H\left(G_1\left(1^k, r_1, r_2\right), \sigma_0\right)\right)$. If $\hat{c}_\pi$ is consistently solvable with accuracy α by H — by definition 15 this is the case for at least β-fraction of users — then it follows that

$$d_k\left(\pi, \pi', \sigma_t\right) \leq \alpha \ ,$$

where $H\left(G_1\left(1^k, r_1, r_2\right)\right)$. For some π_0 (namely $\pi_0 = \pi$) s.t. $d_k\left(\pi_0, \pi'\right) \leq \alpha$ it must be the case that

$$\begin{aligned} h_{pw,0} &= h\left(u, s, pw', \pi_0(1), ..., \pi_0(k)\right) \\ &= h\left(u, s, pw, \pi(1), ..., \pi(k)\right) \\ &= h_p w \ , \end{aligned}$$

and protocol 6.3.2 accepts. □

Reminder of Claim 7. *For all permutations $\pi : [k] \rightarrow [k]$ and $\alpha \geq 0$*

$$\left|\left\{\pi' \mid d_k\left(\pi, \pi'\right) \leq \alpha\right\}\right| \leq 1 + \sum_{i=2}^{\alpha} \binom{k}{i} i! \ .$$

Proof of Claim 7. It suffices to show that $\binom{k}{j} j! \geq \left|\left\{\pi' \mid d_k\left(\pi, \pi'\right) = j\right\}\right|$. We first choose the j unique indices $i_1, \ldots, i_j$ on which π and π' differ — there are $\binom{k}{j}$ ways to do this. Once we have fixed our indices $i_1, \ldots, i_j$ we define $\pi'(k) = \pi(k)$ for each $k \notin \{i_1, \ldots, i_j\}$. Now $j!$ upper bounds the number of ways of selecting the remaining values $\pi'(i_k)$ s.t. $\pi(i_k) \neq \pi'(i_k)$ for all $k \leq j$. □

10.2 HOSP: Pre-Generated CAPTCHAs

The HOSP construction proposed by [51] was to simply fill several high capacity hard drives with randomly generated CAPTCHAs — discarding the solutions. Once we have compiled a database large D of CAPTCHAs we can use algorithm 10.1 as our challenge generator — simply return a random CAPTCHA from D. The advantage of this approach is that we can make use of already tested CAPTCHA solutions so there is no need to make hardness assumptions about new AI problems. The primary disadvantage of this approach is that the size of the database D will be limited by economic considerations — storage isn't free. While $|D|$ the number of CAPTCHAs that could be stored on a hard drive may be large, it is not exponentially large. An adversary could theoretically pay humans to solve every puzzle in D at which point the scheme would be completely broken.

Algorithm 10.1 GenerateChallenge

Input: Random bits $r \in \{0,1\}^n$, Database $D = \{P_1, ..., P_{2^n}\}$ of CAPTCHAs
return P_r

Economic Cost Suppose that two 4 TB hard drives are filled will text CAPTCHAS [1]. Let S be the space required to store one CAPTCHA, and let C_H denote the cost of paying a human to solve a CAPTCHA. We use the values $S = 8$ KB [2] and $C_H = \$0.001$ [3]. In this case $|D| = \frac{4\ TB}{8KB} \approx 10^9$ so we can store a billion unsolved CAPTCHAs on the hard drives. It would cost the adversary $|D|\, C_H = \$1,000,000$ to solve all of the CAPTCHAs — or $\$500,000$ to solve half of them. The up front cost of this attack may be large, but once the adversary has solved the CAPTCHAs he can execute offline dictionary attacks against every user who had an account on the server. Many server breaches have resulted in the release of password records for millions of accounts [5, 9, 11, 13]. If each cracked password is worth between \$4 and \$30 [79] then it may be easily worth the cost to pay humans to solve every CAPTCHA in D.

[1] At the time of submission a 4 TB hard drive can be purchased on Amazon for less than \$162.

[2] The exact value of S may vary slightly depending on the particular method used to generate the CAPTCHA. When we compressed a text CAPTCHA using popular GIF format the resulting files were consistently 8 KB.

[3] Motoyama et al. estimated that spammers paid humans \$1 to solve a thousand CAPTCHAs [110]

Bibliography

[1] Amazon ec2 pricing. http://aws.amazon.com/ec2/pricing/. Retrieved 10/22/2012. 2.5, 7.5.3

[2] Check your password-is it strong? https://www.microsoft.com/security/pc-security/password-checker.aspx. Retrieved 9/8/2011. 7.4.1

[3] Cert incident note in-98.03: Password cracking activity. http://www.cert.org/incident_notes/IN-98.03.html, July 1998. Retrieved 8/16/2011. 1.1, 2.1

[4] Geek to live: Choose (and remember) great passwords. http://lifehacker.com/184773/geek-to-live--choose-and-remember-great-passwords, July 2006. Retrieved 9/27/2012. 1.2, 2.1, 2.3.3, 7.3.1

[5] Rockyou hack: From bad to worse. http://techcrunch.com/2009/12/14/rockyou-hack-security-myspace-facebook-passwords/, December 2009. Retrieved 9/27/2012. 1.1, 1.6, 1.7.1, 5, 2.1, 2, 6.1, 1, 6.1.1, 7.5.1, 10.2

[6] Oh man, what a day! an update on our security breach. http://blogs.atlassian.com/news/2010/04/oh_man_what_a_day_an_update_on_our_security_breach.html, April 2010. Retrieved 8/18/2011. 1.1, 2.1

[7] Sarah palin vs the hacker. http://www.telegraph.co.uk/news/worldnews/sarah-palin/7750050/Sarah-Palin-vs-the-hacker.html, May 2010. Retrieved 9/27/2012. 2.5

[8] Nato site hacked. http://www.theregister.co.uk/2011/06/24/nato_hack_attack/, June 2011. Retrieved 8/16/2011. 1.1, 2.1

[9] Update on playstation network/qriocity services. http://blog.us.playstation.com/2011/04/22/update-on-playstation-network-qriocity-services/, April 2011. Retrieved 5/22/2012. 1.1, 1.7.1, 2.1, 6.1, 6.1.1, 7.5.1, 10.2

[10] Data breach at ieee.org: 100k plaintext passwords. http://ieeelog.com/, September 2012. Retrieved 9/27/2012. 1.1, 1.7.1, 5, 2.1, 6.1, 1, 6.1.1

[11] Zappos customer accounts breached. http://www.usatoday.com/tech/news/story/2012-01-16/mark-smith-zappos-breach-tips/52593484/1, January 2012. Retrieved 5/22/2012. 1.1, 1.7.1, 2.1, 6.1, 6.1.1, 10.2

[12] Apple security blunder exposes lion login passwords in clear text. http://www.zdnet.com/blog/security/apple-security-blunder-exposes-lion-login-passwords-in-clear-text/11963, May 2012. Retrieved 5/22/2012. 1.1, 2.1

[13] An update on linkedin member passwords compromised. http://blog.linkedin.com/2012/06/06/linkedin-member-passwords-compromised/, June 2012. Retrieved 9/27/2012. 1.1, 1.3, 1.7.1, 2.1, 2, 6.1, 6.1.1, 7.5.1, 10.2

[14] Important customer security announcement. http://blogs.adobe.com/conversations/2013/10/important-customer-security-announcement.html, October 2013. Retrieved 2/10/2014. 1.1

[15] Alessandro Acquisti and Ralph Gross. Imagined communities: awareness, information sharing, and privacy on the facebook. In *Proceedings of the 6th international conference on Privacy Enhancing Technologies*, pages 36–58. Springer-Verlag, 2006. 2.4.2

[16] S. Alexander. Password protection for modern operating systems. *;login*, June 2004. 1.7.1, 6.1.1, 7.5.1

[17] N. Alon, D. Moshkovitz, and S. Safra. Algorithmic construction of sets for *k*-restrictions. *ACM Transactions on Algorithms*, 2(2):153–177, 2006. 5.4.2

[18] John R Anderson and Lael J Schooler. Reflections of the environment in memory. *Psychological science*, 2(6):396–408, 1991. 2.4.1, 7.6.1

[19] J.R. Anderson, M. Matessa, and C. Lebiere. Act-r: A theory of higher level cognition and its relation to visual attention. *Human-Computer Interaction*, 12(4):439–462, 1997. 2.1

[20] S. Arora, B. Barak, and D. Steurer. Subexponential algorithms for unique games and related problems. In *Proc. of FOCS*, pages 563–572, 2010. 5.3.4

[21] C. Asmuth and J. Bloom. A modular approach to key safeguarding. *Information Theory, IEEE Transactions on*, 29(2):208–210, 1983. 2.6.1

[22] P. Austrin, S. Khot, and M. Safra. Inapproximability of vertex cover and independent set in bounded degree graphs. *Theory of Computing*, 7(1), 2011. 5.3.4, 5.3.4

[23] A.D. Baddeley. *Human memory: Theory and practice*. Psychology Pr, 1997. ISBN 0863774318. 2.1, 2.2, 2.4.1, 3.2, 4.2, 6.1.1

[24] Boaz Barak, Oded Goldreich, Rusell Impagliazzo, Steven Rudich, Amit Sahai, Salil Vadhan, and Ke Yang. On the (im) possibility of obfuscating programs. In *Advances in Cryptology-CRYPTO 2001*, pages 1–18. Springer, 2001. 11

[25] Calvin Beideman and Jeremiah Blocki. Set families with low pairwise intersection. *CoRR*, abs/1404.4622, 2014. (document), 7.7

[26] M. Bellare and P. Rogaway. The exact security of digital signatures-how to sign with rsa and rabin. In *Advances in CryptologyEurocrypt96*, pages 399–416. Springer, 1996. 1.2.2, 2.1, 2.5

[27] R. Biddle, S. Chiasson, and PC Van Oorschot. Graphical passwords: Learning from the first twelve years. *ACM Computing Surveys (CSUR)*, 44(4):19, 2012. 1.2.1, 2.2, 3.2

[28] Sam. Biddle. Anonymous leaks 90,000 military email accounts in latest antisec attack. `http://gizmodo.com/5820049/anonymous-leaks-90000-military-email-accounts-in-latest-antisec-attack`, July 2011. Retrieved 8/16/2011. 1.1, 1.7.1, 2.1, 6.1, 6.1.1

[29] Jeremiah Blocki, Manuel Blum, and Anupam Datta. Human-computable passwords. ASIACRYPT Rump Session, 2013. URL `http://asiacrypt.2013.rump.cr.yp.to/b0279d7741ad5bab24cf5c55fd292d5c.pdf`. 8.1

[30] Jeremiah Blocki, Manuel Blum, and Anupam Datta. Naturally rehearsing passwords. *CoRR*, abs/1302.5122, 2013. (document), 6.2.1

[31] Jeremiah Blocki, Manuel Blum, and Anupam Datta. Gotcha password hackers! In *Proceedings of the 2013 ACM workshop on Artificial intelligence and security*, pages 25–34. ACM, 2013. (document), 1.1

[32] Jeremiah Blocki, Manuel Blum, and Anupam Datta. Gotcha password hackers! http://www.cs.cmu.edu/ jblocki/papers/aisec2013-fullversion.pdf, 2013. (document)

[33] Jeremiah Blocki, Manuel Blum, and Anupam Datta. Naturally rehearsing passwords. In Kazue Sako and Palash Sarkar, editors, *Advances in Cryptology - ASIACRYPT 2013*, volume 8270 of *Lecture Notes in Computer Science*, pages 361–380. Springer Berlin Heidelberg, 2013. ISBN 978-3-642-42044-3. doi: 10.1007/978-3-642-42045-0_19. URL `http://dx.doi.org/10.1007/978-3-642-42045-0_19`. (document), 1.1, 3.1

[34] Jeremiah Blocki, Saranga Komanduri, Ariel Procaccia, and Or Sheffet. Optimizing password composition policies. In *Proceedings of the fourteenth ACM conference on Electronic commerce*, pages 105–122. ACM, 2013. (document), 1.1, 1.2.1, 2.2, 3.2, 6.1.1, 3

[35] Jeremiah Blocki, Saranga Komanduri, Ariel D. Procaccia, and Or Sheffet. Optimizing password composition policies. *CoRR*, abs/1302.5101, 2013. (document)

[36] Jeremiah Blocki, Manuel Blum, and Anupam Datta. Human computable passwords. *arXiv preprint arXiv:1404.0024*, 2014. (document), 1.1

[37] Avrim Blum, Cynthia Dwork, Frank McSherry, and Kobbi Nissim. Practical privacy: the sulq framework. In *Proceedings of the twenty-fourth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 128–138. ACM, 2005. 3.1, 3.4.1

[38] Hristo Bojinov, Daniel Sanchez, Paul Reber, Dan Boneh, and Patrick Lincoln. Neuroscience meets cryptography: designing crypto primitives secure against rubber hose attacks. In *Proceedings of the 21st USENIX conference on Security symposium*, pages 33–33. USENIX Association, 2012. 1.2.1, 2.2

[39] J. Bonneau. The science of guessing: analyzing an anonymized corpus of 70 million passwords. In *Security and Privacy (SP), 2012 IEEE Symposium on*, pages 538–552. IEEE, 2012. 1.1, 1.2.2, 1.7.1, 2.1, 2.2, 3.2, 5.1.3, 5.6, 5.7, 6.1, 7.4

[40] J. Bonneau and R. Xu. Character encoding issues for web passwords. In *Web 2.0 Security & Privacy*, 2012. 2, 5.1.3

[41] Joseph Bonneau and Sören Preibusch. The password thicket: technical and market failures in human authentication on the web. In *Proc. of WEIS*, volume 2010, 2010. 6.1.1

[42] Joseph Bonneau and Stuart Schechter. "toward reliable storage of 56-bit keys in human memory". In *Proceedings of the 23rd USENIX Security Symposium*, August 2014. 4.2

[43] Joseph Bonneau, Cormac Herley, Paul C van Oorschot, and Frank Stajano. The quest to replace passwords: A framework for comparative evaluation of web authentication schemes. In *IEEE Symposium on Security and Privacy*, pages 553–567. IEEE, 2012. 2.2

[44] George EP Box and Norman R Draper. *Empirical model-building and response surfaces.* John Wiley & Sons, 1987. 1.2.3

[45] S. Boztas. Entropies, guessing, and cryptography. *Department of Mathematics, Royal Melbourne Institute of Technology, Tech. Rep*, 6, 1999. 1.2.2, 2.2, 3.2, 5.1.1, 5.7, 7.4

[46] S. Brand. Department of defense password management guideline. 1985. 1.2, 2.1

[47] Julien Bringer, Hervé Chabanne, and Emmanuelle Dottax. Hbˆ+ˆ+: a lightweight authentication protocol secure against some attacks. In *Security, Privacy and Trust in Pervasive and Ubiquitous Computing, 2006. SecPerU 2006. Second International Workshop on*, pages 28–33. IEEE, 2006. 3.2

[48] S. Brostoff and M.A. Sasse. Are Passfaces more usable than passwords: A field trial investigation. In *People and Computers XIV-Usability or Else: Proceedings of HCI*, pages 405–424, 2000. 1.2.1, 2.2, 3.2

[49] M. Burnett. *Perfect passwords: selection, protection, authentication*. Syngress Publishing, 2005. 1.2, 2.1

[50] W. E. Burr, D. F. Dodson, and W. T. Polk. Electronic authentication guideline. *NIST Special Publication 800-63*, 2006. 5.1.3, 5.6.1

[51] Ran Canetti, Shai Halevi, and Michael Steiner. Mitigating dictionary attacks on password-protected local storage. In *Advances in Cryptology-CRYPTO 2006*, pages 160–179. Springer, 2006. 3.6, 6.1, 6.1.1, 6.2, 6.2, 5, 6.2.1, 6.3.1, 6.5, 10.2

[52] I.A.D. Center. Consumer password worst practices. *Imperva (White Paper)*, 2010. 1.1, 1.2.1, 2.1, 2.1, 2.2, 3.2

[53] Herman Chernoff. A measure of asymptotic efficiency for tests of a hypothesis based on the sum of observations. *The Annals of Mathematical Statistics*, 23(4):493–507, 1952. 7.7.3, 31

[54] Sonia Chiasson, Paul C van Oorschot, and Robert Biddle. A usability study and critique of two password managers. In *Usenix Security*, volume 6, 2006. 2.2

[55] Cheng Chu, Sang Kyun Kim, Yi-An Lin, YuanYuan Yu, Gary Bradski, Andrew Y Ng, and Kunle Olukotun. Map-reduce for machine learning on multicore. *Advances in neural information processing systems*, 19:281, 2007. 3.1, 3.4.1

[56] L. Clair, L. Johansen, W. Enck, M. Pirretti, P. Traynor, P. McDaniel, and T. Jaeger. Password exhaustion: Predicting the end of password usefulness. *Proc. of ICISS*, pages 37–55, 2006. 5.1

[57] G. Cormode and S. Muthukrishnan. An improved data stream summary: The count-min sketch and its applications. *Journal of Algorithms*, 55(1):58–75, 2005. 5.1.3

[58] Mary Cryan and Peter Bro Miltersen. On pseudorandom generators in nc0. In *Mathematical Foundations of Computer Science 2001*, pages 272–284. Springer, 2001. 3.2

[59] Waseem Daher and Ran Canetti. Posh: A generalized captcha with security applications. In *Proceedings of the 1st ACM workshop on Workshop on AISec*, pages 1–10. ACM, 2008. 6.1, 6.1.1, 6.2

[60] Matthew Dailey and Chanathip Namprempre. A text graphics character captcha for password authentication. In *TENCON 2004. 2004 IEEE Region 10 Conference*, pages 45–48. IEEE, 2004. 4, 6.1.1

[61] Cristian Danescu-Niculescu-Mizil, Justin Cheng, Jon Kleinberg, and Lillian Lee. You had me at hello: How phrasing affects memorability. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Long Papers-Volume 1*, pages 892–901. Association for Computational Linguistics, 2012. 2.6

[62] Arthur P Dempster, Nan M Laird, and Donald B Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 1–38, 1977. 3.1, 3.4.1

[63] Solar Designer. John the Ripper. `http://www.openwall.com/john/`, 1996-2010. 1.7.1, 5.6.1, 6.1, 6.2.1, 6.2.1

[64] C. Ding, D. Pei, and A. Salomaa. *Chinese remainder theorem*. World Scientific, 1996. 2.6.1

[65] Y. Dodis, L. Reyzin, and A. Smith. Fuzzy extractors: How to generate strong keys from biometrics and other noisy data. In *Advances in cryptology-Eurocrypt 2004*, pages 523–540. Springer, 2004. 2.2, 7.3.3

[66] K. Doel. Scary logins: Worst passwords of 2012 and how to fix them, 2012. URL `http://www.prweb.com/releases/2012/10/prweb10046001.htm`. Retrieved 1/21/2013. 1.7.1, 5.1, 5.1.3, 6.1

[67] Jeremy Elson, John R Douceur, Jon Howell, and Jared Saul. Asirra: a captcha that exploits interest-aligned manual image categorization. In *Proc. of CCS*. 6.1.1

[68] P Erd6s and H Hanani. On a limit theorem in combinatorical analysis. *Publ. Math. Debrecen*, 10:10–13, 1963. 2.2

[69] P Erdös and A Renyi. On some combinatorial problems. *Publ. Math. Debrecen*, 4:398–405, 1956. 2.2

[70] Paul Erdös, Peter Frankl, and Zoltán Füredi. Families of finite sets in which no set is covered by the union ofr others. *Israel Journal of Mathematics*, 51 (1-2):79–89, 1985. 2.2

[71] Leonhard Euler. *Recherches sur une nouvelle especе de quarres magiques*. Zeeuwsch Genootschao, 1782. 2.2

[72] Vitaly Feldman, Elena Grigorescu, Lev Reyzin, Santosh Vempala, and Ying Xiao. Statistical algorithms and a lower bound for detecting planted cliques. In *Proceedings of the 45th annual ACM symposium on Symposium on theory of computing*, pages 655–664. ACM, 2013. 3.4.1

[73] Vitaly Feldman, Will Perkins, and Santosh Vempala. On the complexity of random satisfiability problems with planted solutions. *arXiv preprint arXiv:1311.4821v2*, 2013. 1.4.1, 3.1, 3.1, 3.2, 2, 9, 3.4.1, 10, 3.4.1, 5, 3.4.1, 1, 3.4.2, 3.6.4, 3.7.3, 8.2, 28, 8.2, 8.2, 8.2, 6, 8.2, 8.4.1, 8.4.1

[74] M. Figurska, M. Stanczyk, and K. Kulesza. Humans cannot consciously generate random numbers sequences: Polemic study. *Medical hypotheses*, 70 (1):182–185, 2008. 3.2, 7.3.3

[75] D. Florencio and C. Herley. A large-scale study of web password habits. In *Proceedings of the 16th international conference on World Wide Web*, pages 657–666. ACM, 2007. 1.1, 1.2.1, 2.1, 2.2, 2.4, 3.2, 6.1.1

[76] D. Florêncio and C. Herley. Where do security policies come from. In *Proc. of SOUPS*, page 10, 2010. 5.1.3, 6.1.1, 6.3.2

[77] Dinei Florencio and Cormac Herley. Is everything we know about password-stealing wrong? *IEEE Security and Privacy*, 2012. 7.5.4

[78] J. Foer. *Moonwalking with Einstein: The Art and Science of Remembering Everything*. Penguin Press, 2011. 2.1, 2.3.1, 2.6

[79] M. Fossi, E. Johnson, D. Turner, T. Mack, J. Blackbird, D. McKinney, M. K. Low, T. Adams, M. P. Laucht, and J. Gough. Symantec report on the undergorund economy, November 2008. URL `http://eval.symantec.com/mktginfo/enterprise/white_papers/b-whitepaper_underground_economy_report_11-2008-14525717.en-us.pdf`. Retrieved 1/8/2013. 1.7.1, 5.1, 6.1, 7.5.4, 10.2

[80] Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. In *Foundations of Computer Science (FOCS), 2013 IEEE 54th Annual Symposium on*, pages 40–49. IEEE, 2013. 6.5, 11

[81] Paolo Gasti and KasperB. Rasmussen. On the security of password manager database formats. In Sara Foresti, Moti Yung, and Fabio Martinelli, editors,

Computer Security ESORICS 2012, volume 7459 of *Lecture Notes in Computer Science*, pages 770–787. Springer Berlin Heidelberg, 2012. ISBN 978-3-642-33166-4. doi: 10.1007/978-3-642-33167-1_44. URL http://dx.doi.org/10.1007/978-3-642-33167-1_44. 1

[82] Shirley Gaw and Edward W. Felten. Password management strategies for online accounts. In *Proceedings of the second symposium on Usable privacy and security*, SOUPS '06, pages 44–55, New York, NY, USA, 2006. ACM. ISBN 1-59593-448-0. doi: http://doi.acm.org/10.1145/1143120.1143127. URL http://doi.acm.org/10.1145/1143120.1143127. 1.2, 2.1

[83] Alan E Gelfand and Adrian FM Smith. Sampling-based approaches to calculating marginal densities. *Journal of the American statistical association*, 85 (410):398–409, 1990. 3.1, 3.4.1

[84] Henri Gilbert, Matthew Robshaw, and Herve Sibert. Active attack against hb+: a provably secure lightweight authentication protocol. *Electronics Letters*, 41(21):1169–1170, 2005. 3.2

[85] O. Goldreich, A. Sahai, and S. Vadhan. Can statistical zero knowledge be made non-interactive? or on the relationship of SZK and NISZK. In *Proc. of CRYPTO*, pages 467–484, 1999. 6.5

[86] Oded Goldreich. Candidate one-way functions based on expander graphs. 2000. 3.2

[87] Dan Goodin. Why passwords have never been weaker-and crackers have never been stronger. http://arstechnica.com/security/2012/08/passwords-under-assault/, August 2012. 1.7.1, 6.1

[88] R. Halprin and M. Naor. Games for extracting randomness. *XRDS: Crossroads, The ACM Magazine for Students*, 17(2):44–48, 2010. 7.3.3

[89] Tzvika Hartman and Ran Raz. On the distribution of the number of roots of polynomials and explicit weak designs. *Random Structures & Algorithms*, 23 (3):235–263, 2003. 7.7.2, 7.7.2, 7.7.2, 7.7.4

[90] Johan Håstad. Clique is hard to approximate within $n^{1\text{-epsilon}}$. In *Proc. of FOCS*, 1996. 5.3.5

[91] N. Hopper and M. Blum. Secure human identification protocols. *Advances in cryptologyASIACRYPT 2001*, pages 52–66, 2001. 3.2, 6.5

[92] Imperva. Consumer password worst practices. 2010. URL `http://www.imperva.com/docs/WP_Consumer_Password_Worst_Practices.pdf`. Retrived 1/22/2013. 1.7.1, 5.1.1, 5.1.3, 5.6, 6.1

[93] Ari Juels and Stephen A Weis. Authenticating pervasive devices with human protocols. In *Advances in Cryptology–CRYPTO 2005*, pages 293–308. Springer, 2005. 3.2

[94] R. M. Karp. Reducibility among combinatorial problems. In R. E. Miller and J. W. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum, 1972. 5.4.2

[95] Jonathan Katz and Ji Sun Shin. Parallel and concurrent security of the hb and hb+ protocols. In *Advances in Cryptology-EUROCRYPT 2006*, pages 73–87. Springer, 2006. 3.2

[96] Michael Kearns and Umesh Virkumar Vazirani. *An introduction to computational learning theory*. The MIT Press, 1994. 3.5.1

[97] 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 *Proc. of Oakland*, pages 523–537, 2012. 5.1, 5.1.3

[98] S. Khot. On the power of unique 2-prover 1-round games. In *Proc. of STOC*, pages 767–775, 2002. 5.3, 5.3.4

[99] S. Khot and O. Regev. Vertex cover might be hard to approximate to within 2- ε. *Journal of Computer and System Sciences*, 74(3):335–349, 2008. 5.3.4

[100] T. Kohonen. *Associative memory: A system-theoretical approach*. Springer-Verlag Berlin; FRG, 1977. 2.1

[101] 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 2011 annual conference on Human factors in computing systems*, pages 2595–2604. ACM, 2011. 1.2.1, 2.2, 3.2, 7.3, 7.4.2

[102] H. Kruger, T. Steyn, B. Medlin, and L. Drevin. An empirical assessment of factors impeding effective password management. *Journal of Information Privacy and Security*, 4(4):45–59, 2008. 1.1, 1.2.1, 2.1, 2.2, 3.2, 5.1, 6.1.1, 3, 7.3.3

[103] E. Kushilevitz and Y. Mansour. Learning decision trees using the Fourier spectrum. *SIAM J. Comput.*, 22(6):1331–1348, 1993. ISSN 0097-5397. 6.5

[104] Qiming Li, Yagiz Sutcu, and Nasir Memon. Secure sketch for biometric templates. In *Advances in Cryptology–ASIACRYPT 2006*, pages 99–113. Springer, 2006. 2.2

[105] D. Malone and K. Maher. Investigating the distribution of password choices. In *Proc. of WWW*, pages 301–310, 2012. 5.1.3

[106] D. Marr. Simple memory: a theory for archicortex. *Philosophical Transactions of the Royal Society of London. Series B, Biological Sciences*, pages 23–81, 1971. 2.1, 2.3.1

[107] J.L. Massey. Guessing and entropy. In *Information Theory, 1994. Proceedings., 1994 IEEE International Symposium on*, page 204. IEEE, 1994. 1.2.2, 2.2, 3.2, 7.4, 7.4.2

[108] G.A. Miller. The magical number seven, plus or minus two: some limits on our capacity for processing information. *Psychological review*, 63(2):81, 1956. 3.3.2, 3.6.3, 7.3

[109] Randall. Monroe. Xkcd: Password strength. `http://www.xkcd.com/936/`. Retrieved 8/16/2011. 1.2, 2.1, 7.4.2

[110] Marti Motoyama, Kirill Levchenko, Chris Kanich, Damon McCoy, Geoffrey M Voelker, and Stefan Savage. Re: Captchas–understanding captcha-solving services in an economic context. In *USENIX Security Symposium*, volume 10, 2010. 6.2.1, 3

[111] Dave. Munger. Is 17 the "most random" number? `http://scienceblogs.com/cognitivedaily/2007/02/is_17_the_most_random_number.php`, 2007. Retrieved 8/16/2011. 3.2, 7.3.3

[112] Moni Naor and Benny Pinkas. Visual authentication and identification. In *Advances in CryptologyCRYPTO'97*, pages 322–336. Springer, 1997. 3.2

[113] Moni Naor and Adi Shamir. Visual cryptography. In *Advances in CryptologyEUROCRYPT'94*, pages 1–12. Springer, 1995. 3.2

[114] Arvind Narayanan and Vitaly Shmatikov. Robust de-anonymization of large sparse datasets. In *Proc. of the 2008 IEEE Symposium on Security and Privacy*, pages 111–125. IEEE, 2008. 6.5

[115] Noam Nisan and Avi Wigderson. Hardness vs randomness. *Journal of Computer and System Sciences*, 49(2):149 – 167, 1994. ISSN 0022-0000. doi: http://dx.doi.org/10.1016/S0022-0000(05)80043-1. URL http://www.sciencedirect.com/science/article/pii/S0022000005800431. 2.2, 7.7, 7.7.2, 4, 7.7.2, 7.7.2, 7.7.2, 27

[116] Ryan ODonnell. Analysis of boolean functions. *Textbook in Progress. Available online at http://analysisofbooleanfunctions.org/*, 2014. 3.1, 3.1, 3.4.2, 8.2, 6

[117] P. Oechslin. Making a faster cryptanalytic time-memory trade-off. *Advances in Cryptology-CRYPTO 2003*, pages 617–630, 2003. 6, 6.1.1, 7.5.1

[118] Lawrence O'Gorman. Comparing passwords, tokens, and biometrics for user authentication. *Proceedings of the IEEE*, 91(12):2021–2040, 2003. 2.2

[119] Ken Perlin. Implementing improved perlin noise. *GPU Gems*, pages 73–85, 2004. 6.3.2

[120] Paul Pimsleur. A memory schedule. *The Modern Language Journal*, 51(2):pp. 73–75, 1967. ISSN 00267902. URL http://www.jstor.org/stable/321812. 4.2

[121] J. Pliam. On the incomparability of entropy and marginal guesswork in brute-force attacks. *Progress in CryptologyINDOCRYPT 2000*, pages 113–123, 2000. 1.2.2, 2.2, 3.2, 5.7, 7.4

[122] N. Provos and D. Mazieres. Bcrypt algorithm. 1.7.1, 2.5, 3.6, 6.1.1, 6, 6.3.1, 7.5.1

[123] Kenneth Radke, Colin Boyd, Juan Gonzalez Nieto, and Margot Brereton. Towards a secure human-and-computer mutual authentication protocol. In *Proceedings of the Tenth Australasian Information Security Conference (AISC 2012)*, volume 125, pages 39–46. Australian Computer Society Inc, 2012. 2.3.3

[124] Srinivasa Ramanujan. A proof of bertrand's postulate. *Journal of the Indian Mathematical Society*, 11:181–182, 1919. 27

[125] G. Rasch. The poisson process as a model for a diversity of behavioral phenomena. In *International Congress of Psychology*, 1963. 2.4.2

[126] Ran Raz, Omer Reingold, and Salil Vadhan. Extracting all the randomness and reducing the error in trevisan's extractors. In *Proceedings of the Thirty-first Annual ACM Symposium on Theory of Computing*, STOC '99, pages 149–158, New York, NY, USA, 1999. ACM. ISBN 1-58113-067-8. doi: 10.1145/301250.301292. URL http://doi.acm.org/10.1145/301250.301292. 7.7.2, 7.7.2, 7.7.2, 7.7.2, 27, 7.7.4

[127] Vojtěch Rödl. On a packing and covering problem. *European Journal of Combinatorics*, 6(1):69–78, 1985. 2.2

[128] Phillip Rogaway. Nonce-based symmetric encryption. In *Fast Software Encryption*, pages 348–358. Springer, 2004. 6.5

[129] Andy Ross. Random ink blot. http://demonstrations.wolfram.com/RandomInkBlot/. 6.5

[130] Blake Ross, Collin Jackson, Nick Miyake, Dan Boneh, and John C Mitchell. Stronger password authentication using browser extensions. In *Usenix security*, pages 17–32. Baltimore, MD, USA, 2005. 2.2

[131] Graig Sauer, Harry Hochheiser, Jinjuan Feng, and Jonathan Lazar. Towards a universally usable captcha. In *Proceedings of the 4th Symposium on Usable Privacy and Security*, 2008. 6.1.1

[132] K. Scarfone and M. Souppaya. Guide to enterprise password management (draft). *National Institute of Standards and Technology*, 800-188(6):38, 2009. 1.1, 1.2, 2.1, 2.1

[133] Karen Scarfone and Murugiah. Souppaya. Nist special publication 800-118: Guide to enterprise password management (draft), April 2009. 5.1.3, 6.1.1, 6.2.1

[134] S. Schechter, A.J.B. Brush, and S. Egelman. It's no secret. measuring the security and reliability of authentication via 'secret' questions. In *2009 30th IEEE Symposium on Security and Privacy*, pages 375–390. IEEE, 2009. 2.5

[135] S. Schechter, C. Herley, and M. Mitzenmacher. Popularity is everything: A new approach to protecting passwords from statistical-guessing attacks. In *Proceedings of the 5th USENIX conference on Hot topics in security*, pages 1–8. USENIX Association, 2010. 5.1.3

[136] D. Seeley. Password cracking: A game of wits. *Communications of the ACM*, 32(6):700–703, 1989. 1.7.1, 5.7, 6.1, 9.2

[137] R. Shaltiel. Recent developments in explicit constructions of extractors. *Current Trends in Theoretical Computer Science: The Challenge of the New Century*, 2004. 7.3.3

[138] Adi Shamir. Ip= pspace. *Journal of the ACM (JACM)*, 39(4):869–877, 1992. 12

[139] C.E. Shannon and W. Weaver. *The mathematical theory of communication*. Citeseer, 1959. 7.4.2

[140] R. Shay, P.G. Kelley, S. Komanduri, M.L. Mazurek, B. Ur, T. Vidas, L. Bauer, N. Christin, and L.F. Cranor. Correct horse battery staple: Exploring the usability of system-assigned passphrases. In *Proceedings of the Eighth Symposium on Usable Privacy and Security*, page 7. ACM, 2012. 1.2.1, 1.2.1, 2.2, 3.2

[141] Abe. Singer. No plaintext passwords. *;login: THE MAGAZINE OF USENIX & SAGE*, 26(7), November 2001. Retrieved 8/16/2011. 1.1, 2.1, 6.1.1

[142] Jonathan Sondow. Ramanujan primes and bertrand's postulate. *American Mathematical Monthly*, 116(7):630–635, 2009. 7.7.2, 23, 7.7.2

[143] J.D. Spence. *The memory palace of Matteo Ricci*. Penguin Books, 1985. 2.1, 2.3.1, 2.6

[144] L.R. Squire. On the course of forgetting in very long-term memory. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, 15(2):241, 1989. 2.1, 2.4.1, 7.6

[145] L. STANDINGT. Learning 10,000 pictures. *Quarterly Journal of Experimental Psychology*, 5(20):7–22, 1973. 2.1, 2.2, 2.4.1, 3.2, 4.2

[146] Joel. Stein. Pimp my password. *Time*, page 62, August 29 2011. 1.2, 2.1

[147] A. Stubblefield and D. Simon. Inkblot authentication. Technical report, Technical Report MSR-TR-2004-85, 2004. 1.7.2

[148] Adam Stubblefield and Dan Simon. Inkblot authentication. Technical report, 2004. 6.1.1

[149] Luca Trevisan. Extractors and pseudorandom generators. *Journal of the ACM*, 48(4):860–879, 2001. 2.2, 7.7.2

[150] L.G. Valiant. Memorization and association on a realistic neural model. *Neural computation*, 17(3):527–555, 2005. 2.1

[151] Hedderik van Rijn, Leendert van Maanen, and Marnix van Woudenberg. Passing the test: Improving learning gains by balancing spacing and testing effects. In *Proceedings of the 9th International Conference of Cognitive Modeling*, 2009. 2.4.1, 7.6.1

[152] L. Von Ahn, M. Blum, N. Hopper, and J. Langford. Captcha: Using hard ai problems for security. *Advances in CryptologyEUROCRYPT 2003*, pages 646–646, 2003. 4, 1.7.2, 2.4.1, 6.1.1, 6.2

[153] Luis Von Ahn, Benjamin Maurer, Colin McMillen, David Abraham, and Manuel Blum. recaptcha: Human-based character recognition via web security measures. *Science*, 321(5895):1465–1468, 2008. 6.1.1

[154] W.A. Wagenaar. Generation of random sequences by human subjects: A critical survey of literature. *Psychological Bulletin*, 77(1):65, 1972. 3.2, 7.3.3

[155] Michael J Watkins and John M Gardiner. An appreciation of generate-recognize theory of recall. *Journal of Verbal Learning and Verbal Behavior*, 18(6):687–704, 1979. 6.1.1

[156] M. Weir, S. Aggarwal, M. Collins, and H. Stern. Testing metrics for password creation policies by attacking large sets of revealed passwords. In *Proc. of CCS*, pages 162–175, 2010. 5.1.3

[157] DJ Willshaw and JT Buckingham. An assessment of marr's theory of the hippocampus as a temporary memory store. *Philosophical Transactions of the Royal Society of London. Series B: Biological Sciences*, 329(1253):205, 1990. 2.1

[158] R.J. Witty, K. Brittain, and A. Allen. Justify identity management investment with metrics. Gartner Group report, 2004. 1.3, 5.1, 6.1.1

[159] Robert Sessions Woodworth and Harold Schlosberg. *Experimental psychology*. Oxford and IBH Publishing, 1954. 4.2

[160] PA Wozniak and Edward J Gorzelanczyk. Optimization of repetition spacing in the practice of learning. *Acta neurobiologiae experimentalis*, 54:59–59, 1994. 1.2.1, 2.1, 2.4.1, 2.7, 4.1, 4.2, 7.6

[161] Piotr Wozniak. Supermemo 2004. *TESL EJ*, 10(4), 2007. 4.2

[162] J. Yan, A. Blackwell, R. Anderson, and A. Grant. Password memorability and security: Empirical results. *Security & Privacy, IEEE*, 2(5):25–31, 2004. ISSN 1540-7993. 1.2, 2.1

[163] Andrew C Yao. Protocols for secure computations. In *Proc. of FOCS*, pages 160–164, 1982. 6.5

[164] Shikun Zhang. Bill gates kissing an igloo – a password management application with provable security and minimal user effort. *Carnegie Mellon University: Senior Research Thesis*, May 2014. Advised by Jeremiah Blocki, Manuel Blum and Anupam Datta. 2.7

[165] Andrew Zonenberg. Distributed hash cracker: A cross-platform gpu-accelerated password recovery system. *Rensselaer Polytechnic Institute*, page 27, 2009. 1.7.1, 6.1, 6.1.1