MODELING RATIONAL ADVERSARIES: PREDICTING BEHAVIOR AND DEVELOPING DETERRENTS

by

Benjamin Harsha

A Dissertation

Submitted to the Faculty of Purdue University

In Partial Fulfillment of the Requirements for the degree of

Doctor of Philosophy

Department of Computer Science

West Lafayette, Indiana

August 2021

THE PURDUE UNIVERSITY GRADUATE SCHOOL
STATEMENT OF COMMITTEE APPROVAL

Dr. Jeremiah Blocki, Chair
Department of Computer Science

Dr. Mikhail Atallah
Department of Computer Science

Dr. Ninghui Li
Department of Computer Science

Dr. Simina Brânzei
Department of Computer Science

Approved by:
Dr. Kihong Park

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

In the field of cybersecurity, it is often not possible to construct systems that are resistant to all attacks. For example, even a well-designed password authentication system will be vulnerable to password cracking attacks because users tend to select low-entropy passwords. In the field of cryptography, we often model attackers as powerful and malicious and say that a system is broken if any such attacker can violate the desired security properties. While this approach is useful in some settings, such a high bar is unachievable in many security applications e.g., password authentication. However, even when the system is imperfectly secure, it may be possible to deter a rational attacker who seeks to maximize their utility. In particular, if a rational adversary finds that the cost of running an attack is higher than their expected rewards, they will not run that particular attack. In this dissertation we argue in support of the following statement: Modeling adversaries as rational actors can be used to better model the security of imperfect systems and develop stronger defenses. We present several results in support of this thesis. First, we develop models for the behavior of rational adversaries in the context of password cracking and quantum key-recovery attacks. These models allow us to quantify the damage caused by password breaches, quantify the damage caused by (widespread) password length leakage, and identify imperfectly secure settings where a rational adversary is unlikely to run any attacks i.e. quantum key-recovery attacks. Second, we develop several tools to deter rational attackers by ensuring the utility-optimizing attack is either less severe or nonexistent. Specifically, we develop tools that increase the cost of offline password cracking attacks by strengthening password hashing algorithms, strategically signaling user password strength, and using dedicated Application-Specific Integrated Circuits (ASICs) to store passwords.

1. INTRODUCTION

In cryptography, we often consider powerful and malicious adversaries and say that a system is broken if such an adversary can violate any desired security property. Indeed, organizations like the National Institute of Standards and Technology (NIST) publish documents with recommendations on key lengths and procedures designed specifically to resist powerful attackers with nation-state level resources, e.g. [1, 2]. Unfortunately, in many practical scenarios resisting such a powerful adversary is unachievable. For example, even a well-designed password authentications system is vulnerable to password guessing attacks because users tend to select low-entropy passwords. If we were to examine a password authentication system from the perspective of a nation-state-level adversary, we would conclude that it is broken and that the adversary will compromise most accounts. In this dissertation, we focus on “imperfectly secure" systems which could be broken by powerful, malicious adversaries.

Even in an imperfectly secure system, there may still be methods of deterring an attacker whose behavior is not fully adversarial. While a malicious adversary will always try to run a maximally damaging attack, a rational adversary instead runs the attack that maximizes its utility. This utility consists of two primary components: the rewards for completing an attack and the expected costs to run it. Even when a system is imperfectly secure, it may be possible to influence the adversary’s behavior by altering the system in a way that increases the attacker’s expected costs or decreases the attacker’s expected reward. For example, if we can increase the adversary’s costs to be higher than the rewards for an attack, the adversary will not run that particular attack. Instead, they may run a different, less severe, attack or choose not to attack at all. In the context of password cracking, we cannot defend against a powerful malicious adversary but we may be able to meaningfully deter a rational attacker and reduce the damage they cause.

In this dissertation we argue in support of the following thesis:

> Modeling adversaries as rational actors can be used to better model the security of imperfect systems and develop stronger defenses.

We present two sets of results in support of this thesis. The first set of results focuses on modeling rational attacker behavior. We use our rational attacker models to quantitatively predict how many users will have their password cracked in offline password cracking attacks [3], estimate the damage caused by (currently widespread) password length leakage [4], determine that rational adversaries are unlikely to run quantum key-recovery attacks [5], and describe the structure of quantum brute force attacks against non-uniform distributions. The second set of results focuses on developing defenses targeted towards rational adversaries. We will show how signaling password strength to a rational adversary can (counter-intuitively!) *decrease* the expected number of cracked passwords [6], how to develop stronger hashing functions to increase attacker costs [7, 8], and describe a new password storage system that takes advantage of the high parallelism and computation speeds of Application Specific Integrated Circuits (ASICs). We show that these defenses can influence a rational attacker to select less harmful actions by increasing attacker costs and/or by strategically revealing information.

1.1 Organization

This document is divided into three chapters covering general background knowledge, rational attacker behavior predictions, and defenses against rational attackers, respectively. In chapters 3 and 4 multiple relevant results surrounding rational adversary models are discussed. Chapter 3 will delve into models of rational adversary behavior and its consequences, and includes summaries of the following works:

1.1.1 Predicting Rational Adversary Behavior

- **The Economics of Offline Password Cracking [3]:** The past decade has seen more than its fair share of password breaches ([9, 10, 11, 12, 13, 14] to name just a few examples). We develop an economic model of a rational offline password cracker which allows us to make quantitative predictions about the fraction of accounts that a rational password attacker would crack in the event of an authentication server breach. Unfortunately, our models predict that with key-stretching parameters seen in practice users are not sufficiently protected against rational offline password cracking adversaries. This prediction holds even if we incorporate an aggressive model of diminishing returns for the attacker (e.g., the total value of 500 million cracked passwords is less than 100 times the total value of 5 million passwords). On a positive note our analysis demonstrates that memory hard functions (MHFs) such as SCRYPT or Argon2i can significantly reduce the damage of an offline attack.

- **Quantifying the Damage of Password Length Leakage** [4]**:** All encryption must necessarily leak some information about the length of the encrypted message. Some common methods, such as AES-GCM, leak the exact length of a message. While this does not always pose a significant risk it can cause issues when the message is short and sensitive, e.g. passwords. We find that password length is widely leaked by many ($> 80\%$) of the Alexa top 100 sites, giving attackers a potential advantage when trying to guess a user's password. To quantify this advantage we extend our economic models to compare an attacker's performance with and without knowledge of a password's length. We find that anywhere from $50 - 100\%$ additional passwords might be cracked if the length is leaked. We describe situations where an attacker can make more in additional profit than it might cost to run such an attack, meaning defenses (e.g. padding) should be put in place to prevent these attacks.

- **An Economic Analysis of Quantum Key-Recovery Attacks** [5]**:** Quantum computers are an emerging technology with the theoretical ability to break several vital cryptographic primitives. One such example is Grover's algorithm, which can run an attack that breaks a symmetric cipher key in square root of the time a classical computer requires. We seek to understand how far quantum computers would need to advance to make such an attack profitable. We consider a case where an adversary has some set valuation function for the stored information (which may decay over time) and a time limit for their attack. Under incredibly optimistic projections for advances in quantum computing we find that *no profit-motivated attacker will want to run a Grover's algorithm attack in almost any scenario, even for 128 bit keys.* This suggests that an appropriate action to defend against profit motivated quantum adversaries is to do nothing, i.e. keep key lengths as they currently are. This contradicts common advice to double key lengths to counter the square root advantage from Grover's algorithm.

- **Grover's Algorithm on Non-Uniform Distributions:** Grovers algorithm allows us to find some input x to any black-box function $f : X \to Y$ giving a desired output x^* s.t. $f(x^*) = y$ in time $O\left(\sqrt{|X|}\right)$. Note that a classical computer requires $O\left(|X|\right)$ to accomplish the same task. Often we can think of the standard inputs to f as being uniformly distributed, e.g. when we are dealing with symmetric key cipher keys. However, in other settings this may not be the case, e.g. user-selected passwords. We seek to better understand what happens when the correct input x^* is selected from these nonuniform distributions. First, we present an example showing that there exist certain distributions and strategies that lower an attacker's expected costs. We hope to characterize the distributions where such strategies exist and provide an algorithm for optimally partitioning distributions to minimize attack costs.

1.1.2 Deterring Rational Adversaries

Chapter 4 covers several defenses designed to deter rational adversary attacks. The following works are summarized in this chapter:

- **Information Signaling in Password Storage** [6] Here we consider an interesting question. Can we protect users from password cracking attacks by giving the attacker *more* information? Specifically, we are interested in providing some sort of signal about the strength of a user's password. We provide a proof-of-concept signaling scheme where, for a specific distribution and signaling strategy, we can reduce the expected number of cracked passwords. Essentially we are able to take advantage of the fact that password cracking is not a zero sum game - allowing us to increase the utility of both attacker and defender at once. We are also able to note that these systems accomplish their goal without causing too much collateral damage (i.e. people who have their password cracked under this system when they would have been secure otherwise). We believe that our results demonstrate that there are likely applications of information signaling in the field of security, and believe that the subject merits further examination.

- **Just In Time Hashing [8]:** In an offline password cracking attack the strength of the hash function serves as a last line of defense against attackers. However, increasing guessing costs often comes as the cost of increasing authentication time, and the longer we spend authenticating the more annoyed users will be. We introduce Just in Time Hashing (JIT), a client side key-stretching algorithm to protect user passwords against offline brute-force cracking attempts *without* increasing delay for the user. The basic idea is to exploit idle time while the user is typing in their password to perform extra key-stretching. As soon as the user types in the first character(s) of their password our algorithm immediately begins filling memory with hash values derived from the character(s) that the user has typed thus far. We conduct a user study to guide the development of JIT e.g. by determining how much extra key-stretching could be performed during idle cycles or how many consecutive deletions JIT may need to handle. Our security analysis demonstrates that JIT can substantially increase guessing costs over traditional key-stretching algorithms with equivalent (or less) authentication delay.

- **Data Independent Memory Hard Functions: New Attacks and Stronger Constructions [7]:** In [7] we look at the details of Memory Hard Function constructions and show multiple results concerning their strength in the face of state-of-the-art attacks. In this thesis we will focus on two specific results that were shown in this paper. First, we will investigate the effectiveness of an attack called the Greedy Pebble attack on proposed constructions, including our previously proposed construction in [15]. We find that while our previously proposed construction resists many attacks, it is particularly vulnerable to one called the Greedy Pebble attack. We propose a new hybrid graph construction and empirically show that it is more resistant to state-of-the-art attacks. Finally, we show a method to modify a specific part of Argon2 [16] to improve its resistance to parallelization on an Application Specific Integrated Circuit.

- **Defending against Password Cracking by Using ASICs:** In [3] we showed that MHFs are a viable defense against password cracking attacks without requiring very high security parameters (as would be needed for hash iteration). The reason for using MHFs is to even the playing field between servers and password crackers, removing any advantage the password cracker was exploiting to run profitable attacker. Here we consider a different method of accomplishing the same goal - adding an ASIC to a server so they can run hash iteration just as fast as an adversary could. We wish to run some cost-benefit analyses on both of these strategies to help provide companies with guidance as to which solution may be more appropriate for their specific defense needs.

2. BACKGROUND

Many of the works discussed in this thesis focus on several common elements that merit some initial discussion. For example, many of the works involve password cracking in some form. These attacks are not only prevalent but are also a situation where attacks are essentially always feasible, leading to rational adversary analysis being a useful tool to understand them. This section is dedicated to providing background relevant to several works, including summaries of relevant work in game theory and decision-theoretic models, password cracking, and quantum computing. This section also contains a section covering notation that will be used throughout the thesis.

2.1 Game-theoretic and Decision-theoretic models

The game or decision theoretic modeling of adversaries attempts to capture the expected behavior of an adversary in some game describing a relevant situation. For a profit-motivated adversary, a game is described where some attack is going to be run. The adversary is given some information about this attack and is asked to make a decision about attack parameters e.g. how many guesses they might make against a particular password during an offline password cracking attack. Usually, these games are described as a Stackelberg game where one player (the defender/server owner) moves first by selecting their security setup and parameters. After these have been selected the adversary is allowed to make their decision about an attack.

Blocki and Datta described a decision-theoretic model for an adversary for Cost-Asymmetric Hashing in 2016 [17]. Here the authors describe how to calculate the optimal pepper (AKA private salt) distribution for password hashing. They model an offline password cracking attack through the description of utility functions, which themselves are based on the reward and cost of an attack. The authors apply this model to several empirical password datasets and find that it is capable of reducing the number of cracked passwords by about 50%.

This model is derived by determining the cost to the server owner, costs of an attacker, and expected rewards based on some value v. Using these building blocks utility functions for the server owner and adversary are constructed in a way that allows for optimization of the distribution of pepper values. Many of the works described in this paper will take the same core step of analyzing the cost and reward functions for various entities, combining them to gain some insight into adversary behavior.

2.2 Rational Actor Models

The concept of a rational adversary or actor is common in several fields, including economics and game theory. Indeed, examples of rational adversary analysis in the field of game theory can be found from Von Neumann and Morgenstern [18]. Recently, rational actors have seen applications in airport security design [19], Environmental Conservation [20], and even Law [21]. In each of these applications, we model at least one (not necessarily adversarial) actor who makes their decisions to increase utility. Often actors are portrayed as being interested in profit, especially in economics, though in works like [21, 22] actors may be interested in some less quantifiable reward like their own freedom from incarceration or a lawyer winning a case.

Rational actors are often found in the subfield of cryptography, especially in the analysis of protocols. Rational adversaries have seen fruitful applications in secret sharing [23, 24] and multiparty computation [25, 26] as well as in more generic analysis frameworks [27]. Some interesting parallels to this exist such as Cleve's work showing fairness in 2PC is unachievable [28], with follow-up work showing that it can be achieved in some cases in a rational model [29]. Password hashing research has seen several interesting applications, especially in the analysis of proposed password storage systems [17, 30].

2.3 Password Cracking

Password cracking is the subject of a large proportion (though not all!) of the profit-motivated adversary analysis that is contained in this proposal. Given its strong influence, the background and some relevant papers are discussed in this subsection.

Password hashing: Password hashing is the process of taking a password pwd, adding some randomized n-bit "salt" value s_n, and using the result as the input to some cryptographic hash function $\mathcal{H}$. The resulting value $\mathcal{H}(pwd||s_n)$ is stored in a table alongside the username and s_n values. This allows for a user to authenticate by providing a password *pwd*. If the result $\mathcal{H}\left(pwd||s_n\right) = \mathcal{H}(pwd||s_n)$ then the user has authenticated successfully.

Obtaining these records $s_n, \mathcal{H}\left(pwd||s_n\right)$ allows an adversary to run an offline password cracking attack, where they can make as many guesses as they like without tripping any online password cracking attack defenses (limited by equipment and time, of course). To defend against these offline attacks a method known as *key-stretching* is used. Key stretching is any method that artificially makes a hash function arbitrarily difficult to compute. Several methods of key stretching exist, with the current most common method being hash iteration. In this method rather than storing the result of the hash function $\mathcal{H}\left(pwd||s_n\right)$ the result is fed back into the hash function k times. The resulting $\mathcal{H}^k\left(pwd||s_n\right)$ is stored in its stead. While this is a common method and is seen in many leaked databases its security has been called into question [3], as it provides little resistance to ASIC-based attacks.

Memory-hard functions: Memory hard functions (MHFs) are a second type of key stretching. With hash iteration, the goal was to artificially increase the computation time of the function. With MHFs the goal is to not only increase computation time but to increase the memory usage of the function at the same time. Motivating this goal is the advantage that Application-Specific Integrated Circuit (ASIC) users have over those using traditional CPUs. As can be seen in bitcoin mining computation power is overwhelmingly dominated by ASICs as they can run the same function faster for a much lower cost. An adversary using an ASIC for password hashing would enjoy a similar advantage, capable of more than ten trillion calls to a base hash function per second [31].

While ASICs enjoy a vast advantage in computing speed they do not enjoy a large advantage for memory usage. That is, while they can calculate functions requiring small amounts of memory incredibly quickly they are not capable of reading or writing to memory any faster than a typical desktop. Behind this is the bottleneck that standard chips e.g. DDRM RAM chips impose. Essentially, they are already very optimized for memory storage and ASICs do not have any additional way to squeeze more performance out of them. So, if a function can be designed to require a large amount of memory the computation speed will be limited by the read/write speed of RAM, leveling the playing field in a way for hash computation.

Early candidates for MHFs were proposed by Percival [32] (SCRYPT) and Boyen [33] (as

Halting Password Puzzles). Since then it has been shown that Percival's SCRYPT construction is maximally memory hard [34]. Of note is that these are a specific type of memory-hard function called a *Data-Dependent* MHF, or dMHF. In these dMHFs the memory access pattern of the MHF is dependent on the input, opening the possibility of side-channel attacks where an adversary can monitor which areas of memory are being accessed during computation.

Contrasted with dMHF are *Data-Independent* MHFs, or iMHFs. In these MHFs the data access pattern is selected independently from the input, preventing side-channel leakage from the memory access patterns. Hybrid constructions also exist that run as an iMHF for the first portion of computation before switching to a dMHF mode - ideally offering the best of both worlds by at least requiring an adversary to complete the iMHF portion before they can begin to use any access pattern data

2.4 Online Password Cracking Attacks

While adversaries have plenty of leaks to choose from if they want to run an offline password cracking attack there is still the concern of online password cracking attacks. In many ways, especially when looking at the economics of these attacks, it closely resembles an offline attack. The primary difference is that the adversary does not have the list of password hashes and is checking password correctness by submitting login attempts to an external server. This can be modeled as the adversary having limited access to some oracle $\mathcal{A}^{\mathcal{H}}$ and being able to submit queries limited by some absolute number or some time limit between queries (or batches of queries). In this case, the adversary does not have access to the salt value and relies on guesses of the value of *pwd* alone. The primary differences in terms of for-profit adversaries when running offline vs online attacks are the cost of an attack and maximum guessing limits. While an attacker can make unlimited guesses during an offline attack due to knowledge of the salt, correct hash value, and hash function they cannot do the same in an online attack. Here they are instead limited by the server they are querying, which is likely not running on any specialized equipment. Rate limiting by these servers is the primary defense against these attacks, where accounts may be locked out until additional conditions are met if too many attempts (k) are made within a short time (currently recommended as 100 consecutive failed logins by NIST [35]). When enforced correctly this limit reduces the success probability of a password guessing attack to at most the sum of the probabilities of the k most common passwords.

Another method of rate limiting in an online attack makes use of CAPTCHAS [36]. In some senses, this method more closely resembles an offline cracking attack in that the adversary is not technically restricted to a maximum number of guesses but rather is required to solve a CAPTCHA every so often. The idea behind this is that some human activity is required to run the attack, preventing much faster attacks. From an economic perspective, this does increase the cost of an attack greatly, but in some cases, they are still feasible. CAPTCHA farms exist and as early as 2010 have been selling CAPTCHA solutions for around $0.80-$1.20 per 1000 solutions. While this seems inexpensive it still dominates the cost of ASIC-based offline attacks, providing some economic protections for users.

2.5 Memory Hard Functions

Memory Hard Functions (MHFs) are a key cryptographic primitive in the design of password hashing, algorithms, and egalitarian proof of work puzzles [37]. In the context of password hashing, we want to ensure that the function can be computed reasonably quickly on standard hardware, but that it is prohibitively expensive to evaluate the function millions or billions of times. The first property ensures that legitimate users can authenticate reasonably quickly, while the purpose of the latter goal is to protect low-entropy secrets (e.g., passwords, PINs, biometrics) against brute-force offline guessing attacks. One of the challenges is that the attacker might attempt to reduce computation costs by employing customized hardware such as a Field Programmable Gate Array (FPGA) or an Application Specific Integrated Circuit (ASIC). Memory-hard functions, intuitively, require both a large amount of computation time *and* require allocating large amounts of memory during the entire computation. Because FPGAs and ASICs would need to interface with standard DRAM chips to have sufficient memory to quickly compute MHFs they do not have as significant an advantage as they would with non-memory-hard hash functions. MHFs were of particular interest in the 2015 Password Hashing Competition [38], where the winner, Argon2 [16] was a MHF. MHFs can be broadly split into two categories: Data-dependent MHFs (dMHFs) and data-independent MHFs (iMHFs). iMHFs offer a natural immunity to side-channel attacks making them attractive for password hashing applications.

An iMHF $f_{G,H}$ can be viewed as a mode of operation over a directed acyclic graph (DAG) $G = (V = [N], E)$ that encodes data-dependencies (because the DAG is static the memory access pattern will be identical for all inputs) and a compression function $H(\cdot)$. Alwen and Serbinenko [39] defined $f_{G,H}(x) = \mathsf{lab}_{G,H,x}(N)$ to be the label of the last node in the graph G on input x. Here, the label of the first node $\mathsf{lab}_{G,H,x}(1) = H(1, x)$ is computed using the input x and for each internal node v with $\mathsf{parents}(v) = v_1, \ldots, v_\delta$ we have

$$\mathsf{lab}_{G,H,x}(v) = H\left(v, \mathsf{lab}_{G,H,x}(v_1), \ldots, \mathsf{lab}_{G,H,x}(v_\delta)\right) .$$

2.6 Graph pebbling

The graph pebbling game is a game played over a directed acyclic graph where the player sets “pebbles" on nodes of the graph according to a set of rules. First introduced by Hewitt and Paterson [40] and Cook [41] the (sequential) black pebbling game (and its relatives) have been used to great effect in theoretical computer science. Some early applications include space/time trade-offs for various computational tasks such as matrix multiplication [42], the FFT [43, 42], integer multiplication [44] and solving linear recursions [45, 46]. More recently, pebbling games have been used for various cryptographic applications including proofs of space [47, 48], proofs of work [49, 50], leakage-resilient cryptography [51], garbled circuits [52], one-time computable functions [53], adaptive security proofs [52, 54] and memory-hard functions [55, 39, 56, 57].

The black pebbling game is played on a fixed DAG G in rounds. Typically, the edges in G represent data dependencies of the function we are trying to compute. At each round, denoted $P_i \subseteq V$, certain vertices are considered to be pebbled if they are contained in P_i. The goal of the game is to pebble a predesignated set of “sink" nodes of G (not necessarily simultaneously). In this document placing a pebble on a node models computing some function and storing the result in memory for later use. In the first round we set $P_0 = \emptyset$. P_i is derived from the previous configuration P_{i-1} according to two simple rules.

1. A node v may be pebbled (added to P_i) if, in the previous configuration all of its parents were pebbled, i.e., parents$(v) \subseteq P_{i-1}$. In this parallel pebbling game, we allow for any number of pebbles to be placed in a single round, while in the sequential version only one pebble may be placed at each round.

2. A pebble can always be removed from P_i.

A sequence of configurations $P = (P_0, P_1, \ldots)$ is a pebbling of G if it adheres to these rules and each sink node of G is contained in at least one configuration. We define the set $\mathcal{P}(G)$ to be the set of all valid sequential pebblings of the graph G and $\mathcal{P}^{||}(G)$ to be the set of all valid parallel pebblings of G.

There are several related metrics for describing the "cost" for a specific pebbling P of the graph G. Cumulative complexity [39] measures the sum of pebbles in each round. Specifically, $CC(P) = \sum_{P_i \in P} |P_i|$. We also define the cumulative complexity of a graph G to be $CC^{||}(G) = \min\limits_{P \in \mathcal{P}^{||}(G)} CC(P)$. The sequential equivalent $CC(G)$ is defined over the set of pebblings $\mathcal{P}(G)$ rather than $\mathcal{P}^{||}(G)$. The problem of finding $CC(G)$ for a DAG G has been proven to be NP-Hard [58]. A related metric known as Amortized Area-Time (aAT) complexity [56] is a modification that charges R times more for adding a pebble than keeping one on a node. Specifically,

$$aAT(P) = \sum_{P_i \in P} |P_I| + R \sum_{P_i \in P} |P_i \setminus P_{i-1}|$$

We define $aAT^{||}(G) = \min\limits_{P \in \mathcal{P}^{||}(G)} aAT(P)$ This specific metric is meant to model computation and storage costs, especially in the context of memory hard functions e.g. in [7, 39, 15]. Alwen and Serbinenko [39] and Alwen and Tackmanm [59] proved that in the parallel random oracle model (PROM) the aAT of a graph G representing data dependencies in a memory-hard function is completely captured by the parallel black pebbling game.

2.7 Quantum Computing

In one work under review and several proposed projects, we investigate some of the economic aspects of quantum computing. This section is intended to provide a high level overview of quantum computing, focusing only on those aspects necessary for the understanding of our results. A much more thorough review of quantum computing is offered by Ronald de Wolf [60] (or any of several other well-written online introductions to quantum computing).

Quantum computing allows for computation over qubits, a quantum analog of classical bits. Each qubit has two state analogous to the classical 0 and 1 states denoted as the $|0\rangle = \binom{1}{0}$ and $|1\rangle = \binom{0}{1}$ states respectively. Rather than each qubit remaining in one classical state they are able to exist as a superposition of states $|\psi\rangle = \alpha_0 |0\rangle + \alpha_1 |1\rangle, \alpha_i \in \mathbb{C}$. Multiple qubits are combined via tensor products and are denoted in this paper as (e.g.) $|01\rangle = |0\rangle \otimes |1\rangle$. When a quantum computer is in a state $|\psi\rangle$ it can be advanced to a new state $|\psi\rangle$ by multiplication with a unitary matrix U (a matrix whose inverse is its conjugate transpose) i.e. $|\psi\rangle = U |\psi\rangle$. Quantum algorithms are described as sequences of unitary transformations on these qubits. For a far more detailed description of the basics of quantum computing, we direct the reader to [60] or another online resource of their choosing.

The ability to operate while in a superposition gives quantum computers an advantage over classical computers - with quantum computers having a significant advantage when solving certain types of problems. In several cases, this improvement is very significant e.g. the ability to solve the integer factorization and discrete log problems in polynomial time - a feat that has not been accomplished with classical computers [61]. The ability to solve these problems efficiently has clear security implications, allowing for quick attacks on several asymmetric or public-key cryptosystems. In other cases, quantum algorithms may provide a significant advantage over classical attacks, but not so significant as a polynomial time attack. Grover's algorithm [62], which we will examine most closely, is an example of this type of attack. Here a quantum computer is capable of providing quadratic speed up which, while not as strong as a polynomial time attack, is still of interest.

2.8 Grover's algorithm

Grover's algorithm [62] is (in Grover's words) "A fast quantum mechanical algorithm for database search". Given black box access to some function $f : X \to Y$ and some value $y \in Y$ we wish to find a value for $x^* \in X$ such that $f(x^*) = y$. Using a classical computer requires $O(N)$, where $N = |X|$. However, if we exploit Grover's algorithm this can be improved from $O(N)$ to $O\left(\sqrt{N}\right)$. While here we are focusing on the case where x^* is unique we note that Grover's algorithm can be modified for the case where multiple solutions are possible, even if the number of solutions is unknown [63]. By setting the function f correctly we can present the problem of finding an AES key for a plaintext/ciphertext pair in terms of a database search solvable using Grover's algorithm. From an asymptotic standpoint this significantly decreases the amount of time it would take to brute force a key e.g. it would only take $\approx 2^{128}$ steps to find a key for AES with a 256 bit key. This is especially significant for AES using shorter key lengths where the number of steps required to find a key becomes more and more feasible.

Grover's algorithm works in rounds, each of which serves to increase the amplitude of the value x^* for which $f(x^*) = 1$ (to a point). It consists of a quantum circuit implementing the function f being inverted and makes use of the following two quantum operations:

- **Hadamard gate:** $H = \frac{1}{\sqrt{N}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$ Maps the $|0\rangle$ and $|1\rangle$ states to $\frac{1}{\sqrt{2}} |0\rangle + \frac{1}{\sqrt{2}} |1\rangle$ and $\frac{1}{\sqrt{2}} |0\rangle - \frac{1}{\sqrt{2}} |1\rangle$ respectively. Importantly - the Hadamard gate is its own inverse.

- **Phase query:** $O_{i,\pm} : |i\rangle \to (-1)^{f(i)} |i\rangle$. Negates the amplitude if the value of f at i is 1.

- **Inversion around mean amplitude U_s:** The unitary transform $U_s = H^{\otimes n} \left(2 |0^n\rangle \langle 0^n| - I_n\right) H^{\otimes n}$ inverts the superposition $|\psi\rangle$ around the mean amplitude i.e. $\alpha_{i+1} = \left(\frac{2}{2^n} \sum_j \alpha_j\right) - \alpha_i$

The algorithm is as follows:

1. Begin in the $|\psi\rangle = |0\rangle^{\otimes N}$ state

2. Apply $H^{\otimes N}$ to get $|\psi\rangle = \frac{1}{\sqrt{N}} \sum\limits_{i=0}^{N-1} |i\rangle$
3. Repeat $O\left(\sqrt{N}\right)$ times:

   (a) Apply the phase query $|\psi\rangle \leftarrow O_{\pm} |\psi\rangle$

   (b) Apply U_s to invert about the mean amplitude

4. Observe the result

Note that because the phase query $O_{\pm}$ negates the amplitude of $|x^*\rangle$ while the mean amplitude stays positive we will end up increasing the amplitude of $|x^*\rangle$ while decreasing the amplitude of all other states. After repeating $O(\sqrt{N})$ times the amplitude α_{x^*} will be likely to be observed. Here the constants are important, as after a time the amplitude α_{x^*} will start decreasing if the inner loop is iterated too much. When N is large (which it will be, for our purposes) the loop should be run about $\frac{\pi}{4}\sqrt{N}$ times for a high probability $\left(\approx 1 - \frac{1}{N}\right)$ of success [63].

Grover's algorithm can be slightly modified to run with k "buckets" in parallel (An indeed this is the optimal method of parallelizing this algorithm, as shown by Zalka [64]). Here rather than having a single oracle that we query during each iteration we have k oracles operating on k buckets. Each bucket runs through the sequential Grover's algorithm steps over a total of $\frac{N}{k}$ elements. At the end we will make k observations rather than one, and some slight work is required to determine if any of these is the correct answer. When running in this mode each of the buckets must run for $\sqrt{\frac{N}{k}}$ iterations, with $k-1$ of these unknowingly running on the empty oracle and one running with an oracle returning 1 as expected. Note that when we run in this mode we require k oracles yet only see a speedup of $\sqrt{k}$. So, while we get our answer faster we must do $\sqrt{k}$ more work to get there. This means that if time is no issue then it would always be better to run the sequential version of Grover's algorithm. However, we will later discuss situations where it may be worth taking the cost increase to obtain information earlier.

Grover's algorithm has clear security implications because of its ability to run a quantum key-recovery attack i.e. it would be able to break symmetric key ciphers like AES. While a classical attack would need to try roughly 2^{n-1} keys before breaking a cipher Grover's algorithm can recover the key in $O(2^{n/2})$ operations - a significant improvement. There is a clear and quick solution to this problem - doubling key lengths. It would be possible to provide security equivalent to a 128-bit key by moving to a 256-bit key, and so on. The downside to this approach is that AES-128 is more efficient than AES-256. Furthermore, it can be challenging to upgrade systems that have already been deployed e.g. when there is some embedded system employing the cipher.

2.9 Post-quantum cryptography

Shor's algorithm can be used to break any public key encryption scheme whose security relies on the hardness of the integer factorization or discrete logarithm problem in polynomial time. This includes the most widely deployed public key encryption/signature schemes including RSA, EC-DSA, Schnorr Signatures, ECDH, etc. Thus, there is a need to migrate towards "Post-Quantum" schemes that resist known quantum attacks like Shor's algorithm [65, 66]. The U.S. National Institute of Standards and Technology (NIST) has been working on developing a set of standards for post-quantum cryptography. NIST first published a report on quantum cryptography in 2016 [66] which outlined their understanding and future plans, and released a call for proposals in 2017 [65] and an update in 2019 [67]. In this document, NIST proposes that an attacker running an attack over one or ten years be limited to a quantum circuit of 2^{40} or 2^{64} layers, respectively. This allows us to implicitly derive speeds for quantum computers running these attacks. The current front-runners as of NIST's update [67] fall into a few common construction categories. Some of these include lattice-based questions like the learning with errors problem (e.g. [68]) which are conjectured to be hard for quantum computers [69] or results from coding theory (e.g. [70]).

2.10 Basic Rational Adversary Model

How we model a rational adversary often involves details specific to one type of attack. However, there are still some characteristics that are common to all rational adversaries discussed in this thesis. To begin, we define a rational adversary as an adversary who is attempting to maximize some function U representing their “utility" from running the attack. The adversary’s utility is defined in terms of an action they take $\vec{x}$ from some set of possibilities X. For example, a password guessing adversary might select the number of guesses they will make on a particular password as their action. This utility function $U(\vec{x}) = R(\vec{x}) - C(\vec{x})$ represents the difference between the attacker’s rewards $R(\vec{x})$ and the attacker’s costs $C(\vec{x})$. A rational adversary selects an action $\vec{x}^*$ such that

$$U(\vec{x}^*) = \max_{\vec{x} \in X} U(\vec{x})$$

While we will often think of utility as a dollar amount in this thesis, this should not be considered the only possible metric. So long as you are willing to accept some quantitative evaluation for some motivation, be that profit, malice, or fun, that is consistent across the R and C you can calculate some measurable utility for the adversary. The specifics of how an adversary selects R, and to a much greater extent how we model their costs C, is dependent on specific properties of the attack. We instantiate these functions with appropriate values to form a rational adversary for a specific attack.

Armed with a specific rational adversary model we can also begin considering the question of defenses. In this thesis, this is primarily done by finding methods of increasing the cost function $C(\vec{x})$. Ideally, we would like to raise the cost function enough that a rational adversary will always select the “do not attack" action to maximize their utility. If that remains infeasible, we may instead try to find some defense that convinces an adversary to choose an action that results in fewer damages to the defender.

3. MODELING RATIONAL ATTACKER BEHAVIOR

This chapter contains sections taken verbatim from [3, 5, 4]

The first category of results we will discuss relates to modeling the behavior of rational adversaries. By examining existing systems with potentially useful results for an adversary using rational adversaries we can begin to develop a picture of how systems may be vulnerable. Using our common example of a password-cracking adversary, we can use a rational adversary model to predict the damage rational adversaries will cause (measured in terms of the proportion of user accounts compromised). By noting that a rational adversary will cause, in our opinion, unacceptable amounts of damage in a password cracking attack we hope to convince the security community that this is an area of research that deserves attention. It is through rational adversary modeling that we can cast a spotlight on potentially problematic situations to help advocate for improving existing systems.

3.1 On the Economics of Offline Password Cracking

In the last few years breaches at organizations like Yahoo!, Dropbox, Lastpass, AshleyMadison, LinkedIn, and eBay have exposed over one billion user passwords to offline password cracking attacks. Password hashing algorithms are a critical last line of defense against an offline attacker who has stolen password hash values from an authentication server. An attacker who has stolen a user's password hash value can attempt to crack each user's password offline by comparing the hashes of likely password guesses with the stolen hash value. Because the attacker can check each guess offline it is no longer possible to lockout the adversary after several incorrect guesses.

An offline attacker is limited only by the cost of computing the hash function. Ideally, the password hashing algorithm should be moderately expensive to compute so that it is prohibitively expensive for an offline attacker to crack most user passwords e.g., by checking millions, billions, or even trillions of password guesses for each user. It is perhaps encouraging that AshleyMadison, Dropbox, LastPass, and Yahoo! had adopted slow password hashing algorithms like BCRYPT and PBKDF2-SHA256 to discourage an offline attacker from cracking passwords. In the aftermath of these breaches, the claim that slow password hashing algorithms like BCRYPT [71] or PBKDF2 [72] are sufficient to protect most user passwords from offline attackers has been repeated frequently. For example, LastPass [73] claimed that "Cracking our algorithms [PBKDF2-SHA256] is extremely difficult, even for the strongest of computers." Security experts have made similar claims about BCRYPT e.g., after the Dropbox breach [74] a prominent security expert confidently stated that "all but the worst possible password choices are going to remain secure" because Dropbox had used the BCRYPT hashing algorithm.

Are these strong claims about the security of BCRYPT and PBKDF2 true? Despite all of their problems passwords remain prevalent and are likely to remain entrenched as the dominant form of authentication on the internet for years to come because they are easy to use and deploy, and users are already familiar with them [75, 76, 77]. It is therefore imperative to develop tools to quantify the damages of password breaches and provide guidance to organizations on how to store passwords. In this work we seek to address the following question:

> Can we quantitatively predict how many user passwords a rational attacker will crack after a breach?

3.1.1 Contributions

We first develop a new decision-theoretic framework to quantify the damage of an offline attack. Our model generalizes the Stackelberg game-theoretic model of Blocki and Datta [17]. A rational password attacker is economically motivated and will quit guessing once his marginal guessing costs exceed his marginal reward. The attacker's marginal reward is given by the probability p_i that the next (ith) password guess is correct times the value of an *additional* cracked password to the adversary e.g., the additional revenue of selling that password on the black market or the expected amount of additional money that could be extorted from this user. Given the average value v of each cracked password for the adversary, the cost k of computing the password hash function, and the probability distribution $p_1 > p_2 > \ldots$ over user-selected passwords, our model allows us to predict exactly how many passwords a rational adversary will crack. Unlike the model of Blocki and Datta [17] we can use our framework to model a setting in which the attacker encounters diminishing returns as we would expect in most (black)markets i.e., the total value of 500 million cracked passwords may be significantly less than 100 times the total value of 5 million passwords.

Second, we present the strongest evidence to date that Zipf's law models the distribution of user-selected passwords (with the possible exception of the tail of the distribution). These findings strongly support previous conclusions of Wang and Wang [78]. In particular, we show that Zipf's law closely fits the Yahoo! password frequency corpus. This dataset was collected by Bonneau [79] and later published by Blocki et al. [80]. In contrast to datasets from password breaches the Yahoo! dataset was collected by trusted parties and is representative of active Yahoo! users (researchers have observed that hacked datasets contain many passwords that appear to be fake [81]). Our sample size, 70 million users, is also more than twice as large as the datasets Wang and Wang[78] used to support their argument that Zipf's law closely models password datasets.

Third, we show that there is a finite threshold $T(.)$ which characterizes the behavior of a rational value v-adversary whenever the distribution over passwords follows Zipf's law. In particular, if the first cracked password has value $v \geq T(.) \times k$ then the adversary's optimal strategy is always to continue guessing until he cracks the user's password. The threshold $T(y, r, a)$ is parameterized by Zipf's law parameters y and r and a parameter a representing the rate of password value decay. We remark that, even if Zipf's law fails to model the tail of the password distribution, the threshold $T(y, r, a)$ still provides a useful characterization of the attacker's behavior. In particular, if $(1 - x)\%$ of passwords in a distribution follow Zip's law and the other $x\%$ follow some unknown (possibly uncrackable) distribution then our bounds imply that an attacker will compromise at least $(1 - x)\%$ of user passwords whenever $v \geq T(y, r, a) \times k$.

Fourth, we also derive model-independent upper and lower bounds on the fraction of passwords that a rational adversary would crack. While these bounds are slightly weaker than the bounds we can derive using Zipf's law these bounds do not require any modeling assumptions e.g., it is impossible to determine for sure whether or not Zipf's law fits the tail of the password distribution. Interestingly, the lower bounds we derive suggest that state-of-the-art password crackers [82] could still be improved substantially.

Fifth, we apply our framework to analyze recent large-scale password breaches including LastPass, AshleyMadison, Dropbox, and Yahoo! Our analysis strongly challenges the claim that BCRYPT and PBKDF2-SHA256 provide adequate protection for user passwords. If the password distribution follows Zipf's law then our analysis indicates that a rational attacker will almost certainly crack 100% of user passwords e.g., unless the value of Dropbox/LastPass/AshleyMadison/Yahoo! passwords is *significantly* less valuable than black market projections [83].

Finally, we derive *model independent* upper and lower bounds on the % of passwords cracked by a rational adversary. These bounds do not rely on the assumption that Zipf's law models the tail of the password distribution[1]. Nevertheless, our predictions are still quite dire e.g., a rational adversary will crack 51% of Yahoo! passwords *at minimum.* Our analysis indicates that to achieve sufficient levels of protection with BCRYPT or PBKDF2, it would be *necessary* to run these algorithms for well over a second on modern CPU which would constitute an unacceptable authentication delay in many contexts [84]. On a more positive note, our analysis suggests that the use of more modern password hashing techniques like memory-hard functions *can* provide strong protection against a rational password attacker *without* introducing inordinate delays for users during authentication. In particular, our analysis suggests that it could be possible to reduce the % of cracked passwords below 22.2% without increasing authentication delays to a full second.

3.1.2 Adversary Model

The Economics of offline password cracking focuses on a profit-motivated adversary playing the following Stackelberg game:

1. The defender (server) selects some key stretching parameter k and hash function H whose difficulty scales with $F(k)$.

---

[1] Wang and Wang [78] observed that the tails of empirical password datasets are not inconsistent with a Zipf's law distribution. However, we cannot be entirely confident that Zipf's law models the tail of the distribution since, by definition, we do not have many samples for passwords in the tail of the distribution.

2. The adversary selects some threshold B for the number of guesses they will make against each password
3. The adversary is given one user-generated password record from the server and wins if they correctly guess the password within B guesses.

We consider an adversary who is playing this game and seeks to maximize some utility function based on this guessing threshold B. To model this we set up a reward function $R(B)$, a cost function $C(B)$, and a utility function $U(B) = R(B) - C(B)$. The reward function is given according to some value of the password v as estimated by the adversary (and in the paper is based on estimates from [83]). We also assume that the adversary has access to the distribution of user-chosen passwords $\Pi = \{\pi_1, \pi_2, \ldots, \pi_w\}$. Given these parameters an adversary making B guesses would have the reward function:

$$R(B) = v \sum_{i=1}^{B} \pi_i$$

Cost is then defined as

$$C(B) = c \sum_{i=1}^{B} i\pi_i$$

where c is the cost to make a single guess. Combining these gives the adversaries expected profit function

$$U(B) = R(B) - C(B) = v \sum_{i=1}^{B} \pi_i - c \sum_{i=1}^{B} i\pi_i$$

An adversary wanting to maximize their profit will select some optimal guessing number B^* that maximizes P. Intuitively this happens at the point where the cost to make one extra guess exceeds the expected reward of making that guess, at which point the adversary stops. This base profit function may be modified to more accurately represent market conditions. During a breach a large number of cracked accounts may be available for sale, flooding the market which reduces the price. To model this supply/demand interaction we introduce a discounting factor $\alpha \in [0, 1]$ that reduces the value of passwords are more are sold. This causes the reward function to slowly taper off, modeling the way prices decrease when a market is flooded with products. The reward and profit functions are respectively modified as:

$$R(B) = v\left(\sum_{j=1}^{t} p_j\right)^{\alpha}$$

$$U(B) = v\left(\sum_{j=1}^{t} p_j\right)^{\alpha} - c\sum_{i=1}^{B} i\pi_i$$

3.1.3 Yahoo! Passwords follow Zipf's Law

Zipf's law states that the frequency of an element in a distribution is related to its rank in the distribution. There are two variants of Zipf's law for passwords: PDF-Zipf and CDF-Zipf. In the CDF-Zipf model we have $\lambda_t = \sum_{j=1}^{t} p_i = y \cdot t^r$, where the constants y and r are the CDF-Zipf parameters. In the PDF-Zipf model we have $f_i = \frac{C}{i^s}$, where s and C are the PDF-Zipf parameters. Normalizing by N the number of users we have $p_i = \frac{z}{i^s}$, where $z = \frac{C}{N}$.

Wang et al. [85] previously found that password frequencies tend to follow PDF-Zipf's law if the tail of the password distribution (e.g., passwords with frequency $f_i < 5$) is dropped. Wang and Wang [78] subsequently found that CDF-Zipf's model is superior in that the CDF-Zipf fits were more stable than PDF-Zipf fits and that the CDF-Zipf fit performed better under Kolmogorov-Smirnov (KS) tests. Furthermore, the CDF-Zipf model can fit the entire password distribution (e.g., without excluding passwords with frequency $f_i < 5$). These claims were based on analysis of several smaller password datasets ($N \leq 32.6$ million users) which were released by hackers.

In 2016 Yahoo! allowed the release of a differentially private list of password frequencies for users of their services [80]. We refer an interested reader to [79, 80] for additional details about how the Yahoo! data was collected and how it was perturbed to preserve differential privacy. The Yahoo! dataset is superior to other datasets in that it offers the largest sample size $N = 70$ million and the dataset was collected and released by trusted parties. We show that the Yahoo! dataset is also well modeled by CDF-Zipf's law. Our analysis comprises the strongest evidence to date of Wang and Wang's premise [78] that password distributions follow CDF-Zipf's law due to the advantages of the Yahoo! dataset. We focus on the CDF-Zipf's law model in this section since it can fit the entire password distribution [78]. We also verified that the Yahoo! dataset is also well modeled by PDF-Zipf's law if we drop passwords with frequency $f_i < 5$ like Wang et al. [85], but we omit this analysis from the submission due to lack of space.

On Ecological Validity

The Yahoo! frequency corpus offers many advantages over breached password datasets such as RockYou or Tianya.

- The Yahoo! password frequency corpus is based on 70 million Yahoo! passwords — more than twice as large as any of the breached datasets analyzed by Wang and Wang [78].
- The records were collected in a trusted fashion. No infiltration, hacking, tricks, or general foul play was used to obtain any of this data. There was no ulterior motive behind collecting these passwords other than to provide valuable data in a way that can be used for scientific research. By contrast, it is possible that hackers strategically omit (or inject) password data before they release a breached dataset like RockYou or Tianya! Why should we trust rogue hackers to provide researchers with representative password data?
- Breached password datasets often contain many passwords/ accounts that look suspiciously fake. In 2016 Yang et al [81] suggested that such passwords can be removed with DBSCAN [86]. Cleansing operations ended up removing a reasonable portion of the dataset (e.g., 5 million passwords were removed from RockYou's data). With the Yahoo! data such cleansing is not needed, as it was collected in a manner that ensured collected passwords were in use. Previous work that has been done on Zipf distributions in breached password datasets [78] did not perform any sort of sanitizing step on the data. It is unclear how such operations would affect the Zipf law fit.
- The information is released in a responsible way that preserves users' privacy. The differential privacy mechanism means that even with the released data it is not pos-

Table 3.1. Impact of Differential Privacy on CDF Fit

List Version	y	σ_y
RockYou Standard	0.0288	
RockYou Diff. Private	0.0302	$1.348 * 10^{-6}$
	r	σ_r
RockYou Standard	0.2108	
RockYou Diff. Private	0.2077	$2.94 * 10^{-6}$
	R^2	σ_{R^2}
RockYou Standard	0.9687	
RockYou Diff. Private	0.9681	$6.50 * 10^{-7}$

sible to determine any new information about Yahoo's users that an adversary would not be able to obtain anyways.

- Data from the Yahoo! password frequency corpus ultimately is derived from the passwords of active Yahoo! users who were logging in during the course of the study as opposed to passwords from throwaway accounts that have been long forgotten.

On the Impact of Differential Privacy on CDF-Zipf Fits

The published Yahoo! password frequency lists were perturbed to ensure differential privacy. Before attempting to fit this dataset using Zipf's law we seek to answer the following question: Does this noise, however small, affect our CDF-Zipf fitting process in any significant way? We claim that the answer is no, and we offer strong empirical evidence in support of this claim. In particular, we took the RockYou dataset ($N \approx 32.6$ million users) and generated 30 different perturbed versions of the frequency list by running the (ϵ, δ)-differentially private algorithm of Blocki et al. [80]. We set $\epsilon = 0.25$, the same value that was used to collect the Yahoo! dataset that we analyze. For each of these perturbed frequency lists, we compute a CDF-Zipf law fit using linear least squares regression. To apply Linear Least Squares regression we apply logarithms to the CDF-Zipf equation $\lambda_t = y \cdot t^r$ to obtain a linear equation $\log \lambda_t = \log y + r \log t$.

Our results, shown in Table 3.1, strongly suggest that the differential privacy mechanism does not impact the parameters y and r in a CDF-Zipf fitting in any significant way. In

Table 3.2. Yahoo! CDF-Zipf with Sub-sampling

Sample Size (Millions)	y	r	R^2
15	0.00949	0.2843	0.9542
30	0.01321	0.2544	0.9531
45	0.01592	0.2384	0.9529
60	0.01810	0.2277	0.9530
Full	0.02112	0.2166	0.9544

particular, the parameters y and r we obtain from fitting the original data with a CDF-Zipf model are virtually indistinguishable from the parameters we obtain by fitting on one of the perturbed datasets. Similarly, differential privacy does not affect the R^2 value of the CDF-Zipf fit. Here, R^2 measures how well the linear regression models the data (R^2 values closer to 1 indicate better fittings). Thus, one can compute CDF-Zipf's law parameters for the Yahoo! data collected by [80] and [79] without worrying about the impact of the (ϵ, δ)-differentially private algorithm used to perturb this dataset. We also verified that the noise added to the Yahoo! dataset will also have a negligible effect on the parameters s and z in a PDF-Zipf fitting.

Testing Stability of CDF-Zipf Fit via Subsampling

There are two primary ways to find a CDF-Zipf fit: Golden Section Search (GSS) and Linear Least Squares (LLS). Wang et al. [78] previously found that CDF-Zipf fits stabilize more quickly with GSS than with LLS. This was particularly important because the largest dataset they tested had size $\approx 3 \times 10^7$. In this section, we test the stability of LLS by subsampling from the much larger Yahoo! dataset. In particular, we subsample (without replacement) datasets of size 15 million, 30 million, 45 million, and 60 million and use LLS to compute the CDF-Zipf parameters y and r for each subsampled dataset. Our results are shown in table 3.2 graphically in Figure 3.1. While the CDF-Zipf fit returned by LLS does take longer to stabilize our results indicate that it does eventually stabilize at larger (sub)sample sizes (e.g., the Yahoo! dataset).

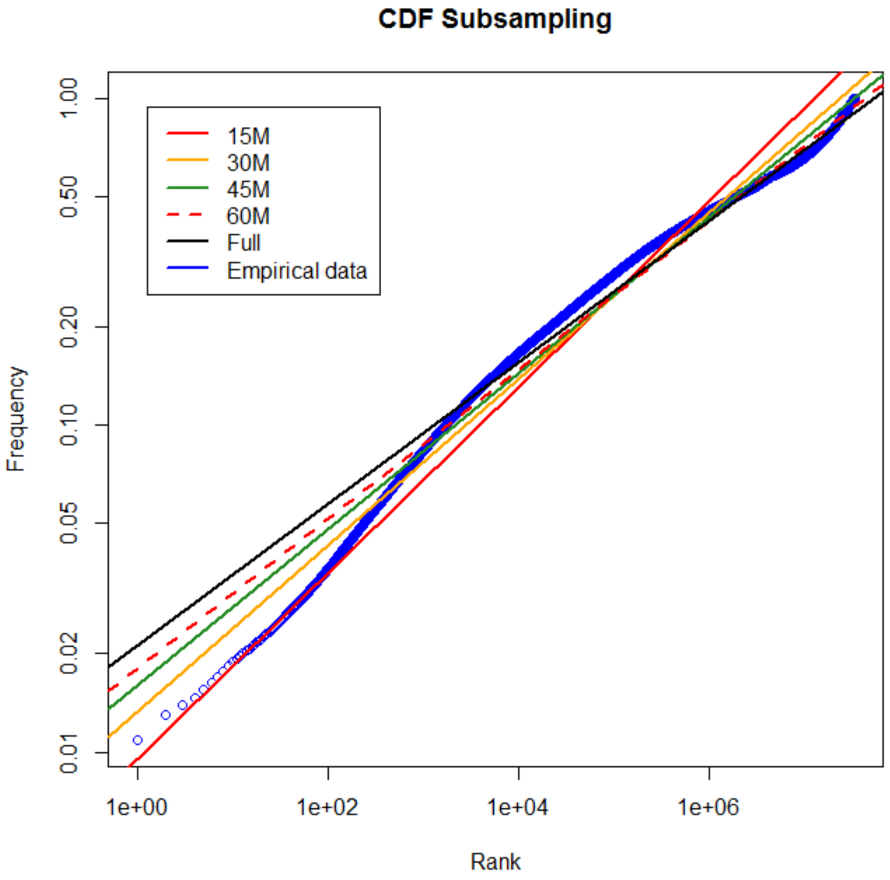

Figure 3.1. Yahoo! CDF-Zipf Subsampling

We also found that the PDF-Zipf parameters s and z stabilize before $N = 7 \times 10^7$ samples.

Fitting the Yahoo! data set with CDF-Zipf

We used both LLS regression and GSS to obtain separate CDF-Zipf fittings for the Yahoo! dataset. The results, shown in table 3.3 show that both methods produce high-quality fittings. In addition to the parameters y and r we report R^2 values and Kolmogorov-Smirnov (KS) distance. The KS test can be thought of as the largest distance between the observed discrete distribution $F_n(x)$ and the proposed theoretical distribution $F(x)$. Formally,

$$D_{KS} = sup\,|F_n(x) - F(x)|$$

Intuitively, smaller D_{KS} values (resp. larger R^2 values) indicates better fits.

Discussion

Both LLS and GSS produce high-quality CDF-Zipf fittings (e.g., $R^2 = 0.9544$) for the Yahoo! dataset. LLS regression outperforms the golden section search under both R^2 and Kolmogorov-Smirnov (KS) tests. Wang and Wang [78] had previously adopted golden section search because the results stabilized quickly. While this was most likely the right choice for smaller password datasets like RockYou, our analysis in the previous section suggests that LLS eventually produces stable solutions when the sample size is large (e.g., $N \geq 60$ million samples) as it is in the Yahoo! dataset. Thus, in the remainder of the paper, we use the CDF-Zipf parameters $y = 0.0211$ and $= 0.2166$ from LLS regression. We stress that the decision to use the CDF-Zipf parameters from LLS instead of the parameters returned by GSS does not affect our findings in any significant way.

We remark that LLS is also more efficient computationally. While we were able to run GSS to find a CDF-Zipf fit for the Yahoo! dataset ($N \approx 7 \times 10^7$), running GSS on a dataset of $N = 1$ billion passwords (e.g., the size of the most recent Yahoo! breach [87]) would be difficult if not intractable. By contrast, LLS could still be used to find a CDF-Zipf fitting and our analysis suggests that the fit would be superior.

Table 3.3. Yahoo! CDF-Zipf Test Results

Method	y	r	R^2	KS
LLS	0.0211	0.2166	0.9544	0.0094328
GSS	0.03315	0.1811	0.9498	0.022282

Table 3.4. CDF-Zipf threshold $T(y, r, a) < v/k$ at which adversary cracks 100% of passwords for $a \in \{1, 0.8\}$.

Dataset	y	r	$T(y, r, 1)$	$T(y, r, 0.8)$
RockYou	0.0374	0.1872	1.70×10^7	2.04×10^7
000webhost	0.0059	0.2816	3.67×10^7	4.27×10^7
Battlefield	0.0103	0.2949	2.37×10^6	2.77×10^6
Tianya	0.0622	0.1555	2.28×10^7	2.76×10^7
Dodonew	0.0194	0.2119	4.92×10^7	5.87×10^7
CSDN	0.0588	0.1486	7.63×10^7	9.24×10^7
Mail.ru	0.0252	0.2182	8.75×10^6	1.04×10^7
Gmail	0.0210	0.2257	1.14×10^7	1.36×10^7
Flirtlife.de	0.0346	0.2916	4.44×10^4	5.19×10^4
Yahoo!	0.0211	0.2166	2.25×10^7	2.69×10^7

3.1.4 Analysis of Rational Adversary Model for Zipf's Law

In this section, we show that there is a finite threshold $T(y, r, a)$ which characterizes the behavior of a rational offline adversary when user passwords follow CDF-Zipf's law with parameters y and r i.e., $\lambda_i = yi^r$. In particular, Theorem 3.1.1 gives a precise formula for computing this threshold $T(y, r, a)$[2]. If $v/k \geq T(y, r, a)$ then a rational value v adversary will proceed to crack all user passwords as marginal guessing rewards will *always* exceed marginal guessing costs for a rational attacker. In Table 3.4 we use this formula to explicitly compute $T(y, r, a)$ for the Yahoo! dataset as well as for nine other password datasets analyzed by Wang and Wang [78].

We note that we choose to focus on CDF-Zipf's law in this section as it is believed to be better than PDF-Zipf models. However, we stress that similar bounds can be derived using PDF-Zipf's law.

Theorem 3.1.1. *Let k denote the cost of attempting a password guess. If*

$$\frac{v}{k} \geq T(y, r, a) = \max_{t \leq Z} \left(\frac{1 - y(t-1)^r}{y^a (ra) t^{ra-1}} \right)$$

---

[2] We remark that when $a = 1$ it is possible to derive a closed-form expressing for the threshold $T(y, r, a)$.

where

$$Z = \left\lceil \left(\frac{1}{y}\right)^{1/r} \right\rceil + 1$$

then a value v *rational attacker will crack* 100% *of passwords chosen from a Zipf's law distribution with parameters* y *and* s.

Proof. Suppose a password frequency distribution follows Zipf's Law, for some parameters $0 < r < 1$ and y, so that $\lambda_n = yn^r$. Since the marginal revenue is $MR(t) = v(\lambda_t^a - \lambda_{t-1}^a)$ and the marginal cost is $MC(t) = k\left(1 - \sum_{n=1}^{t} p_n\right)$, a rational adversary can be assumed to continue attacking as long as $MR(t) \geq MC(t)$. Therefore, the attacker will not quit as long as

$$v(\lambda_t^a - \lambda_{t-1}^a) \geq k\left(1 - \sum_{n=1}^{t} p_n\right)$$
$$v(y^a t^{ra} - y^a(t-1)^{ra}) \geq k(1 - y(t-1)^r)$$

In particular, the attacker will not quit as long as

$$\frac{v}{k} \geq \frac{1 - y(t-1)^r}{y^a t^{ra} - y^a(t-1)^{ra}} \ .$$

Notably, if $\frac{v}{k} \geq \max_t \left(\frac{1-y(t-1)^r}{y^a t^{ra} - y^a(t-1)^{ra}}\right)$ for all t, then a rational adversary will eventually crack *all* passwords. For $g(t) := y^a(ra)t^{ra-1}$, we have $y^a t^{ra} - y^a(t-1)^{ra} = \int_{t-1}^{t} g(x)\,\mathrm{d}x$. Since $ra \leq 1$, then $g(t) \leq g(x) \leq g(t-1)$ for all $x \in [t-1, t]$. Thus we have $y^a(ra)t^{ra-1} \leq y^a t^{ra} - y^a(t-1)^{ra} \leq y^a(ra)(t-1)^{ra-1}$ and

$$\max_t \left(\frac{1 - y(t-1)^r}{y^a t^{ra} - y^a(t-1)^{ra}}\right) \leq \max_t \left(\frac{1 - y(t-1)^r}{y^a(ra)t^{ra-1}}\right) \ .$$

Thus, it suffices to prove that $\frac{v}{k} \geq \max_t f(t)$ where $f(t) := \left(\frac{1-y(t-1)^r}{y^a(ra)t^{ra-1}}\right)$. From the theorem statement we have $\frac{v}{k} \geq f(t)$ holds for any $t \leq Z$; it remains to argue that the same is true when $t > Z$. Since we already know that $f(Z) \leq v/k$, it suffices to show that the function $f(\cdot)$ is decreasing over $[Z, \infty)$ i.e., $f(t) \leq 0$ for all $t \geq Z$.

We calculate the derivative $f(t)$ as follows

$$\begin{aligned} f(t) = &-\frac{(t-1)^{r-1}t^{1-ra}y^{1-a}}{a} \\ &+ \frac{(1-ra)t^{-ra}y^{-a}(1-(t-1)^r y)}{ra}, \end{aligned}$$

so that $f(t) \leq 0$ if and only

$$\begin{aligned} \frac{(1-ra)y^{-a}(1-(t-1)^r y)}{rat^{ra}} &\leq \frac{y^{1-a}t(t-1)^{r-1}}{at^{ra}} \\ (1-ra)(1-(t-1)^r y) &\leq yt(t-1)^{r-1}r \\ (1-ra) &\leq y(t-1)^{r-1}((t-1)(1-ra)+tr). \end{aligned}$$

Since $(t-1)(1-ra) \leq (t-1)(1-ra)+tr$, then the last expression certainly holds true if $(1-ra) \leq y(t-1)^{r-1}(t-1)(1-ra)$ or equivalently, $\frac{1}{y} \leq (t-1)^r$. Since $Z := \left\lceil 1 + \left(\frac{1}{y}\right)^{1/r} \right\rceil$, it follows that $f(t) \leq 0$ for all $t \geq Z$. □

3.1.5 Analysis of Previous Password Breaches

In this section, we apply our economic model to analyze the consequences of recent password breaches and the impact of defenses that could have been adopted. While there have been hundreds of breaches [88] we select the 2014 Yahoo! breach [89], the 2016 Dropbox breach [74], the 2015 LastPass breach [13], and the 2015 AshleyMadison leak [90].

Estimating v

The value v represents the value per password when all passwords are released on the market. Thus, although the actual black market prices may vary with supply, the parameter v is fixed. Our estimate of this value parameter will depend on the current black market price, and model parameter a (diminishing returns). In Table 3.5 we show various estimates of v obtained from multiple estimates of black market password prices. These estimates include measurements from Fossi [83] and more recent estimates from [91], which finds that Yahoo! passwords go for 0.70-1.20 USD on the black market. To obtain the estimates in Table 3.5, we assume that the black market prices were observed when just 1% of the passwords were on the market. This allows us to estimate the value v if all passwords were to be released. We remark that the difference between the two estimates [91] and [83] may be explained due to additional black market supply. We view $a = 0.8$ as substantial diminishing returns e.g., the marginal revenue decreases by a factor of 1/3 when the attacker compromises all accounts. An interesting direction for future work may be to estimate the parameter a from a longitudinal study of black markets.

Translating between v and $v^{\$}$

Bonneau and Schechter [92] observed that in 2013, Bitcoin miners were able to perform approximately 2^{75} SHA-256 hashes in exchange for bitcoin rewards worth about $\$257M$. Correspondingly, one can estimate the cost of evaluating a SHA-256 hash to be approximately $C_H = \$7 \times 10^{-15}$. Alternatively, the cost can be viewed as the economic opportunity cost of evaluating each hash function (for instance, renting a botnet or computing on a cloud platform.) Because Bitcoin mining is almost exclusively performed on application-specific integrated circuits (ASICs) the above cost analysis implicitly assumes that the attacker is willing to fabricate an ASIC to evaluate PBKDF2-SHA256 or BCYRPT. We contend that this is a plausible scenario for a rational attacker since fabrication costs would amortize over the number of user accounts being attacked (e.g., 500+ million). Furthermore, we note that an attacker who is not willing to pay to fabricate an ASIC could obtain similar performance gains using a field-programmable gate array (FPGA).

3.1.6 Empirical Results

In section 3.1.4 we showed that, if passwords follow CDF-Zipf's law with parameters y and r, and $v/k \geq T(y, r, a)$ then a rational adversary will crack 100% of user passwords. Figure 3.3 plots $v = k \times T(y, r, 0.8)$ for various thresholds from Table 3.4 including Yahoo! and RockYou. Thus, for a point (v, τ) lying on the blue line, a value v rational adversary will crack 100% of Yahoo! passwords when he can compute the hash function at cost $k = \tau$. Note that $\tau = k$ for hash functions like BCRYPT and PBKDF2 — the ones used by Yahoo!,

Table 3.5. v conversion chart

$R(t_{1\%})$ (USD)	a = 0.8	a = 0.9	a = 1.0
0.70	0.28	0.44	0.70
1.20	0.48	0.76	1.20
4.00	1.59	2.52	4.00
30.00	11.94	18.93	30.00

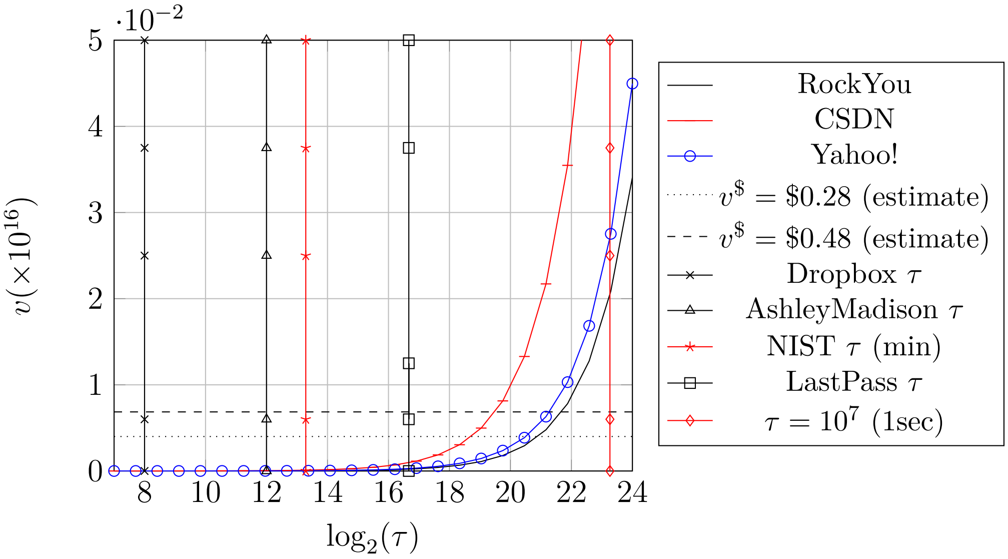

Figure 3.2. $v/k = T(y, r, 0.8)$ for RockYou, CSDN and Yahoo!

Dropbox, AshleyMadison, and LastPass. For reference, Figure 3.3 includes the actual values of τ selected by AshleyMadison, Dropbox and LastPass as well as the value $\tau = 10^7$. Bonneau and Schechter estimated that SHA256 can be evaluated 10^7 times in 1 second on a modern CPU [92]. Thus, 10^7 upper bounds the value of τ that one could select without delaying authentication for more than 1 second when using PBKDF2-SHA256.

The plots predict that, unless we set $\tau \gg 10^7$, the adversary will crack 100% of passwords in almost every instance. In particular, the levels of key-stretching performed by Dropbox, AshleyMadison and even Lastpass are all well below the thresholds necessary to protect Yahoo!, RockYou, or CSDN passwords.

While we do not have CDF-Zipf parameters for other breaches such as AshleyMadison, Dropbox, or LastPass, we do have the value $\tau = k$ for each of these breaches. Figure 3.5 plots $v = k \times T(y, r, 0.8)$ only this time we hold k constant and allow $T(y, r, 0.8)$ to vary.

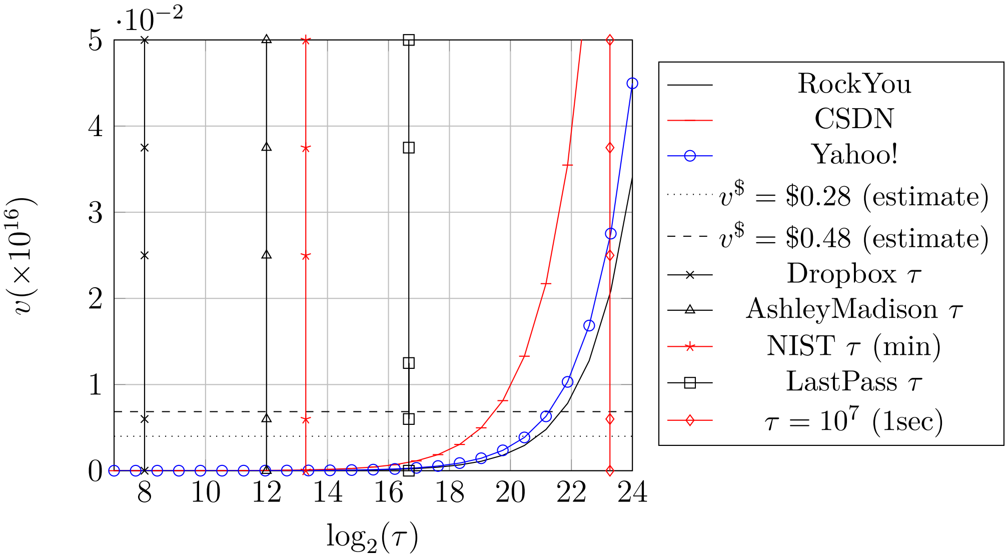

Figure 3.3. $v^{\$}$ vs. τ for $v = k \times T(y, r, 0.8)$

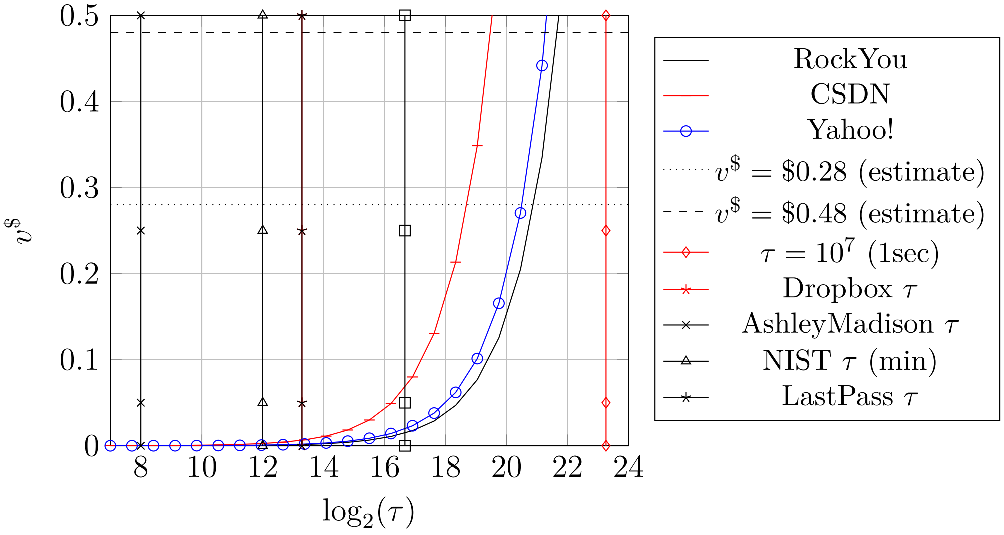

Figure 3.4. Minimum V at which all passwords are cracked by an economic adversary

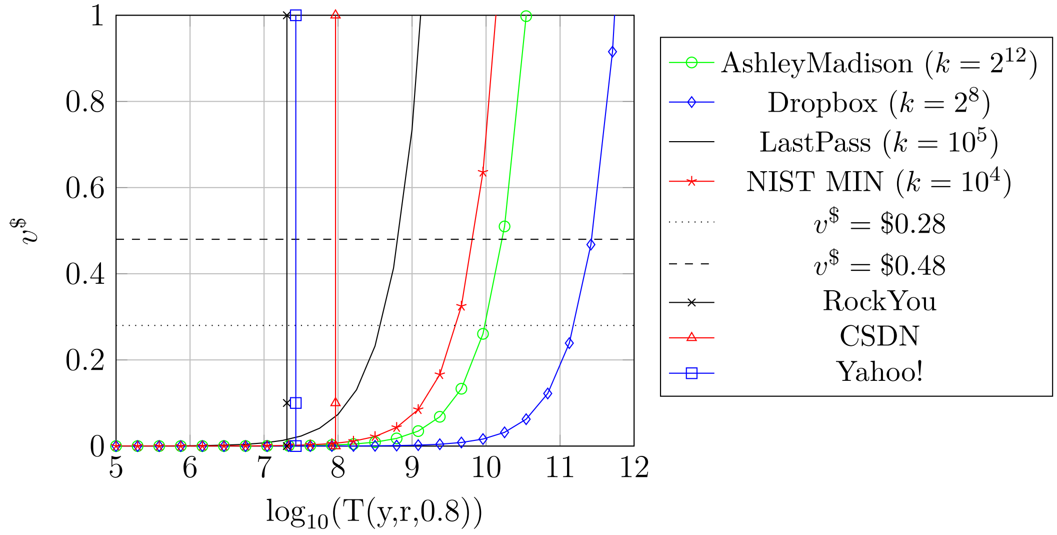

Figure 3.5. $v^{\$}$ versus $T(y, r, 0.8)$ when $v = k \times T(y, r, 0.8)$, at fixed values of k

For example, in the black line we fix $k = \tau = 10^5$ since LastPass used PBKDF2-SHA256 with $\tau = 10^5$ hash iterations and allow $T(y, r, 0.8)$ to vary. The vertical lines represent the thresholds $T(y, r, 0.8)$ we derive from CDF-Zipf's law fits for RockYou, Tianya, and Yahoo! Table 3.4 shows the value of $T(y, r, 0.8)$ obtained from 10 different password datasets. Observe that in all of cases we had $T(y, r, 0.8) \leq 7.64 \times 10^7$. As in Figure 3.4 the y-axis in Figure 3.5 is scaled to show the value $v^{\$}$ in USD (estimated). Thus, if Dropbox (resp. AshleyMadison/LastPass) passwords have comparable strength to Yahoo! passwords (resp. Tianya, RockYou) then a rational adversary would crack 100% of these passwords. Indeed, Figure 3.5 shows that unless the thresholds $T(y, r, a)$ for Dropbox/LastPass/AshleyMadison are significantly larger than the previously observed thresholds, a rational adversary would be compelled to crack all passwords, given the range of password values. For example, even if the threshold $T(y, r, a)$ for Dropbox exceeds the threshold for Yahoo! by four orders of magnitude then the adversary will still crack 100% of these passwords.

3.1.7 Discussion

Figures 3.3, 3.4 and 3.5 paint a grim picture. PBKDF2 and BCRYPT most likely provide dramatically insufficient protection for most AshleyMadison, Dropbox, Yahoo! and LastPass users — even if we used the lowest estimation of the value parameter v from Table 3.5 ($v^{\$} = 0.28$ USD) and we assume that the attacker faces substantial diminishing returns ($a = 0.8$) for additional cracked passwords. Furthermore, it would not have been possible to provide sufficient protection for users using PBKDF2 or BCRYPT without introducing intolerable authentication delays (≥ 1 second).

Our analysis assumes that the password distribution truly follows CDF-Zipf's law. While previous research (e.g., [85, 78] and our results in Section 3.1.3) strongly supports the hypothesis that *most* of the password distribution follows Zipf's law, it is not possible to definitively state that the tail of the password distribution does not follow Zipf's law since each of the passwords in the tail were (by definition) observed with low frequency. We stress that even if CDF-Zipf's law does not fit the tail of the password distribution that $T(y, r, a)$ still characterizes adversary behavior. For example, suppose that the $(100 - x)\%$ of passwords follow a

Zipf's law distribution with parameters y, r while $x\%$ of passwords in the tail of the password distribution do not. In this case, whenever $v/k \geq T(y, r, a)$ we a rational adversary will crack *at least* $(100 - x)\%$ of the user's passwords which follow Zipf's Law.

3.1.8 Memory Hard Functions

Given the bleak picture painted by this analysis, the natural question is how to remedy the problem. One of the strongest factors leading to this result is the absurdly cheap evaluation of a base hash function and the way costs scale linearly with the key stretching parameter when using hash iteration. To solve this issue we show that Memory-Hard Functions are capable of raising the v/k ratio enough to offer users substantially stronger protections in the same amount of time. Because Memory-Hard Functions introduce a memory read/write bottleneck into the computation an adversary loses much of the advantage they could have gained by using an ASIC, drastically increasing the cost per guess. Second, MHFs introduce a τ^2 factor into computation (in an honest evaluation). This further increases costs as τ increases. We find that when identical password distributions are present alongside identical values of τ MHFs offer much more significant protection against offline attacks. As shown in Figure 3.6 when a value is set at $v = \$4$ and about 1 second of computation we can protect $\approx 85\%$ of users who would have been cracked using the same parameters with hash iteration. This provides very strong evidence that MHFs should be required for password hashing, and is one of the official recommendations made in this paper [3].

3.1.9 Discussion

Our economic analysis decisively shows that traditional key-stretching tools like PBKDF2 and BCRYPT fail to provide adequate protection for user passwords, while memory-hard functions do provide meaningful protection against offline attackers. It is time for organizations to upgrade their password hashing algorithms and adopt modern key-stretching such as memory-hard functions [32, 38]. Alternatively, could a creative organization adapt customized Bitcoin mining rigs for use in password authentication? For example, the Antminer S9 [31], currently available on Amazon for approximately $3,000, is capable of computing

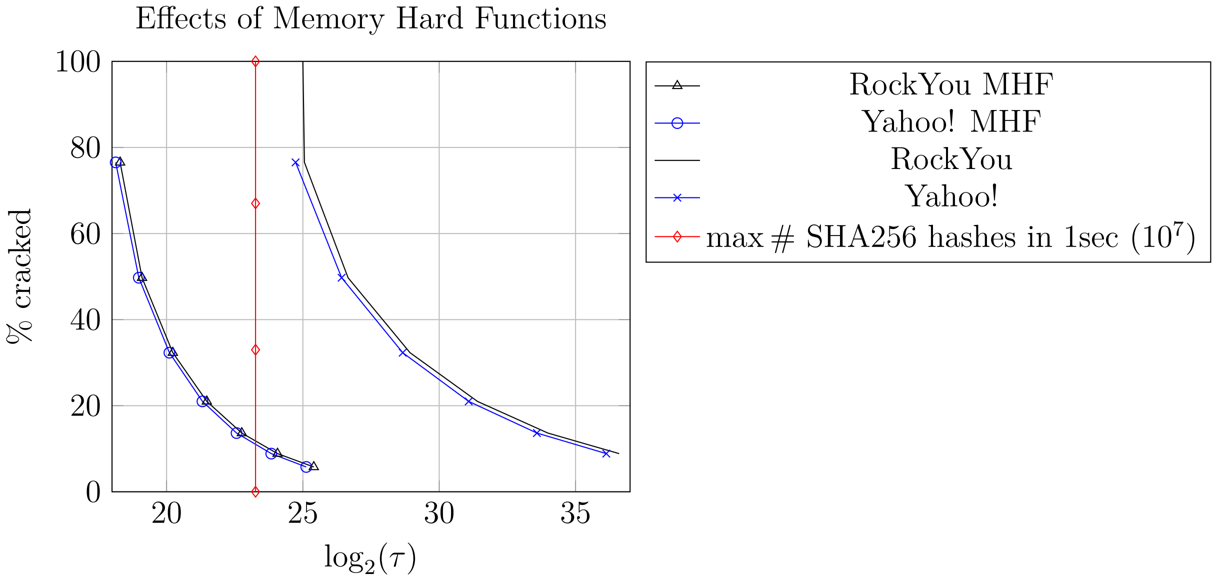

Figure 3.6. Memory Hard Functions: % cracked by value $v = \$4$ adversary against MHF with running time parameter τ.

SHA256 14 trillion times per second. If the organization stored salted and peppered [93, 17] password hash values $u, s_u, SHA256(pwd_u|s_u|p_u)$ then it could potentially use the Antminer S9, or a similar Bitcoin mining rig, to validate a password by quickly enumerating over a (very) large space of secret pepper values p (briefly, a secret salt value that is not stored which even an honest party must brute force).

While our analysis demonstrates that the use of memory-hard functions can significantly reduce the fraction of cracked passwords, the damage of an offline attack may still be significant. Thus, we recommend that organizations adopt distributed password hashing [94, 95, 96, 97] whenever feasible so that an attacker who only breaches one authentication server will not be able to mount an offline attack. Furthermore, we recommend that organizations take additional measures to mitigate the effect of an authentication server breach. Solutions might include mechanisms *detect* password breaches through the use of honey accounts or honey passwords[98], multi-factor authentication, and fraud detection/correction algorithms to prevent suspicious/harmful behavior [99].

While solid options for password hashing and key-derivation exist [32, 38, 16, 15] the reality is that many organizations and developers select suboptimal password hashing functions [100, 101]. Thus, there is a clear need to provide developers with clear guidance about selecting secure password hash functions. On a positive note, recent 2017 NIST guidelines do *suggest* the use of memory-hard functions. However, NIST guidelines still allow for the user of PBKDF2 with just $10,000$ hash iterations. Based on our analysis we advocate that password hashing standards should be updated to require the use of memory-hard functions for password hashing and disallow the use of non-memory hard functions such as BCRYPT or PBKDF2. It may be expedient for policy makers to audit and/or penalize organizations that fail to follow appropriate standards for password hashing.

We recommend that users primarily focus on selecting passwords that are strong enough to resist targeted online attacks [102] as there is often a vast gap between the required entropy to resist online and offline attacks [77]. Extra user effort to memorize a high entropy password might be completely wasted if an organization adopts poor password hashing algorithms like SHA1, MD5 [90] or the identity function [100]. This effort would likely be more productively spent on trying to reduce password reuse [103].

3.2 Quantifying the Damage of Password Length Leakage

In any efficient encryption scheme there is necessarily some relationship between plaintext length and ciphertext length e.g., consider encrypting a 2MB jpeg image vs encrypting a 2GB mp4 movie. Vincent Guido [104] observed that (unpadded) SSL traffic can leak information about password lengths, and introduced the name "bicycle attack" in reference to the fact that a gift-wrapped bicycle still looks like a bicycle. Thus, whenever plaintext-length might be viewed as a sensitive attribute it is recommended that an application developer should pad the plaintext message before encryption [105]. For example, the RFC for TLS 1.2 includes the following caveat:

> Note in particular that the type and length of a record are not protected by encryption. If this information is itself sensitive, application designers may wish to take steps (padding, cover traffic) to minimize information leakage.

Authenticated Encryption with Associated Data (AEAD) ciphers simultaneously guarantees both message integrity and confidentiality. A recent longitudinal study of TLS Deployment found a dramatic rise in the percentage of TLS connections using AEAD cipher such as AES128-GCM, AES256-GCM, and ChaCha20-Poly1305 since 2013 [106] i.e., roughly 80% of TLS connections used either AES128-GCM or AES256-GCM in April, 2018. In all three AEAD schemes there is a one to one relationship between the length of some ciphertext and the length of the original plain text message. This 1-1 relationship can be viewed as a feature of the cipher as it allows for significantly shorter ciphertexts i.e., a 2 byte message would not need to be padded to a 16-byte (AES128-GCM) or 32-byte message (AES256-GCM) before encryption. However, the 1-1 relationship means that an eavesdropping attacker to infer the *exact* length of each transmitted message from the intercepted ciphertext. The responsibility of identifying cases where the length of a plaintext message is potentially sensitive and ensuring that such messages are appropriately padded is left to the application developer.[3]

---

[3] For example, the RFC for TLS 1.3 [107] explicitly says "Selecting a padding policy that suggests when and how much to pad is a complex topic and is beyond the scope of this specification."

3.2.1 Attacker Model

We consider three types of attackers, their capabilities, and motivations to compromise an account. This will help us create a threat model before gaining information leaked from GCM. We then can compare any increases in capability or willingness to understand the severity of the information leakage.

- **Hacker** The first attacker is a hacker and is capable of controlling local networks, either on the Sender or Recipient side of the connection. The hackers are motivated by the challenge of compromising an account and does not look to directly profit from the hack. The hacker's willingness to compromise an account waivers in the face of systems with multiple security controls or a monetary cost of more than 0-100 (USD), because the account's perceived value to the hacker's ego does not exceed its cost of 100.

- **Criminal** The second attacker is a criminal with the capability to control local networks and several organization-wide networks. The criminal is financially motivated to only compromise accounts whereby the payoff is at least ten times the cost [108]. In most cases, the criminal will spend between 100-1,000 (USD) to compromise an individual account, because its perceived value to financial profit does not exceed its cost of 1,000 (USD).

- **Nation-State** The third attacker is a nation-state. The nation-state can compromise local and organizational networks and also control large segments of the Internet such as Internet Service Providers. The nation-state is primarily motivated by understanding threats to its citizens or interests and is only willing to conduct multi-year campaigns against accounts threatening its security or sovereignty. In most cases, we assume the nation-state is willing to spend between 1000-10,000 (USD) to compromise an individual account because the perceived value in the information gained from the compromised account does not exceed its cost of 10,000. While we do not know the targeted account's subjective worth, we assume a rational adversary is not willing to pay more than its perceived value.

A nation-state (and perhaps a sophisticated criminal) might have the capability to eavesdrop on network traffic on a broader scale to obtain (username, password length) pairs at scale e.g., using a DNS Hijacking attack. Similarly, a nation-state (and perhaps a sophisticated criminal) might have the ability to coerce an internet service provider to reveal logs associating specific users with the different IP addresses they were assigned over time.

Password spray campaigns typically target single sign-on (SSO) and cloud-based applications using federated authentication. For example - in February 2018, the United States Department of Justice indicted nine Iranian nationals a part of the Mabna Institute for computer intrusions using password spraying [109]. While nation-states might be intrinsically motivated to conduct persistent and long-term campaigns, would an extrinsically motivated attacker being as willing to use a password-spraying technique? To help answer this question, we conduct an economic analysis. If the information leaked doubles an attacker's capability or willingness to conduct the attack, then we consider the information leak to be severe, because our data suggests doubling the adversarial advantage begins to show a clear change in the related monetary, decision-making, and temporal characteristics. We want to create a threat model whereby the attackers have a spectrum of capabilities and willingness to compromise targets of interest. Examples of severe information leakage might include the compromise of the session key, allowing an attacker to understand the underlying plain text without decryption, or increasing the capability beyond their stated initial set of capabilities to bypass a security control.

3.2.2 Security and Privacy Impact of Password Length Leakage

In this section, we aim to quantify the damages of password length leakage. In our analysis, we suppose that an online password attacker attempts to crack the user's password by repeatedly attempting the most popular passwords from a dictionary. We assume that the authentication server uses secure CAPTCHAs to rate limit the attacker (e.g., Gmail authentication). Thus, an attacker must pay human workers to solve a CAPTCHA after each incorrect guess. If the attacker knows the length of the password in advance then the attacker can eliminate passwords from the dictionary.

We aim to answer the following questions: (1) How many additional passwords will an online attacker crack when given the length of each password? (2) How much does password length leakage monetarily benefit the attacker? We stress that questions (1) and (2) ask very different questions. The answer to the first question tells us how many additional user accounts will be compromised if password lengths are revealed to a rational attacker. The answer to the second question allows us to predict whether or not the cost of eavesdropping (equipment, manpower) on network traffic outweighs the benefit of learning password lengths. To address these questions, we adapt a game-theoretic model introduced by Blocki and Datta [110] to model an offline password attacker.

Constrained Attacker

Before introducing our decision-theoretic model, we first consider a constrained online password attacker who either gives up or gets locked out after B incorrect guesses. This will give us the chance to introduce key notation.

Notation Let $\mathcal{D}$ denote the distribution over user selected passwords $\mathcal{P}$ and we let $p_i = \Pr_{pwd \leftarrow \mathcal{D}}\left[pwd = pwd_i\right]$ denote the probability that a user selects the i'th most popular password $pwd_i \in \mathcal{P}$ e.g. ,$p_i \geq p_{i+1}$. We also $\mathcal{P}^\ell = \{x \in \mathcal{P} : |x| = \ell\} \subseteq \mathcal{P}$ denote the set of all passwords with length ℓ and

$$p_i^\ell = \Pr_{pwd \leftarrow \mathcal{D}}\left[pwd = pwd_{i_\ell} \,\middle|\, pwd \in \mathcal{P}^\ell\right]$$

where i_ℓ is the index of the i'th most popular password in the set $\mathcal{P}^\ell$. Observe that we have $p_1^\ell \geq p_2^\ell \geq \ldots$ for each $\ell \geq 1$. For notational convenience we also write $\Pr\left[\mathcal{P}^\ell\right] = \Pr_{pwd \leftarrow \mathcal{D}}\left[pwd \in \mathcal{P}^\ell\right]$.

Experiment 1: Unknown Lengths with B Guesses.

A user selects a random password $pwd \leftarrow \mathcal{D}$ and an attacker attempts to guess the password online. We assume an attacker who knows the distribution $\mathcal{D}$, but not the specific password and that the attacker either gives up or gets locked out after B guesses. The attacker best strategy is to try the B most likely guesses in the distribution $pwd_1, \ldots, pwd_B$. The attacker will succeed with probability $\lambda_B = \sum_{i=1}^{B} p_i$. Conditioning on the event that the user's password has length ℓ (i.e., $pwd \in \mathcal{P}^\ell$) and that the length ℓ is *unknown* to the attacker, the attacker will succeed with probability

$$\lambda_{B,\ell} = \Pr_{pwd \leftarrow \mathcal{D}}\left[pwd \in \bigcup_{i=1}^{B} \{pwd_i\} \;\middle|\; pwd \in \mathcal{P}^\ell \right] .$$

Experiment 2: Known Lengths with B Guesses

This experiment is exactly like experiment one except that after we sample $pwd \leftarrow \mathcal{D}$ the length $|pwd|$ of the password is revealed to the attacker. If the attacker knows the password length is ℓ (i.e., $pwd \in \mathcal{P}^\ell$), then the attacker succeeds with probability

$$\lambda^*_{B,\ell} = \sum_{i=1}^{B} p_i^\ell .$$

If the attacker attempts the B most likely guesses before giving up then he will succeed with probability

$$\lambda^*_B = \sum_{\ell} \Pr\left[\mathcal{P}^\ell\right] \lambda^*_{B,\ell} .$$

Analysis

Table 3.6 compares the success rate of the attacker with (λ^*_B) and without (λ_B) knowledge of the passwords length for various guessing limits B. The results show that the attacker's success rate increases significantly when the password length is known, e.g., a criminal attempting $B = 10^5$ guesses per user using the LinkedIn distribution would crack nearly 35% of passwords with knowledge of password length compared to 24% without this knowledge, or about 50%. Table 3.7 compares the attacker's conditional success rate with ($\lambda^*_{B,\ell}$) and without ($\lambda_{B,\ell}$) knowledge of the passwords length conditioning on the event that the user's password has length ℓ[4]. Surprisingly, even for longer lengths such as $\ell = 30$ an attacker

---

[4] Table 3.7 only shows information from the Rockyou leak. This is because the calculation of $\lambda_{B,\ell}$ requires a list showing the exact order of passwords as well as their frequencies. It is not enough to know something

still may have a reasonably high success rate with a smaller guessing limit B. For example, $\lambda^*_{B,\ell} \approx 4.5\%$ at just $B = 10$ guesses against the Rockyou list when password length is a large as $\ell = 30$.[5] By contrast, when the attacker does not know the length we have $\lambda_{B,\ell} = 0$ when $\ell = 30$ even if the attacker tries up to $B = 288,046$ guesses! This is because there are $288,046$ passwords with length $\ell \neq 30$ that are more popular than the most popular password of length 30.

Limitations of the LinkedIn and Rockyou Datasets.

A general limitation of empirically defined password distributions is that they almost certainly overestimate the probability of passwords at the tail of the distributions, e.g., for *any* password pwd_i that was observed once in the Linkedin dataset we estimate that p_i is at most $p_i = 1/N \geq 5.73 \times 10^{-9}$. The Linkedin dataset contains 1.7×10^8 unique passwords and about 2.1×10^7 of these passwords are observed exactly one time. Unfortunately, there is no way to be confident about the true probability of an event that has only been observed once. Thus, for $B > 1.5 \times 10^7$ our estimate of λ_B may be too high, and for $B > 3.1 \times 10^6$ our estimate for $\lambda^*_{B,6}$ may be too high (there are about 3.1×10^6 length six passwords that were observed more than once so our estimate of p^6_i may be too high for $i > 3.1 \times 10^6$). When $B \leq 10^6$ we believe the estimate for λ^*_B is reasonable[6]. This means that there is some uncertainty about our estimates of λ^*_B for a nation-state attacker ($B \in [10^6, 10^7]$). In our analysis, we use the symbol ⚠to indicate that it is affected by uncertainty about the tail of the password distribution.

As part of our analysis, we used empirical data from the RockYou dataset [12], released in 2009, to define our password distribution. The dataset was released in 2009 by hackers and

---

like “there are 5 passwords picked with frequency 100". While you could identify which lengths made up those 5, we do not have the specific order. It is possible to construct an estimated list by arbitrarily ordering those 5 passwords, but this would produce a noisy estimate. Thus we do not have accurate values of $\lambda_{B,\ell}$ for the LinkedIn data

[5] Of the $1,052$ users in the RockYou dataset with length $\ell = 30$ passwords eight users selected “bebelicouz_05_mistme@yahoo.com” and seven users selected “111111111111111111111111111111.” Another popular password of length $\ell = 30$ was the URL http://www.rockyou.com/tos.php — selected by five users.

[6] For each $\ell \in [6, 10]$ we have more than 1.6×10^6 (2×10^5) passwords of that length that were observed multiple times. These lengths account for $\sum_{\ell=6}^{10} \Pr\left[\mathcal{P}^\ell\right] \approx 81.4\%(86.5)\%$ of all passwords in the Linkedin (Rockyou) dataset. If we include $\ell = 5, 11$ then we have $\sum_{\ell=5}^{11} \Pr\left[\mathcal{P}^\ell\right] \geq 85.9\%(94.1\%)$ and for both lengths $\ell = 5, 11$ we have over 7.8×10^6 (8.75×10^4) passwords of that length that were observed multiple times.

Table 3.6. Attacker Success Rate at Various Guessing Limits with and without knowledge of the password length (LinkedIn)

LinkedIn						
Adversary type	Hacker		Criminal		Nation-state	
Guess limit B	10^2	10^3	10^4	10^5	10^6	10^7 ⚠
λ_B^*	0.058	0.119	0.214	0.352	0.571	0.973
$\lambda_B^* - \lambda_B$	0.030	0.048	0.072	0.112	0.195	0.394
λ_B^*/λ_B	2.074	1.672	1.505	1.466	1.517	1.682
Rockyou						
Adversary type	Hacker		Criminal		Nation-state ⚠	
Guess limit B	10^2	10^3	10^4	10^5	10^6	10^7
λ_B^*	0.089	0.192	0.330	0.519	0.796	1.000
$\lambda_B^* - \lambda_B$	0.043	0.080	0.107	0.153	0.255	0.133
λ_B^*/λ_B	1.938	1.712	1.479	1.418	1.471	1.154

Table 3.7. Attacker Conditional Success Rate at Various Guessing Limits with and without knowledge of the password length

Lengths	$\lambda_{B,\ell}^*$			$\lambda_{B,\ell}^* - \lambda_{B,\ell}$			$\lambda_{B,\ell}^*/\lambda_{B,\ell}$		
limit	10^2	10^4	10^6 ⚠	10^2	10^4	10^6 ⚠	10^2	10^4	10^6 ⚠
5	0.228	0.670	1.000	0.183	0.447	0.458	5.067	3.004	1.845
6	0.116	0.430	0.888	0.071	0.207	0.346	2.578	1.928	1.638
7	0.077	0.353	0.760	0.032	0.130	0.218	1.711	1.583	1.402
8	0.080	0.281	0.698	0.035	0.058	0.156	1.778	1.260	1.288
9	0.083	0.262	0.698	0.038	0.039	0.156	1.844	1.175	1.288
λ_B	0.045	0.223	0.542						

remains one of the largest available **plaintext** password datasets. One potential downside is that many RockYou users may have viewed their account as low-value. While Bonneau [79] found that account value did not appear to be correlated with password strength in his analysis of Yahoo! passwords, we cannot rule out the possibility that RockYou users were less motivated to pick strong passwords because the account had low-value. However, we remark that it is possible that a stronger password distribution would result in an even bigger advantage $\lambda_B^* - \lambda_B$ for an attacker who learns the password length since both λ_B^* and λ_B would decrease.

3.2.3 Decision Theoretic Model

Experiments 1 and 2 consider an attacker that gives up after a fixed number B of incorrect guesses. While this model may be appropriate in some scenarios where the attacker is eventually locked out, it does not model scenarios in which guessing is throttled using CAPTCHA puzzles (e.g., Gmail). This approach has the advantage in that legitimate users will never be locked out (at worst they will be bothered to solve a CAPTCHA puzzle). In this section, we model a rational online attacker who will select a threshold B^{opt} which maximizes his expected gain (expected reward minus expected guessing costs). In other words, the attacker will continue attacking as long as marginal reward exceeds marginal guessing costs.

Marginal Guessing Reward

Suppose that a rational attacker has value v for a cracked password. The attacker's expected reward is v times the probability he successfully cracks the password. If, as in experiment 1 (resp. experiment 2), the attacker doesn't (resp. does) know the password length ℓ then the expected reward after B guesses is $R(v,B) \doteq v\lambda_B$ (resp. $R^{\ell}(v,B) \doteq v\lambda^{*}_{B,\ell}$). The marginal reward of one more guess when the attacker doesn't (resp. does) know the password length is $MR(v,B) = R(v,B+1) - R(v,B) = vp_{B+1}$ (resp. $MR^{\ell}(v,b) = R^{\ell}(v,B+1) - R^{\ell}(v,B) = vp^{\ell}_{B+1}$).

Marginal Guessing Costs

Assume that the cost of each additional password guess is k (e.g., the amortized cost of paying a human to solve one more CAPTCHA puzzle). If, as in experiment 1 (resp. experiment 2), the attacker doesn't (resp. does) know the password length ℓ then the expected guessing cost is

$$C(k,B) = (1-\lambda_B)Bk + k\sum_{i=1}^{B} i \times p_i \ ,$$

or if password length is known

$$C^{\ell}(k,B) = (1-\lambda^*_{B,\ell})Bk + k\sum_{i=1}^{B} i \times p_i^{\ell}.$$

To understand this formula, we first observe that the attacker incurs maximum guessing cost Bk when he fails to crack the password, which happens with probability $1-\lambda_B$ (resp. $1-\lambda^*_{B,\ell}$) when the attacker is not told (resp. is told) the password length ℓ. If the attacker is successful on guess $i < B$ then the attacker only incurs cost ik and this happens with probability p_i (resp. p_i^{ℓ}) when the attacker is not told (resp. is told) the password length ℓ.

Attacker Gain

We $G(v,k,B) \doteq R(v,B) - C(k,B)$ (resp. $G^{\ell}(v,k,B) \doteq R^{\ell}(v,B) - C^{\ell}(k,B)$) to denote the attackers expected gain (guessing reward minus guessing cost) when the attacker is not told (resp. is told) the password length ℓ. If the attacker is rational the attacker will select a guessing threshold B which maximizes his gain. We use $B^{opt}_{v,k} \doteq \arg\max_B G(v,k,B)$ resp. $B^{opt}_{v,k,\ell} \doteq \arg\max_B G^{\ell}(v,k,B)$ to denote the attackers optimal guessing threshold when not told (resp. is told) the password length ℓ. We use $G(v,k) \doteq G(v,k,B^{opt}_{v,k})$ denotes the expected gain of a rational attacker in experiment 1. Finally, we use

$$G^*(v,k) \doteq \sum_{\ell} \Pr\left[\mathcal{P}^{\ell}\right] G^{\ell}(v,k,B^{opt}_{v,k,\ell})$$

the expected gain of a rational attacker in experiment 2.

Attacker's Monetary Benefit when Learning Password Lengths: We remark that $G^*(v,k) - G(v,k)$ denotes the expected (per user) benefit to the online attacker when learning the password length ℓ. If this benefit $\#(\text{AttackedUsers} \times (G^*(v,k) - G(v,k))$ exceeds the cost of eavesdropping on network traffic then it will be worthwhile for the attacker to exploit the password length-leakage attacks described earlier. Table 3.8 plots the value $G^*(v,k) - G(v,k)$ for different v/k ratios. Tables showing the optimal thresholds $B^{OPT}_{\dots}$ which maximize gains at various v/k ratios can be found in Table 3.9. From Table 3.8 we can see that the monetary benefit of learning password length can be quite substantial. The

Table 3.8. Attacker gains for various account value to marginal guessing cost ratios

LinkedIn						
Adversary type	Hacker		Criminal		Nation-state	
v/k ratio	10^2	10^3	10^4	10^5	10^6	10^7 ⚠
G^*/k	0.511	17.89	605.63	14652	296826	8024681
G/k	0.000	5.803	187.67	7448.19	165335	3085617
$(G^* - G)/k$	0.511	12.097	417.96	7203.81	131491	4939064
G^*/G	∞	3.083	3.227	1.967	1.795	2.600
Rockyou						
Adversary type	Hacker		Criminal		Nation-state ⚠	
v/k ratio	10^2	10^3	10^4	10^5	10^6	10^7
G^*/k	0.959	30.168	1225.5	27053	552447	9.5E6
G/k	0.000	12.141	428.182	14576	297611	6.7E6
$(G^* - G)/k$	0.959	18.026	797.31	23577	254836	2.8E6
G^*/G	∞	2.485	2.862	1.856	1.856	1.422

Table 3.9. Optimal Attacker Guessing Limit for various account value to marginal guessing cost ratios

LinkedIn						
Adversary type	Hacker		Criminal		Nation-state	
v/k ratio	10^2	10^3	10^4	10^5	10^6	10^7 ⚠
$B_{v,k,6}^{OPT}$	1	10	736	10327	271903	5418647
$B_{v,k,7}^{OPT}$	0	4	570	8818	134534	6672366
$B_{v,k,8}^{OPT}$	0	4	199	4771	107888	14886616
$B_{v,k,9}^{OPT}$	1	2	168	4164	81901	8750881
$B_{v,k}^{OPT}$	0	3	164	3537	58638	1063939
Rockyou						
Adversary type	Hacker		Criminal		Nation-state ⚠	
v/k ratio	10^2	10^3	10^4	10^5	10^6	10^7
$B_{v,k,5}^{OPT}$	1	121	1928	259169	259169	259169
$B_{v,k,6}^{OPT}$	1	30	981	20963	1947797	1947797
$B_{v,k,7}^{OPT}$	0	18	759	11077	227299	2506271
$B_{v,k,8}^{OPT}$	0	15	417	7919	168816	2966037
$B_{v,k,9}^{OPT}$	1	20	366	6708	2191039	2191039
$B_{v,k}^{OPT}$	0	7	320	6327	114760	14344391

Table 3.10. Advantages for several guessing limits

LinkedIn						
Adversary type	Hacker		Criminal		Nation-state	
Guess limit	10^2	10^3	10^4	10^5	10^6	10^7 ⚠
$\overline{\lambda}^*_{v,k}$	0.007	0.028	0.090	0.191	0.378	1.000
$\overline{\lambda}^*_{v,k} - \overline{\lambda}_{v,k}$	0.000	0.019	0.055	0.084	0.164	0.619
$\overline{\lambda}^*_{v,k}/\overline{\lambda}_{v,k}$	∞	3.182	2.586	1.785	1.766	2.625
Rockyou						
Adversary type	Hacker		Criminal		Nation-state ⚠	
Guess limit	10^2	10^3	10^4	10^5	10^6	10^7
$\overline{\lambda}^*_{v,k}$	0.014	0.052	0.177	0.363	0.823	1.000
$\overline{\lambda}^*_{v,k} - \overline{\lambda}_{v,k}$	0.014	0.034	0.104	0.163	0.446	0.000
$\overline{\lambda}^*_{v,k}/\overline{\lambda}_{v,k}$	∞	2.874	2.417	1.817	2.183	1.000

Table 3.11. Cracking estimates for attacks over time (LinkedIn)

LinkedIn			
Days	1 guess/day	10 guesses/day	100 guesses/day (LinkedIn)
30	0.04(0.02)	0.08(0.05)	0.16(0.10)
90	0.06(0.03)	0.11(0.07)	0.21(0.14)
180	0.7(0.04)	0.14(0.09)	0.24(0.16)
360	0.9(0.5)	0.17(0.11)	0.28(0.19)
Rockyou			
Days	1 guess/day	10 guesses/day	100 guesses/day (Rockyou)
30	0.06(0.03)	0.13(0.07)	0.26(0.16)
90	0.09(0.04)	0.19(0.11)	0.32(0.22)
180	0.11(0.06)	0.23(0.14)	0.37(0.26)
360	0.14(0.08)	0.27(0.17)	0.43(0.30)

increased monetary benefit might entice additional criminals to entire the password cracking game provided that increased monetary benefit outweighs the cost of eavesdropping and linking IP addresses to user ids.

Number of Compromised User Accounts: We now seek to quantify the damages of leaking password lengths to a rational attacker. In particular, we use $\overline{\lambda}_{v,k} = \lambda_{B^{opt}_{v,k}}$ (resp. $\overline{\lambda}^*_{v,k,\ell} = \lambda^*_{B^{opt}_{v,k,\ell},\ell}$) to denote the probability a rational value v attacker succeeds without (resp. with) knowledge of the password length ℓ and given guessing costs k. We use $\overline{\lambda}^*_{v,k} = \sum_\ell \Pr\left[\mathcal{P}^\ell\right]\overline{\lambda}^*_{v,k,\ell}$ to denote the probability that a password is cracked in experiment 2 (We will also use $\overline{\lambda}_{v,k,\ell} = \lambda_{B^{opt}_{v,k},\ell}$ to denote the probability that an attacker cracks the user's password without knowledge of password length conditioning on the event that the user's password has length ℓ.). Finally, we note that $\overline{\lambda}^*_{v,k} - \overline{\lambda}_{v,k}$ denotes the increase in the attacker's success rate when the attacker learns the password length.

Table 3.7 compares the attackers success rate with $\left(\overline{\lambda}^*_{v,k}\right)$ and without $\left(\overline{\lambda}_{v,k,\ell}\right)$ knowledge of the password length for various v/k ratios, and Table 3.8 compares the attacker's success rate conditioning on the event that the the password has length ℓ (Table 3.9 shows the optimal thresholds $B^{OPT}_{v,k}$ as well as $B^{OPT}_{v,k,\ell}$ for various lengths ℓ). These tables show that a rational attacker will crack many more passwords when given the password length. For example, a criminal attacker with $v/k = 10^5$ who knows the password length will crack over 19% of targets from the Linkedin data compared to just 10.7% of targets without knowledge of password length. As another example consider a hacker using the Rockyou data with $v/k = 10^2$. If the attacker does not know the password length then his optimal strategy is to give up immediately without attempting any password guesses. However, if the attacker does know the length then he will crack about 1.4% of passwords. We once again wish to stress the ⚠ symbol towards higher v/k ratios, which denotes situations where overestimates are likely - especially where values show the adversary would guess 100% of passwords. We remark that if the price k of a CAPTCHA solving services increase (resp. decrease) by an order of magnitude then the value to cost ratio v/k will also increase (resp. decrease) correspondingly and we can still refer to Table 3.9 to infer how many passwords a rational attacker will crack with and without knowledge of the password length. .

Online time-delayed attacks The models we have introduced are based on the notion that an attacker can continuously try passwords. However, in many situations, there is some sort of lockout that limits the number of attempts that can be made. In this case, the adversary can run an attack over time to bypass lockout mechanisms. Rather than being rate limited by a service like CAPTCHA, a set number of guesses may be run per day. The models introduced also provide insight into these types of attacks. To provide an idea of what sort of advantage an adversary may have in this case, we take Brostoff and Sasse's recommendations that 10 attempts should be allowed [111], however, we note that Bonneau and Preibusch found that the vast majority of sites they surveyed allowed over 100 guesses with no restrictions [100]. In addition, the National Institute of Standards and Technology recommendations allow for no more than 100 consecutive failed login attempts before a lockout [35]. Table 3.11 shows the estimated proportion of passwords that would be cracked over set periods given 1, 10, and 100 daily guesses.

3.2.4 Solutions

Since QUIC's adoption is widespread, there are many stakeholders and systems with different interests to balance usability, performance, and security. As such, we believe this defies an easy fix in the short term and probably is not suitable for an immediate patch. We have provided some short-term recommendations for both users and system administrators as well as longer-term solutions to help avoid similar problems in the future.

Short-term Recommendations

Until an industry solution occurs, the team suggests to not use security transport protocols like QUIC that have chosen to use AES-GCM to pass sensitive information like credentials over the Internet. Unfortunately, users do not have control over this option in some situations until alternatives or patches are developed. We recommend the following user steps 1) users should disable QUIC in Chrome; 2) users should enable two-step verification with their Gmail account for additional protection; 3) system administrators should block QUIC to their servers with their firewall.

Long-term Recommendations

The team suggests that industry should look into padding sensitive, short communication. In the ideal case, padding should begin before the password is transmitted. We realize this could introduce a host of other issues such as making browsers and servers unable to communicate. The padding may be more appropriate to implement into future versions of HTTP/2, because this would ensure future client and server communication does not break. As we know, many users and administrators do not keep servers patches up-to-date, and therefore, any short-term patch would need to be installed on both clients and servers. We also do not recommend necessarily reverting to AES-CBC + HMAC, because past implementations have introduced a host of vulnerabilities. We believe this defies an easy fix in the short term and probably is not suitable for an immediate patch. We suggest future design choices for transport security protocols carefully weigh whether or not to use cipher suites that expose plaintext length. To further help industry make better design choices, we recommend NIST note of the potential danger in choosing counter mode or similar ciphers that expose plaintext length. NIST should explicitly highlight how counter mode and similar ciphers may not be appropriate for implementation into transport security protocols.

3.2.5 Conclusions

In TLS 1.3 and TLS 1.2 (some ciphers) there is a 1-1 relationship between the length of a ciphertext and the length of the corresponding plaintext. The responsibility of identifying and padding length-sensitive data is pushed to the application developer. We conducted an observational study of AES-GCM traffic (the most commonly used cipher in TLS) which uncovered a widespread failure to pad passwords. In particular, we found multiple high-profile instances where password lengths can be directly inferred from encrypted web traffic. If an eavesdropping attacker can link the source IP address with a particular user name (e.g., via unencrypted traffic) then the attacker can directly infer the length of that user's password. We used a decision-theoretic model to analyze the advantage a password attacker obtains by learning the length of a user's password. Our analysis shows that the advantage is substantial.

While there are good metrics about how much a particular cipher reduces latency, there are fewer reliable models to help security professionals quantify how many users might be susceptible to intrusions and from which class of intruders (i.e. nation-states, criminals, or hackers). Without this information, it will be difficult for security professionals to make informed decisions about the trade-offs between speed and security. As a necessary first step, this research begins to quantify the cost an intruder would incur to perform an online attack. This helps better understand which intruders might be willing to endure the cost to intrude on when conducting a campaign against a set of targeted accounts. As the push for faster security transport security protocols continues to grow, researchers will be challenged to balance the need for speed and security. Ultimately, this research adds value by helping security researchers better quantify and compare the specific trade-offs between speed and security for a particular cipher suite used for a security transport protocol.

Future work: We have begun to quantify the risks of leaking password lengths. There is a need to quantify the risks of leaking plain text lengths in other contexts (e.g., short chat communications). In general, it would be helpful to formulate clear guidelines to help developers evaluate when plain text length should be viewed as a sensitive attribute. When plain text length is sensitive there is a need to provide developers with an easy method to obfuscate sensitive data lengths, and there is a need to develop automated tools which could audit code and identify instances where length-sensitive data might not be hidden.

Recommendations for Users: We offer the following suggestions to users to protect themselves against password length leakage attacks[7]. First, enable two-factor authentication whenever possible. Strong two-factor authentication will prevent an attacker from mounting an online attack whether or not the attacker knows the length ℓ of your password. Second, we recommend that users select strong passwords which don't occur in a password cracking dictionary. We recognize that this advice, while easy to give, can be challenging to follow since users typically have many password-protected accounts [103]. However, the additional risks from password length leakage may justify the extra effort for many users. Mnemonic techniques [112, 113] and spaced repetition [92, 114] may help to reduce the extra user burden. Finally, users could also begin to use a password manager to generate unique passwords from one master password. This solution potentially reduces user burden since the user will now only need to remember a few master passwords. Second, the length of the derived password for each domain is not necessarily correlated with the length of the user's master password. For example, PwdHash [115] derives a unique fixed-length password for each domain based on the cryptographic hash of the user's master password along with the domain itself.

---

[7] Even in the absence of password length leakage attacks the following advice can help a user to secure his accounts. However, the advice takes on a greater urgency due to the stronger threat of online attackers.

3.3 An Economic Model for Quantum Key-Recovery Attacks

As the field of quantum computing progresses it is crucial for security practitioners to understand the potential risks posed to deployed cryptosystems. In this work, we focus on quantum key-recovery attacks for symmetric-key primitives e.g., AES. Classically a symmetric key-recovery attack requires $\approx 2^n$ queries in the ideal cipher model where n is the size of the secret key (bits). By contrast, Grover's algorithm only requires $\approx 2^{n/2}$ queries in the (quantum) ideal cipher model. While the attack requires exponential work[8], it constitutes a dramatic reduction compared to classical attacks. For example, it would be infeasible for a powerful nation-state attacker to make 2^{128} AES queries, but 2^{64} might be feasible even for much less sophisticated attackers!

Traditional wisdom says that one can ensure n bits of security for an ideal cipher by simply selecting $2n$ bit keys instead of n. However, this conservative advice might dramatically overestimate the capability of the attacker. In particular, Grover's search requires $2^{n/2}$ *sequential* queries meaning that the attack might not finish in our lifetime. We remark that, in the ideal cipher model, any quantum key-recovery attack making at most $O\left(2^{n/2}/\sqrt{k}\right)$ sequential queries requires at least $\Omega\left(2^{n/2}\sqrt{k}\right)$ total queries. Thus, while one can parallelize Grover's search to reduce the running time by a factor of $\sqrt{k}$, but this approach necessarily increases the total amount of work by a factor of $\sqrt{k}$.

In this paper, we advocate for an economic approach to evaluate the security of symmetric-key primitives (e.g., AES-128) in a post-quantum world instead of focusing only on the running time of the fastest attack. Wiener argued that the "full cost" of a cryptanalytic attack [116] should account for all of the required resources e.g., the cost of the circuit running the attack amortized over the number of instances that can be solved over the lifetime of the circuit. This view has guided the design and analysis of secure memory-hard functions for protecting low entropy secrets like passwords against brute-force attacks [32, 16, 39, 56, 3]. Taking this view we aim to model (and lower bound) the cost of running a quantum key-

---

[8] By contrast, Shor's algorithm can be used to break any public key encryption scheme whose security relies on the hardness of the integer factorization or discrete logarithm problem in polynomial time. This includes most widely deployed public key encryption/signature schemes including RSA, EC-DSA, Schnorr Signatures, ECDH, etc. Thus, there is a need to migrate towards "Post-Quantum" schemes that resist known quantum attacks like Shor's algorithm [65, 66].

recover attack. We take the view that an attacker will only run a brute-force attack if the "full cost" of the attack is less than the value of the information that can be decrypted at the time when the quantum brute-force attack completes i.e., information decrypted 10 years in the future may be worth less than if the documents had been decrypted today.

We now introduce an economic model that estimates the gain (or loss) of a quantum key-recovery attack. Our model includes the following components: (1) The initial value v_0 of the encrypted information and a function $R(T, v_0)$ which describes how this value decays over time T. (2) A time limit T_y (years) for the attack e.g., 1–100 years. (3) The width and depth of a quantum circuit implementing the cipher we are analyzing e.g., see [117, 118] for estimates of AES. (4) The (predicted) speed of a universal quantum computer (gates/sec), (5) The (predicted) cost of renting a single quantum circuit capable of evaluating this cipher (dollars/year). Given these parameters our model allows us to determine whether or not a profitable attack exists. Fixing all of the parameters except for the initial value of the encrypted information we can determine how valuable the information would need to be for a quantum key-recovery attack to be profitable. Alternatively, fixing the initial value of the information (and a decay function) we can ask how fast/cheap a quantum computer must be to make a quantum key-recovery attack profitable.

We remark that components three and four of our model (speed/cost of future quantum computers) are arguably the most difficult to predict. We advocate for a conservative approach where we attempt to upper bound (resp. lower bound) the speed (resp. cost) of a future quantum computer. We remark that NIST considers 2^{64} to be a safe upper bound on the depth of any quantum circuit which can be evaluated in 10 years which would correspond to a speed of 5.8×10^{10} gates per second. Thus, we might take 60 GHz as our conservative upper bound on the speed of a quantum computer.

The attacker will select a desired time T (years) for the key-recovery attack to complete. We can infer the level of parallelism necessary to complete the attack in time T given additional information about the depth of our quantum circuit implementing our cipher (e.g., AES) as well as the gate propagation speed of our quantum computer. We use $C(T)$ to denote the minimum possible cost of a quantum key-recovery attack with a time bound T. Intuitively, as T decreases the level of parallelism increases as well as the cost $C(T)$. We use the reward function $R(T, v_0)$ to describe the attacker's benefit when the encrypted information is recovered at time T. Here, $v_0 = R(0, v_0)$ denotes the initial value of the encrypted information which may decrease over time. Thus, the profit of the attacker is $P(T, v_0) = R(T, v_0) - C(T)$ and the attacker will select the time parameter $T^* = \arg\max_T P(T, v_0)$ to optimize profit. If $P(T^*, v_0) < 0$ a rational attacker will choose not to attack. For notational convenience we use $P(0, v_0) = 0$ to denote the profit of an adversary who does not run the attack i.e., $T = 0$.

3.3.1 Cipher Circuit Year

To estimate the costs of running an attack we first define the concept of a Cipher Circuit Year (CCY). Intuitively, a CCY represents the annual rental cost (which factors in equipment, labor, electricity, and any other expenses) of a quantum computer capable of evaluating our cipher (e.g., AES)[9]. We can use CCY as a way to examine the monetary cost of a key recovery attack. For example, if we can complete a key recovery attack (e.g using Grover's algorithm) with no parallelism (i.e. using only one circuit) in 10 years then this attack would cost 10 CCY. However, if the same attack was completed in 1 year (which will require the use of 100 circuits running Grover's algorithm in parallel) we would have a cost of 100 CCY. Similar notions such as full cost [116] or aAT complexity [39] have been very fruitfully applied in the area of password hashing as a method of estimating costs of computation. Throughout this work we are considering attacks in the (quantum) ideal cipher model i.e. we do not concern ourselves with (quantum) structural attacks against a cipher like AES e.g. [119].

3.3.2 Required Level of Parallelism and Attack Costs

Suppose that we have a time bound T_y (unit: years) for our key-recovery attack. Given the gate propagation speed s (Hz) of a quantum circuit we can use T_y to upper bound the

---

[9] Alternatively, we could think of CCY as representing the opportunity cost when this quantum computer used to running our key-recovery attack instead of performing other computation. Finally, we could think of CCY as representing this cost of building the quantum computer divided by the (expected) number of years before the quantum computer breaks.

total depth $t = T_y \times s$ of our computation (quantum gates) e.g., if $s = 1GHz$ and $T_y = 1$ year then $t = 3.15 \times 10^{16}$. Supposing that our cipher can be implemented as a depth d circuit our key-recovery attack can make at most t/d sequential oracle queries to the cipher. If we partition our search space $\{0,1\}^n$ into k buckets of size N/k and run Grover's attack on each bucket in parallel then we require at least $\frac{\pi}{4}\sqrt{\frac{N}{k}}$ sequential oracle calls in each bucket ($\frac{\pi}{4}\sqrt{Nk}$ total oracle calls). Thus, $t/d \geq \frac{\pi}{4}\sqrt{\frac{N}{k}}$ which means that we require parallelism $k \geq \frac{\pi^2 N}{16\left(\frac{t}{d}\right)^2}$. The total cost will be minimized when equality holds. The total cost will be $C(T_y) = T_y \times k \times C_{CCY}$, where C_{CCY} is the cost of a CCY e.g., in USD. We remark that the value of k will depend on the time bound T_y, the depth d of our cipher and the speed s (Hz) of our quantum computer. Substituting into the above formula we get

$$C(T_y) = \frac{C_{CCY}\pi^2 N d^2}{16 T_y s^2}$$

Intuitively, the cost decreases as we relax the time bound T_y. If $T_y \geq \frac{\pi}{4}\sqrt{N} * \frac{d}{s}$ is sufficiently large to set $k = 1$ we have $C(T_y) = C\left(\frac{\pi}{4}\sqrt{N} * \frac{d}{s}\right)$.

We note that attack costs are directly linked to an attacker's strategy. If an attacker considers the value of information to be less than the cost to run the attack we say that a rational attacker will choose to not run the attack, leaving the information secure.

3.3.3 Time-Value of Information and Reward Functions

We first discuss several different instantiations of the reward function $R(T, v_0)$ which defines the time-value of the encrypted information. We will always assume that the function is monotonic i.e., $R(T, v_0) \leq R(T - \epsilon, v_0)$. Intuitively, obtaining the secret information earlier (e.g., at time $T - \epsilon$) is preferable to obtaining the secret information later[10]. In our analysis we consider three types of reward functions: (1) Constant functions $R(T, v_0) = v_0$ i.e., the time-value of the information does not diminish over time. (2) Threshold Functions where the information has value v_0 before time T and value 0 afterwards i.e., $R_T(T, v_0) = v_0$ whenever $T < T$ and $R_T(T, v_0) = 0$ for $T > T$. (3) Delta Discounting where the time-value of the

---

[10] If the attacker prefers to wait to time t to recover the secret information he could always run the attack and then wait ϵ seconds to measure the quibits

information smoothly decays with some fixed rate $0 < \delta < 1$ i.e., $R_{\delta,T}(T, v_0) = v_0\delta^T$. While this is not an exhaustive list of all possible reward functions we believe our list constitutes a reasonable range of behaviors.

We remark that a threshold function is appropriate in settings where the encrypted information will become public at some time t in the future e.g., scripts for soon-to-be-released blockbuster movies or plans for an upcoming military campaign. The constant reward function can be seen as a special case of delta-discounting with $\delta = 1$ and threshold $T = \infty$. Below we analyze the attacker's optimal strategies with respect to each reward function.

3.3.4 Rational Attacker Strategies

A symmetric key-recovery attacker can pick a desired parallelism parameter k. Larger values of k reduce the running time T . Thus, by picking large k we can potentially earn a larger reward $R(T, v_0)$, but at the expense of total cost $C(T)$. However, as long as the total profit $P(T, v_0) = R(T, v_0) - C(T)$ increases it is in the adversary's best interest to pick a larger value of k.

Constant valuation: For constant reward functions profit is maximized whenever $C(T, k)$ is minimized. As our total time and work only increase with the addition of more oracles, C is minimized by setting $k = 1$ i.e. running a sequential attack. We argue that constant valuation is rarely an appropriate model e.g., we expect that the value of information will not be useful after 100 years since most people who are currently alive won't be around to benefit.

Threshold function: We next consider the threshold reward functions where information has value v_0 before time T and value 0 afterwards e.g., plot points for an upcoming movie.

$$R_T(T, v_0) = \begin{cases} v_0 & T \leq T \\ 0 & T > T \end{cases}$$

, where v_0 is the value of the information if it is recovered in time. In such a case there is no need to decrypt the information after time T so the attacker effectively faces a time limit of T. Since the reward is constant before time T the attacker will maximize profit by selecting the minimum possible level of parallelism necessary to finish in time exactly T i.e., $k = \frac{\pi^2 N}{16 \frac{t}{d}^2}$ where $t = T \cdot s$.

Delta discounting with Threshold We now analyze the behavior of the attacker with smooth δ-discounting reward functions i.e., $R_{\delta,T}(T, v_0) = v_0 \delta^T$ for $T \leq T$ and $R_{\delta,T}(T, v_0) = 0$ if $T \geq T$. Here, $0 < \delta \leq 1$ is our decay parameter and T is our threshold. The attacker wants to pick a time T which maximizes profit $P(T, v_0) = R_{\delta,T}(T, v_0) - C(T)$. We show that there are three possible ways to maximize the profit function $P(T, v_0)$. (1) If the attacker does not run the attack $T = 0$ then $P(0, v_0) = 0$. (2) The attacker sets $T = \min\{T_{seq}, T\}$ where $T_{seq} = \frac{\pi d}{4s}\sqrt{N}$ is the time to run the sequential version of Grover's algorithm ($k = 1$) when the speed is s and the depth of the underlying cipher circuit is d. (3) The attacker sets $T = T^*$ for a special value

$$T^* = \frac{2W\left(\frac{1}{2}\sqrt{c}\log\delta\right)}{\log\delta}.$$

Here we let $c = \frac{\Lambda}{v \ln \delta^{-1}}$ and $W(\cdot)$ denotes the analytic continuation of the product log function i.e., the Lambert W function. We note that this function can be efficiently evaluated. The derivation is as follows: Here we seek to maximize:

$$\begin{aligned} P(T, v_0) &= R_{\delta,T}(T, v_0) - C(T) \\ &= v\delta^T - \frac{C_{CCY}\pi^2 N d^2}{16 T s^2} \end{aligned}$$

We compress via $\Lambda = \frac{C_{CCY}\pi^2 N d^2}{16 s^2}$. Profit can be maximized as:

$$\begin{aligned} P(T, v_0) &= \frac{d}{dT_y}\left(v\delta^T - \frac{\Lambda}{T_y}\right) \\ &= v\delta^{T_y}\ln\delta + \frac{\Lambda}{T^2}. \\ 0 &= v\delta^T \ln\delta + \frac{\Lambda}{T^2} \\ \frac{\Lambda}{T^2} &= v\delta^T \ln\delta^{-1} \\ \delta^T T^2 &= \frac{\Lambda}{v\ln\delta^{-1}} \end{aligned}$$

3.3.5 On the Future Cost and Speed of Quantum Computers

Our economic model utilizes predictions of the future speed/cost of quantum computers. However, it is difficult (or impossible) to predict what these values may be. Instead, we consider a range of possible future worlds: quantum mania, optimistic improvements, and steady improvements. Arguably, all of these worlds represent optimistic predictions of the future power of quantum computers. We could add a fourth pessimistic world where the field of quantum computing is stuck for decades due to insurmountable technical barriers e.g., decoherence, temperature maintenance. However, in such a world it would not be interesting to analyze quantum attacks. We advocate for a conservative approach where we attempt to upper bound (resp. lower bound) the speed (resp. cost) of a future quantum computer. In particular, if an attack is not profitable in our quantum mania scenario then it is reasonable to assume that no attack will be profitable.

- **Quantum Mania:** Here we assume that quantum computers have enjoyed incredible advances, both in gate speed, number of qubits, and cost. In particular, we assume that quantum circuits can be evaluated at a gate propagation speed of 60GHz which we derive from NIST's proposed upper bound on the maximum depth (2^{64}) of a quantum circuit which could be evaluated in 10 years [65] i.e., $60GHz \approx 2^{64}/(10 \times 3.154x10^7)$ where $3.154x10^7$ is the number of seconds a year.

We also assume that dramatic advancements in QC technology e.g. temperature maintenance and construction costs making it possible to rent a quantum AES circuit for \$50 per year i.e., $C_{CCY} = \$50$.

- **Optimistic improvements:** We assume a slightly slower gate propagation speed of 1GHz for quantum computers comparable to the clock speed of current desktop computers. We also assume that $C_{CCY} = 500USD$. This price is meant to be in line with a budget desktop one can currently purchase.
- **Steady improvements:** We assume that future quantum circuits can be evaluated at a gate propagation speed of 100MHz. We set $C_{CCY} = 50000USD$ here. In this scenario, the future speed/cost of quantum computers is not comparable to current classical machines. However, this future world would still constitute a dramatic increase in QC technology.

3.3.6 Case study: AES128

In this section, we use our economic model to analyze the cost of breaking a 128 bit AES key. To apply our model we first require a concrete implementation of AES-128 as a quantum circuit. Multiple groups have considered the problem of implementing AES-128 as a quantum circuit resulting in a series of increasingly efficient constructions [117, 120, 118]. Specifically, Langenberg et al. [118] provide the implementation with the smallest depth $d \approx 5.8 \times 10^4$. This corresponds to 3.27×10^{13} sequential AES oracle calls per year in our quantum mania scenario.

In our analysis we consider an attacker with a threshold reward function $R_T(T, v_0)$ for thresholds $T \in \{1, 10, 100\}$ years. Here we aim to determine how valuable the encrypted information v_0 must be for a profitable attack to exist. We repeat this analysis for each of our quantum scenarios: quantum mania, optimistic improvements, and steady improvements. Similarly, we can analyze the behavior of a profit-motivated attacker when faced with δ-discounting rewards $R_{T,\delta}(T, v_0)$ for thresholds $T \in \{1, 10, 100\}$ years. Here, we plot the minimum reward v_0 for a profitable attack vs. δ. Intuitively, as δ increases (slower diminishing rewards) the minimum value v_0 will increase. Finally, if we fix v_0 we can ask how fast/cheap would a quantum computer need to be for a profitable attack to exist.

Threshold Functions

We first begin by examining what the costs would look like if the value follows a threshold function. When considering our 100, 10, and 1-year attacks we first convert this to some value d, which in this case is representing the number of circuit layers we have available given the quantum power estimates and the time available. For example, in the incredible improvements scenario, we have $t = 1.892 \times 10^{20}$, which is derived from the 60GHz figure combined with the 100-year time span. This t, combined with the AES-128 circuit depths from [117], allows us to derive the number of oracle calls that can be made in the set time. With some set number of oracle calls possible in the time we derive k such that the attack would finish in the allotted time. k times the attack length in years gives us our CCY cost. A final substitution for the cost ratios described earlier puts a cost in USD to run each attack. These threshold results are described in Tables 3.12, 3.13, and 3.14.

- **100-year attack:** A 100-year attack represents the most persistent of adversaries. This is an attack that spans generations and would represent an enormous effort to recover some piece of information. In many ways, this is an impractical attack, as there are not many cases where it is worth protecting information for 100 years. Still, we include this type of attack to make a point about the costs of a quantum key-recovery attack. The estimated costs for this attack with a threshold function are in Table 3.12, 3.13, and 3.14.

- **Ten-year attack:** A ten-year attack still represents a fairly long-term attack. Essentially, this is an attack against information that is not time-sensitive, which might include something like bank account access credentials. The estimated cost with a threshold function can be found in Table 3.13.

Table 3.12. 100 Year Attack, Threshold value function

Advancement	t	k
Mania	1.892×10^{20}	1.962×10^{7}
Optimistic	3.154×10^{18}	7.064×10^{10}
Steady	3.154×10^{17}	7.064×10^{12}
	Cost(CCY)	Cost(USD)
Mania	1.962×10^{9}	9.810×10^{10}
Optimistic	7.064×10^{12}	3.532×10^{15}
Steady	7.064×10^{14}	3.532×10^{19}

Table 3.13. 10 Year Attack, Threshold value function

Advancement	t	k
Mania	1.89×10^{19}	1.962×10^{9}
Optimistic	3.154×10^{17}	7.064×10^{12}
Steady	3.154×10^{16}	7.064×10^{14}
	Cost(CCY)	Cost(USD)
Mania	1.962×10^{10}	9.810×10^{11}
Optimistic	7.064×10^{13}	3.532×10^{16}
Steady	7.064×10^{15}	3.532×10^{20}

Table 3.14. 1 Year Attack, Threshold value function

Advancement	t	k
Mania	1.89×10^{18}	1.962×10^{11}
Optimistic	3.154×10^{16}	7.064×10^{14}
Steady	3.154×10^{15}	7.064×10^{16}
	Cost(CCY)	Cost(USD)
Mania	1.962×10^{11}	9.810×10^{12}
Optimistic	7.064×10^{14}	3.532×10^{17}
Steady	7.064×10^{16}	3.532×10^{21}

- **One year attack** Here we consider attacks that may be of interest to an adversary wanting information that is valuable in the near future. This might include things like business strategies, financial plans, etc. The estimated cost with a threshold function can be found in Table 3.14.

δ discounting method

For the δ discounting method we present the information in a slightly different way. We look for the minimum value such that an attack would be profitable to run. As we have $P(T_Y, v_0) = R_{T,\delta}(T_y, v_0) - C(T_y) = v\delta^{T_y} - c_{CCY}\frac{\pi^2 N T_y}{16(t/d)^2}$, where $t = T_y \cdot s$. We have a viable attack if $v \geq \frac{\pi^2 N d^2}{16 T_y s^2 \delta^{T_y}}$. Here we can set T_y appropriately, as we did in the threshold experiments. When this is set we have the option to let δ range from 0 to 1, or equivalently to allow δ^{T_y} to range from 0 to 1. For the sake of demonstration we will be examining this kind of attack in the "incredible improvements" scenario. To find this point v we first require the conversion factor between T_y and t. We have $d = 57894$ as our circuit depth for AES [118] (taking their round-depth estimates for a full Grover's attack). Supposing that $s = 6\times 10^{10} Hz$ (quantum mania) we are able to evaluate a circuit of depth $t = T_y \times 6 \times 10^{10} Hz$ where T_y is given in years e.g., 1 year, 10 year, or 100 years (here we take 100). Substitution gives $\frac{t}{d} = \frac{T_y \times 1 \text{ year} \times 6 \times 10^{10} Hz}{57894} = T_y \times 3.271 \times 10^{13}$. We know that an attack is profitable only if

$$v \geq \frac{C_{CCY}\pi^2 N}{16(3.271 \times 10^{13})^2 T_y \delta^{T_y}}$$

Fixing $T_y \in \{1, 10, 100\}$ and letting $\beta = \frac{C_{CCY}\pi^2 N}{16(3.271\times 10^{13})^2 T_y}$ we have $v \geq \frac{\beta}{\delta_t}$. We plot the minimum value required for this to be true for the given T_y, and the results for the quantum mania world are shown in Figure 3.7. We note that the case $\delta^t = 1$ is identical to the case where $\delta = 1$, and that these values will match those from the threshold function. For other values in this chart, δ^{T_y} can be understood as the remaining value at the *end* of the attack e.g. δ^{T_y} of 0.2 in the 100 year case means that 20% of the value remains after 100 years, while in the 1 year case δ^{T_y} of 0.2 means only 20% of the value remains after a single year. In the case $\delta^{T_y} = 0.2$ the specific value of $\delta = (0.2)^{\frac{1}{T_y}}$. Thus, our model predicts that

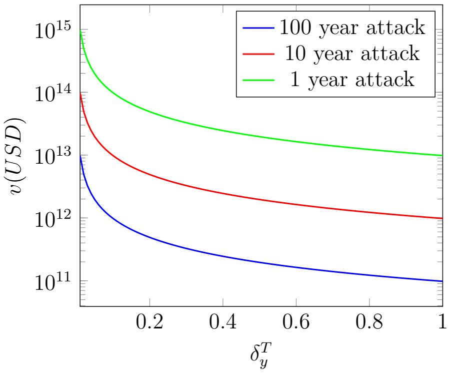

Figure 3.7. Minimum value required for positive profit. δ^{T_y} represents the value left after T_y years

AES128 provides sufficient protection provided that the initial value of the information is under $\approx 10^{11.5}$ USD.

Improvements in circuit width and depth

We now consider the following question. What happens if we can develop smaller quantum circuits to compute a cipher (e.g. the quantum AES circuit)? Improvements might be made either by reducing the width or the depth of the circuit e.g., by exploiting feasible time-space tradeoffs for the function. We first note that if the width is reduced by some factor c then the cost of running the attack also drops by the same factor c. More interesting is the case where we can reduce the depth of our cipher circuit. As we reduce depth d of the quantum circuit (holding width constant) an attacker running at the same gate speed can make more cipher queries in the same amount of time. Thus the attacker saves cost on multiple fronts — the circuit itself is smaller by some factor c and the attacker can reduce parallelism because s/he can make more sequential queries in the same amount of time. Improvements in circuit depth offer quadratic improvements in attack cost, meaning that it is worth reducing the depth of a quantum circuit even if it comes at the cost of increasing the width by a sub-quadratic amount. Specifically, if we reduce d by some factor $0 < \beta < 1$ we now have a more relaxed requirement for our time t. Previously for some set real-time limit T_y we had time to make xt oracle calls. We now have time for $\frac{1t}{d\beta} > \frac{t}{d}$ calls. Thus our previous $k = \frac{\pi^2 N}{16 \frac{t}{d}^2}$ becomes $k = \frac{\pi^2 N}{16(\frac{t}{d}/\beta)^2} = \beta^2 k$. Thus a reduction by β offers quadratic returns. Note that increases in QC speed identically affect how many calls per year we can make and offers the same tradeoff. Suppose that we reduce depth by a factor of 10 (as is claimed from [117] to [118]), holding width constant. This allows for a 100 fold reduction in the level of parallelism and a 100 fold reduction in costs. In our 100 year quantum mania example from Table 3.12 this could lower attack cost to $\approx 9.8 \times 10^8$ USD. Far smaller than the previous estimates, but still infeasible in practice.

3.3.7 Computers capable of running profitable attacks

Under the assumptions from our three worlds of quantum development, we found that any quantum key recovery attack for a 128-bit key is not economically feasible. Here we seek to answer a related question: If we want an economically feasible attack, what kind of quantum computer would be required? We follow a similar strategy of proposing three attacks but note that this analysis works for any relevant parameters. We begin here not by assuming any level of advancement in quantum computing but by assuming an attacked values some piece of information at a particular level. Here we select some USD amount e.g. 100,000, 1,000,000, or 10,000,000 as the value of information. We also allow the attacker to select 1, 10, and 100-year attacks in the same manner as in Section 3.3.6. When we set the cost and time limit for these attacks we arrive not at a single quantum computer that would suffice to run the attack but a family of quantum computers with varying speeds and costs. This fact arises from the theoretical ability to bring the cost per cubit down if we allow for a computer to have a higher clock speed. So, when we set the total budget and time limit for an attack we arrive at some family of computers described in terms of the cost/speed tradeoffs. These tradeoffs are subject to the quadratic increase in cost seen when increasing the level of parallelism. We denote a family of quantum computers $\mathcal{Q}_{b,T_y,n}$ based on a budget b, time limit in years T_y, and key length n. This describes the set of quantum computers capable of running a quantum key-recovery attack with the relevant restrictions. The family contains all quantum computers satisfying the property: $\mathcal{Q}_{b,T_y,n} = \left\{ q : C_{CCY} \leq \frac{16b\left(\frac{s}{d}\right)^2}{\pi^2 2^n T_y} \right\}$ where d is the depth of the oracle circuit and s is the number of circuit layers that can be processed in

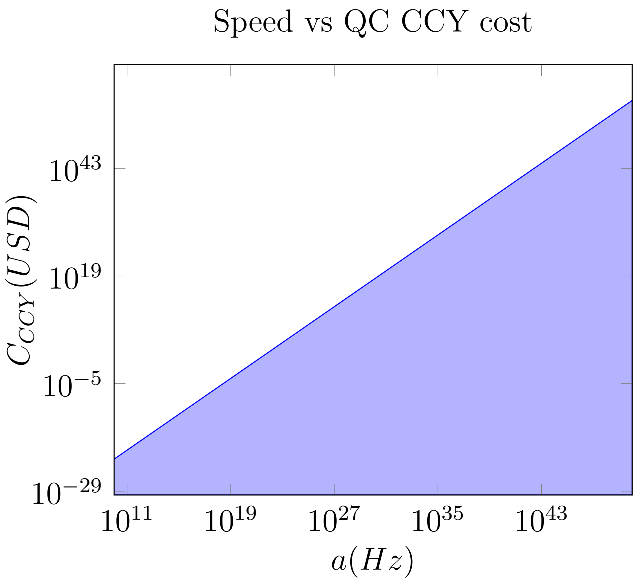

Figure 3.8. Possible values of C_{CCY} based on estimated QC speed ($b = 1.0 \times 10^8, T_y = 100, n = 128$

the given time limit. We can now begin to look at some families of quantum computers based on some reasonable attack budgets. Consider an attack that is of vital importance - where an attacker is willing to spend USD 100 million on an attack on AES-128, and needs it within 100 years. A quantum computer capable of doing so is in $\mathcal{Q}_{1.0\times10^8,100,128}$, which contains all quantum computers such that $C_{CCY} \leq \frac{1.6\times10^9\left(\frac{s}{57854}\right)^2}{\pi^2 2^{128} 100} = 1.423 \times 10^{-42} s^2$. Consider a case where we would like $C_{CCY} \leq USD1000$. This would require the computer to run at a speed of $s = 2.65 \times 10^{22}$ in 100 years, corresponding to a gate propagation speed of $8.403 \times 10^{12} Hz$, well beyond NIST's estimates of around 60GHz [65]. Any required parameters might be inserted here to see what would be required to make the existing parameters work - a plot of the family that can solve this problem is shown in Fig 3.9. We note that no matter which parameters you pick for this attack you end up with a computer that is either impossibly fast or impossibly cheap, meaning that no quantum computer that can run an attack with this requested budget and time limit is feasible.

Under the assumptions from our three worlds of quantum development, we found that any quantum key recovery attack for a 128-bit key is not economically feasible. Here we seek to answer a related question: If we want an economically feasible attack, what kind of quantum computer would be required? We follow a similar strategy of proposing three attacks but note that this analysis works for any relevant parameters. We begin here not by assuming any level of advancement in quantum computing but by assuming an attacked values some piece of information at a particular level. Here we select USD 100,000, 1,000,000, and 10,000,000 as the value of information. We also allow the attacker to select 1, 10, and 100-year attacks in the same manner as in Section 3.3.6. When we set the cost and time limit for these attacks we arrive not at a single quantum computer that would suffice to run the attack but a family of quantum computers with varying speeds and costs. This fact arises from the theoretical ability to bring the cost per cubit down if we allow for a computer to have a higher clock speed. So, when we set the total budget and time limit for an attack we arrive at some family of computers described in terms of the cost/speed tradeoffs. These tradeoffs are subject to the quadratic increase in cost seen when increasing the level of parallelism. We denote a family of quantum computers $\mathcal{Q}_{b,T_y,n}$ based on a budget b, time limit in years T_y, and key length n. This describes the set of quantum computers capable of running a quantum key-recovery attack with the relevant restrictions. The family contains all quantum computers satisfying the property: $\mathcal{Q}_{b,T_y,n} = \left\{ q : C_{CCY} \leq \frac{16b\left(\frac{\sigma}{d}\right)^2}{\pi^2 2^n T_y} \right\}$ where d is the depth of the oracle circuit and σ is the number of circuit layers that can be processed in the given time limit. We can now begin to look at some families of quantum computers based on some reasonable attack budgets. Consider an attack that is of vital importance - where an attacker is willing to spend USD 100 million on an attack on AES-128, and needs it within 100 years. A quantum computer capable of doing so is in $\mathcal{Q}_{1.0\times10^8,100,128}$, which contains all quantum computers such that $C_{CCY} \leq \frac{1.6\times10^9\left(\frac{\sigma}{15040}\right)^2}{\pi^2 2^{128} 100} = 2.11\times10^{-41}\sigma^2$. Consider a case where we would like $C_{CCY} \leq USD1000$. This would require the computer to run at a speed that computes $\sigma = 6.89 \times 10^{21}$ layers over 100 years, corresponding to a clock speed of $2.185 \times 10^{12} Hz$. This greatly exceeds even NIST's most optimistic predictions [65]. Any required parameters might be inserted here to see what would be required to make the existing parameters work - a plot of the family that can solve this problem is shown in Fig 3.9. We note that no matter which parameters you pick for this attack you end up with a computer that is either impossibly fast or impossibly cheap, meaning that no quantum computer that can run an attack with this requested budget and time limit is feasible.

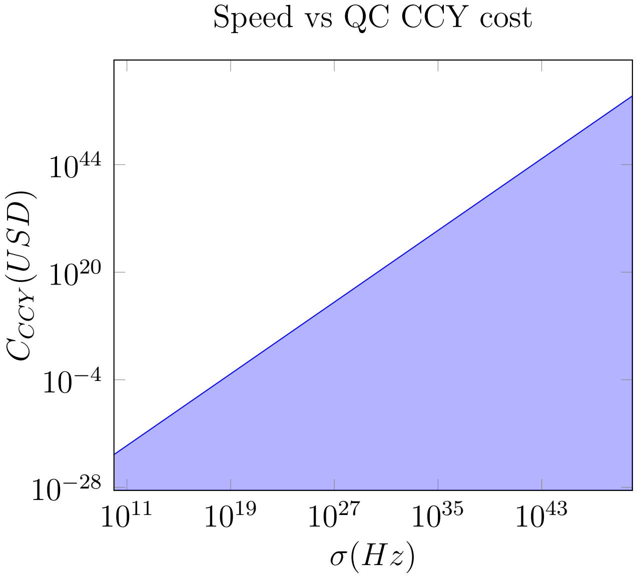

Figure 3.9. Possible values of C_{CCY} based on estimated QC speed ($b = 1.0 \times 10^8, T_y = 100, n = 128$

3.3.8 m to 1 Key Recovery Attacks

With a chosen plaintext attack where multiple keys have been used and a single chosen plaintext can be used with multiple keys it is possible to "batch" key-recovery attacks for a more effective attack. This might be considered where any one of m keys would be sufficient for an adversary to accomplish their goals e.g. to access some specific set of data that had been sent to multiple people using multiple different keys with the same nonce. We consider a chosen-plaintext attack where an attacker has managed to obtain M ciphertexts $c_1, \dots c_M$ all encrypting the same known plaintext m i.e., $c_i = Enc_{k_i}(m; r)$ where $m = m_1 || m_2$ consists of two blocks and the randomness r (i.e., nonce) is the same. Such a scenario might arise if we have multiple embedded devices using a stateful mode of operation like AES-CTR with a fixed initialization vector. Modes like AES-GCM would not be susceptible to this attack as long as the nonces are selected appropriately i.e., with strong PRGs.

Our attacker will be content to crack *any* of the keys $k_1, \dots, k_m$. To run Grover's algorithm the attacker would need to implement the function

$$f_{k_1,\dots,k_M}(x) = \begin{cases} 1 & x \in \{k_1, \dots, k_M\} \\ 0 & \text{otherwise} \end{cases} .$$

This function could be implemented as follows

$$F_{c_1,\dots,c_M}(x) = \begin{cases} 1 & Enc_x(m) \in \{c_1, \dots, c_M\} \\ 0 & \text{otherwise} \end{cases} .$$

We first note that (except with negligible probability) we will have $F_{c_1,\dots,c_M}(x) = f_{k_1,\dots,k_M}(x)$ for all inputs x i.e., because m is two blocks long we expect that for each c_i there is only one key k (namely $k = k_i$) s.t. $Enc_k(m) = c_i$. Note that each call to $F_{c_1,\dots,c_M}$ generates just 2 calls to the underlying cipher-circuit to obtain $c = Enc_x(m)$ — both of these calls can be evaluated in parallel. We then need to check whether or not $c \in \{c_1, \dots, c_m\}$. Since we want to compute $F_{c_1,\dots,c_M}$ on the same quantum hardware used to evaluate the cipher we require that the width of our circuit is not larger that the width of our AES circuit. When we add this restriction $F_{c_1,\ldots,c_M}$ can be evaluated on a Quantum Circuit of depth $\mathcal{O}\left(d_{AES} + \frac{Mn}{w_{AES}}\right)$ where $d_{AES} \approx 1.5 \times 10^4$ and $w_{AES} \approx 10^3$ are the depth and width of the quantum AES circuit. Since $n = 128$ in our analysis, whenever $M < 10^5$ the depth of the circuit is dominated by the depth of the AES circuit.

Theorems 3.3.1 and 3.3.2 below upper and lower bound the total number of ideal cipher queries necessary to recover one out of M keys.

Theorem 3.3.1. *There exists a k-parallel quantum algorithm $A^{F_{c_1,\ldots,c_M}(\cdot)}$ such that $\Pr\left[A^{F_{c_1,\ldots,c_M}(\cdot)}(x)(1^n) \in S\right] > \frac{1}{2}$ in sequential time $O\left(\sqrt{\frac{N}{kM}}\right)$ and makes $O\left(\sqrt{\frac{kN}{M}}\right)$ oracle queries, where probability is taken over selection of a random subset $S \subseteq \{0,1\}^n$ of size M as well as the randomness of $A^{f_S(\cdot)}$.*

Proof. WLOG we assume that $M = 2^m$ is a power of two to simplify exposition. Let $A^{f_S(\cdot)}(1^n)$ do the following:

1. Partition the search space into m blocks $B_{0^m}, \ldots, B_{1^m}$ where we have $B_x = \{xy : y \in \{0,1\}^{n-m}\}$ for each $x \in \{0,1\}^m$.
2. Select a block uniformly random B_x for $x \in \{0,1\}^m$.
3. Run a modified k-Parallel Grover's algorithm on the block B_x.

Straightforward balls and bins analysis tells us that $\Pr\left[|B_x \cap S| \geq 1\right] \geq 1 - \frac{1}{e}$. Boyer et al [63] adapted Grover's algorithm to handle the case where there are an unknown number of solutions t. Their algorithm runs in sequential time $O\left(\sqrt{\frac{|B_x|}{t}}\right)$. Since, $\sqrt{\frac{|B_x|}{t}} = O\left(\sqrt{|B_x|}\right)$ and $\sqrt{|B_x|} = \sqrt{N/M}$ the running time would be at most $O\left(\sqrt{\frac{N}{M}}\right)$. If the attacker is k-parallel we can use the standard trick of further dividing B_x into k blocks $B_{x,1}, \ldots, B_{x,k}$ of equal size and running an independent search on each of these blocks. Each of these searches requires sequential time $O\left(\sqrt{|B_{x,i}|}\right) = O\left(\sqrt{\frac{N}{Mk}}\right)$ with $O\left(\sqrt{\frac{N}{Mk}}\right)$ queries to $f_S(\cdot)$ total number of oracle queries would be $O\left(\sqrt{\frac{N}{Mk}}\right)$.

We remark that if $|B_x \cap S| \geq 1$ then the search will succeed with high probability. Thus, we have $\Pr\left[A^{f_S(\cdot)}(x)(1^n) \in S\right] \geq \frac{1}{2}$ as required.

□

Theorem 3.3.2. *Given any constant* $c \in (0,1]$ *there is no* k*-parallel quantum algorithm* A^{Enc} *running in sequential time* $o\left(\sqrt{\frac{N}{Mk}}\right)$ *and making at most* $o\left(\sqrt{\frac{Nk}{M}}\right)$ *queries to the ideal cipher that can find an element* $x \in S$ *with probability* $\Pr\left[A^{f_S}(1^n) \in S\right] > c$*, where probability is taken over selection of a random subset* $S \subseteq \{0,1\}^n$ *of size* m *as well as the randomness of* $A^{f_S(\cdot)}$.

Proof. (sketch) We assume that $M = 2^m$ is a power of two to simplify presentation. We first note that the problem of selecting a susbset S of size M is equivalent to randomly partitioning the search space $\{0,1\}^N$ into M blocks $B_{0^m}, \ldots, B_{1^m}$ of size $2^{n-m} = N/M$ and then constructing S by randomly selecting one element from each block B_x. If we offer to reveal the partition $B_{0^m}, \ldots, B_{1^m}$ this can only help the attacker. Thus, without loss of generality we can assume that S is constructed by selecting one random element from each of the sets $B_x = \{yx \ : \ y \in \{0,1\}^{n-m}\}$ for each $x \in \{0,1\}^m$.

We will argue by contradiction. In particular, we show that if such a k-parallel quantum algorithm A^{f_S} exists such that (1) $\Pr\left[A^{f_S}(1^n) \in S\right] > c$, (2) A^{f_S} runs in sequential time $o\left(\sqrt{\frac{N}{Mk}}\right)$ and (3) A^{f_S} makes at most $o\left(\sqrt{\frac{Nk}{M}}\right)$ oracle queries then we can devise a new k-parallel quantum algorithm A to solve the regular quantum search problem over the search space $\{0,1\}^{n-m}$ such that A runs in sequential time $o\left(\sqrt{\frac{N}{k}}\right)$ and makes at most $o\left(\sqrt{kN}\right)$ queries contradicting a result of Zalka [64].

Given an indicator function $f_x : \{0,1\}^{n-m} \to \{0,1\}$ such that $f_x(x) = 1$ and $f_x(y) = 0$ for all $y \neq x$ the quantum search problem is to find the secret value x given oracle access to $f_x(\cdot)$. Our algorithm A^{f_x} will select random values $y_z \in \{0,1\}^n$ for each $z \in \{0,1\}^m$ subject to the constraint that for any $z \neq z$ the last m bits of y_z and y_z are distinct. We can implicitly define the set $S = \{(xz) \oplus y_z \ : \ z \in \{0,1\}^m\}$. The set S cannot be constructed explicitly, but A can simulate the oracle $f_S(\cdot)$ using two queries to the oracle $f_x(\cdot)$. In particular, given an input $w \in \{0,1\}^n$ for $f_S(\cdot)$ there are at most two values $z \in \{0,1\}^m$ such that $w_z = w \oplus y_z = x_w z$ for some string $x_w \in \{0,1\}^{n-m}$ and then let x_z denote the first $n-m$ bits of w_z. We remark that $w \in S$ if and only if we can find z, w such that $w_z = w \oplus y_z = x_w z$ and $f_x(x_w) = 1$. A will now simulate A^{f_S} to recover $w \in S$ with probability at least $c > 0$. The sequential running time of A will still be $o\left(\sqrt{\frac{N}{Mk}}\right) = o\left(\sqrt{Nk}\right)$ and the total number of queries will be $q_A = 2 * q_A = o\left(\sqrt{Nk}\right)$. Given $w \in S$ we can find the unique value $z \in \{0,1\}^m$ such that $w_z = w \oplus y_z = x_w z$ for some string $x_w \in \{0,1\}^{n-m}$ and recover x from the first $n - m$ bits of w_z. $\square$

These theorems show that when considering an M key batch attack running on multiple quantum computers in parallel Grover's algorithm is an asymptotically optimal solution. This also shows that as you obtain M keys to batch together you can speed up attacks by a factor of $\sqrt{M}$. This can cause some significant reductions in attack costs, bringing some attacks closer to economically feasible ranges. For example, consider a setting where the attacker has access to $M = 10^6$ encryptions of the same message under different AES keys. In this case, the cost of cracking one of these keys within 100 years would be around 100 million (USD) under our quantum mania assumption. This is still quite expensive, but significantly cheaper than the 100 billion (USD) it would take to crack each key individually.

3.3.9 Discussion

We introduced an economic model to analyze the efficacy of quantum key-recover attacks. Our results (for threshold scenarios) are summarized in table 3.12, 3.13 and 3.14. Within these tables consider the attacker's most optimistic scenario. Suppose that we are in the "quantum mania" world in which the cost/speed of quantum computers improves at a rapid pace. Further suppose that the only time restriction that the attacker faces is that the key-recovery attack must be completed within 100 years. Even in the attacker's best-case scenario the cost of a key-recovery attack is estimated at $\$9.81 \times 10^{10}$, a very significant amount. While this is certainly less than the expected classical cost of $\$9.24 \times 10^{29}$ we still see a significant financial barrier to these attacks. Under less optimistic scenarios the attacker's costs only increase e.g., if the attacker needs to recover the key in 10 years under our "optimistic" assumption on advances in quantum computing the attacker's costs will be at least 3.352×10^{16}, well beyond the capabilities of any adversary. Given that the cost of a quantum key-recovery attack is so high we argue that for almost all use cases AES-128 should remain safe in a post-quantum world. We additionally stress that the values we provide should be considered lower bounds. We have ignored many significant issues that arise for quantum computers like error correction, decoherence, and electricity costs.

We advocate for rethinking the common strategy of defending against a quantum key-recovery attack by doubling the key length. In fact, not only do we find that doubling key length is usually unnecessary, we also find that adding a constant number of bits to the key is not needed as suggested in [121]. In settings where computational overhead is paramount (e.g., embedded devices) and the secret is under our lowest attack cost estimate ($\$6.63 \times 10^9$) it may be better to opt for smaller key lengths.

3.4 Grover's Algorithms Attacks Against Nonuniform Distributions

Grover's algorithm [62] for database search is a well-known quantum algorithm capable of outperforming classical computers for the database search problem. Specifically, it is capable of finding some input $x^* \in X$ to a function $f : X \to Y$ that produces a specified output $y^* \in Y$ in time $O\left(\sqrt{|X|}\right)$ (More details in Section 2). By appropriately selecting this function, domain, and codomain Grover's algorithm can accomplish goals like finding a symmetric cipher key (e.g., for AES) much faster than a classical computer could brute force the same problem. Grover's algorithm was proven to be asymptotically optimal [64], however, some methods have been shown to slightly reduce the expected cost of Grover's algorithm, e.g., [63]. Here we consider another method of reducing the expected cost of Grover's algorithm in specific scenarios - those where the distribution of elements in X are not uniformly distributed.

For use cases of Grover's algorithm like a key-recovery attack against AES we generally do not require any information about the distribution over elements in X. Indeed, Grover's algorithm itself is blind to any information about this distribution and performs at the same speed over a uniform distribution as it would over a distribution with all weight on a single element. What we show in this work is that there are ways to take advantage of non-uniform distributions in some cases to reduce the expected cost of a Grover's algorithm attack. Essentially, we note that while Grover's algorithm is blind to the distribution of elements in X while it is being run, we can take advantage of a distribution's structure to run multiple smaller instances of Grover's algorithm in a way that reduces the expected cost of an attack.

When a distribution is sufficiently far from uniform it becomes possible to partition the domain X based on the distribution D_X that elements from X are selected. The general idea is to find a partition that takes advantage of grouping high-probability sections of the distribution D_X and runs a smaller instance of Grover's algorithm that has a higher probability of finding the correct solution than it would under a uniform distribution. While normally running a parallel version of Grover's algorithm imposes a significant cost overhead [64, 5] it is possible to *reduce* the expected total cost of an attack if we have a distribution that sufficiently increases success probability within a subdomain of D_X.

We show in Theorem 3.4.1 that any partition of D_X that reduces the total cost of Grover's algorithm is structured with blocks in the partition forming consecutive contiguous blocks, a property that makes it possible to efficiently determine what the optimal cost-reducing partition would be. Using these properties, we show that there is a dynamic programming algorithm that produces a cost-minimizing partition P of D_X efficiently. Finally, we apply this dynamic programming algorithm to an example of a non-uniform distribution - password cracking. Because users do not select passwords at random, and indeed generally select passwords only from a small subdomain of all possible passwords Σ^* and rather select from a skewed distribution with a small number of possible passwords making up a large proportion of selected passwords. Using the dynamic programming algorithm on distributions obtained from existing password datasets we find that the partitioned version of Grover's algorithm is capable of decreasing costs enough to make previously unprofitable attacks profitable.

For many practical uses of Grover's algorithm, like key-recovery attacks against symmetric-key ciphers, the distribution D_X can be considered to be uniform, or at least close enough to uniform that we believe it unlikely that any partitioning strategy would be able to reduce the cost of a key-recovery attack. Even in certain situations where a distribution over keys is non-uniform, e.g. if a programmer accidentally selected a 256-bit key using only a 64-bit seed to a pseudorandom number generator, we would still have a uniform subdomain of D_X on which the standard version of Grover's algorithm could be run. However, in cases where we do have a skewed domain, such as in password cracking, this new method of structuring a Grover's algorithm attack offers a new, more efficient, method of increasing attack speed and reducing attack cost.

3.4.1 Partitioned Grover's Algorithm

The partitioned version of Grover's algorithm is not a modification of Grover's algorithm itself. Rather, it is a sequence of smaller instances of Grover's algorithm that has been designed to reduce the expected cost of running the algorithm over some distribution D_X over domain X using a circuit for some function f. We first note that, in the case of a uniform distribution over the domain X, there is a higher total cost to run Grover's algorithm over sections of the domain compared to running it over the entire domain at once [121, 5]. However, in the parallel versions of Grover's algorithm discussed in these works, it is assumed that each instance of Grover's algorithm is being run concurrently. In this case, all instances are run regardless of the outcome of other instances. When a parallel attack is structured in this way with k individual instances of Grover's algorithm the algorithm suffers an overhead of $O\left(\sqrt{k}\right)$ compared to running the attack at the entire domain at once. However, it does complete its work $O\left(\sqrt{k}\right)$ times faster. Even though there is overhead there may be circumstances where time limitations make this tradeoff worthwhile, as shown in [5].

Here we note that there is another situation where a tradeoff may be worthwhile. Whereas in [5] this tradeoff involved time limits and reducing rewards over time as a motivation for parallelizing Grover's algorithm we instead consider a case where we want to minimize the expected time for an attack without any time limit or reducing rewards over time. In doing this, we do not parallelize Grover's algorithm but rather try to find a partition $P = \{B_1, B_2, \ldots, B_{|P|}\}$ that reduces the expected number of total oracle calls to Grover's algorithm when we run Grover's sequentially on each block in the partition. Specifically, the adversary runs Grover's search to check the first bucket. If they find the correct result the search terminates, otherwise the attacker can continue the attack on the next block. In the traditional version of Grover's algorithm, we make a total of $\frac{\pi}{4}\sqrt{|X|}$ calls to the oracle for f. However, if the adversary partitions the domain and checks the partition blocks in descending order of probability mass (and assuming they check the entire partition) we pay cost

$$Cost(D_X, P) = \sum_{i=1}^{|P|} \Pr[B_i] \sum_{j=1}^{i} \frac{\pi}{4}\sqrt{|B_j|},$$

where $\Pr[B_i]$ represents the total probability mass in B_i according to the distribution D_X. Intuitively, with probability $\Pr[B_i]$ an attacker would halt the attack after block B_i has been checked because they found the correct answer, and had to pay $\sum_{j=1}^{i} \frac{\pi}{4}\sqrt{|B_j|}$ to reach that point. An attacker running this attack can select any partition they want when running an attack, with a rational attacker selecting the partition that minimizes the expected attack costs.

As a practical example, consider the domain $X = Y = [1, 80]$ and a random function $f : X \to Y$. Suppose we want to find some input that generates 1 and know that it was generated with the input to f being drawn from the distribution D_X where $D_X = 0.04$ on elements $[1, 10]$ and $\frac{0.6}{70}$ on $[11, 80]$. Consider the partition $P = \{B_1, B_2\}, B_1 = [1, 10], B_2 = [11, 80]$. Note that if we ran Grover's algorithm over the entire domain X we would require $\frac{\pi}{4}\sqrt{80} \approx 7$ iterations[11]. However, if we were to run two separate instances of Grover's algorithm in sequence on B_1 and B_2 the expected number of oracle calls would be $\frac{\pi}{4}\sqrt{10} + 0.60\frac{\pi}{4}\sqrt{70} \approx 6$. Although this is admittedly a contrived case, it does show that it is possible to strategize a Grover's algorithm instance by first attempting a smaller subdomain with higher probability before moving on to other blocks.

While the above example demonstrates that there exist some possible partitions that can reduce the cost of Grover's algorithm in expectation it does not yet provide a method of determining a good partitioning strategy. In section 3.4.3 we will demonstrate a dynamic programming algorithm that finds an optimal partition for the partitioned version of Grover's algorithm. However, before the dynamic programming algorithm is demonstrated we require a few facts about the structure of an optimal partition. Thankfully, any optimal partition has a structure that makes finding an optimal partition efficient.

---

[11] Grover's algorithm requires the number of rounds to be a natural number. In reality the closest natural number to $\pi/4\sqrt{n}$ would be selected, though for convenience we use the exact value in our calculations.

3.4.2 Bucketizing Contiguous Sections is optimal

To construct our algorithms that finds the optimal cost-reducing partition P^* we first require some structural theorems to characterize the optimal distribution. First, we denote the domain that we are searching over as X, with size $|X| = n$. We denote a probability distribution over the elements of X as D_X where we will search over the support $supp(D_X)$. Elements in the support of D_X are labelled in descending order as $d_1, d_2, \ldots$. A partition $P = \{B_1, B-2, \ldots, B_{|P|}\}$ of $supp(D_X)$ has blocks B_i labelled in descending order of probability mass. For convenience, we denote $|B_i| = s_i$ and $S_i = \sum_{j=1}^{i} s_j$. This fact follows from a series of swapping arguments of the following form:

Lemma 1. *Let D_X be a distribution over X. Given a partition $P = \{B_1, B_2, \ldots, B_k\}$ of Supp(D_X) and two blocks $B_x, B_y \in P$, $x < y$, containing a pair of elements $d_x \in B_y$ and $d_y \in B_y$ such that $d_y > d_x$, there exists a partition P' such that $Cost(D_X, P') < Cost(D_X, P)$.*

Proof. Let $P' = \{B'_1, B'_2, \ldots, B'_{|P'|}\}$ where $B_i = B'_i$ for all $i \neq x, y$ and $B'_x = (B_x \setminus d_x) \cup \{d_y\}$ and $B'_y = (B_y \setminus d_y) \cup \{d_x\}$. The difference between the expected running costs using P vs P' is $\mathbb{E}\left[Cost(D_X, P) - Cost(D_X, P')\right] = \sum_{i=1}^{k} \Pr[B_i] \sum_{j=1}^{i} \frac{\pi}{4}\sqrt{|B_j|} - \sum_{i=1}^{k} \Pr[B'_i] \sum_{j=1}^{i} \frac{\pi}{4}\sqrt{|B'_j|}$. note that block sizes remain unchanged.

$$\mathbb{E}\left[Cost(D_X, P) - Cost(D_X, P')\right] =$$

$$(\Pr[B_x] - \Pr[B'_x]) \sum_{j=1}^{X} \frac{\pi}{4}\sqrt{|B_j|}$$

$$-\left(\left(\Pr[B_y] - \Pr\left[B'_y\right]\right) \sum_{j=1}^{y} \frac{\pi}{4}\sqrt{|B_j|}\right)$$

Let $p = d_x - d_y$, note $p < 0$.

$$= p\left(\sum_{j=1}^{x} \frac{\pi}{4}\sqrt{|B_j|}\right) - p\left(\sum_{j=1}^{y} \frac{\pi}{4}\sqrt{|B_j|}\right)$$

$$= -p \sum_{j=x+1}^{y} \frac{\pi}{4}\sqrt{|B_j|}$$

Thus P' has a lower expected cost. ☐

Theorem 3.4.1. *For all distributions D_X over X all minimal-cost partitions $P^* \in \underset{P\in\mathcal{P}}{\operatorname{argmax}} C(D_X, P)$ of Supp(D_X) have of the form $P^* = \{B_1, B_2, \ldots\}$ where $B_i = [S_{i-1}+1, S_i]$ for each $i \geq 1$ with $S_0 = 0$ and $S_i = S_{i-1} + |B_i|$.*

Proof. Assume for contradiction that we have an optimal partition P that is *not* of the form $B_i = [S_{i-1}+1, S_i]$ for each $i \geq 1$ where $S_0 = 0$ and $S_i = S_{i-1} + |B_i|$ (i.e., not contiguous). Then we have at least one pair of elements $d_x, d_y, d_x < d_y, x > y$ with $d_x \in B_y, d_y \in B_x$. By Lemma 1 we can construct a partition P' such that $C(D_X, P') < C(D_X, P)$. Thus, P is not optimal. By contradiction, the optimal partition must be contiguous. ☐

3.4.3 A Dynamic Programming Algorithm for Optimal Partitioned Grover's Algorithm

In this section we demonstrate a dynamic programming algorithm that finds an optimal cost-reducing partition P^* for the partitioned version of Grover's algorithm. For the purposes of this section we will fix a distribution D_X. We use the following notation in our definition of the optimal cost partition and in the derivation of a recurrence relation that finds P^*. First, we denote $\overline{B_i}$ as the event where the target element t was *not* in blocks B_1 through B_i. We let $\overline{B_0} = 0$. Similarly let $\overline{d_i}$ denote the event that the target element is known to not be drawn from any distribution element $\leq i$. Additionally, let $d_{\leq i} = \{d_1, \ldots, d_i\}$ and fixing $B_1, \ldots, B_k$ we have $B_{\leq j} = \bigcup_{i=1}^{j} B_j$ and $B_{\leq 0} = \{\}$.

Let $D_{\geq i}$ be the conditional distribution over elements $\{d_i, \ldots, d_n\}$ conditioning on the event that we do not sample $d_{\leq i-1}$ and let $OPT(i)$ denote the cost of an optimal partition of $D_{\geq i}$ i.e.,

$$OPT(i) = \min_{P\in\mathcal{P}(D_{\geq i})} \sum_{j=1}^{|P|} \Pr\left[B_j | \overline{d_{\leq i-1}}\right] \sum_{x=1}^{j} \frac{\pi}{4}\sqrt{|B_x|}.$$

We present the following recurrence relation that allows for a dynamic programming algorithm to find $OPT(i)$:

Theorem 3.4.2.

$$OPT(i) = \min_{s \le n-i+1} \left\{ \frac{\pi}{4}\sqrt{s} + \left(1 - \sum_{j=i}^{i+s-1} \Pr\left[d_j | \overline{d_{\le i-1}}\right]\right) OPT(i+s) \right\}.$$

Proof. We begin with the base cases $OPT(|X|) = \frac{\pi}{4}\Pr\left[d_{|X|}|\overline{d_{\le |X|-1}}\right] = \frac{\pi}{4}$ and, for convenience, $OPT(|X|+1) = 0$. Supposing that $B_1, \ldots, B_k$ is the optimal partition for $OPT(i)$. By Theorem 3.4.1 we can assume that $B_i = \{d_i, \ldots, d_{i-1+s_1}\}$ for some $s_1 = |B_1|$. We can rewrite $OPT(i) =$

$$\begin{aligned}
& \textstyle\sum_{j=1}^{|P|} \Pr\left[B_j|\overline{d_{\le i-1}}\right] \sum_{x=1}^{i} \frac{\pi}{4}\sqrt{|B_x|} \\
= \quad & \textstyle\frac{\pi}{4}\sqrt{B_1} + \sum_{j=2}^{|P|} \Pr\left[B_j|\overline{d_{\le i-1}}\right] \sum_{x=2}^{j} \frac{\pi}{4}\sqrt{|B_x|} \\
= \quad & \textstyle\frac{\pi}{4}\sqrt{B_1} + \left(1 - \Pr\left[B_1|\overline{d_{\le i-1}}\right]\right) \times \\
& \textstyle\sum_{j=2}^{|P|} \Pr\left[B_j|\overline{d_{\le i-1}} \cup \overline{B_1}\right] \sum_{x=2}^{j} \frac{\pi}{4}\sqrt{|B_x|} \\
= \quad & \textstyle\frac{\pi}{4}\sqrt{B_1} + \left(1 - \Pr\left[B_1|\overline{d_{\le i-1}}\right]\right) \times \\
& \textstyle\sum_{j=2}^{|P|} \Pr\left[B_j|\overline{d_{\le i-1+|B_1|}}\right] \sum_{x=2}^{j} \frac{\pi}{4}\sqrt{|B_x|} \\
= \quad & \textstyle\frac{\pi}{4}\sqrt{B_1} + \left(1 - \Pr\left[B_1|\overline{d_{\le i-1}}\right]\right) OPT(i + |B_1|)\ .
\end{aligned}$$

In the final step, we note that if $B_2, \ldots, B_k$ were not an optimal partition for $OPT(i+|B_1|)$ then we would have been able to find a better partition for $OPT(i)$ (contradiction).

Thus, $OPT(n) = \pi/4$ and for $i < n$ we obtain the recurrence relationship

$$OPT(i) = \min_{s \le n-i+1} \left\{ \frac{\pi}{4}\sqrt{s} + \left(1 - \sum_{j=i}^{i+s-1} \Pr\left[d_j | \overline{d_{\le i-1}}\right]\right) OPT(i+s) \right\}$$

. Note that when picking $s = n - i + 1$ in the recurrence above we have cost $\frac{\pi}{4}\sqrt{s} = \frac{\pi}{4}\sqrt{n-i+1}$ since $OPT(i+s) = OPT(n+1) = 0$. □

We remark that calculating $OPT(1)$ requires calculating $OPT(i)$ for each i. For each i we remark that calculating $OPT(i)$ can be calculated in linear time by storing previous results from each partial sum. The total cost to calculate $OPT(1)$ is $O(n^2)$. Interestingly, we find that this dynamic programming algorithm produces solutions for the uniform distribution that exceed the performance of the typical single-threaded solution for Grover's algorithm. Specifically we note the following:

Theorem 3.4.3. *The cost of the partitioned version of Grover's algorithm running on the uniform distribution U_X over X, $Cost\,(U_X, P^*)$ is $\frac{\pi x}{4}\sqrt{n}$ where $0.978 < x < 0.9787$ for domains of size $n \geq 8^3$.*

Proof sketch: For the lower bound, let $C(n)$ be the cost of the optimal partition of U_n. Note $C(n) = OPT(1)$ and $C(i = OPT(n-i)$ under the uniform distribution. Plugging in $s = 7n/8$ into the recurrence gives $C(n) = OPT(1) \leq \frac{\pi}{4}\sqrt{\frac{7n}{8}} + \frac{1}{8}OPT\left(\frac{7n}{8}\right) = \frac{\pi}{4}\sqrt{\frac{7n}{8}} + \frac{1}{8}C\left(\frac{n}{8}\right)$. We can recursively argue that $C\left(\frac{n}{8^i}\right) \leq \frac{\pi}{4}\sqrt{\frac{7n}{8^{i+1}}} + \frac{1}{8}C(\frac{n}{8^{i+1}})$ and that we always have $C\left(\frac{n}{8^i}\right) \leq \frac{\pi}{4}\sqrt{\frac{n}{8^i}}$. Three levels of recursion produces the upper bound in the theorem statement.

The lower bound is generated through differential calculus optimization argument. We first consider whether $x > 0.978$. Consider the case $|X| = 8$. Note that $2.7662 = 0.978\sqrt{8} < \sqrt{7} + 0.125 = 2.7707$, so the claim holds for $|X| = 8$.
Assume $C(t) > 0.978\sqrt{t}\ \forall t < n$.
Let $c_1 = 0.978$. Now,

$$\begin{aligned} C(n) &= \min_{t \leq n}\left\{\sqrt{t} + \frac{t-n}{n}C(n-t)\right\} \\ &> \min_{t \leq n}\left\{\sqrt{t} + \frac{c_1(n-t)^{3/2}}{n}\right\} \end{aligned}$$

Let

$$f(t) = \sqrt{t} + \frac{c_1(n-t)^{3/2}}{n}$$

.

$$f(t) = \frac{1}{2\sqrt{t}} - \frac{3c_1\sqrt{n-t}}{2n}$$

Thus we can minimize at

$$t = \frac{9c_1^2 n \pm \sqrt{81c_1^4 n^2 + 36c_1^2 n^2}}{18c_1^2}$$

Note that the second root is negative, so we focus only on the first. Here we find that $t* = \frac{1647+\sqrt{1152089}n}{2934} \approx 0.8658n$.

3.4.4 Utility-maximizing Partitions

In Section 3.4.3 we noted the dynamic programming algorithm that minimizes the work (measured in iterations within Grover's algorithm) to run a Grover's algorithm attack for some element sampled from a distribution. Here we ask a related question: what if, rather than minimizing work, we wished to maximize the profit gained from running an attack?

First, we note that when maximizing profit instead of minimizing work we introduce the possibility of not checking all possible inputs with Grover's algorithm. We define $\widehat{\mathcal{P}}$ as the set of all possible subsets of partitions in $\mathcal{P}$. We now modify our earlier model by having an attacker $\mathcal{A}$ determine a value v representing the reward they obtain if the attack is successful. Second, we note that the attacker must fund the attack, which includes factors like labor, equipment costs, electricity costs, etc... Let $R(D_X, \widehat{P}, v)$ be the expected reward the attacker receives for running the attack with some partial partition $\widehat{P} \in \widehat{\mathcal{P}}$. If we consider a case where the attacker is running a typical Grover's algorithm attack on an entire distribution we have $R(D_X, \widehat{P}, v) = v$, as the attacker recovers the correct information with high probability. For some partial partition $\widehat{P}$ we have

$$R(D_X, v) = v \sum_{B_i \in \widehat{P}} \Pr[B_i].$$

The adversary also considers the cost function $C(D_X, \widehat{P}, k)$, where k is defined as the cost to run one iteration of Grover's algorithm. If we again consider an attacker running the traditional version of Grover's algorithm we would find that $C(D_X, k) = \frac{k\pi}{4}\sqrt{|X|}$. For an adversary operating over $\widehat{P} = \{B_1, B_2, \ldots, B_{|\widehat{P}|}\} \subseteq P$ we have

$$C(D_X, \widehat{P}, k) = \frac{k\pi}{4} \sum_{i=1}^{|\widehat{P}|} \Pr[B_i] \sum_{j=1}^{i} \sqrt{|B_j|}.$$

Finally, we define the Attacker $\mathcal{A}$'s profit $\texttt{Profit}(D_X, \widehat{P}, v, k) = R(D_X, \widehat{P}, v) - C(D_X, \widehat{P}, k)$. With this new function, we can turn our attention from minimizing work to the maximization of profit. We note that this change in the model allows for slightly more flexibility on the side of the attacker. For example, in Section 3.4.3 we assume that the attacker is optimizing an attack with the intent of trying every possible input. However, when optimizing profit we allow the attacker to choose to stop the attack partway through, a move they would want to take if their expected marginal profit for continuing the attack at some block drops below zero.

A Dynamic Programming Algorithm for Optimizing Profit

As in Section 3.4.3 we find that there is a dynamic programming algorithm that finds the optimal subset of a partition (which we refer to as a partial partition) that maximizes an attacker's profit. Fixing $D_X, v > 0, k > 0$ we define

$$OPT(i) = \max_{\widehat{P} \in \widehat{\mathcal{P}}(D_{\geq i})} \texttt{Profit}(D_{\geq i}, \widehat{P}, v, k).$$

where $OPT(1)$ is the maximum profit over all partial partitions of $D_{\geq i}$ as $D_{\geq 1} = D_X$. We first require a slight modification of Lemma 1 and Theorem 3.4.1 before we can derive a recurrence.

Lemma 2. *Let $\widehat{P} = \{B_1, B_2, \ldots, B_k\}$ be a subset of a partition $P \in \mathcal{P}$ of D_X. Given the work per Grover's iteration $k > 0$, reward valuation $v > 0$, and two blocks $B_x, B_y \in P$, $x < y$, containing a pair of elements $d_x \in B_x$ and $d_y \in B_y$ such that $d_y > d_x$, let the subset $\widehat{P'}$ be defined as a partial partition containing the same elements as $\widehat{P}$ where the blocks B_x and B_y are replaced with $B'_x = (B_x \setminus d_x) \cup \{d_y\}$ and $B'_y = (B_y \setminus d_y) \cup \{d_x\}$. $\mathbb{E}\left[\texttt{Profit}(D_X, \widehat{P'}, v, k)\right] > \mathbb{E}\left[\texttt{Profit}(D_X, \widehat{P}, v, k)\right]$.*

Proof. The difference between the expected profits using P vs P' is

$$\mathbb{E}\left[\texttt{Profit}(D_X, \widehat{P'}, v, k) - \texttt{Profit}(D_X, \widehat{P}, v, k)\right] = \mathbb{E}\left[C(D_X, \widehat{P}, k) - C(D_X, \widehat{P'}, k)\right]$$

due to the equal expected reward between the two partial partitions.

$$\sum_{i=1}^{|P|} \Pr[B_i] \sum_{j=1}^{i} \frac{\pi k}{4}\sqrt{|B_j|}$$
$$-\sum_{i=1}^{|P'|} \Pr[B_i'] \sum_{j=1}^{i} \frac{\pi k}{4}\sqrt{|B_j'|}.$$

Note that block sizes remain unchanged.

$$(\Pr[B_x] - \Pr[B_x']) \sum_{j=1}^{x} \frac{\pi k}{4}\sqrt{|B_j|}$$

$$-\left(\Big(\Pr[B_y] - \Pr\big[B_y'\big]\Big) \sum_{j=1}^{y} \frac{\pi k}{4}\sqrt{|B_j|}\right)$$

Let $p = d_X - d_Y$, note $p < 0$.

$$= p\left(\sum_{j=1}^{x} \frac{\pi k}{4}\sqrt{|B_j|}\right) - p\left(\sum_{j=1}^{y} \frac{\pi k}{4}\sqrt{|B_j|}\right)$$

$$= -p \sum_{j=x+1}^{y} \frac{\pi k}{4}\sqrt{|B_j|} > 0$$

Thus P' has a lower expected cost. ☐

We also note that in the case where some element in $supp(D_X) \setminus \bigcup \widehat{P}$ is greater than the smallest element in $\bigcup \widehat{P}$ a similar swapping lemma exists.

Lemma 3. *Let the partial partition* $\widehat{P} = \{B_1, B_2, \ldots, B_k\}$ *over* D_X *with* $v > 0, k > 0$ *have some block* $B_x \in \widehat{P}$ *and element* $d_y \in supp(D_X) \setminus \bigcup \widehat{P}$ *that is larger than an element* $d_x \in B_x$*. Let the partition* $\widehat{P'}$ *be the same as* $\widehat{P}$ *with the block* B_x *replaced with* $B_x' = (B_x \setminus d_x) \cup \{d_y\}$*.* $\mathbb{E}\left[\texttt{Profit}(D_x, \widehat{P'}, v, k)\right] > \mathbb{E}\left[\texttt{Profit}(D_X, \widehat{P}, v, k)\right]$*.*

Proof. Consider

$$\mathbb{E}\left[\texttt{Profit}\Big(D_X, \widehat{P'}, v, k\Big) - \texttt{Profit}\Big(D_X, \widehat{P}, v, k\Big)\right]$$

$$
\begin{aligned}
&= \left(\sum_{i=1}^{|\widehat{P}'|} v \Pr[B'_i] - \frac{k\pi}{4} \sum_{i=1}^{|\widehat{P}|} \Pr[B'_i] \sum_{j=1}^{i} \sqrt{|B'_j|} \right) \\
&- \left(\sum_{i=1}^{|\widehat{P}|} v \Pr[B_i] - \frac{k\pi}{4} \sum_{i=1}^{|\widehat{P}|} \Pr[B_i] \sum_{j=1}^{i} \sqrt{|B_j|} \right) \\
&= v \left(\sum_{i=1}^{|\widehat{P}'|} \Pr[B'_i] - \sum_{i=1}^{|\widehat{P}|} \Pr[B_i] \right) - \frac{k\pi}{4} (d_x - d_y) \sum_{j=1}^{x} \sqrt{|B_j|} \\
&= (d_y - d_x) \left(v - \frac{k\pi}{4} \sum_{j=1}^{x} \sqrt{|B_j|} \right) > 0
\end{aligned}
$$

where the last line follows as $d_y > d_x$.

$\square$

Theorem 3.4.4. *Any profit maximizing partition* $\widehat{P^*} \in \underset{\widehat{P} \in \widehat{\mathcal{P}}(D_X)}{\operatorname{argmax}} \texttt{Profit}(D_X, \widehat{P}, v, k)$ *has the form* $\widehat{P^*} = \{B_1, B_2, \ldots\}$ *where* $B_i = [S_{i-1} + 1, S_i]$ *for each* $i \geq 1$ *with* $S_0 = 0$ *and* $S_i = S_{i-1} + |B_i|$.

Proof. First note that a direct consequence of Lemma 3 have have
$D_X \backslash \bigcup \widehat{P^*} = \{d_{S_{|\widehat{P^*}|}+1}, d_{S_{|\widehat{P^*}|}+2}, \ldots, d_{|D_X|}\}$. If not, we would have some element $d_x \, in D_X \backslash \bigcup \widehat{P^*}$ greater than some element in $\bigcup \widehat{P^*}$. By Lemma 3 this was not the optimal partition.

Next, assume that the remaining elements in $\bigcup \widehat{P^*}$ are not contiguous, i.e., there is some pair of elements d_X, d_Y such that $d_x \in B_X$ and $d_y \in B_Y$, $X > Y$ and $d_y > \min_{d' \in B_X}$. By Lemma 2 we can contruct a partial partition with higher profit. Thus $\widehat{P^*}$ was not the optimal partial partition.

From these two facts we can see that $\widehat{P^*}$ is composed of blocks of the form $B_i = \{d_{S_{i-1}+1}, d_{S_{i-1}+2}, \ldots, d_{S_i}\}$. $\square$

Theorem 3.4.5. *If* $\widehat{P^*} \in \underset{\widehat{P} \in \widehat{\mathcal{P}}(D_{\geq i})}{\operatorname{argmax}} \texttt{Profit}(D_X, \widehat{P}, v, k)$ *and* $\widehat{P^*} = \{B_1, B_2, \ldots, B_{|\widehat{P^*}|}\}$ *has* $|B_1| = j$ *then* $\widehat{P^*}' = \{B_2, B_3, \ldots, B_{|\widehat{P^*}|}\} \in \underset{\widehat{P} \in \widehat{\mathcal{P}}(D_{\geq i+j})}{\operatorname{argmax}} \texttt{Profit}(D_X, \widehat{P}, v, k)$.

Proof. Assume $\widehat{P^*} \in \mathcal{P}_{D_{\geq i}}$ is the optimal partial partition for $D_{\geq i}$ and $\widehat{P^*}$ has $|B_1| = j$ but $\widehat{P^*}' = \{B_2, B_3, \ldots, B_{\widehat{P^*}}\}$ is not optimal for $D_{\geq i+j}$. Then let $\widehat{P}'' = \{B_1', B_2', \ldots, B_{|\widehat{P}'|}\}$ be the optimal partition for $D_{\geq i+j}$. Define $\widehat{P^*}'' = \{B_1\} \cup \widehat{P}''$.

$$
\begin{aligned}
&\mathbb{E}\left[\texttt{Profit}(D_X, \widehat{P^*}', v, k) - \texttt{Profit}(D_X, \widehat{P^*}, v, k)\right] \\
&= \texttt{Profit}(D_X, \{B_1\}, v, k) + (1 - \Pr[B_1])\,\texttt{Profit}(D_X, \widehat{P^*}'', v, k) \\
&- \texttt{Profit}(D_X\{B_1\}, v, k) - (1 - \Pr[B_1])\,\texttt{Profit}(D_X, \widehat{P^*}', v, k) \\
&= (1 - \Pr[B_1])\left(\texttt{Profit}(D_X, \widehat{P^*}'', v, k) - \texttt{Profit}(D_X, \widehat{P^*}', v, k)\right) > 0
\end{aligned}
$$

Meaning P^* was not the optimal partition for $D_{\geq i}$. By contradiction, if $|B_1| = j$ $\widehat{P^*}'$ is optimal and $P^* = \{B_1\} \cup \widehat{P^*}'$. ☐

The process for deriving a recurrence relation here is very similar to the one used in Section 3.4.3. We begin with the base case $OPT(|X|) = \max(0, v - \frac{\pi k}{4})$. We set $OPT(\xi) = 0$ where $\xi > |X|$ for convenience. Note that we have three possibilities for the optimal solution: (1) Check nothing i.e., the first bucket B_1 is empty, (2) The first bucket B_1 contains everything, or (3) $|B_1| = j$ for some j. In the third case the optimal partition will combine B_1 with the optimal partition for the remaining partition. By Theorem 3.4.5, the profit will be

$$
\begin{aligned}
&\max_{j>i}\left(v \sum_{x=i}^{j} \Pr\left[d_x | \overline{d_{\leq i-1}}\right] - \frac{k\pi}{4}\sqrt{n-i}\right) \\
&+ \left(1 - \sum_{x=i}^{j} \Pr\left[d_x | \overline{d_{\leq i-1}}\right]\right) OPT(j)
\end{aligned}
$$

The largest of these options is the maximum profit, thus

$$
OPT(i) = \max \begin{cases} v - \frac{k\pi}{4}\sqrt{n-i}, \\ \max\limits_{j>i}\left(v \sum\limits_{x=i}^{j} \Pr\left[d_x | \overline{d_{\leq i-1}}\right] - \frac{k\pi}{4}\sqrt{n-i}\right) \\ + \left(1 - \sum\limits_{x=i}^{j} \Pr\left[d_x | \overline{d_{\leq i-1}}\right]\right) OPT(j), \\ 0 \end{cases} .
$$

3.4.5 Empirical Analysis

We empirically evaluated password breaches using our dynamic programming algorithm to see what an optimal bucketized Grover's algorithm attack would look like in practice. Here we examine the results from the Phpbb leak [122] in the context of a partitioned Grover's algorithm attack. In our analysis we examine two different costs an adversary might consider when running this attack. The first, c_q, represents the cost, in USD, to compute a single round of Grover's algorithm. Similarly, c_c represents the cost to make one guess on a classical machine. We fix $v = 1$ for the purposes of this analysis. In our previous recurrence $OPT(i)$ we note you can update the recurrence to include an option where we check one password classically before continuing. The new recurrence with the new case is:

$$OPT(i) = \max \begin{cases} 1 - \frac{c_q \pi}{4}\sqrt{n-i}, \\ \max\limits_{j>i} \left(\sum\limits_{x=i}^{j} \Pr\left[d_x | \overline{d_{\leq i-1}}\right] - \frac{c_q \pi}{4}\sqrt{n-i} \right) \\ + \left(1 - \sum\limits_{x=i}^{j} \Pr\left[d_x | \overline{d_{\leq i-1}}\right] \right) OPT(j), \\ c_c + (1 - \Pr\left[d_i | \overline{d_{\leq i-11}}\right]) OPT(i+1), \\ 0 \end{cases} .$$

In our analysis we allow the attacker to run a hybrid attack where they are permitted to run certain partition blocks on a classical machine before switching to a quantum attack, or vice versa. Results for the Phpbb leak are shown in Figure 3.10. hatched areas represent classical guesses, while solid represents quantum attacks.

To highlight a significant example, consider the case in Figure 3.10 with $c_q = 3 \times 10^{-3}$ and $c_c = 3 \times 10^{-5}$. Here we find that the optimal partition will crack almost all passwords in the dataset in a perfect knowledge attack. However, running Grover's algorithm on the entire set (i.e., non-partitioned) is unprofitable since $1 - \frac{c_q \pi}{4}\sqrt{|X|} \approx -0.01$!

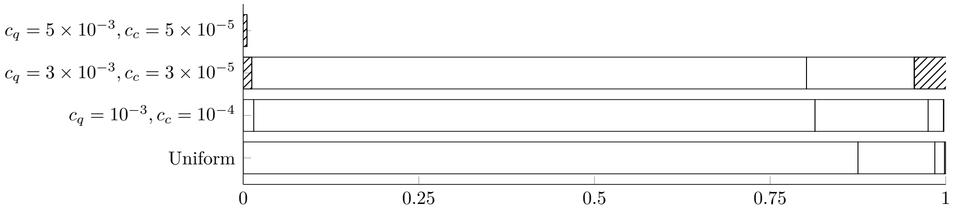

Figure 3.10. Partitions for phpbb leak

3.4.6 Password-cracking circuits

Grover's algorithm is often demonstrated as a search over some domain $D = \{0,1\}^n$ for an element $d \in D$, as might be appropriate when searching for a randomly generated symmetric encryption key. However, in this paper, we are generally considering attacks in the context of password cracking. While it may be theoretically possible to search over the entire space of passwords over some alphabet Σ, the sheer size of this domain would make such a task unwise. Instead, we would like to describe some method of only checking a predetermined subset of all possible passwords e.g., some set of passwords an attacker has obtained from a password breach.

To search over the support of some distribution $\mathit{supp}(D_P)$ over $P \subseteq \Sigma^*$ representing a password breach for some known salted hash value v we must compose several functions before it is in a form suitable for Grover's algorithm. Whatever function f we use must be in the final form $f : \{0,1\}^k \to \{0,1\}$. To find a specific element $p \in P$ that generates we construct the following functions which we then compose:

- $m : \{0,1\}^{\lceil \log_2(|\mathit{supp}(P)|)\rceil} \to P$ is an arbitrary mapping function that takes some index and maps it on to an element in P in a one-to-one fashion.

- $h : \Sigma^* \times \{0,1\}^s \to \{0,1\}^\mu$ is the password hashing function that was used in the leaked password database. $\{0,1\}^s$ represents the salt value for the password we are trying to crack.

- $\mathbb{1}_v : \{0,1\}^\mu \to \{0,1\}$ is an indicator function that outputs 1 if $h \circ m(x) = v$ and 0 otherwise.

With these functions we can use the function $\mathbb{1}_v \circ h \circ m(x) : \{0,1\}^{\lceil \log_2(|\mathit{supp}(P)|)\rceil} \to \{0,1\}$ as our search function for Grover's algorithm. Though this function satisfies the requirements to use Grover's algorithm, the functions m and h merit further description and discussion, as they constitute the bulk of the circuit size for this attack.

m serves to translate from a superposition of $\{0,1\}^{\lceil \log_2(|\mathit{supp}(P)|)\rceil}$ logical qubits to some element in our database of passwords P that we are checking. The straightforward solution to constructing m is to build it as a linear-size dictionary we can reference. To build this, we rely on a gadget $L_p(i)$ that copies a string $p_i \in P$ into a set of target qubits T if the input index i matches the input corresponding with p. Intuitively all this circuit does is read the input x and use it to index into a table, writing its output to some set of target qubits T.

The gadget L_p serves to check if the input index i matches an index associated with the password p, i_p. If $p = i_p$ then the encoded text of p is placed into a block of designated scratch qubits. It takes as input $\lambda = \lceil \log_2 |P| \rceil$ qubits representing an index and $2\lambda + 2\ell \lceil \log_2 |\Sigma| \rceil + \ell$ $\langle 0|\rangle$ qubits, where ℓ is the maximum length of a password in the set P. At the end of this gadget the password p has been placed into a designated set of ℓ qubits iff $i = i_p$.

The L_p gadget contains two main stages of computation - the index comparison and a conditional XOR into a set of target qubits.

A diagram of $L_p(i)$ is found in Figure 3.11.

The circuit that selects the appropriate password p based on an input index i requires a total of $|P|$ copies of L_p, one for each password $p \in P$. There is some flexibility in how these individual gadgets are combined. One extreme is to run all L_p gadgets in parallel as a single layer, while another is to run a single gadget at a time sequentially. Mixtures of these two strategies are also possible, running a subset of the L_p gadgets in one layer and stacking enough layers to check all elements. Diagrams of these strategies can be seen in Figures 3.11 and 3.12.

While this linear-size dictionary strategy for constructing a superposition of passwords accomplishes its goal it may also be possible to consider more advanced methods of generating this superposition of input passwords. More state-of-the-art password cracking methods take advantage of neural networks [82] or PCFGs [123], to generate large numbers of candidate passwords for use in a brute force attack.

In the context of the full Grover iteration circuit, we would like to know how these password-checking circuits affect the total circuit size. We have a total circuit width of $|0\rangle^{|P|width(L_{p_i}(i)} + \lambda$, as noted in Figure 3.12. Memory hard functions like Argon2 can require up to *billions* of qubits under the naive algorithm for computation. However, password lists like LinkedIn [10] can contain tens of millions of passwords with long passwords (e.g., > 20 characters) mixed in. In terms of width, this circuit may be comparable with the width of MHF computation when large password lists are used. However, the depth of these circuits

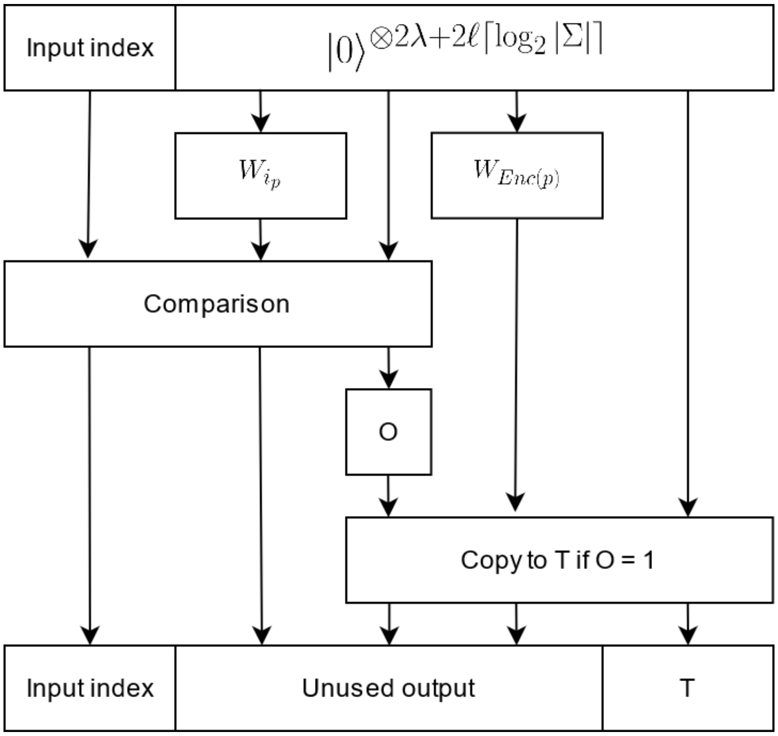

Figure 3.11. A lookup gadget to copy a single password into a set of target qubits

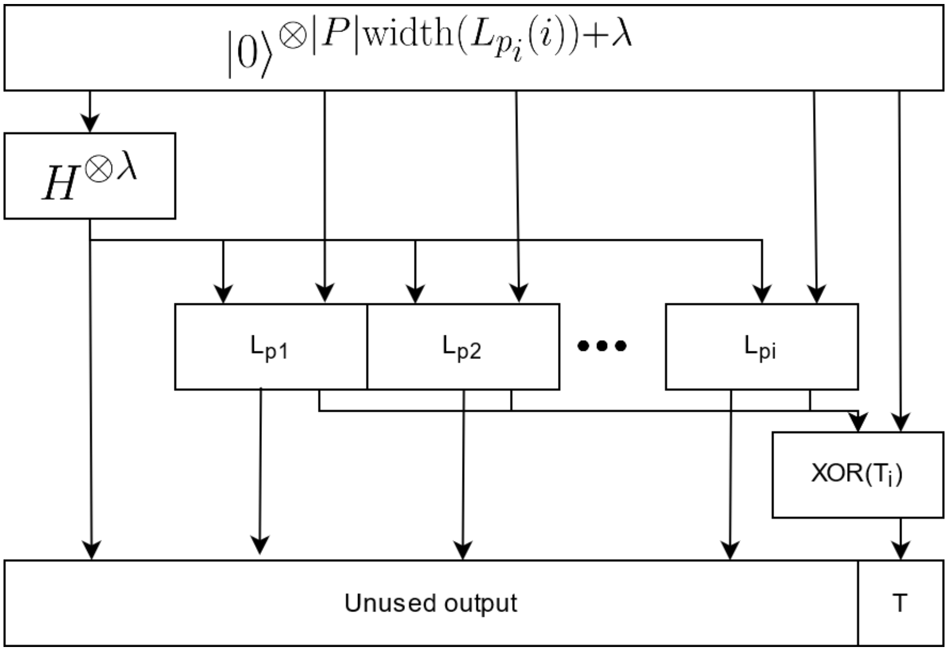

Figure 3.12. A circuit to place a superposition of passwords into an output target

is proportional to the log of password length from the comparison stages. This is *significantly* less than the depth required to compute a function like Argon2. In terms of the full cost to compute a Grover's algorithm iteration, the function is still dominated by the cost of computing the password hashing function, especially when MHFs are used.

3.4.7 Discussion

In this section, we have noted that Grover's algorithm attacks need not remain blind to the distribution over possible input elements. We have shown that there is a dynamic programming algorithm that can generate an optimal partition for a Grover's algorithm attack. This dynamic algorithm was able to find cases where attacks with negative attacker utility can be made positive by applying the partitioned version of the attack. These cases can be considered even more significant because they not only make an attack possible, they increase utility enough that the attacker is predicted to crack almost all passwords in the distribution. If these cost estimates were applied only in the context of a standard Grover's algorithm attack there would be a woeful underestimate of the dangers posed by a rational adversary. Thankfully, with this new model, we can predict how rational attackers behave when they are taking advantage of the partitioned version of Grover's algorithm. These new predictions may help us properly apply several of the results in Chapter 4, where methods to deter these attackers may depend on the specific costs related to an attack.

4. DETERRING RATIONAL ATTACKERS

This chapter contains sections taken verbatim from [6, 8, 7]

In Chapter 3 several papers describing methods of modelling rational attackers were discussed. In several of these cases, such as in [3], we find from our models that there exist attacks that rational attackers will perform. The natural next question is whether or not we can do anything to deter these attackers. In this chapter we will investigate several rational attacker deterrents, especially in cases like password hashing where our previous results have identified problem areas. Recall from our rational adversary model in Section 2 that we have two components of a rational adversary's utility that we can focus on to influence their behavior. The first is their reward function $R(a)$. While it is possible to influence the adversaries perceived reward (e.g., by trying to make the products of an attack harder to use and thus less desirable to someone who might buy it), it tends to be a more difficult approach than trying to affect their attack costs $C(a)$. Raising costs has a similar effect to reducing reward, by affecting the difference between these two factors we can influence rational adversaries to select actions that we consider less harmful. In each of these four papers we specifically focus on the context of password cracking, one of the most common attacks against imperfectly secure systems. By using rational adversary analysis, we are able to demonstrate how work to increase adversary costs influences them to select actions that we consider more desirable (in this case, reducing the number of compromised user accounts).

4.1 Information Signaling in Password Storage

Password hashing serves as a last line of defense against an offline password attacker. A good password hash function H should be moderately expensive to compute so that it becomes prohibitively expensive to check millions or billions of password guesses. However, we cannot make H too expensive to compute as the honest authentication server needs to evaluate H every time a user authenticates. In this paper, we explore a highly counter-intuitive defense against rational attackers which does not impact hashing costs: information signaling! In particular, we propose to have the authentication server store a (noisy) signal sig_u which is correlated with the strength of the user's password.

Traditionally, an authentication server stores the tuple $(u, salt_u, h_u)$ for each user u where $salt_u$ is a random salt value and $h_u = H(salt_u, pw_u)$ is the salted hash. We propose to have the authentication server instead store the tuple $(u, salt_u, sig_u, h_u)$, where the (noisy) signal sig_u is sampled based on the strength of the user's password pw_u. The signal sig_u is simply recorded for an offline attacker to find if the authentication server is breached. In fact, the authentication server never even uses sig_u when the user u authenticates[1]. At first glance, it seems highly counter-intuitive that the signal sig_u could help to protect user's against offline attackers. The attacker will only use the signal sig_u if it is beneficial — at minimum the attacker could always choose to ignore the signal.

To provide some intuition for why information signaling might be beneficial we observe that password cracking is not a zero-sum game between the defender and the password cracker. The defender's utility is inversely proportional to the fraction of user passwords that are cracked. By contrast, the attacker's utility is given by his reward, i.e., the value of all of the cracked passwords, minus his guessing costs. Thus, it is possible that password strength signaling would result in the attacker cracking fewer passwords.

Example 1

Suppose that we add a signal $sig_u = 1$ to indicate that user u's password pw_u is uncrackable (e.g., the entropy of the password is over 60-bits) and we add the signal $sig_u = 0$ otherwise. In this case, the attacker will simply choose to ignore accounts with $sig_u = 1$ to reduce his total guessing cost. However, the number of cracked user passwords stays unchanged.

---

[1] If a user u attempts to login with password pw the authentication server will lookup $salt_u$ and h_u and accept pw if and only if $h_u = H(salt_u, pw)$.

Example 2

Suppose that we modify the signaling scheme above so that even when the user's password pw_u is *not* deemed to be uncrackable we still signal $sig_u = 1$ with probability ϵ and $sig_u = 0$ otherwise. If the user's password is uncrackable we always signal $sig_u = 1$. Assuming that ϵ is not too large a rational attacker might still choose to ignore any account with $sig_u = 1$ i.e., the attacker's expected reward will decrease slightly, but the attacker's guessing costs will also be reduced. In this example, the fraction of cracked user passwords is reduced by up to ϵ i.e., any lucky user u with $sig_u = 1$ will not have their password cracked.

In this work, we explore the following questions: Can information signaling be used to protect passwords against rational attackers? If so, how can we compute the optimal signaling strategy?

Contributions

We introduce information signaling as a novel, counter-intuitive, defense against rational password attackers. We adapt a Stackelberg game-theoretic model of Blocki and Datta [17] to characterize the behavior of a rational password adversary and to characterize the optimal signaling strategy for an authentication server (defender). We analyze the performance of information signaling using several large password datasets: Bfield, Brazzers, Clixsense, CSDN, Neopets, 000webhost, RockYou, Yahoo! [79, 80], and LinkedIn [10]. We analyze our mechanism both in the idealistic setting, where the defender has perfect knowledge of the user password distribution $\mathcal{P}$ and value v of each cracked password, as well as in a more realistic setting where the defender only is given approximations of $\mathcal{P}$ and v. In our experiments, we analyze the fraction $x_{sig}(v)$ (resp. $x_{no-sig}(v)$) of passwords that a rational attacker would crack if the authentication server uses (resp. does not use) information signaling. We find that the reduction in the number of cracked passwords can be substantial e.g., $x_{no-sig}(v) - x_{sig}(v) \approx 8\%$ under empirical distribution and 13% under Monte Carlo distribution. We also show that information signaling can be used to help deter online attacks when CAPTCHAs are used for throttling.

An additional advantage of our information signaling method is that it is independent of the password hashing method and requires no additional hashing work. Implementation involves some determination of which signal to attach to a certain account, but beyond that, any future authentication attempts are handled exactly as they were before i.e. the signal information is ignored.

Password Representation

We use $\mathbb{P}$ to denote the set of all passwords that a user might select and we use $\mathcal{P}$ to denote a distribution over user-selected passwords i.e., a new user will select the password $pw \in \mathbb{P}$ with probability $\Pr_{x\sim\mathcal{P}}[x = pw]$ — we typically write $\Pr[pw]$ for notational simplicity.

Password Datasets

Given a set of N users $\mathcal{U} = \{u_1, \ldots, u_N\}$ the corresponding password dataset D_u is given by the multiset $D_u = \{pw_{u_1}, \ldots, pw_{u_N}\}$ where pw_{u_i} denotes the password selected by user u_i. Fixing a password dataset D we let f_i denote the number of users who selected the ith most popular password in the dataset. We note that that $f_1 \geq f_2 \geq \ldots$ and that $\sum_i f_i = N$ gives the total number N of users in the original dataset.

Empirical Password Distribution

Viewing our dataset D as N independent samples from the (unknown) distribution $\mathcal{P}$, we use f_i/N as an empirical estimate of the probability of the ith most common password pw_i and $D_f = (f_1, f_2, \ldots)$ as the corresponding frequency list. In addition, $\mathcal{D}_e$ is used to denoted the corresponding empirical distribution i.e., $\Pr_{x\sim\mathcal{D}_e}[x = pw_i] = f_i/N$. Because the real distribution $\mathcal{P}$ is unknown we will typically work with the empirical distribution $\mathcal{D}_e$. We remark that when $f_i \gg 1$ the empirical estimate will be close to the actual distribution i.e., $\Pr[pw_i] \approx f_i/N$, but when f_i is small the empirical estimate will likely diverge from the true probability value. Thus, while the empirical distribution is useful to analyze the performance of information signaling, when the password value v is small this analysis will be less accurate for larger values of v i.e., once the rational attacker has the incentive to start cracking passwords with lower frequency.

Monte Carlo Password Distribution

Following [30] we also use the Monte Carlo Password Distribution $\mathcal{D}_m$ to evaluate the performance of our password signaling mechanism when v is large. The Monte Carlo distribution is derived by subsampling passwords from our dataset D, generating guessing numbers from state-of-the-art password cracking models, and fitting a distribution to the resulting guessing curve. See more details in section 4.1.

Password Equivalence Set

It is often convenient to group passwords having (approximately) equal probability into an *equivalence set es* both for empirical distribution and Monte Carlo distribution. Suppose there are n' equivalence set, we typically have $n \ll N$ in password datasets e.g., for the LinkedIn empirical dataset we have $N \geq 1.7 \times 10^8$ while $n = 3639$. Thus, an algorithm whose running time scales with n is much faster than an algorithm whose running time scales with N.

Differential Privacy and Count Sketches

As part of our information signaling, we need a way for the authentication server to estimate the strength of each user's passwords. We propose to do this with a (differentially private) Count-Sketch data structure, which allows us to approximately determine how many users have selected each particular password. As a side-benefit, the authentication server could also use the Count-Sketch data structure to identify/ban overly popular passwords [124] and to defend against online guessing attacks [125, 126]. We first introduce the notion of differential privacy.

ϵ-Differential Privacy

ϵ-Differential Privacy [127] is a mechanism that provides strong information-theoretic privacy guarantees for all individuals in a dataset. Formally, an algorithm $\mathcal{A}$ preserves ϵ-differential privacy iff for all datasets D and D that differ by only one element and all subsets S of Range($\mathcal{A}$):

$$\Pr\left[\mathcal{A}(D) \in S\right] \leq \mathrm{e}^{\epsilon} \Pr\left[\mathcal{A}(D) \in S\right].$$

In our context, we can think of D (resp. D) as a password dataset which does (resp. does not) include our user u's password pw_u and we can think of $\mathcal{A}$ as a randomized algorithm that outputs a noisy count-sketch algorithm. Intuitively, differential privacy guarantees that an attacker cannot even tell if pw_u was included when the count-sketch was generated. In particular, (up to a small multiplicative factor e^{ϵ}) the attacker cannot tell the difference between $\mathcal{A}(D)$ and $\mathcal{A}(D)$ the count-sketch we sample when pw_u was (resp. was not) included. Thus, whatever the attacker hopes to know about u's from $\mathcal{A}(D)$ the attacker could have learned from $\mathcal{A}(D)$.

Count-sketch

A count sketch over some domain E is a probabilistic data structure that stores some information about the frequency of items seen in a stream of data — in our password context we will use the domain $E = \mathbb{P}$. A count-sketch functions as a table T with width w_s columns and depth d_s rows. Initially, $T[i, j] = 0$ for all $i \leq w_s$ and $j \leq d_s$. Each row is associated with a hash function $H_i : \mathbb{P} \to [w_s]$, with each of the hash functions used in the sketch being pairwise independent.

To insert an element $pw \in \mathbb{P}$ into the count sketch we update $T[i, H_i(pw)] \leftarrow T[i, H_i(pw)] + 1$ for each $i \leq d_s$ [2]. To estimate the frequency of pw we would output $f\left(T[1, H_1(pw)], \ldots, T[d_s, H_{d_s}(pw)]\right)$ for some function $f : \mathbb{N}^{d_s} \to \mathbb{N}$. In our experiments we instantiate a Count-Mean-Min Sketch where $f = \texttt{median}\left\{T[i, H_i(pw)] - \frac{\#total - T[i, H_i(pw)]}{d_w - 1} : i = 1, \ldots, d_s\right\}$ ($\#total$ is the total number of

---

[2] In some instantiations of count sketch we would instead set $T[i, H_i(pw)] \leftarrow T[i, H_i(pw)] + G_i(pw)$ where the hash function $G_i : \mathbb{P} \to \{-1, 1\}$

elements being inserted) so that bias is subtracted from overall estimate. Other options are available too, e.g., $f = \texttt{min}$ (Count-Min), $f = \texttt{mean}$ (Count-Mean-Sketch) and $f = \texttt{median}$ (Count-Median) [3].

Oserve that adding a password only alters the value of $T[i, j]$ at d_s locations. Thus, to preserve ϵ-differential privacy we can initialize each cell $T[i, j]$ by adding Laplace noise with scaling parameter d_s/ϵ [128].

Other Notation

Given a permutation π over all allowable passwords $\mathbb{P}$ we let $\lambda(\pi, B) := \sum_{i=1}^{B} \Pr\left[pw_i^\pi\right]$ denote the probability that a randomly sampled password $pw \in \mathbb{P}$ would be cracked by an attacker who checks the first B guesses according to the order π — here pw_i^π is the ith password in the sequence π. Given an randomized algorithm $\mathcal{A}$ and a random string r we use $y \leftarrow \mathcal{A}(x; r)$ to denote the output when we run $\mathcal{A}$ with input x fixing the outcome of the random coins to be r. We use $y \overset{\$}{\leftarrow} \mathcal{A}(x)$ to denote a random sample drawn by sampling the random coins r uniformly at random. Given a randomized (signaling) algorithm $\mathcal{A} : \mathbb{P} \to [0, b-1]$ (where b is the total number of signals) we define the conditional probability $\Pr[pw \mid y] := \Pr_{x \sim \mathcal{P}, r}\left[x = pw \mid y = \mathcal{A}(pw)\right]$ and

$$\lambda(\pi, B; y) := \sum_{i=1}^{B} \Pr[pw_i^\pi \mid y] \ .$$

We remark that $\Pr[pw \mid y]$ can be evaluated using Bayes Law given knowledge of the signaling algorithm $\mathcal{A}(x)$.

In this section, we overview our basic signaling mechanism deferring until later how to optimally tune the parameters of the mechanism to minimize the number of cracked passwords.

---

[3] Count-Median Sketch uses a different insersion method

Account Creation and Signaling

When users create their accounts they provide a user name u and password pw_u. First, the server runs the canonical password storage procedure—randomly selecting a salt value $salt_u$ and calculating the hash value $h_u = H(salt_u, pw_u)$. Next, the server calculates the (estimated) strength $str_u \leftarrow \mathsf{getStrength}(pw_u)$ of password pw_u and samples the signal $sig_u \overset{\$}{\leftarrow} \mathsf{getSignal}(st_u)$. Finally, the server stores the tuple $(u, salt_u, sig_u, h_u)$ — later if the user u attempts to login with a password pw the authentication server will accept pw if and only if $h_u = H(salt_u, pw)$. The account creation process is formally presented in Algorithm 1.

Algorithm 1 Signaling during Account Creation

```
Input: $u$, $pw_u$, $L$, $d$
$salt_u \overset{\$}{\leftarrow} \{0,1\}^L$
$h_u \leftarrow H(salt_u, pw_u)$
$str_u \leftarrow \mathsf{getStrength}(pw_u)$
$sig_u \overset{\$}{\leftarrow} \mathsf{getSignal}(str_u)$
$\mathsf{StoreRecord}(u, salt_u, sig_u, h_u)$
```

A traditional password hashing solution would simply store the tuple $(u, salt_u, h_u)$ i.e., excluding the signal sig_u. Our mechanism requires two additionally subroutines $\mathsf{getStrength}()$ and $\mathsf{getSignal}()$ to generate this signal. The first algorithm is deterministic. It takes the user's password pw_u as input and outputs str_u — (an estimate of) the password strength. The second randomized algorithm takes the (estimated) strength parameter str_u and outputs a signal sig_u. The whole signaling algorithm is the composition of these two subroutines i.e., $\mathcal{A} = \mathsf{getSignal}(\mathsf{getStrength}(pw))$. We use $s_{i,j}$ to denote the probability of observing the signal $sig_u = j$ given that the estimated strength level was $str_u = i$. Thus, $\mathsf{getSignal}()$ can be encoded using a signaling matrix $\mathbf{S}$ of dimension $a \times b$, i.e.,

$$\begin{bmatrix} s_{0,0} & s_{0,1} & \cdots & s_{0,b-1} \\ s_{1,0} & s_{1,1} & \cdots & s_{1,b-1} \\ \vdots & \vdots & \ddots & \vdots \\ s_{a-1,0} & s_{a-1,1} & \cdots & s_{a-1,b-1} \end{bmatrix},$$

where a is the number of strength levels that passwords can be labeled, b is the number of signals the server can generate and $\mathbf{S}[i,j] = s_{i,j}$.

We remark that for some signaling matrices (e.g., if $\mathbf{S}[i,0] = 1$ for all i [4]) then the actual signal sig_u is *uncorrelated* with the password pw_u. In this case our mechanism is equivalent to the traditional (salted) password storage mechanism where $\mathsf{getSignal}()$ is replaced with a constant/null function. $\mathsf{getStrength}()$ is password strength oracle that outputs the actual/estimated strength of a password. We discuss ways that $\mathsf{getStrength}()$ could be implemented in Section 4.1. For now, we omit the implementation details of strength oracle $\mathsf{getStrength}()$ for sake of readability.

Generating Signals

We use $[a] = 0, 1, \ldots, a-1$ (resp. $[b] = 0, 1, \ldots, b-1$) to denote the range of $\mathsf{getStrength}()$ (resp. $\mathsf{getSignal}()$). For example, if $[a] = \{0, 1, 2\}$ then 0 would correspond to weak passwords, 2 would correspond to strong passwords and 1 would correspond to medium strength passwords. To generate signal for pw_u, the server first invokes subroutine $\mathsf{getStrength}(pw_u)$ to get strength level $str_u = i \in [a]$ of pw_u, then signals $sig_u = j \in [b]$ with probability $\Pr[\mathsf{getSignal}(pw_u) = j \mid \mathsf{getStrength}(pw_u) = i] = \mathbf{S}[i,j] = s_{i,j}$.

Bayesian Update

An attacker who breaks into the authentication server will be able to observe the signal sig_u and $\mathbf{S}$. After observing the signal $sig_u = y$ and $\mathbf{S}$ the attacker can perform a Bayesian update. In particular, given any password $pw \in \mathbb{P}$ with strength $i = \mathsf{getStrength}(pw)$ we have

$$\begin{aligned} \Pr\left[pw \mid y\right] &= \frac{\Pr[pw]\mathbf{S}[i,y]}{\sum_{pw \in \mathbb{P}} \Pr\left[\mathsf{getSignal}\left(\mathsf{getStrength}(pw)\right)\right] \cdot \Pr\left[pw\right]} \\ &= \frac{\Pr[pw]\mathbf{S}[i,y]}{\sum_{i \in [a]} \Pr_{pw \sim \mathbb{P}}\left[\mathsf{getStrength}(pw) = i\right] \cdot \mathbf{S}[i,y]} \end{aligned} \tag{4.1}$$

---

[4] The index of matrix elements start from 0

If the attacker knew the original password distribution $\mathcal{P}$ then s/he can update posterior distribution $\mathcal{P}_y$ with $\Pr_{x\sim\mathcal{P}_y}[x = pw] := \Pr[pw \mid y]$. We extend our notation, let $\lambda(\pi, B; y) = \sum_{i=1}^{B} \Pr[pw_i^\pi \mid y]$ where pw_i^π is the ith password in the ordering π. Intuitively, $\lambda(\pi, B; y)$ is the conditional probability of cracking the user's password by checking the first B guesses in permutation π.

Delayed Signaling

In some instances, the authentication server might implement the password strength oracle $\mathsf{getStrength}()$ by training a (differentially private) Count-Sketch based on the user-selected passwords $pw_u \sim \mathcal{P}$. In this case, the strength estimation will not be accurate until a larger number N of users have registered. In this case, the authentication server may want to delay signaling until after the Count-Sketch has been initialized. In this case the authentication server will store the tuple $(u, salt_u, sig_u = \perp, h_u)$. During the next (successful) login with the password pw_u we can update $sig_u = \mathsf{getSignal}(\mathsf{getStrength}(pw_u))$.

4.1.1 Adversary Model

We adapt the economic model of [17] to capture the behavior of a rational attacker. We also make several assumptions: (1) there is a value v_u for each password pw_u that the attacker cracks; (2) the attacker is untargeted and that the value $v_u = v$ for each user $u \in U$; (3) by Kerckhoffs's principle, the password distribution $\mathcal{P}$ and the signaling matrix are known to the attacker.

Value/Cost Estimates

One can derive a range of estimates for v based on black market studies e.g., Symantec reported that passwords generally sell for \$4—\$30 [83] and [91] reported that Yahoo! e-mail passwords sold for $\approx$ \$1. Similarly, we assume that the attacker pays a cost k each time he evaluates the hash function H to check a password guess. We remark that one can estimate $k \approx \$1 \times 10^{-7}$ if we use a memory-hard function [5].

Adversary Utility: No Signaling

We first discuss how a rational adversary would behave when is no signal is available (traditional hashing). We defer the discussion of how the adversary would update his strategy after observing a signal y to the next section. In the no-signaling case, the attacker's strategy (π, B) is given by an ordering π over passwords $\mathbb{P}$ and a threshold B. Intuitively, this means that the attacker will check the first B guesses in π and then give up. The expected reward for the attacker is given by the simple formula $v \times \lambda(\pi, B)$, i.e., the probability that the password is cracked times the value v. Similarly, the expected guessing cost of the attacker is

$$C(k, \pi, B) = k \sum_{i=1}^{B} (1 - \lambda(\pi, i-1)), \tag{4.2}$$

Intuitively, $(1 - \lambda(\pi, i-1))$ denotes the probability that the adversary actually has to check the ith password guess at cost k. With probability $\lambda(\pi, i-1)$ the attacker will find the password in the first $i - 1$ guesses and will not have to check the ith password guess pw_i^{π}. Specially, we define $\lambda(\pi, 0) = 0$. The adversary's expected utility is the difference of expected gain and expected cost, namely,

$$U_{adv}(v, k, \pi, B) = v \cdot \lambda(\pi, B) - C(k, \pi, B). \tag{4.3}$$

Sometimes we omit parameters in the parenthesis and just write U_{adv} for short when the v, k and B are clear from context.

---

[5] The energy cost of transferring 1GB of memory between RAM and cache is approximately $0.3J$ on an [129], which translates to an energy cost of $\approx \$3 \times 10^{-8}$ per evaluation. Similarly, if we assume that our MHF can be evaluated in 1 second [16, 7] then evaluating the hash function 6.3×10^{7} times will tie up a 1GB RAM chip for 2 years. If it costs \$5 to rent a 1GB RAM chip for 2 years (equivalently purchase the RAM chip which lasts for 2 years for \$5) then the capital cost is $\approx \$8 \times 10^{-8}$. Thus, our total cost would be around $\$10^{-7}$ per password guess.

Optimal Attacker Strategy: Without Signaling

A rational adversary would choose $(\pi^*, B^*) \in \arg\max U_{adv}(v, k, \pi, B)$. It is easy to verify that the optimal ordering π^* is always to check passwords in descending order of probability. The probability that a random user's account is cracked is

$$P_{adv} = \lambda(\pi^*, B^*). \tag{4.4}$$

We remark that in practice $\arg\max U_{adv}(v, k, \pi, B)$ usually returns a singleton set (π^*, B^*). If instead the set contains multiple strategies then we break ties adversarially i.e.,

$$P_{adv} = \max_{(\pi^*, B^*) \in \arg\max U_{adv}(v,k,\pi,B)} \lambda(\pi^*, B^*).$$

4.1.2 Information Signaling as a Stackelberg Game

We model the interaction between the authentication server (leader) and the adversary (follower) as a two-stage Stackelberg game. In a Stackelberg game, the leader moves first and then the follower may select its action after observing the action of the leader.

In our setting the action of the defender is to commit to a signaling matrix $\mathbf{S}$ as well as the implementation of $\mathsf{getStrength}()$ which maps passwords to strength levels. The attacker responds by selecting a cracking strategy $(\vec{\pi}, \vec{B}) = \{(\pi_0, B_0), \ldots, (\pi_{b-1}, B_{b-1})\}$. Intuitively, this strategy means that whenever the attacker observes a signal y he will check the top B_y guesses according to the ordering π_y.

Attacker Utility

If the attacker checks the top B_y guesses according to the order π_y then the attacker will crack the password with probability $\lambda(\pi_y, B_y; y)$. Recall that $\lambda(\pi_y, B_y; y)$ denotes the probability of the first B_y passwords in π_y according to the posterior distribution $\mathcal{P}_y$ obtained by applying Bayes Law after observing a signal y. Extrapolating from no signal case, the expected utility of adversary conditioned on observing the signal y is

$$U_{adv}(v,k,\pi_y,B_y;\mathbf{S},y) = v\cdot\lambda(\pi_y,B_y;y) - \sum_{i=1}^{B_y} k\cdot\left(1-\lambda(\pi_y,i-1;y)\right), \tag{4.5}$$

where B_y and π_y are now both functions of the signal y. Intuitively, $(1-\lambda(\pi_y,i-1;y))$ denotes the probability that the attacker has to pay cost k to make the ith guess. We use $U^s_{adv}\left(v,k,\{\mathbf{S},(\vec{\pi},\vec{B})\}\right)$ to denote the expected utility of the adversary with information signaling,

$$U^s_{adv}\left(v,k,\{\mathbf{S},(\vec{\pi},\vec{B})\}\right) = \sum_{y\in[b]} \Pr[Sig=y]U_{adv}(v,k,\pi_y,B_y;\mathbf{S},y)\ , \tag{4.6}$$

where

$$Pr[Sig=y] = \sum_{i\in[b]} \Pr_{pw\sim\mathcal{P}}\left[\mathsf{getStrength}(pw)=i\right]\cdot S[i,y]\ .$$

Optimal Attacker Strategy

Now we discuss how to find the optimal strategy $(\vec{\pi}^*,\vec{B}^*)$. Since the attacker's strategies in reponse to different signals are independent. It suffices to find
$(\pi^*_y,B^*_y)\in\arg\max_{B_y,\pi_y}U_{adv}(v,k,\pi_y,B_y;y)$ for each signal y. We first remark that the adversary can obtain the optimal checking sequence π^*_y for pw_u associated with signal y by sorting all $pw\in\mathcal{P}$ in descending order of posterior probability according to the posterior distribution $\mathcal{P}_y$.

After the optimal checking sequence π^*_y being specified, the adversary can determine the optimal budget B^*_y for signal y such that $B^*_y = \arg\max_{B_y}U_{adv}(v,k,\pi^*_y,B_y;y)$. It is proved in [30] that the optimal budget can be found more efficiently when given a compact representation of password dataset.

We observe that an adversary who sets $\pi_y=\pi$ and $B_y=B$ for all $y\in[b]$ is effectively ignoring the signal and is equivalent to an adversary in the no signal case. Thus we have

$$\max_{\vec{\pi},\vec{B}} U^s_{adv}\left(v,k,\{\mathbf{S},(\vec{\pi},\vec{B})\}\right) \geq \max_{\pi,B} U_{adv}(v,k,\pi,B),\ \forall\mathbf{S}, \tag{4.7}$$

implying that adversary's expected utility will never decrease by adapting its strategy according to the signal.

Optimal Signaling Strategy

Once the function `getStrength`() is fixed we want to find the optimal signaling matrix $\mathbf{S}$. We begin by introducing the defender's utility function. Intuitively, the defender wants to minimize the total number of cracked passwords.

Let $P_{adv}^{s}\left(v,k,\mathbf{S}\right)$ denote the expected adversary success rate with information signaling when playing with his/her optimal strategy, then

$$P_{adv}^{s}\left(v,k,\mathbf{S}\right) = \sum_{y \in SL} \Pr[Sig = y] \lambda(\pi_y^*, B_y^*; \mathbf{S}, y), \tag{4.8}$$

where (π_y^*, B_y^*) is the optimal strategy of the adversary when receiving signal y, namely,

$$(\pi_y^*, B_y^*) = \arg\max_{\pi_y, B_y} U_{adv}(v, k, \pi_y, B_y; \mathbf{S}, y).$$

If $\arg\max_{\pi_y, B_y} U_{adv}(v, k, \pi_y, B_y; y)$ returns a set, we break ties adversarially.

The objective of the server is to minimize $P_{adv}^{s}\left(v,k,\mathbf{S}\right)$, therefore we define

$$U_{ser}^{s}\left(v, k, \{\mathbf{S}, (\vec{\pi}^*, \vec{B}^*)\}\right) = -P_{adv}^{s}\left(v,k,\mathbf{S}\right). \tag{4.9}$$

Our focus of this paper is to find the optimal signaling strategy, namely, the signaling matrix $\mathbf{S}^*$ such that $\mathbf{S}^* = \arg\min_{\mathbf{S}} P_{adv}^{s}\left(v,k,\mathbf{S}\right)$. Finding the optimal signaling matrix $\mathbf{S}^*$ is equivalent to solving the mixed strategy Subgame Perfect Equilibrium (SPE) of the Stackelberg game. At SPE no player has the incentive to derivate from his/her strategy. Therefore,

$$\begin{cases} U_{ser}^{s}\left(v, k, \{\mathbf{S}^*, (\vec{\pi}^*, \vec{B}^*)\}\right) \geq U_{ser}^{s}\left(v, k, \{\mathbf{S}, (\vec{\pi}^*, \vec{B}^*)\}\right), & \forall \mathbf{S} \in \mathbf{S}^{a \times b}, \\ U_{adv}^{s}\left(v, k, \{\mathbf{S}^*, (\vec{\pi}^*, \vec{B}^*)\}\right) \geq U_{adv}^{s}\left(v, k, \{\mathbf{S}^*, (\vec{\pi}, \vec{B})\}\right), & \forall (\vec{\pi}, \vec{B}). \end{cases} \tag{4.10}$$

Notice that a signaling matrix of dimension $a \times b$ can be fully specified by $a(b-1)$ variables since the elements in each row sum up to 1. Fixing v and k, we define $f : \mathbb{R}^{a(b-1)} \to \mathbb{R}$ to be the map from $\mathbf{S}$ to $P_{adv}^{s}(v, k, \mathbf{S})$. Then we can formulate the optimization problem as

$$\begin{aligned} \min_{\mathbf{S}} \quad & f(s_{0,0}, \dots s_{0,(b-2)}, \dots, s_{(a-1),0}, s_{(a-1),(b-2)}) \\ \text{s.t.} \quad & 0 \le s_{i,j} \le 1,\ \forall 0 \le i \le a-1,\ 0 \le j \le b-2, \\ & \sum_{j=0}^{b-2} s_{i,j} \le 1,\ \forall 0 \le i \le a-1. \end{aligned} \tag{4.11}$$

The feasible region is a $a(b-1)$-dimensional probability simplex. Notice that in 2-D ($a = b = 2$), the second constraint would be equivalent to the first constraint. In our experiments, we will treat f as a black box and use derivative-free optimization methods to find good signaling matrices $\mathbf{S}$.

4.1.3 Theoretical Example

Having presented our Stackelberg Game model for information signaling we now give an (admittedly contrived) example of a password distribution where information signaling can dramatically reduce the percentage of cracked passwords. We assume that the attacker has value $v = 2k + \epsilon$ for each cracked password where the cost of each password guess is k and $\epsilon > 0$ is a small constant.

Password Distribution

Suppose that $\mathbb{P} = \{\mathrm{pw}_i\}_{i \ge 1}$ and that each password pw_i has probability 2^{-i} i.e., $\Pr_{pw \sim \mathcal{P}}[pw = i] = 2^{-i}$. The weakest password pw_1 would be selected with probability 1/2.

Optimal Attacker Strategy without Signaling

By checking passwords in descending order of probability (the checking sequence is π) the adversary has an expected cost of:

$$C(k, \pi, B) = k \sum_{i=1}^{B} i \times 2^{-i} + 2^{-B} \times B \times k = k(2 - 2^{1-B}),$$

and an expected reward of

$$R(v, k, \pi, B) = v \sum_{i=1}^{B} i 2^{-i},$$

which leads to expected profits of

$$U_{adv}(v, k, \pi, B) = R(v, k, B) - C(k, \pi, B) = (v - 2k) + (2k - v)2^{-B}.$$

A profit-motivated adversary is interested in calculating $B^* = \underset{B}{\operatorname{argmax}}\ U_{adv}(v, k, \pi, B)$. With our sample distribution we have

$$B^* = \begin{cases} 0, v <= 2k, \\ \infty, v > 2k. \end{cases}$$

Since we assume that $v = 2k + \epsilon > 2k$ the attackers optimal strategy is $B^* = \infty$ meaning that 100% of passwords will be cracked.

Signaling Strategy

Suppose that $\mathsf{getStrength}$ is define such that $\mathsf{getStrength}(pw_1) = 0$ and $\mathsf{getStrength}(pw_i) = 1$ for each $i > 1$. Intuively, a the strength level is 0 if and only if we sampled the weakest password from the distribution. Now suppose that we select our signaling matrix

$$\mathbf{S} = \begin{bmatrix} 1/2 & 1/2 \\ 0 & 1 \end{bmatrix},$$

such that $\Pr[Sig = 0 \mid pw = pw_1] = \frac{1}{2} = \Pr[Sig = 1 \mid pw = pw_1]$ and $\Pr[Sig = 1 \mid pw \neq pw_1] = 1$.

Optimal Attacker Strategy with Information Signaling

We now analyze the behavior of a rational attacker under signaling when given this signal matrix and password distribution. Consider the strategy of an attacker who has $v = 2k + \epsilon$. As noted above their optimal guessing number B^* with no signal is $B^* = \infty$.

We now consider the case that the attacker facing $Sig = 1$. Note that $\Pr[Sig = 1] = \Pr[pw = pw_1] \Pr[Sig = 1 \mid pw = pw_1] + \Pr[pw \neq pw_1] \Pr[Sig = 1 \mid pw \neq pw_1] = \frac{3}{4}$.

We have the following posterior probabilities for each of the passwords in the distribution:

$$\begin{aligned} \Pr[pw = pw_1 \mid Sig = 1] &= \frac{0.5 * 0.5}{0.75} = \frac{1}{3}, \\ \Pr[pw = pw_i, i > 1 \mid Sig = 1] &= \frac{1 * 2^{-i}}{0.75} = \frac{4 * 2^{-i}}{3}. \end{aligned}$$

Now we compute the attacker's expected costs conditioned on $Sig = 1$.

$$C(k, \pi, B; \mathbf{S}, 1) = k\left(\frac{1}{3} + \frac{4}{3}\sum_{i=2}^{B} i * 2^{-i}\right) + kB\left(1 - \frac{1}{3} - \frac{4}{3}\left(\sum_{i=2}^{B} 2^{-i}\right)\right) = k\left(\frac{7}{3} - \frac{2^{3-B}}{3}\right),$$

when $B > 0$, with $C(k, \pi, 0; \mathbf{S}, 1) = 0$. For expected reward we have:

$$R(v, k, \pi, B; \mathbf{S}, 1) = v\left(\frac{1}{3} + \frac{4}{3}\sum_{i=2}^{B} 2^{-i}\right) = v\left(1 - \frac{2^{2-B}}{3}\right),$$

where $R(v, k, 0; \mathbf{S}, 1) = 0$ in the case where no guesses are made. Thus, the attacker's profit is given by:

$$U_{adv}(v, k, \pi, B; \mathbf{S}, 1) = R(v, k, \pi, B; \mathbf{S}, 1) - C(k, \pi, B; \mathbf{S}, 1) = v\left(1 - \frac{2^{2-B}}{3}\right) - k\left(\frac{7}{3} - \frac{2^{3-B}}{3}\right).$$

Plugging in $v = 2k + \epsilon$ we have

$$\begin{aligned} U_{adv}(2k+\epsilon, k, \pi, B; \mathbf{S}, 1) &= (2k+\epsilon)\left(1 - \frac{2^{2-B}}{3}\right) - k\left(\frac{7}{3} - \frac{2^{3-B}}{3}\right) \\ = (2k+\epsilon)\left(1 - \frac{2^{2-B}}{3}\right) - k\left(\frac{7}{3} - \frac{2^{3-B}}{3}\right) &= -\frac{1}{3}k + \epsilon\left(1 - \frac{2^{2-B}}{3}\right), \end{aligned}$$

if $\epsilon < \frac{1}{3}k$ then this value will always be negative and the optimal strategy is to select $B^* = 0$ i.e. to not run the attack[6]

If the attacker observes the signal $Sig = 0$ we know for sure that the user selected the most common password as $\Pr[pw = pw_1 \mid Sig = 0] = 1$ so as long as $v \geq k$ the attacker will crack the password.

Discussion

In our example an attacker with value $v = 2k + \epsilon$ cracks 100% of passwords when we don't use information signaling. However, if our information signaling mechanism (above) were deployed, the attacker will only crack 25% of passwords — a reduction of 75%! Given this (contrived) example it is natural to ask whether or not information signaling produces similar results for more realistic password distributions. We explore this question in the next sections.

4.1.4 Empirical Results

We now describe our empirical experiments to evaluate the performance of information signaling. Fixing the parameters v, k, a, b, a password distribution $\mathcal{D}$ and the strength oracle $\mathsf{getStrength}(\cdot)$ we define a procedure $\mathbf{S}^* \leftarrow \mathsf{genSigMat}(v, k, a, b, \mathcal{D})$ which uses derivate-free optimization to find a good generate a signaling matrix $\mathbf{S}^*$ of dimension $a \times b$ using the optimization problem defined in equation (4.11). Similarly, given a signaling matrix $\mathbf{S}^*$ we define a procedure $\mathsf{evaluate}(v, k, a, b, \mathbf{S}^*, \mathcal{D})$ which returns the percentage of passwords that

---

[6] If we set $v = 4k$ instead we would have $U_{adv}(4k, k, \pi, B; \mathbf{S}, 1) = k\left(\frac{5}{3} - \frac{2^{3-B}}{3}\right)$ which is maximized at $B^* = \infty$.

a rational adversary will crack given that the value of a cracked password is v, the cost to check each password is k. To simulate settings where the defender has imperfect knowledge of the password distribution we use different distributions $\mathcal{D}_1$ (training) and $\mathcal{D}_2$ (evaluation) to generate the signaling matrix $\mathbf{S}^* \leftarrow \mathsf{genSigMat}(v, k, a, b, \mathcal{D}_1)$ and evaluate the success rate of a rational attacker $\mathsf{evaluate}(v, k, a, b, \mathbf{S}^*, \mathcal{D}_2)$. We can also set $\mathcal{D}_1 = \mathcal{D}_2$ to evaluate our mechanism under the idealized setting in which defender has perfect knowledge of the distribution.

In the remainder of this section, we describe how the oracle $\mathsf{getStrength}()$ is implemented in different experiments, the password distribution(s) derived from empirical password datasets, and how we implement $\mathsf{genSigMat}()$.

Password Distribution

We evaluate the performance of our information signaling mechanism using 9 password datasets: Bfield (0.54 million), Brazzers ($N = 0.93$ million), Clixsense (2.2 million), CSDN (6.4 million), LinkedIn (174 million), Neopets (68.3 million), RockYou (32.6 million), 000webhost (153 million) and Yahoo! (69.3 million). The Yahoo! frequency corpus ($N \approx 7 \times 10^7$) was collected and released with permission from Yahoo! using differential privacy [80] and other privacy-preserving measures [79]. All the other datasets come from server breaches.

Empirical Distribution

For all 9 datasets we can derive an empirical password distribution $\mathcal{D}_e$ where $\Pr_{pw \sim \mathcal{D}_e}[pw_i] = f_i/N$. Here, N is the number of users in the dataset and f_i is the number of occurrences of pw_i in the dataset. We remark that for datasets like Yahoo! and LinkedIn where the datasets only include frequencies f_i without the original plaintext password we can derive a distribution simply by generating unique strings for each password. The empirical distribution is useful to analyze the performance of information signaling when the password value v is small this analysis will be less accurate for larger values of v i.e., once the rational attacker has the incentive to start cracking passwords with lower frequency. Following an approach taken in [30], we use Good-Turing frequency estimation [130] to identify and highlight regions of uncertainty where the CDF for the empirical distribution might significantly diverge from the real password distribution. To simulate an attacker with imperfect knowledge of the distribution we train a differentially private Count-Mean-Min-Sketch. In turn, the Count-Sketch is used to derive a distribution $\mathcal{D}_{train}$, to implement $\mathsf{getStrength}()$ and to generate the signaling matrix $\mathbf{S}^* \leftarrow \mathsf{genSigMat}(v, k, a, b, \mathcal{D}_{train})$ (see details below).

Monte Carlo Distribution

To derive the Monte Carlo password distribution from a dataset we follow a process from [30]. In particular, we subsample passwords $D_s \subseteq D$ from the dataset and derive guessing numbers $\#guessing_m(pw)$ for each $pw \in D_s$. Here, $\#guessing_m(pw)$ denotes the number of guesses needed to crack pw with a state of the art password cracking model m e.g., Probabilistic Context-Free Grammars [131, 132, 133], Markov models [134, 135, 136, 137], and neural networks [82]. We used the password guessing service [137] to generate the guessing numbers for each dataset. We then fit our distribution to the guessing curve i.e., fixing thresholds $t_0 = 0 < t_1 < t_2 \ldots$ we assign any password pw with $t_{i-1} < \min_m\{\#guessing_m(pw)\} \leq t_i$ to have probability $\frac{g_i}{|D_s|(t_i - t_{i-1})}$ where g_i counts the number of sampled passwords in D_s with guessing number between t_{i-1} and t_i. Intuitively, the Monte Carlo distribution $\mathcal{D}_m$ models password distribution from the attacker's perspective. One drawback is that the distribution would change if the attacker were to develop an improved password cracking model.

We extract Monte Carlo distribution from 6 datasets (Bfield, Brazzers, Clixsense, CSDN, Neopets, 000webhost) for which we have plain text passwords so that we can query Password Guessing Service [137] about password guessing numbers. In the imperfect knowledge setting, we repeated the process above twice for each dataset with different sub-samples to derive two distributions $\mathcal{D}_{train}$ and D_{eval}.

Differentially Private Count-Sketch

When using the empirical distribution $\mathcal{D}_e$ for evaluation we evaluate the performance of an imperfect knowledge defender who trains a differentially private Count-Mean-Min-Sketch.

As users register their accounts, the server can feed passwords into a Count-Mean-Min-Sketch initialized with Laplace noise to ensure differential privacy.

When working with empirical distributions in an imperfect knowledge setting we split the original dataset D in half to obtain D_1 and D_2. Our noise-initialized Count-Mean-Min-Sketch is trained with D_1. We fix the width d_w (resp. depth d_s) of our count sketch to be $d_w = 10^8$ (resp. $d_s = 10$) and add Laplace Noise with scaling factor $b = d_s/\epsilon_{pri} = 5$ to preserve $\epsilon_{pri} = 2$-differential privacy. Since we were not optimizing for space we set the number of columns d_w to be large to minimize the probability of hash collisions and increase the accuracy of frequency estimation. Each cell is encoded by a 4-byte int type so the total size of the sketch is 4 GB.

We then use this count sketch along with D_2 to extract a noisy distribution $\mathcal{D}_{train}$. In particular, for every $pw \in D_2$ we query the the count sketch to get $\tilde{f}_{pw}$, a noisy estimate of the frequency of pw in D_2 and set $\Pr_{\mathcal{D}_{train}}[pw] \doteq \frac{\tilde{f}_{pw}}{\sum_{w \in D_2} \tilde{f}_w}$. We also use the Count-Mean-Min Sketch as a frequency oracle in our implementation of $\mathsf{getStrength}()$ (see details below). We then use $\mathcal{D}_{train}$ to derive frequency thresholds for $\mathsf{getStrength}()$ and to generate the signaling matrix $\mathbf{S}^* = \mathsf{genSigMat}(v, k, a, b, \mathcal{D}_{train})$. Finally we evaluate results on the original empirical distribution $\mathcal{D}_e$ for the original dataset D i.e., $P^s_{adv} = \mathsf{evaluate}(v, k, a, b, \mathbf{S}^*, \mathcal{D}_e)$.

Implementing getStrength()

Given a distribution $\mathcal{D}$ and a frequency oracle $\mathcal{O}$ which outputs $f(pw)$ in the perfect knowledge setting and an estimate of frequency $\hat{f}(pw)$ in the imperfect knowledge setting, we can specify $\mathsf{getStrength}()$ by selecting thresholds $x_1 > \ldots > x_{a-1} > x_a = 1$. In particular, if $x_{i+1} \leq \mathcal{O}(pw) < x_i$ then $\mathsf{getStrength}(pw) = i$ and if $\mathcal{O}(pw) \geq x_1$ then $\mathsf{getStrength}(pw) = 0$. Let $Y_i \doteq \Pr_{pw \sim \mathcal{D}}[x_i \leq \mathsf{getStrength}(pw) < x_{i-1}]$ for $i > 1$ and $Y_1 = \Pr_{pw \sim \mathcal{D}}[\mathsf{getStrength}(pw) > x_1]$. We fix the thresholds $x_1 \geq \ldots \geq x_{a-1}$ to (approximately) balance the probability mass of each strength level i.e., to ensure that $Y_i \approx Y_j$. In imperfect (resp. perfect) knowledge settings we use $\mathcal{D} = \mathcal{D}_{train}$ (resp. $\mathcal{D} = \mathcal{D}_{eval}$) to select the thresholds.

Derivative-Free Optimization

Given a value v and hash cost k we want to find a signaling matrix which optimizes the defenders utility. Recall that this is equivalent to minimizing the function $f(\mathbf{S}) = \mathsf{evaluate}(v, k, a, b, \mathbf{S}, \mathcal{D})$ subject to the constraints that $\mathbf{S}$ is a valid signaling matrix. In our experiments we will treat f as a black box and use derivative-free optimization methods to find good signaling matrices $\mathbf{S}^*$.

Derivative-free optimization is active research area with many mature solvers with simple interface, e.g., CMA-ES [138], NOMAD [139, 140], DAKOTA [141]. In our experiment, we choose BITmask Evolution OPTimization (BITEOPT) algorithm [142] to compute the quasi-optimal signaling matrix $\mathbf{S}^*$[7]. BITEOPT is a free open-source stochastic non-linear bound-constrained derivative-free optimization method (heuristic or strategy). BiteOpt took 2nd place (1st by sum of ranks) in BBComp2018-1OBJ-expensive competition track [143].

In each experiment we use BITEOPT with 10^4 iterations to generate signaling matrix $\mathbf{S}^*$ for each different v/C_{max} ratio, where C_{max} is server's maximum authentication cost satisfying $k \leq C_{max}$. We refer to the procedure as $\mathbf{S}^* \leftarrow \mathsf{genSigMat}(v, k, a, b, \mathcal{D}_1)$.

We describe the results of our experiments. In the first batch of experiments, we evaluate the performance of information signaling against an offline and an online attacker where the ratio v/C_{max} is typically much smaller.

4.1.5 Password Signaling against Offline Attacks

We consider four scenarios using the empirical/Monte Carlo distribution in a setting where the defender has perfect/imperfect knowledge of the distribution.

---

[7] BITEOPT maintains a population list of previously evaluated solutions that are ordered in cost (objective function value). The whole population evolves towards a lower cost. On every iteration, the solution with the highest cost in the list can be replaced with a new solution, and the list is reordered. The solution vectors are spanned apart from each other to cover a larger parameter search space collectively. Besides that, a range of parameter randomization and the "step in the right direction" (Differential Evolution "mutation") operations are used that probabilistically make the population evolve to be “fittest” ones (lower cost solutions). BITEOPT's hyper-parameters (probabilities) were pre-selected and are not supposed to be changed.

Empirical Distribution

From each password dataset we derived an empirical distribution $\mathcal{D}_e$ and set $\mathcal{D}_{eval} = \mathcal{D}_e$. In the perfect knowledge setting we also set $\mathcal{D}_{train} = \mathcal{D}_e$ while in the imperfect knowledge setting we used a Count-Min-Mean Sketch to derive $\mathcal{D}_{train}$ (see details in the previous section).

We fix dimension of signaling matrix to be 11 by 3 (the server issues 3 signals for 11 password strength levels) and compute attacker's success rate for different value-to-cost ratios $v/C_{max} \in \{i * 10^j : 1 \le i \le 9, 3 \le j \le 7\} \cup \{(i + 0.5) * 10^j : 1 \le i \le 9, 6 \le j \le 7\}$. In particular, for each value-to-cost ratio v/C_{max} we run $\mathbf{S}^* \leftarrow \mathsf{genSigMat}(v, k, a, b, \mathcal{D}_e)$ to generate a signaling matrix and then run $\mathsf{evaluate}(v, k, a, b, \mathbf{S}^*, \mathcal{D}_e)$ to get the attacker's success rate. We plot the attacker's success rate vs. v/C_{max} . Results for the BField data set are shown in Fig. 4.1. The same experiment is repeated for all 9 password datasets, which can be found in the main paper at [6]. We follow the approach of [30], highlighting the uncertain regions of the plot where the cumulative density function of the empirical distribution might diverge from the real distribution. In particular, the red (resp. yellow) region indicates $E > 0.1$ (resp. $E > 0.01$) where E can be interpreted as an upper bound on the difference between the two CDFs.

Fig. 4.1 demonstrates that information signaling reduces the fraction of cracked passwords. The mechanism performs best when the defender has perfect knowledge of the distribution (blue curve), but even with imperfect knowledge, there is still a large advantage. For example, for the Neopets dataset when $v/C_{max} = 5 \times 10^6$ the percentage of cracked passwords is reduced from 44.6% to 36.9% (resp. 39.1%) when the defender has perfect (resp. imperfect) knowledge of the password distribution. Similar results hold for other datasets. The green curve (signaling with imperfect knowledge) curve generally lies in between the black curve (no signaling) and the blue curve (signaling with perfect knowledge), but sometimes has an adverse effect when v/C_{max} is large. This is because the noisy distribution will be less accurate for stronger passwords that were sampled only once.

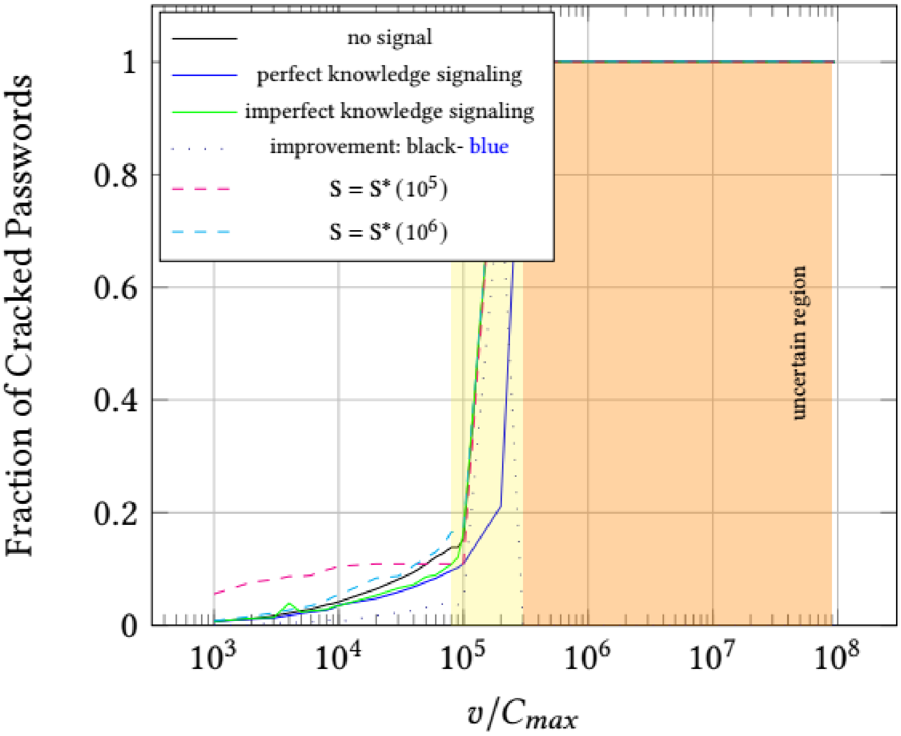

Figure 4.1. Adversary Success Rate vs v/k for the BField empirical distribution the red (resp. yellow) shaded areas denote low confidence regions where the the empirical distribution might diverges from the real distribution $E \geq 0.1$ (resp. $E \geq 0.01$).

Which accounts are cracked?

As Fig 4.1 demonstrates information signaling can substantially reduce the overall fraction of cracked passwords i.e., many previously cracked passwords are now protected. It is natural to ask whether there are any unlucky users u whose password is cracked after information signaling *even though* their account was safe before signaling. Let X_u (resp. L_u) denote the event that user u is unlucky (resp. lucky) i.e., a rational attacker would originally not crack pw_u, but after information signaling the account is cracked. We measure $E[X_u]$ and $E[L_u]$ for various v/C_{max} values under each dataset. Generally, we find that the fraction of unlucky users $E[X_u]$ is small in most cases e.g. ≤ 0.04. For example, when $v/k = 2 * 10^7$ we have that $E[X_u] \approx 0.03\%$ and $E[L_u] \approx 6\%$ for LinkedIn. In all instances the net advantage $E[L_u] - E[X_u]$ remains positive.

Robustness

We also evaluated the robustness of the signaling matrix when the defender's estimate of the ratio v/C_{max} is inaccurate. In particular, for each dataset we generated the signaling matrix $\mathbf{S}(10^5)$ (resp. $\mathbf{S}(10^6)$) which was optimized with respect to the ratio $v/C_{max} = 10^5$ (resp. $v/C_{max} = 10^6$) and evaluated the performance of both signaling matrices against an attacker with different v/C_{max} ratios. We find that password signaling is tolerant even if our estimate of v/k is off by a small multiplicative constant factor e.g., 2. In the "downhill" direction, even if the estimation of v/k deviates from its true value up to 5×10^5 at anchor point 10^6 it is still advantageous for the server to deploy password signaling.

Monte Carlo Distribution

We use the Monte Carlo distribution to evaluate information signaling when v/C_{max} is large. In particular, we subsample 25k passwords from each dataset for which we have plain text passwords (excluding Yahoo! and LinkedIn) and obtain guessing numbers from the Password Guessing Service. Then we split our 25k subsamples in half to obtain two guessing curves and we extract two Monte Carlo distributions $\mathcal{D}_{train}$ and $\mathcal{D}_{eval}$ from these curves (see details in the last section). In the perfect knowledge setting the signaling matrix is both optimized and tested on $\mathcal{D}_{eval}$ i.e., $\mathbf{S}^* = \mathsf{genSigMat}(v, k, a, b, \mathcal{D}_{eval})$, $P^s_{adv} = \mathsf{evaluate}(v, k, a, b, \mathbf{S}^*, \mathcal{D}_{eval})$. In the imperfect knowledge setting the signaling matrix is tuned on $\mathcal{D}_{train}$ while the attacker's success rate is evaluated on $\mathcal{D}_{eval}$. One advantage of simulating Monte Carlo distribution is that it allows us to evaluate the performance of information signaling against state of the art password cracking models when the v/C_{max} is large. We consider $v/C_{max} \in \{i * 10^j : 1 \leq i \leq 9, 5 \leq j \leq 10\}$ in performance evaluation for Monte Carlo distribution. As before we set $a = 11$ and $b = 3$ so that the signaling matrix is in dimension of 11×3.

In the full paper [6] we show that information signaling can significantly reduce the number of cracked passwords. In particular, for the Neopets dataset when $v/C_{max} = 6 \times 10^7$ the number of cracked passwords is reduced from 52.2% to 40% (resp. 43.8%) when the defender has perfect (resp. imperfect) knowledge of the distribution. The green curve (signaling with imperfect knowledge) generally lies between the black curve (no signaling) and the blue curve (signaling with perfect information) though we occasionally find points where the green curve lies slightly above the black curve.

4.1.6 Password Signaling against Online Attacks

We can extend the experiment from password signaling with perfect knowledge to an online attack scenario. One common way to throttle online attackers is to require the attacker to solve a CAPTCHA challenge [36], or provide some other proof of work (PoW), after each incorrect login attempt [144]. One advantage of this approach is that a malicious attacker cannot lockout an honest user by repeatedly submitting incorrect passwords [145]. However, the solution also allows an attacker to continue trying to crack the password as long as s/he is willing to continue paying the cost to solve the CAPTCHA/PoW challenges. Thus, information signaling could be a useful tool to mitigate the risk of online attacks.

When modeling a rational online password we will assume that $v/C_{max} \leq 10^5$ since the cost to pay a human to solve a CAPTCHA challenge (e.g., \$$10^{-3}$ to 10^2 [146]) is typically much larger than the cost to evaluate a memory-hard cryptographic hash function (e.g.,

\$$10^{-7}$). Since $v/C_{max} \leq 10^5$ we use the empirical distribution to evaluate the performance of information signaling against an online attacker. In the previous subsection, we found that the uncertain regions of the curve started when $v/C_{max} \gg 10^5$ so the empirical distribution is guaranteed to closely match the real one.

Since an online attacker will be primarily focused on the most common passwords (e.g., top 10^3 to 10^4) we modify getStrength() accordingly. We consider two modifications of getStrength() which split passwords in the top 10^3 (resp. 10^4) passwords into 11 strength levels. By contrast, our prior implementation of getStrength() would have placed most of the top 10^3 passwords in the bottom two strength levels. As before we fix the signaling matrix dimension to be 11×3.

Our results demonstrate that information signaling can be an effective defense against online attackers as well. For example, when $v/C_{max} = 9 \times 10^4$ in the Brazzers dataset our mechanism reduces the fraction of cracked passwords from 20.4% to just 15.3%. Similar observations hold true for other datasets.

Implementing Password Signaling

One naive way to implement password signaling in an online attack would simply be to explicitly send back the signal noisy signal sig_u in response to any incorrect login attempt. As an alternative, we propose a solution where users with a weaker signal sig_u are throttled more aggressively. For example, if sig_u indicates that the password is strong then it might be reasonable to allow for 10 consecutive incorrect login attempts before throttling the account by requiring the user to solve a CAPTCHA challenge before every login attempt. On the other hand if the signal sig_u indicates that the password is weak the server might begin throttling after just 3 incorrect login attempts. The attacker can indirectly infer the signal sig_u by measuring how many login attempts s/he gets before throttling begins. This solution might also motivate users to pick stronger passwords.

4.1.7 Discussion

While our experimental results are positive, we stress that several questions would need to be addressed before we recommend deploying information signaling to protect against offline attacks.

- Can we accurately predict the value to cost ratio v/C_{max}? Our results suggest that information signaling is useful even when our estimates deviate by a factor of 2. However, if our estimates are wildly off then information signaling could be harmful.
- While information signaling reduced the total number of cracked passwords a few unlucky users might be harmed i.e., instead of being deterred the unlucky signal helps the rational attacker to crack a password that they would not otherwise have cracked. Knowing this is a possibility how would users react to such a solution? One possibility would be to allow users to opt-in/out of information signaling. However, each user u would need to make this decision without observing their signal — otherwise, the signal might be strongly correlated with the decision to opt-in/out allowing the attacker to perform another Bayesian update.
- Can we analyze the behavior of rational targeted attackers? We only consider an untargeted adversary who has a constant password value expectation for all accounts. In some settings, an attacker might place a higher value on some passwords e.g., celebrity accounts. Can we predict how a targeted attacker would behave if the value v_u varied from user to user? Similarly, a targeted adversary could exploit demographic and/or biographical knowledge to improve password guessing attacks e.g., see [102].

4.2 Just In Time Hashing

In the past few years, billions of user passwords have been exposed to the threat of offline cracking attempts. Recent high-profile examples include Yahoo!, Dropbox, Lastpass, AshleyMadison, LinkedIn, AdultFriendFinder, and eBay. Once such a breach occurs the attacker can check as many password guesses as s/he wants offline. The attacker is only limited by the resources s/he invests to crack user passwords and by the underlying cost of computing the hash function.

Offline brute-force cracking attacks are increasingly dangerous as password cracking hardware continues to improve and as many users continue to select low-entropy passwords finding it too difficult to memorize multiple strong passwords for each of their accounts. Key stretching serves as a last line of defense for users after a password breach. The basic idea is to increase guessing costs for the attacker by performing hash iteration (e.g., BCRYPT[71] or PBKDF2 [72]) or by intentionally using a password hash function that is memory-hard (e.g., SCRYPT [147], Argon2 [16]).

Unfortunately, these key-stretching algorithms are fundamentally constrained by user patience. Specifically, authentication latency is a usability constraint that upper bounds the maximum number of calls that can be made to an underlying hash function (e.g., SHA256, Blake2b) as well as the amount of RAM that can be filled. For example, LastPass had been using PBKDF2-SHA256 with 10^5 SHA256 iterations when they were breached. While LastPass [73] claimed that "Cracking our algorithms is extremely difficult, even for the strongest computers"[8], it has been estimated that the cost to evaluate the SHA256 hash function on customized hardware [31] is as low as $\$10^{-15}$ (USD) [92], which means that it could potentially cost an attacker as little as \$1 (USD) to validate 10^{10} password guesses under PBKDF2-SHA256 with 10^5 iterations. Even with more advanced key-stretching mechanisms such as memory-hard functions, it is not clear whether or not it is possible to perform sufficient key-stretching to protect most (lower entropy) user passwords without *substantially increasing* authentication delay.

Contributions

We introduce a novel client-side key stretching technique that we call Just In Time hashing (JIT) which can substantially increase key-stretching *without* increasing authentication delay for the user. JIT is suitable for applications such as password managers, disk encryption, and mobile encryption. The basic idea is to exploit idle time while the user is typing

---

[8] Similar claims were made after the Dropbox breach [74].

in their password to perform extra key-stretching. As soon as the user types in the first character(s) of their password our algorithm immediately begins filling memory with hash values derived from the character(s) that the user has typed thus far. The key challenge in designing such a function is that the final output must be a deterministic function of the input password, while users do not always enter their password at the same speed.

We conduct a user study to investigate password typing habits and inform the design of JIT. In particular, we aimed to answer the following questions: How fast do users type when entering passwords? Is this password typing speed equivalent to regular typing speed? How often do users press backspace when entering a password? Unlike previous user studies, we asked users to type in their actual passwords. To minimize risk to users the actual passwords were never transmitted to us. Instead, we only measured typing speeds while the user typed their password and a sample paragraph. We find that for over 95% of pc users (resp. mobile users) the delay between consecutive key-strokes during password entry is over 180 ms (resp. 319 ms). While users do occasionally press backspace during password entry we find that the pattern is highly predictable (e.g., a user either erases ≤ 3 characters or erases the entire password). Both of these observations are encouraging trends for JIT since we have lots of time to perform key-stretching between consecutive key-presses, and at any time we only need to be able to restore the JIT state for the last three characters of the password that the user typed.

Several of our findings from the user study may be of independent interest. For example, we find that password typing speed is only weakly correlated with regular typing speed, which may have implications for the design and evaluation of implicit authentication mechanisms based on keystroke dynamics during password entry. We conjecture that the differences in typing times are due to muscle memory as well as the use of less common characters and/or character sequences in passwords.

We analyze the security of a JIT password hashing algorithm using graph pebbling arguments. On the negative side our analysis demonstrates that JIT password hashing with hash iteration as the underlying key-stretching mechanism provides minimal benefits over traditional key-stretching algorithms based on hash iteration (e.g., BCRYPT, PBKDF2). On the positive side, we find that JIT hashing can be combined with memory-hard functions to *dramatically* increase guessing costs for an offline attacker. In particular, we find that if users select passwords similarly to those found in the Rockyou data set [148] JIT hashing requires more than 6 times as much work to evaluate than non-JIT hashing functions in the worst case. For XKCD-Style Passwords [149] this advantage is, 11.7. We remark that these advantages are pessimistic and are based on the assumption that the adversary has an unbounded amount of parallelism available. If the adversary is restricted to a model without parallelism these advantages increase to 13.3 and 25.4 respectively.

Finally, we provide a proof-of-concept implementation of JIT highlighting key design decisions made along the way. Our implementation is based on a modification of Argon2 [16], winner of the password hashing competition [38]. The execution of JIT can remain hidden from the user to provide the benefit of increased key-stretching without affecting the user's authentication experience.

4.2.1 Related Work

Password Cracking

The issue of offline password cracking is far from new, and has been studied for many years [150]. Password cracking tools have been created and improved through the exploration of new strategies and techniques such as probabilistic password models [151], probabilistic context-free grammars [131, 132, 133], Markov chain models [135, 136, 137], and neural networks [82]. For sentence-based passwords, attackers may turn to public or online resources as a source of password possibilities, or they may use training data from previous large breaches like Rockyou [152]. Public and open-source password cracking tools like John the Ripper are easily available online and can be modified or provided with specific strategies to attempt to crack lists of passwords offline [153].

Improving Password Strength

It has proven difficult to convince or force users to choose stronger passwords [154, 155, 156, 157, 158, 159], and methods that do work have usability issues [160]. Strategies to convince users to select stronger passwords have included providing continuous feedback (e.g. password strength meters [161, 162, 163]) and providing instructions or enforcing composition policies [152, 114, 154, 155, 156, 157, 159, 164]. However it has been shown that these methods also suffer from usability issues [158, 165, 166, 160] and in some cases can even lead to users selecting weaker passwords [167, 155]. Password strength meters have also been shown to provide inconsistent feedback to users, often failing to persuade them to select a stronger password [162, 163].

Key Stretching

Key stretching, the process of artificially increasing the difficulty of computation of a hash function, is designed to protect low entropy passwords and secrets from offline cracking attempts. By making each guess more expensive, it becomes more difficult for an adversary to crack each password as each attempt costs them more. The method was proposed by Morris in 1979 [150] who used it in the context of password security. Key stretching was originally performed by repeated calculations of the hashing function, i.e. rather than storing the hash of the password and salt the result is first run through the hash function many more times. This method is still used by the functions BCRYPT [71] and PBKDF2 [72]. However these functions require small amounts of memory, and the base hash functions that these are based on can now be computed very quickly for a reasonable cost using hardware such as the Antminer [31], which can computer trillions of base functions per second. Current levels of key stretching being used have raised concern - Bonneau estimated that a human would need to memorize a 56-bit secret to provide themselves with adequate security [92]. Additional key stretching methods have been proposed to introduce asymmetric costs by keeping part of the salt secret and requiring that it be guessed iteratively [17].

Memory Hard Functions

Memory Hard Functions (MHFs) were introduced in 2009 by Percival [32]. The key insight behind MHFs is that, while computation power is asymmetric between users and adversaries, the cost of using memory is more equitable. Ideally, a MHF should have τ^2 area-time complexity, where τ is a parameter setting the amount of time and memory the function should use. Functions like BCRYPT or PBKDF2 would instead have an area-time complexity of τ as they use a constant amount of space. Data-independent MHFs are a particular class of MHF that are designed to help prevent side-channel attacks. MHFs such as Argon2d [16] or SCRYPT [32].Data-dependent MHFs have a data access pattern that depends on the input, meaning they are potentially vulnerable to side-channel attacks that determine memory access patterns [168, 55]. It has recently been shown that SCRYPT is optimally memory-hard [34].

Other defenses against offline attacks

It is possible to distribute the storage and computation of password hashes across multiple servers [94, 95, 96]. Juels and Rivest [98] proposed storing the hashes of fake passwords (honeywords) and using a second auxiliary server to detect authentication attempts that come from cracking the fake passwords. These methods require the purchase of additional equipment, which may prevent those with more limited financial resources from employing them. A second area of research has investigated the use of hard artificial intelligence problems that require a human to solve [169, 170, 171]. This would require an offline attacker to employ human oversight throughout the process by having them solve a puzzle (e.g. a CAPTCHA [169, 171]). In comparison, data-independent MHFs have a set memory access pattern that does not involve the input. These data-independent MHFs are typically the recommended type to use for password hashing [38, 16]. Several of the most prominent iMHFs from the literature are (1) Argon2i [16], the winner of the password hashing competition [38], (2) Catena [55], a PHC contestant which received special recognition from the PHC judges, and (3) Balloon Hashing [172]. Several attacks have been found for Catena [173, 39, 56] and for Argon2i and Balloon Hashing [56, 174]. Constructions for iMHFs with cumulative complexity $\Omega\left(n^2/\log n\right)$ have been shown [175] using a concept called depth-robust graphs [176]. This is asymptotically the best possible result given the attack shown by Alwen and Blocki [56] showing that any “natural" iMHF has cumulative complexity $O(n^2 \log\log n/\log n)$, but the construction remains theoretical at this time.

4.2.2 Just in Time Hashing

The proposed solution to this problem is Just in Time Hashing, a method that allows for extended key stretching without a user being aware that it is being done. That is, where a user may have noticed that a large amount of key stretching was taking several seconds before, using just in time hashing they could perform the same amount of key stretching and barely notice any delay once they have finished entering their password.

Formally we define a k-limited Just In Time hashing function as a streaming algorithm $\mathcal{A}$, with a random oracle H, and an initial state q_0 that makes at most k sequential calls to the random oracle for each state update. As each character c_i of the input enters the algorithms the state is updated, up until a special terminating character \$. The input must be of the form $C \in (\Sigma \setminus \$)^* |\$$, where once the terminating character is read the output τ based on the final state is returned. On each update, a just-in-time hashing algorithm returns one of two types of outputs. If the character was not the terminating character \$ then the function returns a new state q_i from the set of possible states Q. If it is then it returns an output t from the set of possible outputs T. The just in time algorithm transitions between states according to the following function:

$$\mathcal{A}^H(q_{i=1}, c_i) = \begin{cases} q_i \in Q & c_i \neq \$ \\ \tau \in T & c_i = \$ \end{cases}$$

We use $\mathcal{A}^H(C)$ to denote the final output given a sequence of the form $C \in (\Sigma \setminus \$)^* |\$$.

The Backspace Challenge: We allow the character set Σ to include a special character b (backspace). We require that a $\mathcal{A}^H$ is consistent meaning that we should get the same output when the user types $1, 2, f, g, b, b, 3$ that we would if the user had typed the sequence $1, 2, 3, \$$ the output τ. Formally, for all input sequence $C \in (\Sigma \setminus \$)^* |\$$ we require that $\mathcal{A}^H(C) = \mathcal{A}^H(\textbf{Prune}(C))$, where $\textbf{Prune}(C) \in (\Sigma \setminus \{\$, b\})^* |\$$ is the character sequence we obtain after applying each backspace operation b.

A naive way to handle backspaces would be to revert to state q_0 and repeat the entire computation, but this approach would result in noticeably large authentication delays for the user. A second way to handle backspaces would be to store all previous states so that we can quickly revert to a prior state. The key challenge is that states can quickly become very large (e.g., 1GB) because our instantiation of $\mathcal{A}^H$ is memory-hard.

We can relax the requirement that $\mathcal{A}^H$ *always* updates after at most k sequential calls to the random oracle to say that $\mathcal{A}^H$ *always* updates after at most k sequential calls for β-good input sequences. Intuitively, a sequence is β good if it does not contain too many backspaces b within a short interval so that once we are in state $q_{i+\beta}$ we will never be asked to revert to a state q_j for $j < i$. We allow for one exception: if the user wipes out the entire password then the sequence is *not* β-bad because it is easy to revert to state q_0.

Definition 4.2.1. *We say that a sequence $C = c_1, \ldots, c_t, \$ \in (\Sigma \setminus \$)^* |\$$ is β-bad if we can find indices $i \leq j \leq t$ such that*

$$\beta < \sum_{i=1}^{j} \left(\mathbb{1}_{c_i = b} - \mathbb{1}_{c_j \neq b} \right) \ ,$$

and $\textbf{Prune}(c_1, \ldots, c_j) \neq \emptyset$*. If no such indices exist then we say that the sequence is β-good. We say that a sequence is β-bad if $\mathcal{A}^H$ is a β-tolerant k-limited Just In Time hashing function if for all β-good sequences $C \in (\Sigma \setminus \$)^* |\$$ the algorithm $\mathcal{A}^H(C)$ never requires more than k-sequential calls to the random oracle between updates.*

Discussion: In this paper we focus on the context of password hashing and key stretching, specifically using the time users spend typing in their passwords. However, in the broadest sense, JIT is a method to hide computation within idle cycles by streaming input instead of working in batches, and thus potential applications are not necessarily limited to password hash computation. For example, the JIT technique could be used to generate proofs of work for email. As the user types his e-mail the JIT algorithm could continually update the proof of work for the current email message. Using this approach could help deter spammers by making it prohibitively expensive to generate the proof of work for each message. One intriguing challenge would be to develop a JIT proof of work with a more efficient verification algorithm in case the receiver does not have time to regenerate the entire JIT proof. Another possible application domain for authentication would be to take advantage of the longer delays induced by two-factor authentication.

Usability Analysis

In the last section, we introduced the notion of a β-tolerant k-limited JIT scheme which updates the state at most k times given any β-good input sequence. Before instantiating any JIT scheme it is crucial to understand how people type passwords in practice. In particular, to avoid delays during authentication we need to tune k so that the time to update the state is less than the expected delay between consecutive keystrokes. Thus, the parameter k will depend on the user's password typing speed. Furthermore, we also need to ensure that JIT is β-tolerant for a sufficiently large value of β to ensure that the input sequence we receive when a user types their password is β-good.

In this section, we aim to answer the following questions. How quickly do users type their passwords? To what extent is password typing speed correlated with regular typing speed? What fraction of login attempts are β-good for $\beta = 1, 2, 3$? And to what extent does password typing speed Change over time?

To answer these questions we first analyze two publicly available datasets [177, 178]. While we can extract useful insights from both datasets, there are significant methodological limitations when we attempt to use these datasets to answer each of our questions e.g., users in the passwords typo dataset [177] were not typing their own passwords. To address these limitations we also conduct our own user study in which we asked users to type in their real passwords so that we could measure password typing speed.

Study Design

To address the previous limitations we designed a user study to investigate user's typing speed and correction habits on their real passwords. Briefly, in the study users were asked to type their password, type a paragraph and then type their password again. The instructions emphasized that we wanted users to type in their actual password and reassured users that we only collected statistics on typing speeds and would never receive their actual password.

Previous work has found that conducting password studies poses many challenges and that care must be taken when analyzing the results [179]. Thus we strived to ensure that we were learning valuable information while taking care to design the study properly.

To give an idea of how much key stretching could be performed with JIT hashing the specific data that is needed is how quickly people type their passwords in practice. While previous work did have people type in passwords, they were either typing a pre-defined password list [178] or randomly generated passwords [177]. To give an idea of how much time we have for key stretching in practice we need to know how long users spend typing per character on their own passwords. To obtain this information we performed an IRB-approved user study on MTurk in which we collected the time it took for people to type in their real passwords. We recruited 400 participants, each of whom were paid $0.50 for an estimated 5-minute survey. For more details on the construction of this study please see [8].

The main data that were collected were per character password and standard typing speeds. To record this, an action was triggered when the first character was entered into the provided field that recorded the starting time. If at any point, a user cleared out the field, the timer was reset. Once the user hit enter or clicked the continue button the timer stopped, calculated the total time over the number of characters, and transmitted the per character speed over the encrypted connection. In addition to timing data, we also collected data on how many consecutive backspaces occurred in the worst case, as well as how many times users cleared the entire field. The results were stored in a database at our institution for analysis.

Ethical Considerations As this study involved the use of human subjects and sensitive information great care was taken to ensure this study was designed and run in a way that would offer the most benefit with the least risk to users. A large number of security precautions to prevent password theft were put in place, described in the following paragraph. Another potential concern is that an attacker might conduct a copycat “study" to phish for user passwords. To minimize the risk of such copycat studies we provided full contact information for the PI and the IRB board at Purdue. The study site was also hosted on an HTTPS server using a domain name affiliated with Purdue University. Finally, we note that in user studies in which users are asked to create a new password that many users simply type in one of their passwords [155]. Thus the risk of phishing 'studies' is present whether or not the user is explicitly asked to type in their password. The study was submitted to and approved by the IRB board at Purdue before the study was conducted.

Security precautions

We took several precautions to ensure that at no point would a user's password be revealed, either to us or even to someone monitoring the user's network traffic. The first step for ensuring security is to make sure that all data involving the user's password was computed locally on the user's machine. To accomplish this we wrote Javascript code to monitor the time between key presses and watched for the enter key to be pressed when the user was done typing. Once they finished our code transmitted only the time typed per character, the number of field clears, and the maximum number of consecutive backspaces to the server. At no point was the password or its length transmitted, only the time it takes to type each character, which is the relevant information for tuning JIT parameters.

As a second layer of protection, we required that all connections to our server be encrypted. Thus, even if secure data was sent it would not be retrievable by observing network traffic.

As a final precaution, all of the code for the survey was subjected to independent third-party analysis. The third-party used the automated tool Checkmarx to test for security vulnerabilities. The analysis found no vulnerabilities that would expose any sensitive user data.

Results

Of the 400 MTurk participants recruited 335 self-reported that they had completed the study and used one of their own passwords. In our analysis, we dropped data from the 65 users who self-reported not using their own password in the study. Additionally, we discarded data from the 7 PC users who left the password field blank (all mobile users filled in the password field). Of the remaining 335 users, 313 reported using a desktop or a laptop while 22 reported using some mobile device (phone, tablet, etc...).

Several users had exceptionally long typing times (2000ms+ per character typed). In each of these cases either the password per-character speed or the typing per-character speed were unusually large, never both. These values are excluded from the charts and tables in this section as they make it difficult to visualize the more common results. Statistical analysis was performed using the statistical package R [180], where one of the first things we looked at is whether or not the time taken to type each user's password had anything to do with their typing speed.

We split the analysis into mobile and non-mobile users. One of the first things to notice, especially in Figure 4.2, is that there isn't a very strong correlation ($R^2_{adj} = 0.1289, p < 0.001$) between observed typing time and password typing time. The non-mobile data showed the same weak correlation, meaning that typing time is not a particularly good measure of how quickly someone might type their passwords.

We noted that we found similar deletion habits to those from Chatterjee et al's data [177]. In particular, we observe that people rarely have more than 3 consecutive deletions without deleting the entire password, with only about 1% of participants doing so[9]. In total, we saw that, of the 612 entries from 306 users (two entries per user) who self-reported using their real password and were on a non-mobile device, only 4 showed more than 3 consecutive deletions. Thus, we maintain that a large majority of users will not run into more than a small number of deletions. In particular, it should be sufficient to set $\beta = 3$ when implementing a β-tolerant JIT scheme.

Of particular interest to JIT hashing are some of the typing time percentiles, marked on Figure 4.2. For just in time hashing it is valuable to know how long we can safely run the key stretching per character so that users will not notice any odd delays or slowdowns from the system. From the provided data we can see that it should suffice to stop after 183 or 213 milliseconds of computation for a non-mobile user so that 95% and 90%, respectively, of users will notice no delay from the key stretching. With mobile data it does seem like we may have a bit more time to run key stretching due to overall slower password typing speeds, however, due to the small mobile sample size, this will likely require further study to

---

[9] The four users that did have larger numbers of consecutive deletions without wiping out the entire password had very large numbers (26,19,22 and 27) for the maximum number of consecutive deletions.

come up with statistically significant claims. If it does turn out that we have more time on mobile devices this may be a benefit, as we can make up for some of the slower processing speeds with additional computation time.

Using Regular Typing Speeds to Select Cutoffs We further investigated the possibility of predicting typing speeds by categorizing users into broader categories. We began with the non-mobile users and then split this group into those with speeds under 250ms/ch, those between 250 and 500, and those taking more than 500ms per character typed. Each of these groups was further split into a training set and a testing set. Each training set contained 70% of the time group's results, with the remainder reserved for testing. Using the training data split by typing speed, we determined each group's 5th and 10th password typing speed percentiles. We then looked at the cutoff line for the percentiles and determined what proportion of the training data fell below the cutoff line, giving an idea of how accurate the predictions came out. The results are shown in Table 4.1, which shows the percentile cutoffs from the training data and the percentage of the testing data that fell below each cutoff. We observe that we obtain reasonable predictions of the testing percentiles, with the exception of the final timing category. This category turned out to be more difficult to predict due to the outliers contained in the set.

The practical benefit of being able to make some predictions based on larger standard typing time categories is the potential optimization of JIT hashing times per character by giving a user a typing speed test. If their typing speed is known, and if they turn out to be in one of the slower groups, our data suggests that it is possible to run just in time hashing for more time per character for that individual. While possible, usability may be an issue with this optimization. Users may become impatient with a required typing test before registering, and those with faster typing times can argue that they are being cheated out of additional key-stretching due to their typing speed. That is, they may prefer the extra security they would have gained by increasing the per-character running time. The benefit of this method over a universal set time would be that those who would have experienced some annoying delay when typing would no longer see this delay.

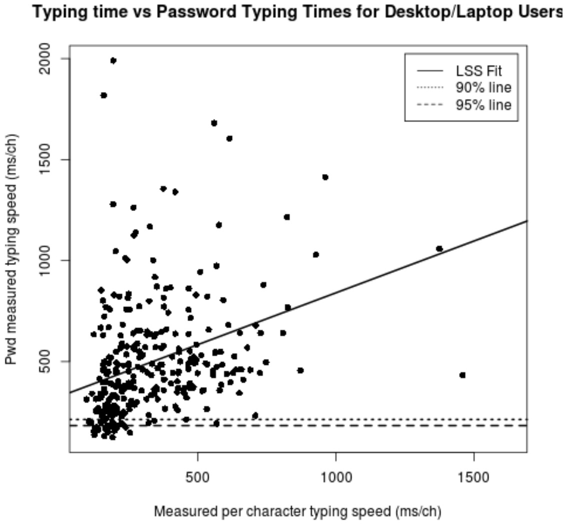

Figure 4.2. Results for non-mobile users

Table 4.1. Summarized results from subsampling tests

Typing speed range	Train 5'th (ms)	% $\leq$ Pred	Train 10th(ms)	% $\leq$ Pred
$0 \leq x < 250$	170.01	0.078	180.025	0.100
$250 \leq x < 500$	264.11	0.059	299.18	0.118
$500 \leq x$	310.12	0.083	356.400	0.167

4.2.3 Security analysis

In this section, we investigate the performance of JIT hashing with and without memory hardness. On the negative side, our analysis demonstrates that the benefits of JIT hashing without memory hardness are marginal. In particular, if an iterated hash function is run in JIT mode we will show that the adversary has a fairly efficient method to guess passwords i.e., the cost of checking every guess in a dictionary with JIT is only marginally higher than the costs the attacker would incur if a comparable iterated hash function (inducing the same authentication delay) were used. Given this attack, we warn that JIT hashing does not offer its full benefits without memory hardness. In the second section, we will examine JIT hashing when implemented with memory hardness. In this case, we use a pebbling argument to demonstrate that JIT hashing substantially increases guessing costs for an attacker.

Adversarial model

For analysis we assume the adversary is:

1. **Offline:** The adversary has obtained a hash and salt of a password and can verify password guesses offline.

2. **Informed:** The adversary is familiar with the specific implementation of JIT hashing being used, and knows exactly how the hash value they have was obtained. The adversary is also assumed to possess a reasonably large password dictionary containing all of the most likely user password choices. The adversary is interested in cracking the password using the minimum possible number of guesses and will use their knowledge to optimize their strategy to crack the password with the minimal possible amount of work.

3. **Rational:** An attacker is willing to continue cracking as long as marginal guessing benefits (i.e., the value of a cracked password times the probability that the next guess is correct) exceed marginal guessing costs. If expected guessing costs exceed the expected reward then the attacker will quit his attack. In particular, it is possible to discourage the attacker by increasing the cost to validate each password guess.

(a) **Infinitely Parallel, Memory Unbounded:** The adversary has no time limit to their computation, although there is an opportunity cost to allocating additional resources (memory/processing cores) to password cracking. Since the adversary is rational the attacker may stop attacking if the opportunity costs exceed the expected reward. This model may be overly pessimistic since a real-world attacker does not have infinite memory.

(b) **Sequential, Memory Unbounded:** The attacker has limited memory and each memory chip is associated with a single processor. While this model may be overly optimistic we note that in practice it is difficult to route messages from a single shared memory chip to many different cores.

Password Model

We consider three types of password distributions:

1. **Empirical:** The user selects a password from the RockYou dictionary. Probabilities are weighted by their empirical frequency (e.g., in the RockYou dictionary contains passwords from $N = 32.6$ million user accounts and 291×10^3 users in the dataset selected '123456' so the probability our user selects the password '123456' is $\Pr[\text{`123456}] \approx 0.009$).

2. **XKCD (Random Words):** The user selects several words uniformly at random from a dictionary of English words. In particular, we use Google's list of the 10,000 most common English words in our analysis.

3. **Cracking Dictionary:** Passwords are taken from a cracking dictionary created by Openwall and intended for use with John the Ripper [153]. This is designed to mimic how a criminal may perform an online attack against a standard password.

An additional analysis of uniform passwords is available in the full version of this paper. Briefly, our analysis shows that when we are protecting uniformly random passwords JIT offers no advantage against a parallel memory bounded attacker. However, JIT can increase costs for a sequential attacker by an order of magnitude. We defer the analysis to the full version of the paper because real users tend not to pick uniformly random passwords.

JIT without memory hardness

In this section, we analyze the performance of JIT without memory hardness. We will present an executive summary of our results and refer an interested reader to the full version of this paper for more details. To begin, assume that we have an unbounded adversary attempting to run through their list of possible passwords as quickly as they can and that JIT hashing is being run using a hash function H and key stretching is performed through hash iteration. Note that under the JIT model the adversary can think of the list of possible passwords as forming a trie of possibilities. To explore all passwords the adversary simply needs to calculate the entire trie, ensuring that they visit every node at least once.

If we define the cost to traverse each edge to be $W = \mathbb{C}(\mathbb{H}) = 1$ and define our alphabet as Σ and assume that no password is of length $1 \leq \text{length} \leq \ell$ then the adversaries total work to check all password guesses is given by the number of nodes in the trie. By comparison, if we had not used JIT and instead simply hashed the final password with $\mathbb{H}$ then the total work is given by the total number of passwords in the dataset that the attacker wants to check (e.g., the number of *leaf* nodes in the trie). The advantage of JIT is given by the ratio: $\#nodes/\#leaves$.

Empirical Distribution: For each value of T we computed a trie from the T most popular passwords in the RockYou list. Figure 4.3 plots the ratio $\#nodes/\#leaves$ for each point T. A typical value of the ratio is about 1.5. Thus, JIT slightly increases the work that an attacker must do to check the T most popular passwords.

XKCD (Random Words): We computed the ratio $\#nodes/\#leaves$ for the trie for the dictionary containing all i-tuples of the $10,000$ English words for each $i \leq 5$. The typical value for the ratio (i.e. at 4 words) is 2.417 meaning that JIT yields a modest increase in

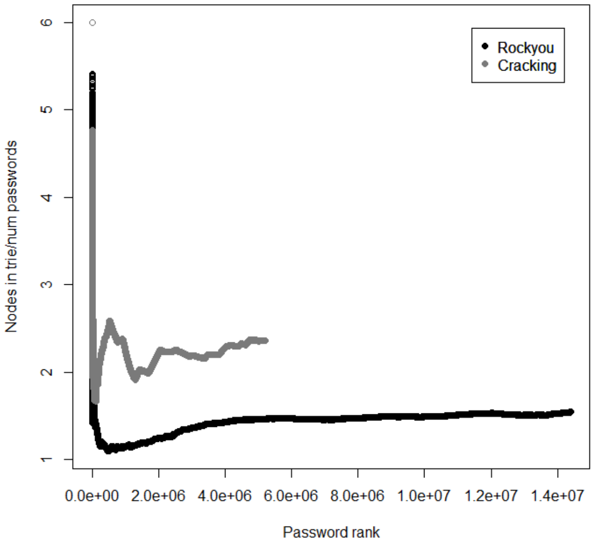

Figure 4.3. Average number of nodes added to pwd trie over time

the work that an attacker must do to crack an XKCD style password.

Cracking Dictionary: The cracking dictionary was analyzed in the same manner as Rockyou, with the results shown in Figure 4.3. We note that the ratio is slightly higher, closer to 2.4, for this dictionary.

Just in Time Memory Hard Hashing

Memory hard functions are functions that require the user to dedicate a set amount of memory to compute a function in addition to performing the computation cost, or at the very least to suffer an extreme runtime penalty if they do not want to store the function in memory. Several of these memory-hard functions, such as SCRYPT, have a tunable memory use parameter that allows the user to specify how much memory they would like the hashing function to take up. We note that if a user were to simply select a value that a reasonable computer would have, such as using 1GB of memory, we already require more memory to calculate a single hash than it took to store the entire trie under the iterated hashing scheme. The question arises as to just how much would it cost to run an offline attack against a JIT password hash that used a memory-hard function. To accomplish this we first require the notion of graph representations of MHFs, graph pebbling, and cumulative complexity (See Chapter 2.6 for these base definitions).

JIT Cumulative Complexity When running a pebbling game on a DAG we have the concept of cumulative complexity (Defined by Alwen and Serbinenko in [39]). In this model, we are thinking of each pebble as some unit of memory, and each round as a unit of time. Cumulative complexity of a DAG G, $CC(G)$, and a pebbling sequence $P = P_0, P_1, \ldots, P_k$ is a measure of the space-time complexity of the pebbling. To model cumulative memory usage in a JIT MHF, we slightly modify the standard pebbling cost definition to capture the properties of just-in-time hashing. In a JIT MHF placing a new pebble involves filling an array of size m over m steps. During the placement, the cumulative memory usage of this process is $1+2+\ldots = \frac{m^2}{2}$ units of memory over the entire placement process. When keeping a pebble on the graph, we keep m units of memory filled for the m steps it will take to fill up a new pebble, meaning each pebble costs m^2 to keep around for an additional round. Note that in the original definition of cumulative complexity the graph being used represents nodes as blocks of memory in a memory-hard function and the edges as the dependencies required to fill that block. For JIT hashing the graph represents a different point of view. Each node does not represent a single memory block but rather represents the state that the JIT hashing function is in once a sequence of characters has been entered. For each character entry, the JIT function represents the filling of m blocks of memory rather than a single block in the original definition. It is this distinction that leads to these modifications in the definition of cumulative complexity. Essentially, rather than a single node representing a single operation, it represents the sequence of operations required to update the JIT state from the previous state to a new state.

With these definitions in place we can now redefine the cumulative complexity of a sequence of pebbling moves P:

$$CC(P) = \sum_{i=0}^{k} \left(\left(\frac{m^2}{2} |P_i \setminus P_{i-1}| \right) + \Big(m^2 \, |P_{i-1} \cap P_i) \Big| \right)$$

Next, we define the cumulative complexity of an entire graph. Denote the set of all possible pebbling sequences P_G (resp. $P_G^{||}$ for parallel pebbling).

$$CC(G) =^{\min}_{P \in P_G} CC(P) \ , \quad \text{and} \quad CC^{||}(G) =^{\min}_{P \in P_G^{||}} CC(P) \ .$$

With parallel pebbling cumulative complexity ($CC^{||}(G)$) being defined in the same way when parallel pebbling has been used.

Cumulative Cost for JIT Hashing

In JIT hashing we have a window of size w that only allows us to select dependencies from the previous $w-1$ memory blocks. Recall that there is a trie representing the list of passwords that an adversary wants to try, with each node representing the addition of a new character. To create our JIT pebbling graph we start with this base trie. For each node, any node at distance at most $w-1$ may depend on it. To represent this for each node in the graph we add an edge to each of its descendants up to distance $w-1$. We set our list of sink nodes to be each node that corresponds to a password that is being guessed (e.g. in the path 1-2-3-4-5-6 we may set the nodes for 5 and 6 to be sink nodes, as they correspond to common passwords). We denote $T_{D,w}$ as the directed acyclic graph created in this manner using a dictionary D to form the base trie and windows size w. From this graph we derive two bounds on the cumulative complexity of running a brute force attack on a JIT hashed password. The upper bound on time is derived using a parallel pebbling argument while the similar lower bound is derived using a sequential pebbling game.

Notation: Given a node $v \in T_{D,w}$ we use height(v) (resp. depth(v)) to denote the height (resp. depth) of a node in the tree $T_{W,v}$ e.g., a leaf node is defined to have height 1 and the root node is defined to have depth 0.

Theorem 4.2.1. *For a parallel attacker with unbounded memory we have*

$$CC^{||}(T_{D,w}) = |T_{D,w}| \frac{m^2}{2} + \sum_{h>1} \sum_{v:\ height(v)=h} \left(m^2 \min\{h-2, w-2\} \right)$$

For a sequential (memory bounded) adversary, we have

$$CC(T_{D,w}) \geq \left(\sum_{d} \sum_{v:depth(v)=d} m^2 \min\{d-1, w-1\}\right) \\ - |T_{D,w}| \frac{m^2}{2}$$

Proof. (sketch) Consider the graph $T_{D,w}$ under the parallel black pebbling game. Let t be the height of the root node. We first observe that the there is a simple legal parallel pebbling strategy $P \in P_{T_{D,w}}$ with

$$CC^{||}(P) \leq |T_{D,w}| \frac{m^2}{2} \\ + \sum_{h>1} \sum_{v:\ \mathrm{height}(v)=h} \left(m^2 \min\{h-2, w-2\}\right) .$$

In particular, we set $P_0 = \emptyset$ and we set $P_i = \{v :\ t+w-i \geq height(v) \geq t+1-i\}$. To see that the pebbling is legal we observe that P_t contains all leaf nodes in $T_{D,w}$ and that $P_{i+1} \setminus P_i = \{v\ :\ \mathrm{height}(v) = t-i\}$ and that therefore $\mathsf{parents}(P_{i+1} \setminus P_i) \subseteq \{v : height(v) = t-i\} \subseteq \mathsf{parents}(P_i)$. Furthermore, since $P_{i+1} \cap P_i = \{v :\ t+w-i-1 \geq \mathrm{height}(v) \geq t+1-i\}$, we have $CC^{||}(T_{D,w}) \leq CC(P) =$

$$\sum_{i=1}^{t} \left(\left(\frac{m^2}{2}|P_i \setminus P_{i-1}|\right) + \left(m^2\left(P_{i-1} \cap P_i\right)\right)\right)$$

$$= \sum_{h} \sum_{v:\ \mathrm{height}(v)=h} \left(\frac{m^2}{2} + m^2 \min\{\max\{h-2, 0\}, w-2\}\right)$$

$$= |T_{D,w}| \frac{m^2}{2} \\ + \sum_{h>1} \sum_{v:\ \mathrm{height}(v)=h} \left(\frac{m^2}{2} + m^2 \min\{h-2, w-2\}\right)$$

Note that if a node v is at height 1 (e.g., a leaf) or 2 then we keep a pebble on that node for exactly one round (total cost $m^2/2$). If a node v is at height > 2 then we keep a pebble on that node for exactly $\min\{h-2, w-2\}$ additional rounds *after* we initially place the pebble (total cost $m^2/2 + m^2 \min\{h-2, w-2\}$).

To see that $CC^{||}(T_{D,w}) \geq CC(P)$ we note that in *any* legal pebbling of $T_{D,w}$ we must place a pebble on each node in $T_{D,w}$ at some point and that the total cost of placing these pebbles on each node for the first time is at least $|T_{D,w}|\, m^2/2$. After we first place a pebble on node v we must keep a pebble on node v for an additional $\min\{w-2, \max\{h-2, 0\}\}$ steps to pebble the $\min\{w-1, h-1\}$ children of node v. The total additional cost is $m^2 \min\{w-2, \max\{h-2, 0\}\}$for each node v. Therefore, $CC^{||}(T_{D,w}) \geq$

$$\sum_h \sum_{v:\ \text{height}(v)=h} \left(\frac{m^2}{2} + m^2 \min\{\max\{h-2, 0\}, w-2\} \right) \ .$$

Now consider an arbitrary sequential pebbling strategy and in particular consider the unique round i_v during which we first place a pebble on node v. We note that during round $i_v - 1$ we must have pebbles on all of v's parents, thus $|P_{i_v-1}| \geq |\mathsf{parents}(v)| \geq \min\{w-1, \text{depth}(v) - 1\}$. It follows that

$$\begin{aligned} CC(P) &\geq \left(\sum_v m^2 \min\{w-1, \text{depth}(v) - 1\} \right) \\ &\quad - |T_{D,w}| \frac{m^2}{2} \\ &= \left(\sum_d \sum_{v:\text{depth}(v)=d} m^2 \min\{d-1, w-1\} \right) \\ &\quad - |T_{D,w}|\, m^2/2 \ . \end{aligned}$$

□

Under the sequential black pebbling game a similar approach works. In this case, rather than the height of each node consider the depth of each node, which is the distance to the root of the trie. For each node realize that you must pay the initial pebbling cost and also pay $min\{w-1, d-1\}$ to keep its parents in the trie. Thus we gain a similar bound for the sequential pebbling game:

Table 4.2. Advantages of JIT hashing with selected dictionaries

List	$adv^{\|\|}(D, w)$	$adv(D, w)$
Rockyou	6.028	13.260
Cracking(1k)	10.100	15.542
Cracking (10k)	7.057	10.654
Cracking (100k)	3.048	6.068
Cracking ($\sim$ 5 mill)	9.154	12.468
XKCD (4 word)	11.674	25.399

To give some perspective we calculate the pebbling complexities for several password lists, including rockyou, xkcd-style passwords, and a cracking password list from Openwall, designed for use with John the Ripper[153]. We look at the advantage in terms of the CC of the JIT pebbling graph and the work required to calculate all passwords using memory hard hashing but not a JIT model i.e. each password requires only $m^2/2$ work to compute. We specifically define our advantages as $adv^{||}(D, w)$ and $adv(D, w)$ under parallel and sequential models for a dictionary of passwords D with a JIT hashing algorithm using a window of size w as

$$
\begin{aligned}
adv^{||}(D, w) &= \frac{CC^{||}(T_{D,w})}{m^2|D|/2} \\
adv(D, w) &= \frac{CC(T_{D,w})}{m^2|D|/2} \ .
\end{aligned}
$$

This can be thought of intuitively as the amount of work necessary for the attacker to check all passwords in the dictionary when passwords are protected with JIT divided by the work the attacker must perform when passwords are protected by a standard memory-hard function with equivalent authentication delay. We calculate these advantages over several password dictionaries in Table 4.2. Note that some of these require theoretical analysis rather than empirical. For full details please see [8].

4.2.4 Discussion

Usability advantages

A great benefit of JIT hashing is that, from a user's perspective, there is nothing new to learn. So long as the system has been implemented correctly most users should expect to be able to authenticate with no detectable delay. In a case when a user does notice a delay this would be because they are either typing much faster than expected, such as faster than 95% of users, or because they have deleted a significant amount, but not all, of the characters that they entered. In these cases, while there is a delay it is not a very significant delay, and should only last for a few seconds while the algorithm restarts computation from the beginning to catch up with the user.

From a developer's perspective, JIT hashing will require some modification to their existing authentication systems. Current password hashing functions are set up to take the entire password at once, while JIT is a streaming algorithm. Developers would need to modify their existing systems to accept passwords one character at a time, which may vary from simple to complex depending on the current systems they are working with. Beyond modification to be a streaming algorithm, the replacement of the function itself would be quick, only requiring the developer to import the function and set a few additional parameters.

Client vs Server-Side

In earlier sections, we described JIT Hashing as a client-side hashing algorithm. The reason for this is that a naive implementation for a server-side version could include several serious security or usability risks. For example, a naive implementation may send (encrypted) characters to the server one at a time. An adversary could eavesdrop in this scenario to learn the exact length of a user's password. One way to address this issue would be to have the client send an encrypted packet every few milliseconds whether or not a character was typed. A second consideration is that of server resources. Since JIT hashing involves extra work any server-side implementation must also consider the potentially increased risk of denial-of-service attacks.

4.3 Data-independent Memory Hard Functions: New Attacks and Stronger Constructions

As discussed in the Economics of Offline Password Cracking [3], one of the primary concerns for offline password cracking is an attacker's ability to guess for very low costs. One solution to raising attacker costs is to employ MHFs, which impose a higher space-time cost and are designed to be resistant to cheap ASIC computation. These properties are dependent on MHFs ability to *require* such large amounts of memory for efficient computation (i.e., there should not be some way to cheat and compute the function for a much lower cost). As of this writing, the only MHF proven to be maximally memory-hard was shown by Alwen et al in 2017 [34].

This work on iMHFs [7] improves the state of the art on independent MHF construction through proposed changes to existing iMHFs like Argon2 [16]. Due to the length of the main work the portions summarized in this thesis document focus on the portions I was most involved in. We will discuss our new MHF candidate, a new look at old attacks against MHFs and their effectiveness, and a minor change that we note can increase Argon2's resistance to parallelization. This work contains many additional relevant results and constructions, all of which can be found in the full version of the paper.

4.3.1 Existing constructions and attacks

In general, iMHFs can be described using a DAG where edges encode data dependencies required as an array of memory is filled. These DAGs can be analyzed using pebbling games (See Section 2.6 where the pebbles intuitively represent storing data in memory for later use. Several iMHF construction candidates have been proposed, with many appearing in the 2015 Password Hashing Competition [38]. Of the candidates proposed in this competition Argon2 [16] was selected as the winning construction. Subsequent work found flaws in the construction that reduced the cost of computation to $O\left(n^{1.768}\right)$ [174, 181], well below the theoretical maximum cost of $O\left(N^2 \log\log n / \log n\right)$ [56]. Alwen, Blocki, and Harsha proposed a new construction called DRSample (Definition 4.3.3) which resisted all *tested* attacks and provably has aAT cost $\Omega\left(\frac{N^2}{\log\log n}\right)$ [15].

A variety of possible attacks against iMHF constructions exists going back to Valiant's results from 1977 [182]. Additional attacks that have been tested in the past include the Layered attack [56] and greedy pebbling attack [172]. Importantly, this work finds that the greedy pebble can *significantly* reduce the cost of computing DRsample.

4.3.2 Greedy Pebbling Attack

In this section we present an empirical analysis of the greedy pebbling attack [172] that *reverses* previous conclusions about the *practical* security of Argon2i vs DRSample [15]. Additional theoretical results are presented as additional support of these conclusions in the full paper. The greedy pebble attack is an attack on MHFs that attempts to reduce the full cost of computation by strategically selecting certain blocks of memory to keep filled. Rather than storing all labels on a node (as would be done in the naive version of the algorithm) we identify nodes that are used more often than others and keep these blocks loaded while not storing the rest and computing them on the fly each time they are required. The formal definition is:

$\mathsf{gc}(v)$: For each node $v < N$ we let $\mathsf{gc}(v) = \max\{w | \ (v, w) \in E\}$ denote the maximum child of node v — if $v < N$ then the set $\{w | \ (v, w) \in E\}$ is non-empty as it contains the node $v + 1$. If node v has no children then set $\mathsf{gc}(v) := v$.

$\chi(i)$: This represents what we call the crossing set of the ith node. It is defined as $\chi(i) = \{v | v \leq i \ \wedge \ \mathsf{gc}(v) > i\}$. Intuitively this represents the set of nodes $v \leq i$ incident to a directed edge (v, u) that "crosses over" node i i.e. $u > i$.

4.3.3 Empirical Analysis of the GP Attack

We ran the greedy pebbling attack against several iMHF DAGs including Argon2i, DRSample and our new construction DRSample+BRG (introduced in Section 4.3.4) and compared the attack quality of the greedy pebbling attack with prior depth-reducing attacks. Here we define attack quality for an adversary $\mathcal{A}$ running pebbling P as $AT-quality(P) = \frac{n(n+1)/2+nR}{aAT(P)}$ where $n(n+1)/2 + nR$ is the cost of the naive pebbling strategy i.e. on round i

place a pebble on node i so nodes $[1, i]$ are pebbled. The results, seen in Figure 4.4, show that the GP attack was especially effective against the DRSample DAG, improving attack quality by a factor of up to 7 (at $n = 24$) when compared to previous state-of-the-art depth-reducing attacks (Valiant, Layered, and various hybrid approaches) [182, 56, 15].

The most important observation about Figure 4.4 is simply how effective the greedy pebbling attack is against DRSample. While DRSample may have the strongest asymptotic guarantees (i.e. $aAT(G) = \Omega(N^2/\log N)$ for DRSample vs. $aAT(G) = \mathcal{O}(N^{1.767})$ for Argon2i) Argon2i seems to provide better resistance to known pebbling attacks for *practical* parameter ranges. In our full paper, we show that the greedy pebble attack against DRSample has cost approximately $O\left(\frac{n^2}{\log n}\right)$.

Our tests found that while the Greedy Pebbling attack does sometimes outperform depth-reducing attacks at smaller values of n, the depth-reducing attacks appear to be superior once we reach graph sizes that would likely be used in practice. As an example, when $n = 20$ we find that the attack quality of the greedy pebbling attack is just 2.99, while the best depth-reducing attack achieved attack quality 6.25 [15]. However, we note that when we overlay BRG over DRS, we can capture the greedy pebbling resistant features of the BRG, and can effectively reduce the threat of this attack.

4.3.4 New MHF Candidate

In this paper, we are interested in understanding the cost of MHF computation from the perspective of Full Cost.Wiener [116] defined the full cost of an algorithm's execution to be the number of hardware components multiplied by the duration of their usage e.g., if the algorithm needs to allocate $\Omega(N)$ blocks of memory for $\Omega(N)$ time steps then full evaluation costs would scale quadratically. Several candidate constructions attempting to raise Full Cost for computation have been proposed, several of which are relevant to our analysis here. First, we have the current Argon2 construction [16]. Second is an earlier work introducing a new method of selecting predecessor blocks DRSample [15]. We will soon see that there exists an attack against both Argon2 and DRSample that causes concern, prompting our development of a new MHF candidate that is resistant to all tested attacks.

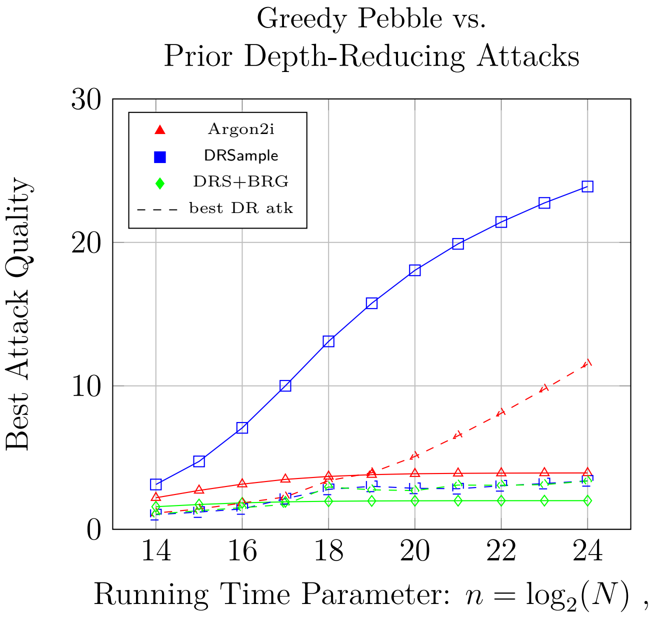

Figure 4.4. Attack Quality for Greedy Pebble and Greedy Depth Reduce

The new construction is obtained by overlaying a bit-reversal graph BRG_n [183] on top of a random DRSample DAG. If G denotes a random DRSample DAG with $N/2$ nodes then we will use $\mathsf{BRG}(G)$ to denote the bit-reversal overlay with N nodes. Intuitively, the result is a graph that resists both the greedy pebble attack (described in the next section, and which is effective against DRSample alone) and depth-reducing attacks (which DRSample was designed to resist). An even more exciting result is that we can show that DRSample+BRG is the first practical construction to provide strong sustained space complexity guarantees. Interestingly, neither graph (DRSample or BRG) is individually known to provide strong sustained space guarantees. Instead, several of our proofs exploit the synergistic properties of both graphs. The formal definition is given in two stages. First, we define the BRG and the related overlay version. Second, we define DRSample. Finally, we define the combination of the two "$DRS + BRG$".

Given a sequence of bits $X = x_1 \circ x_2 \circ \cdots x_n$, let $\mathsf{ReverseBits}(X) = x_n \circ x_{n-1} \circ \cdots \circ x_1$. Let $\mathsf{integer}(X)$ be the integer representation of bit-string X starting at 1 so that $\mathsf{integer}\left(\{0,1\}^n\right) = [2^n]$ i.e., $\mathsf{integer}(0^n) = 1$ and $\mathsf{integer}(1^n) = 2^n$. Similarly, let $\mathsf{bits}(v, n)$ be the length n binary encoding of $(v-1) \mod 2^n$ e.g., $\mathsf{bits}(1, n) = 0^n$ and $\mathsf{bits}\left(2^n, n\right) = 1^n$ so that for all $v \in [2^n]$ we have $\mathsf{integer}(\mathsf{bits}(v, n)) = v$.

Definition 4.3.1. *We use the notation* BRG_n *to denote the bit reversal graph with* 2^{n+1} *nodes. In particular,* $\mathsf{BRG}_n = \left(V = \left[2^{n+1}\right], E = E_1 \cup E_2\right)$ *where* $E_1 := \{(i, i+1) \;:\; 1 \leq i < 2^{n+1}\}$ *and* $E_2 := \{(x, 2^n + y) \;:\; x = \mathsf{integer}(\mathsf{ReverseBits}(\mathsf{bits}(y, n)))\}$. *That is,* E_2 *contains an edge from node* $x \leq 2^n$ *to node* $2^n + y$ *in* BRG_n *if and only if* $x = \mathsf{integer}(\mathsf{ReverseBits}(\mathsf{bits}(y, n)))$.

We now define the bit-reversal overlay of the bit reversal graph on a graph G_1. If the graph G_1 has N nodes then $\mathsf{BRG}(G_1)$ has $2N$ nodes, and the subgraph induced by the first N nodes of $\mathsf{BRG}(G_1)$ is simply G_1.

Definition 4.3.2. *Let* $G_1 = (V_1 = [N], E_1)$ *be a fixed DAG with* $N = 2^n$ *nodes and* $\mathsf{BRG}_n = (V = [2N], E)$ *denote the bit-reversal graph. Then we use* $\mathsf{BRG}(G_1) = (V, E \cup E_1)$ *to denote the bit-reversal overlay of* G_1.

Definition 4.3.3. *DRSample*$(N) = (V, E)$ *is defined as a set of* N *vertices* $V = \{1, 2, \ldots, N\}$ *with edges* $(v-1, v) \in E$ *for all* $v > 1$ *and additional edges leading to each vertex* $v \in V$

selected by first picking some $g \underset{\$}{\leftarrow} [1, \lfloor \log_2(v) \rfloor]$*. Next, set* $g = \min(v, 2^g)$*. Finally select* $r \underset{\$}{\leftarrow} [\max(g/2, 2), g]$ *and add* $(v - r, v)$ *to* E*.*

Definition 4.3.4. $DRS + BRG = \mathsf{BRG}(DRSample(N))$*.*

The full paper contains several theoretical results about the strength of DRS+BRG. Specifically, we show that any parallel pebbling P against DRS+BRG must have either $aAT(P) \in \omega(N^2)$ or the pebbling has high sustained space complexity i.e., requires $\Omega\left(\frac{N^2}{\log N}\right)$ pebbles for $\Omega(N)$ steps. This means any parallel pebbling has $aAT(P)\ \sigma\ \Omega\left(\frac{N^2}{\log N}\right)$. In addition, any sequential pebbling (or even any nearly sequential pebbling) P of DRS+BRG has $aAT(P) \in \Omega(N^2)$. Since the greedy pebbling attack is sequential, this shows that the GP attack (or simple variants) will not be effective against DRS+BRG.

4.3.5 Antiparallelism in the Argon2 Compression Function

In this section, we show how a parallel attacker could reduce aAT costs by nearly an order of magnitude by computing the Argon2i round function in parallel. We then present a tweaked round function to ensure that the function must be computed *sequentially*. Empirical analysis indicates that our modifications have *negligible* impact on the running time performance of Argon2 for the honest party (sequential), while the modifications will *increase* the attackers aAT costs by nearly an order of magnitude.

Review of the Argon2 Compression Function. We begin by briefly reviewing the Argon2 round function $\mathcal{G} : \{0,1\}^{8092} \rightarrow \{0,1\}^{8092}$ which takes two 1KB blocks X and Y as input and outputs the next block $\mathcal{G}(X, Y)$. $\mathcal{G}$ builds upon a second function $\mathcal{BP} : \{0,1\}^{128} \rightarrow \{0,1\}^{128}$, which is the Blake2b round function [184]. In our analysis we treat $\mathcal{BP}$ as a blackbox. For a more detailed explanation including the specific definition of $\mathcal{BP}$, we refer the readers to the Argon2 specification [16].

To begin, $\mathcal{G}$ takes the intermediate block $R = X \oplus Y$ (which is being treated as an 8x8 array of 16 byte values $R_0, \ldots, R_{63}$), and runs $\mathcal{BP}$ on each row to create a second intermediate stage Q. We then apply $\mathcal{BP}$ to Q column-wise to obtain one more intermediate value Z: Specifically:

$$\begin{aligned}
(Q_0, Q_1, \ldots, Q_7) &\leftarrow \mathcal{BP}(R_0, R_1, \ldots, R_7) & (Z_0, Z_8, \ldots, Z_{56}) &\leftarrow \mathcal{BP}(Q_0, Q_8, \ldots, Q_{56}) \\
(Q_8, Q_9, \ldots, Q_{15}) &\leftarrow \mathcal{BP}(R_8, R_9, \ldots, R_{15}) & (Z_1, Z_9, \ldots, Z_{57}) &\leftarrow \mathcal{BP}(Q_1, Q_9, \ldots, Q_{57}) \\
&\ldots \\
(Q_{56}, Q_{57}, \ldots, Q_{63}) &\leftarrow \mathcal{BP}(R_{56}, R_{57}, \ldots, R_{63}) & (Z_7, Z_{15}, \ldots, Z_{63}) &\leftarrow \mathcal{BP}(Q_7, Q_{15}, \ldots, Q_{63})
\end{aligned}$$

To finish, we have one last XOR, giving the result $\mathcal{G}(X, Y) = R \oplus Z$.

ASIC vs CPU AT cost. From the above description, it is clear that computation of the round function can be parallelized. In particular, the first (resp. last) eight calls to the permutation $\mathcal{BP}$ are all independent and could easily be evaluated in parallel i.e., compute $\mathcal{BP}(R_0, R_1, \ldots, R_7), \ldots, \mathcal{BP}(R_{56}, R_{57}, \ldots, R_{64})$ then compute $\mathcal{BP}(Q_0, Q_8, \ldots, Q_{56})$, $\ldots, \mathcal{BP}(Q_7, Q_{15}, \ldots, Q_{63})$ in parallel. Similarly, XORing the 1KB blocks in the first ($R = X \oplus Y$) and last ($\mathcal{G}(X, Y) = R \oplus Z$) steps can be done in parallel. This if we let $t^{ASIC}_{\mathcal{BP}}$ (resp. $t^{CPU}_{\mathcal{BP}}$) denote the time to compute $\mathcal{BP}$ on an ASIC (resp. CPU) we have $t^{ASIC}_{\mathcal{G}} \approx 2t^{ASIC}_{\mathcal{BP}}$ whereas $t^{CPU}_{\mathcal{G}} \approx 16 \times t^{CPU}_{\mathcal{BP}}$ since the honest party (CPU) must evaluate each call to $\mathcal{BP}$ sequentially. Suppose that the MHF uses the round function $\mathcal{G}$ to fill N blocks of size 1KB e.g., $N = 2^{20}$ is 1GB. Then the total area-time product on an ASIC (resp. CPU) would approximately be $\left(A^{ASIC}_{mem} N\right) \times \left(t^{ASIC}_{\mathcal{G}} N\right) \approx 2N^2 \times A^{ASIC}_{mem} t^{ASIC}_{\mathcal{BP}}$ (resp. $\left(A^{CPU}_{mem} N\right) \times \left(16 t^{CP}_{\mathcal{BP}} N\right)$ where A^{ASIC}_{mem} (resp. A^{ASIC}_{mem}) is the area required to store a 1KB block in memory on an ASIC (resp. CPU). Since memory is egalitarian we have $A^{ASIC}_{mem} \approx A^{CPU}_{mem}$ whereas we may have $t^{ASIC}_{\mathcal{BP}} \ll t^{CPU}_{\mathcal{BP}}$. If we can make $\mathcal{G}$ inherently sequential then we have $t^{ASIC}_{\mathcal{G}} \approx 16 t^{ASIC}_{\mathcal{BP}}$, which means that the new AT cost on an ASIC is $16N^2 \times A^{ASIC}_{mem} t^{ASIC}_{\mathcal{BP}}$ which is eight times higher than before. We remark that the change would not necessarily increase the running time $N \times t^{CPU}_{\mathcal{G}}$ on a CPU since evaluation is already sequential. We stress that the improvement (resp. attack) applies to *all* modes of Argon2 both data-dependent (Argon2d,Argon2id) and data-independent (Argon2i), and that the attack could potentially be combined with other pebbling attacks [56, 172].

We remark that the implementation of $\mathcal{BP}$ in Argon2 is heavily optimized using SIMD instructions so that the function $\mathcal{BP}$ would be computed in parallel on *most* computer architectures. Thus, we avoid trying to make $\mathcal{BP}$ sequential as this would slow down *both* the attacker *and* the honest party i.e., both $t_{\mathcal{BP}}^{CPU}$ and $t_{\mathcal{BP}}^{ASIC}$ would increase.

Inherently Sequential Round Function. We present a small modification to the Argon2 compression function which prevents the above attack. The idea is simply to inject extra data-dependencies between calls to $\mathcal{BP}$ to ensure that an attacker must evaluate each call to $\mathcal{BP}$ sequentially just like the honest party would. In short, we require the first output byte from the $i-1^{th}$ call to $\mathcal{BP}$ to be XORed with the i^{th} input byte for the current (i^{th}) call.

In particular, we now compute $\mathcal{G}(X,Y)$ as:

$$
\begin{array}{cc}
(Q_0, Q_1, \ldots, Q_7) \leftarrow \mathcal{BP}(R_0, R_1, \ldots, R_7) & (Z_0, Z_8, \ldots, Z_{56}) \leftarrow \mathcal{BP}(Q_0, Q_8, \ldots, Q_{56}) \\
(Q_8, Q_9, \ldots, Q_{15}) \leftarrow \mathcal{BP}(R_8, R_9 \oplus Q_0, \ldots, R_{15}) & (Z_1, Z_9, \ldots, Z_{57}) \leftarrow \mathcal{BP}(Q_1, Q_9 \oplus Z_0, \ldots, Q_{57}) \\
\ldots & \ldots \\
(Q_{56}, Q_{57}, \ldots, Q_{63}) \leftarrow \mathcal{BP}(R_{56}, R_{57}, \ldots, R_{64} \oplus Q_{48}) & (Z_7, Z_{15}, \ldots, Z_{63}) \leftarrow \mathcal{BP}(Q_7, Q_{15}, \ldots, Q_{63} \oplus Z_6)
\end{array}
$$

where, as before, $R = X \oplus Y$ and the output is $\mathcal{G}(X,Y) = Z \oplus R$.

We welcome cryptanalysis of both this round function and the original Argon2 round function. We stress that the primary threat to passwords is brute-force attacks (not hash inversions/collisions etc...) so increasing evaluation costs is arguably the primary goal.

4.3.6 Conclusions

We have seen about these new MHF constructions we can investigate their impact on a rational attacker's decisions. As we saw in [3], setting hashing costs too low can put users at significant risk of losing an account to an offline password cracking attack. Attacks such as the greedy pebble attack we saw here have the capability of letting a rational adversary cheat and reduce their costs. With these reduced costs they may change their optimal attack decisions in a manner we would prefer to avoid i.e., they may decide to attack more users or spend more effort per user, resulting in more damage from an attack. Thankfully, we can reduce the effect of state-of-the-art cost-reduction attacks using the new constructions we have just seen. By preventing cost reduction we can require an adversary to pay more egalitarian computation costs, allowing us to continue to reap the attack damage reductions from MHFs that we first demonstrated in [3].

4.4 Timely and Effective Key-Stretching with ASICs

When a password database is breached and an attacker obtains a service's password records, the final line of defense for the users is the strength of the storage method used. The standard method of storing passwords involves keeping a record for each user containing their username, a salt value, and a cryptographic hash of their password and salt. An attacker running a typical brute force guessing attack is mainly limited by how quickly they can compute this cryptographic hash function to check if a password guess matches the user's record in the database.

Because this hash function serves as a bottleneck for an adversary, various strategies have been employed to reduce the number of guesses per second an attacker can make. Typically, these strategies are based on increasing the computation time and space for the hash function used to store each password, a method called key-stretching. As of this writing, there are two commonly used approaches to increase the cost of computation. The first, hash iteration, works by iteratively feeding the output of some base hash function H (e.g., a version of SHA) back into itself some set number of times k. In this construction both the total cost to compute this function and authentication time scale linearly with k. A second method, Memory Hard Functions (MHFs) increases guessing costs by requiring some set amount of memory time and memory m, t that is filled during computation in t steps. While these two variables can be set separately they The total space-time cost for the naive algorithm to compute these functions is $O\left(m^2\right)$[10].

One area of concern regarding password cracking involves a possible asymmetry between service providers and offline password crackers. While service providers typically use

---

[10] Several additional parameters may slightly change the space-time cost for these functions such as requiring multiple passes over the memory blocks. The details here depend on the MHF in question.

standard commercially available hardware to perform password hashing, an adversary can construct an ASIC to assist their cracking efforts. While to our knowledge no such ASIC is currently commercially available, similar products such as the Antminer line [185] have shown that base hash functions can be called at a very high rate using an ASIC. This would allow an attacker who constructed such a device to compute password hashes much faster than a service provider using their standard hardware.

Blocki et al. [3] suggest that this asymmetry combined with linear cost scaling would render typical levels of hash iteration insufficient to defend against an ASIC-powered attacker. They argue that MHFs offer a reasonable solution to this asymmetry. By requiring large amounts of memory, even an ASIC would be bottlenecked by slow memory access rather than by computation speed. This results in a password hashing function where an ASIC would not have as strong an advantage as with hash iteration and that has quadratic cost scaling. In this thesis, we will explore an additional method of countering this asymmetry: using an ASIC to authenticate users.

We compare a proposed ASIC-based authentication system with existing MHFs used for the same purpose. MHFs have the advantage of requiring a much lower investment in equipment compared to ASIC-based hashing. However, they require more time than an ASIC to provide hashing at a similar cost. In the following sections, we will explore the tradeoffs between these two factors along with possible usability constraints, as users are not infinitely patient. To compare these two solutions, we consider a defender who is interested in the following three factors:

1. How much does it cost for the adversary to make a single password guess (C_{guess}) in an offline password cracking attack?
2. What is the authentication delay imposed by the password hashing algorithm?
3. How much does it cost the defender per authentication (C_{auth})?

Intuitively, we want to increase C_{guess} as much as possible to deter attackers, but may be constrained by server resources from C_{auth} or user patience during authentication. We develop a model that allows us to answer the question of costs given certain authentication times, allowing us to decide which of these two systems is more appropriate in a given context.

Contributions: In this paper we introduce a per-guess and per-authentication model for both MHF and ASIC-based authentication. By examining the differences in cost-per-guess (which models attacker cost) and cost-per-authentication (which models defender costs), our models can advise a server owner on which authentication method maximizes attacker cost subject to system requirements (C_{auth} and authentication delay). We find that for smaller authentication times (e.g., less than one second) ASIC-based systems can provide significantly higher attacker costs in the context of an offline password cracking attack. We also note that MHF based systems remain a reasonable budget option for smaller-scale applications. This is because amortized C_{auth} is much higher for ASIC-based systems. In this situation, MHF-bases systems are still capable of offering reasonable user protection. Finally, we provide some example scenarios demonstrating when each authentication method would be most appropriate.

4.4.1 ASIC-Based Password Hashing

A typical server hardware-based hashing system achieves key-stretching either through hash iteration or through memory-hard functions. A third proposed system is to keep some secret nonce called a pepper value (sometimes called a secret salt) which is not stored on any machine and which must be guessed on every authentication [17, 186, 93]. These solutions increase the guessing cost per password guess by forcing at least a small brute force attack for every authentication attempt or offline password guess. Kedem and Ishihara [186] specifically argue that using pepper in a password storage system may be a good defense against highly parallel attacks, as one would see in a modern ASIC-based attack.

We propose using a pepper system for key stretching in an ASIC-based authentication system. This method of key stretching allows us to take advantage of the high level of parallelism within an ASIC while also providing a straightforward method of partitioning the work to send to individual cores within an ASIC. Given c cores and a λ-bit pepper value we can easily assign $\frac{2^\lambda}{c}$ values to each core to be checked in parallel. If any one of the cores finds that a given pepper value is valid then the authentication succeeds.

User Registration

User registration in an ASIC-based authentication system is very similar to the typical $(user, salt, \mathcal{H}(salt||pwd_{user}))$ entry stored per person, where $\mathcal{H}$ is some cryptographic hashing function. The primary change is an addition of some secret pepper value included in the hash input but not stored outright. The new record stored per person is $(user, salt, \mathcal{H}(salt||pepper||pwd_{user}))$. User registration itself does not require the use of an ASIC, as we only need to select our pepper value $pepper \in \{0,1\}^\lambda$ from a distribution, e.g., from uniform (which we use here) or an optimized distribution as described in [17].

User Authentication and Password Guessing

When a user submits an authentication request $r = (username, pwd)$ we must begin a search over the elements in $\{0,1\}^\lambda$. At this point, the server can make a query to a connected ASIC designed to brute force a pepper value. The values $(\mathcal{H}(salt||pepper||pwd), salt, pwd)$ are forwarded to the ASIC which distributes some subspace of $\{0,1\}^\lambda$ to each core. Each core iterates through its subspace until it either reaches the end or it has found a matching value that authenticates the user. If a matching value is found the authentication is successful. We note that password cracking follows an almost identical process, with many possible values *pwd* forwarded to the ASIC to check if they match the record in a database. As noted by Blocki and Datta [17], the costs of doing this exceed the costs of typical authentications due to most guesses being incorrect. The adversary therefore must check the entire pepper range most times whereas the server will typically stop early due to a correct input.

4.4.2 Modeling Authentication Costs

Before we can develop a model that meets the three requirements our defender specifies we require quite a few parameters representing various aspects of the authentication system. In many cases, these parameters are easily fixed by searching for commercially available equipment and utilities such as existing hashing ASICs or RAM chips. Additionally, several parameters are independent of the authentication system used, and primarily represent the defender's speed and cost requirements. These parameters common to both systems are:

- C_e: The cost of electricity (USD per kW/hr).
- A_{max}: The maximum number of authentications per second the server must handle during peak load.
- A_{avg}: The average number of authentications per second the server is expected to handle.

The first of these, C_e, can be quickly estimated based upon a service provider's current rate. We use a value of $C_e = 0.12USD$ as a general estimate for the cost of electricity for commercial purposes [187]. This value will of course vary depending on the electricity consumer's exact location.

ASIC-based Hashing Parameters

Many of the parameters required to model an ASIC-based password hashing system represent the capabilities and costs of the specific ASIC used. We assume that these parameters are instantiated with values representing a single ASIC, not as general representations of the tools as a whole. Because our proposed ASIC-based hashing system is based upon a pepper system, we also require parameters specifying pepper length.

- C_{ASIC}: The cost of purchasing an ASIC designed to authenticate using the pepper-based method described in Section 4.4.
- L_{ASIC}: The lifespan of the ASIC measured in seconds.
- S_{ASIC}: The speed (measured in $\mathcal{H}/s$) of the password hashing ASIC.
- ℓ_p: The length, in bits, of the pepper value.
- E_{ASIC}: The energy usage (measured in Watts) of the ASIC.

MHF-based Hashing Parameters

Unlike ASIC-based hashing, MHF based hashing is not significantly impacted by the choice of a specific hardware device. Compared to commercially available ASICS, commercially available RAM cards do not vary as much in cost or performance. We consider the following factors in an MHF-based authentication system.

- C_{GB}: The cost for one GB of RAM .
- L_{RAM}: The lifespan of a RAM card, in seconds.
- E_B: The power required (in joules) to transfer/access one byte in RAM.
- m: The memory/time parameter specifying the number of GBs the MHF should use.
- t_{GB}: The time it takes to fill one GB of memory.

4.4.3 Modeling ASIC-Based Password Hashing Costs

To construct our ASIC cost model we consider two primary factors in the cost of both guessing and authentication: Capital costs and energy costs (with energy costs similar to [129, 116]). Capital costs represent some amortized component per attacker guess or defender authentication representing the initial investment in equipment required for an ASIC-based system. We begin with the costs per guess for an attacker $C_{guess,ASIC}$. If we have an ASIC that costs C_{ASIC} USD and has a lifetime of L_{ASIC} we have an equipment cost per second of life of $\frac{C_{ASIC}}{L_{ASIC}}$. The product of this value and the time it takes per guess $\frac{2^{\ell_p}}{S_{ASIC}}$ gives our equipment cost per guess. The second factor measures the electricity cost per guess and is equal to the time per guess times the cost per second $\frac{2^{\ell_p} C_e E_{ASIC}}{S_{ASIC}}$. Combining these gives the cost per attacker guess

$$C_{Guess,ASIC} = \frac{C_{ASIC} 2^{\ell_p}}{L_{ASIC} S_{ASIC}} + \frac{2^{\ell_p} E_{ASIC} C_E}{S_{ASIC}}$$

The costs to authenticate a user are slightly different due to a) the lower expected authentication time for legitimate authentication attempts and b) the need to purchase equipment for peak traffic instead of average traffic. The former decreases our time per authentication to $\frac{2^{\ell_P-1}}{S_{ASIC}}$. To handle a defender amortizing the purchase of additional equipment we note that they require at least $\left\lceil \frac{2^{\ell_P-1}A_{max}}{S_{ASIC}} \right\rceil$ ASICs to handle peak loads. On average, we assign a (capital) cost of $\left\lceil \frac{2^{\ell_P-1}A_{max}}{S_{ASIC}} \right\rceil \frac{C_{ASIC}}{L_{ASIC}A_{avg}}$ per authentication to the defender. The electrical costs per authentication remain the same for a defender, giving the authentication cost model

$$C_{Auth,ASIC} = \left\lceil \frac{2^{\ell_P-1}A_{max}}{S_{ASIC}} \right\rceil \frac{C_{ASIC}}{L_{ASIC}A_{avg}} + \frac{2^{\ell_p-1}E_{ASIC}C_e}{S_{ASIC}}$$

4.4.4 Modeling MHF-based Password Hashing Costs

As with our ASIC model, we break the cost of computing a MHF into two components: hardware costs and software costs. For an attacker, as we increase our memory parameter m we require $m^2 t_{GB}$ gigabyte-seconds to compute the function at a cost of $\frac{C_{GB}}{L_{GB}}$ per gigabyte-second.

We use existing estimates of the amount of energy required to transfer/access memory per byte C_t with an estimate of two transfers per byte (as is required to computer MHFs like Argon2i [16] and SCRYPT [147]). The electrical cost estimates for $2m$ transfers for C_t joules per transfer comes out to $2mC_tC_e$. Combining gives our attacker guessing cost of

$$C_{Guess,MHF} = m^2 t_{GB} \frac{C_{GB}}{L_{GB}} + 2mC_tC_e$$

Similar to how we assigned higher hardware costs per authentication to a defender in an ASIC-based system we assign a higher cost per authentication in the MHF model as well. However, due to the ability to buy memory in much more specific quantities, we allow the defender to purchase the exact amount they require. This is in contract with ASIC-based systems where a very large investment is required to even slightly increase capacity above current capabilities.

$$C_{Auth,MHF} = \frac{A_{max}}{A_{avg}} m^2 t_{GB} \frac{C_{GB}}{L_{GB}} + 2mC_tC_e$$

4.4.5 Selection of Parameters

We begin our comparison of ASIC and MHF-based authentication by modeling a few example authentication systems. This requires us to instantiate several cost parameters which can vary based on the specific model. ASIC prices for cryptocurrency miners, for example, can vary by thousands of USD for similar products, which can lead to large differences in authentication costs. Even with these large costs, however, the base cost per hash computation, $C_{\mathcal{H}}$, remains within an order of magnitude between devices (see Figure 4.3 for sample values). To account for variability in prices and computational power we examine multiple existing commercially available ASIC (See Table 4.3). We select these values out of a desire to not underestimate attacker guessing costs, which themselves are strongly influenced by hardware costs.

For MHF-based authentication, we have ready access to commercially available equipment. While prices vary slighting, as of this writing in June 2021 8GB DDR4 SDRAM cards were available for around 40 USD from sites like Newegg [188] or Amazon [189]. Taking this as a typical price we set $C_{GB} = 5.00USD$. Setting t_{GB} is not as straightforward as calculating the value from manufacturer specifications. Here we estimate t_{GB} by running an instance of Argon2i [16] set to 1GB of RAM usage in single-pass, single-threaded mode on a machine with 16GB DDR4 RAM and an Intel i5-6600k 3.50GHz processor. Observed running time values varied from 1.05 to 1.07 seconds, which is consistent with previous estimates [16]. Given these observations, we estimate the running time for a 1GB MHF evaluation to be $t_{GB} = 1s$. For electrical usage, we take the same approach as Ren and Devadas [129] using the Intel Power Gadget [190]. Using the same Argon2i settings used for time estimates and assuming 2 transfers per byte we saw energy consumption consistent with Ren and Devada's estimate of $0.3nJ/Byte$ for RAM transfer energy costs. For 2 billion transfers over 1 second this comes out to $E_{GB} = 0.6J/GB$. For lifespan estimates we let $L_{ASIC} = L_{GB} = 2$ years, as has been estimated in previous work involving MHFs [7]. We note that this may not represent the time to hardware failure, and may include system upgrades or equipment deprecation as reasons for hardware replacement. These estimates are summarized in Table 4.3.

Table 4.3. Example cost parameters

MHF Parameters	
C_{GB}	5 USD / GB
L_{RAM}	2 years
E_{RAM}	$0.6J/GB$
t_{GB}	1 second

Antminer S19	
C_{ASIC}	12000 USD
L_{ASIC}	2 years
E_{ASIC}	$3250W$
S_{ASIC}	$110e12\mathcal{H}/s$
$C_{\mathcal{H}}$	$1.73e-18$

Antminer S9	
C_{ASIC}	750 USD
L_{ASIC}	2 years
E_{ASIC}	$1323W$
S_{ASIC}	$13.5e12\mathcal{H}/s$
$C_{\mathcal{H}}$	$8.84e-19$

ASIC estimates can vary much more than those we use for MHFs. While many RAM cards are standard equipment with consistent speeds and market prices, ASICs have prices and hash calculations speeds that can vary by orders of magnitude. As mentioned in Section 4.4 we primarily consider miners designed to find a nonce while calculating instances of SHA-256 as would be done for cryptocurrencies like Bitcoin.

Because bitcoin mining ASICs are so varied in price and capabilities we will consider two example miners as a source for cost estimates. The first miner we consider is the Antminer S19 Pro manufactured by Bitmain [191]. Table 4.3 contains the manufacturer specifications and prices based on June 2021 Amazon prices [192]. Similarly, Table 4.3 shows specifications and prices for an older, less costly model.

Using the estimates from Section 4.4.5 we plot both the authentication and guessing costs for the example parameters in Table 4.3. Attacker guessing costs can be found in Figure 4.5 and related defender authentication costs are shown in Figure 4.6. We compare MHF authentication and S19 ASIC cost estimates along with their cost ratios (The ratio of the attacker's cost to the defender's cost) in Figure 4.6. In this figure, we also include a cost estimate for a hash iteration strategy to offer some perspective on both MHF and ASIC-based methods. To calculate hash iteration costs we use Bonneau's estimate of 8.5×10^{-16} USD per iteration running on a typical CPU accomplishing a hash rate of 10MHz [92].

4.4.6 Discussion

From our experiments we note a few significant findings:

- ASIC-based authentication can provide higher attacker costs than MHFs when holding authentication time constant. These systems also provide higher attacker costs than defender costs for the same parameters. This is significant as it may provide a reasonable method for service providers to provide high attacker guessing costs without the need to spend unreasonable amounts themselves.

- MHF-based systems provide a lower level of cost scaling compared to ASIC-based systems. However, they have a lower operating cost overall compared to ASIC-based systems. Even with these lower costs, existing work [3] still notes that they are well-suited to deterring rational offline password cracking attacks.

- The ratio of attacker to defender costs is higher for ASIC-based systems (about 1.5 in the example shown in Figure 4.6 compared to 0.5 for MHFs). The primary cause of this disparity is the properties of pepper-based systems, which provide lower expected costs for service providers when compared to adversaries [17].

- *Both* MHF and ASIC-based systems have enormous cost advantages compared to hash iteration. The costs of hash iteration are many orders of magnitude smaller than both MHF and ASIC-based authentication systems *even if we restrict the adversary to using the same equipment as the defender*. For example, if we use even a low-cost ASIC to set guessing costs at $10e-6$ USD and assume accounts are worth 1 USD, then we can protect *at least* 60% of users from a rational offline cracking attacker with perfect distribution knowledge. In comparison, a rational attacker with perfect knowledge would be expected to crack 100% of passwords if less than 2 seconds of hash iteration time is used.

We include iteration costs and point 3 to emphasize that, although it seems MHF-based systems have much lower costs than ASIC-based systems, they still offer a higher authentication cost for reasonable authentication times compared to hash iteration. As demonstrated in [3] MHFs remain a reasonable choice for authentication.

Example use cases: We can now use these estimates to recommend different solutions for different companies. Let us consider two companies: One a financial institution and the other a small hobby-based forum.

- Suppose a financial company requires a guessing cost of at least 1.0×10^{-5} USD with authentication times under 1 second. Additionally, suppose this company has a usage profile of $A_{max} = 1000$ and $A_{avg} = 500$. From figure 4.5 we see that both ASIC options presented would offer reasonable authentication options for this company. Next, let us consider the much different case of a hobby forum. Due to the company's given restrictions, memory-hard functions would not be able to provide sufficiently high guessing costs in the allotted time. The smaller-scale forum sees many fewer people per day, with $A_{max} = 2$ and $A_{avg} = 0.01$ (corresponding to about 1000 active daily users). They would like to keep authentication times under 1 second, similar to the financial institution, but due to the nature of their much smaller scale have a maximum authentication budget of 2.7×10^{-4} (which comes out to about 10 USD per year). Due to the higher amortized cost per authentication, even an Antminer S9 would have an authentication cost of 0.001 USD, nearly an order of magnitude out of budget. In this case, the hobbyist website would be best served by choosing to use MHF-based authentication.

The two example illustrations demonstrate that ASIC-based systems are desirable when we desire 1) high attacker guessing costs with limited authentication times and 2) the organization sees a sufficiently large number of authentications per second. Smaller-scale situations can maximize security within budget constraints by using a MHF-based solution where ASIC-based solutions may be prohibitively expensive. Because previous work has

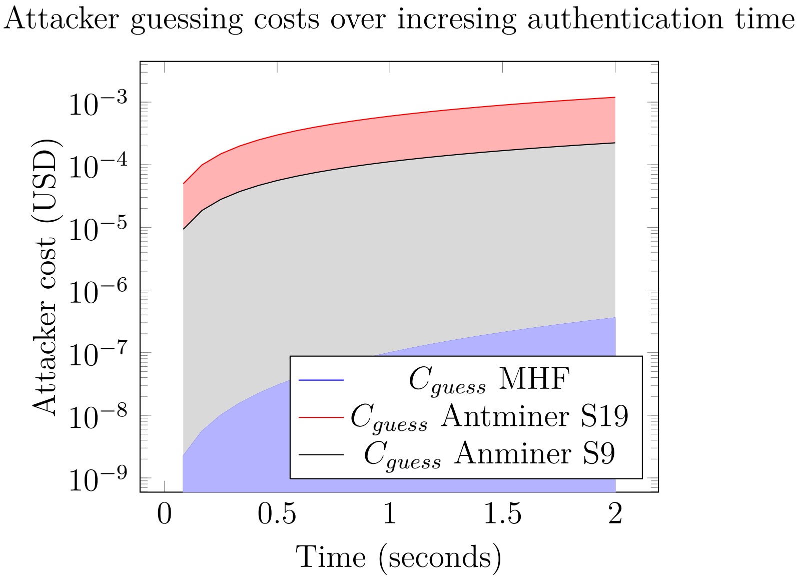

Figure 4.5. Maximum Achievable Attacker guessing costs for the variables listed in Table 4.3.

shown that MHF-based systems are likely to protect a majority of users [3] and given that our models predict even higher guessing costs for ASIC-based systems, we believe that both MHF and ASIC based systems are reasonable methods of protecting users from rational offline password cracking attackers.

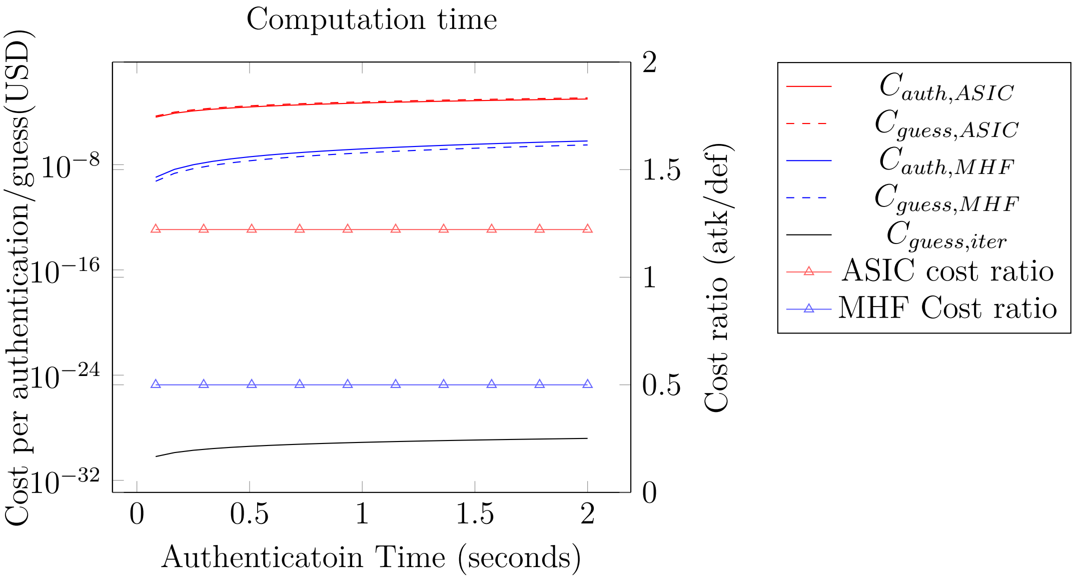

Figure 4.6. Cost scaling based on desired adversary guessing cost $C_{guess,\cdot}$. $A_{max} = 1000$, $A_{avg} = 500$.

5. FINAL REMARKS

In this thesis, I have demonstrated two key points regarding rational adversaries. In Chapter 3 we saw how we can model rational attackers in the context of password cracking and Grover's algorithm attacks against ideal ciphers, both of which are imperfectly secure systems due to the feasibility of brute-force attacks. By examining these systems from the lens of a rational adversary we have identified problem areas (such as insufficient key-stretching and length leakage) and also provided reassurance that existing systems like AES-128 would be sufficiently secure against rational attackers. Due to the astounding scale and frequency [88] of attacks like password cracking attacks, the security community must ensure systems are designed to resist these attacks.

In Chapter 4 we focused on methods of deterring rational attackers by increasing attackers' costs. Through increased attacker costs we can lower an attacker's utility so long as the reward remains fixed - ideally causing them to select an action that we would prefer. We presented modifications to password storage systems, stronger hashing function constructions, and methods to increase key-stretching computation time without causing undue burdens on end-users. Through these cost increases, we have shown that the severity of attacks like offline password cracking attacks by rational adversaries can protect many users from the consequences of an attack.

The results from this document demonstrate the value of rational adversary models. The security problems we have examined in this thesis seem dire when examined in the context of powerful, malicious adversaries. Even though these are imperfectly secure systems, persistent problems like password breaches make improving security as much as possible an important goal. Through the works in this thesis, we have used rational adversaries to show how these improvements can be accomplished. Though these improvements may not offer total security, they help ensure we can at least secure as many users as possible.

Funding Acknowledgments

The work in this thesis was made possible by generous funding from:

- **Rolls-Royce** through the Rolls-Royce Doctoral Fellowship
- **Intel** through a funded Research Assistantship
- **The National Science Foundation** through NSF grants 1704587 and 1755708.

The opinions expressed in this thesis are the opinions of the authors and do not necessarily reflect those of the National Science Foundation, Intel, or Rolls-Royce.

REFERENCES

[1] E. Barker and Q. Dang, "Nist special publication 800-57 part 1, revision 4," *National Institute of Standards & Technology*, 2016.

[2] E. B. Barker and A. L. Roginsky, "Sp 800-131a. transitions: Recommendation for transitioning the use of cryptographic algorithms and key lengths," 2011.

[3] J. Blocki, B. Harsha, and S. Zhou, "On the economics of offline password cracking," in *2018 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, May 2018, pp. 853–871.

[4] B. Harsha, J. Blocki, J. Springer, and M. Dark, "Bicycle attacks considered harmful: Quantifying the damage of widespread password length leakage," *Computers & Security*, vol. 100, p. 102068, 2021.

[5] B. Harsha and J. Blocki, "An economic model for quantum key-recovery attacks against ideal ciphers," *Workshop on the Economics of Information Security*, Dec 2020.

[6] W. Bai, J. Blocki, and B. Harsha, "Information signaling: A counter-intuitive defenseagainst password cracking," *arXiv preprint arXiv:2009.10060*, 2020.

[7] J. Blocki, B. Harsha, S. Kang, S. Lee, L. Xing, and S. Zhou, "Data-independent memory hard functions: New attacks and stronger constructions," in *CRYPTO 2019, Part II*, ser. LNCS, A. Boldyreva and D. Micciancio, Eds., vol. 11693. Springer, Heidelberg, Aug. 2019, pp. 573–607.

[8] B. Harsha and J. Blocki, "Just in time hashing," in *2018 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, 2018, pp. 368–383.

[9] N. Perlroth, "Yahoo says hackers stole data on 500 million users in 2014," Sep 2016. [Online]. Available: https://www.nytimes.com/2016/09/23/technology/yahoo-hackers.html

[10] S. Perez, "117 million linkedin emails and passwords from a 2012 hack just got posted online," May 2016. [Online]. Available: https://techcrunch.com/2016/05/18/117-million-linkedin-emails-and-passwords-from-a-2012-hack-just-got-posted-online/

[11] R. Hackett, "Linkedin lost 167 million account credentials in data breach," May 2016. [Online]. Available: https://fortune.com/2016/05/18/linkedin-data-breach-email-password/

[12] N. Cubrilovic, "Rockyou hack: From bad to worse," Dec 2009. [Online]. Available: https://techcrunch.com/2009/12/14/rockyou-hack-security-myspace-facebook-passwords/

[13] J. Siegrist, "Lastpass hacked – identified early and resolved," 2015.

[14] Z. Whittaker, "Home chef confirms breach after 8 million user records found on dark web," May 2020.

[15] J. Alwen, J. Blocki, and B. Harsha, "Practical graphs for optimal side-channel resistant memory-hard functions," in *ACM CCS 2017*, B. M. Thuraisingham, D. Evans, T. Malkin, and D. Xu, Eds. ACM Press, Oct. / Nov. 2017, pp. 1001–1017.

[16] A. Biryukov, D. Dinu, and D. Khovratovich, "Argon2: new generation of memory-hard functions for password hashing and other applications," in *2016 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, 2016, pp. 292–302.

[17] J. Blocki and A. Datta, "CASH: A cost asymmetric secure hash algorithm for optimal password protection," in *CSF 2016Computer Security Foundations Symposium*, M. Hicks and B. Köpf, Eds. IEEE Computer Society Press, 2016, pp. 371–386.

[18] O. Morgenstern and J. Von Neumann, *Theory of games and economic behavior*. Princeton university press, 1944.

[19] J. Pita, M. Tambe, C. Kiekintveld, S. Cullen, and E. Steigerwald, "Guards—innovative application of game theory for national airport security," in *Twenty-Second International Joint Conference on Artificial Intelligence*, 2011.

[20] S. Gholami, S. Mc Carthy, B. Dilkina, A. Plumptre, M. Tambe, M. Driciru, F. Wanyama, A. Rwetsiba, M. Nsubaga, J. Mabonga *et al.*, "Adversary models account for imperfect crime data: Forecasting and planning against real-world poachers (corrected version)," in *17th International Conference on Autonomous Agents and Multiagent Systems*, 2018.

[21] D. R. Songer, C. M. Cameron, and J. A. Segal, "An empirical test of the rational-actor theory of litigation," *The Journal of Politics*, vol. 57, no. 4, pp. 1119–1129, 1995.

[22] E. Kamenica and M. Gentzkow, "Bayesian persuasion," *American Economic Review*, vol. 101, no. 6, pp. 2590–2615, October 2011. [Online]. Available: https://www.aeaweb.org/articles?id=10.1257/aer.101.6.2590

[23] S. D. Gordon and J. Katz, "Rational secret sharing, revisited," in *International Conference on Security and Cryptography for Networks*. Springer, 2006, pp. 229–241.

[24] J. Halpern and V. Teague, "Rational secret sharing and multiparty computation," in *Proceedings of the thirty-sixth annual ACM symposium on Theory of computing*, 2004, pp. 623–632.

[25] A. Lysyanskaya and N. Triandopoulos, "Rationality and adversarial behavior in multi-party computation," in *Annual International Cryptology Conference*. Springer, 2006, pp. 180–197.

[26] J. R. Wallrabenstein and C. Clifton, "Equilibrium concepts for rational multiparty computation," in *International Conference on Decision and Game Theory for Security*. Springer, 2013, pp. 226–245.

[27] J. Garay, J. Katz, U. Maurer, B. Tackmann, and V. Zikas, "Rational protocol design: Cryptography against incentive-driven adversaries," in *2013 IEEE 54th Annual Symposium on Foundations of Computer Science.* IEEE, 2013, pp. 648–657.

[28] R. Cleve, "Limits on the security of coin flips when half the processors are faulty," in *Proceedings of the eighteenth annual ACM symposium on Theory of computing*, 1986, pp. 364–369.

[29] A. Groce and J. Katz, "Fair computation with rational players," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques.* Springer, 2012, pp. 81–98.

[30] W. Bai and J. Blocki, "Dahash: Distribution aware tuning of password hashing costs," in *Financial Cryptography and Data Security.* Springer International Publishing, 2021.

[31] A. D. Europe, "Antminer s9," http://www.antminerdistribution.com/antminer-s9/, (Retrieved November 13, 2016).

[32] C. Percival, "Stronger key derivation via sequential memory-hard functions," in *BSDCan 2009*, 2009.

[33] X. Boyen, "Halting password puzzles: Hard-to-break encryption from human-memorable keys," in *USENIX Security 2007*, N. Provos, Ed. USENIX Association, Aug. 2007.

[34] J. Alwen, B. Chen, K. Pietrzak, L. Reyzin, and S. Tessaro, "Scrypt is maximally memory-hard," in *EUROCRYPT 2017, Part III*, ser. LNCS, J.-S. Coron and J. B. Nielsen, Eds., vol. 10212. Springer, Heidelberg, Apr. / May 2017, pp. 33–62.

[35] P. A. Grassi, J. L. Fenton, E. M. Newton, R. A. Perlner, A. R. Regenscheid, W. E. Burr, J. P. Richer, N. B. Lefkovitz, J. M. Danker, Y.-Y. Choong, K. K. Greene, and M. F. Theofanos, "Digital identity guidelines: authentication and lifecycle management," June 2017. [Online]. Available: http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-63b.pdf

[36] L. von Ahn, M. Blum, N. J. Hopper, and J. Langford, "CAPTCHA: Using hard AI problems for security," in *EUROCRYPT 2003*, ser. LNCS, E. Biham, Ed., vol. 2656. Springer, Heidelberg, May 2003, pp. 294–311.

[37] C. Lee, "Litecoin," 2011.

[38] J.-P. A. et al., "Password hashing competition," 2015, https://password-hashing.net/.

[39] J. Alwen and V. Serbinenko, "High parallel complexity graphs and memory-hard functions," in *47th ACM STOC*, R. A. Servedio and R. Rubinfeld, Eds. ACM Press, Jun. 2015, pp. 595–603.

[40] C. E. Hewitt and M. S. Paterson, "Record of the Project MAC Conference on Concurrent Systems and Parallel Computation," J. B. Dennis, Ed. New York, NY, USA: ACM, 1970, ch. Comparative Schematology, pp. 119–127. [Online]. Available: http://doi.acm.org/10.1145/1344551.1344563

[41] S. A. Cook, "An observation on time-storage trade off," in *Proceedings of the Fifth Annual ACM Symposium on Theory of Computing*, ser. STOC '73. New York, NY, USA: ACM, 1973, pp. 29–33.

[42] M. Tompa, "Time-space tradeoffs for computing functions, using connectivity properties of their circuits," in *Proceedings of the Tenth Annual ACM Symposium on Theory of Computing*, ser. STOC '78. New York, NY, USA: ACM, 1978, pp. 196–204.

[43] J. E. Savage and S. Swamy, "Space-time trade-offs on the fft algorithm," *IEEE Transactions on Information Theory*, vol. 24, no. 5, pp. 563–568, 1978.

[44] S. Swamy and J. E. Savage, "Space-time tradeoffs for linear recursion," in *POPL*, 1979, pp. 135–142.

[45] A. K. Chandra, "Efficient compilation of linear recursive programs," in *SWAT (FOCS)*, 1973, pp. 16–25.

[46] J. E. Savage and S. Swamy, "Space-time tradeoffs for oblivious interger multiplications," in *ICALP*, 1979, pp. 498–504.

[47] S. Dziembowski, S. Faust, V. Kolmogorov, and K. Pietrzak, "Proofs of space," in *CRYPTO 2015, Part II*, ser. LNCS, R. Gennaro and M. J. B. Robshaw, Eds., vol. 9216. Springer, Heidelberg, Aug. 2015, pp. 585–605.

[48] L. Ren and S. Devadas, "Proof of space from stacked expanders," in *TCC 2016-B, Part I*, ser. LNCS, M. Hirt and A. D. Smith, Eds., vol. 9985. Springer, Heidelberg, Oct. / Nov. 2016, pp. 262–285.

[49] C. Dwork, M. Naor, and H. Wee, "Pebbling and proofs of work," in *CRYPTO 2005*, ser. LNCS, V. Shoup, Ed., vol. 3621. Springer, Heidelberg, Aug. 2005, pp. 37–54.

[50] M. Mahmoody, T. Moran, and S. P. Vadhan, "Publicly verifiable proofs of sequential work," in *ITCS 2013*, R. D. Kleinberg, Ed. ACM, Jan. 2013, pp. 373–388.

[51] S. Dziembowski, T. Kazana, and D. Wichs, "Key-evolution schemes resilient to space-bounded leakage," in *CRYPTO 2011*, ser. LNCS, P. Rogaway, Ed., vol. 6841. Springer, Heidelberg, Aug. 2011, pp. 335–353.

[52] B. Hemenway, Z. Jafargholi, R. Ostrovsky, A. Scafuro, and D. Wichs, "Adaptively secure garbled circuits from one-way functions," in *CRYPTO 2016, Part III*, ser. LNCS, M. Robshaw and J. Katz, Eds., vol. 9816. Springer, Heidelberg, Aug. 2016, pp. 149–178.

[53] S. Dziembowski, T. Kazana, and D. Wichs, "One-time computable self-erasing functions," in *TCC 2011*, ser. LNCS, Y. Ishai, Ed., vol. 6597. Springer, Heidelberg, Mar. 2011, pp. 125–143.

[54] Z. Jafargholi and D. Wichs, "Adaptive security of Yao's garbled circuits," in *TCC 2016-B, Part I*, ser. LNCS, M. Hirt and A. D. Smith, Eds., vol. 9985. Springer, Heidelberg, Oct. / Nov. 2016, pp. 433–458.

[55] C. Forler, S. Lucks, and J. Wenzel, "Catena: A memory-consuming password scrambler." *IACR Cryptology ePrint Archive*, vol. 2013, p. 525, 2013.

[56] J. Alwen and J. Blocki, "Efficiently computing data-independent memory-hard functions," in *CRYPTO 2016, Part II*, ser. LNCS, M. Robshaw and J. Katz, Eds., vol. 9815. Springer, Heidelberg, Aug. 2016, pp. 241–271.

[57] J. Alwen, P. Gaži, C. Kamath, K. Klein, G. Osang, K. Pietrzak, L. Reyzin, M. Rolínek, and M. Rybár, "On the memory-hardness of data-independent password-hashing functions," Cryptology ePrint Archive, Report 2016/783, 2016, https://eprint.iacr.org/2016/783.

[58] J. Blocki and S. Zhou, "On the computational complexity of minimal cumulative cost graph pebbling," in *FC 2018*, ser. LNCS, S. Meiklejohn and K. Sako, Eds., vol. 10957. Springer, Heidelberg, Feb. / Mar. 2018, pp. 329–346.

[59] J. Alwen and B. Tackmann, "Moderately hard functions: Definition, instantiations, and applications," in *TCC 2017, Part I*, ser. LNCS, Y. Kalai and L. Reyzin, Eds., vol. 10677. Springer, Heidelberg, Nov. 2017, pp. 493–526.

[60] R. de Wolf, "Quantum computing: Lecture notes," Jan 2018. [Online]. Available: https://homepages.cwi.nl/~rdewolf/qcnotes.pdf

[61] P. W. Shor, "Algorithms for quantum computation: Discrete logarithms and factoring," in *35th FOCS*. IEEE Computer Society Press, Nov. 1994, pp. 124–134.

[62] L. K. Grover, "A fast quantum mechanical algorithm for database search," in *28th ACM STOC*. ACM Press, May 1996, pp. 212–219.

[63] M. Boyer, G. Brassard, P. Hωyer, and A. Tapp, "Tight bounds on quantum searching," *Fortschritte der Physik: Progress of Physics*, vol. 46, no. 4-5, pp. 493–505, 1998.

[64] C. Zalka, "Grover's quantum searching algorithm is optimal," *Physical Review A*, vol. 60, no. 4, p. 2746, 1999.

[65] L. Chen, D. Moody, and Y.-k. Liu, "Submission requirements and evaluation criteria for the post-quantum cryptography standardization process," 2017.

[66] L. Chen, S. Jordan, Y.-K. Liu, D. Moody, R. Peralta, R. Perlner, and D. Smith-Tone, *Report on post-quantum cryptography*. US Department of Commerce, National Institute of Standards and Technology, 2016.

[67] G. Alagic, G. Alagic, J. Alperin-Sheriff, D. Apon, D. Cooper, Q. Dang, Y.-K. Liu, C. Miller, D. Moody, R. Peralta *et al.*, *Status report on the first round of the NIST post-quantum cryptography standardization process.* US Department of Commerce, National Institute of Standards and Technology . . . , 2019.

[68] R. Avanzi, J. Bos, L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, J. M. Schanck, P. Schwabe, G. Seiler, and D. Stehlé, "Crystals-kyber," *NIST, Tech. Rep*, 2017.

[69] O. Regev, "On lattices, learning with errors, random linear codes, and cryptography," *Journal of the ACM (JACM)*, vol. 56, no. 6, pp. 1–40, 2009.

[70] M. Baldi, A. Barenghi, F. Chiaraluce, G. Pelosi, and P. Santini, "Ledacrypt: Qc-ldpc code-based cryptosystems with bounded decryption failure rate," in *Code-Based Cryptography Workshop.* Springer, 2019, pp. 11–43.

[71] N. Provos and D. Mazieres, "Bcrypt algorithm." USENIX, 1999.

[72] B. Kaliski, "Pkcs# 5: Password-based cryptography specification version 2.0," 2000.

[73] L. Breech, "Lastpass security notice," https://blog.lastpass.com/2015/06/lastpass-security-notice.html/ (retrieved 11/10/2016), 2015.

[74] D. Meyer, "How to check if you were caught up in the dropbox breach," http://fortune.com/2016/08/31/dropbox-breach-passwords/ (retrieved 11/10/2016).

[75] J. Bonneau, C. Herley, P. C. Van Oorschot, and F. Stajano, "The quest to replace passwords: A framework for comparative evaluation of web authentication schemes," in *Security and Privacy (SP), 2012 IEEE Symposium on.* IEEE, 2012, pp. 553–567.

[76] C. Herley and P. C. van Oorschot, "A research agenda acknowledging the persistence of passwords," *IEEE Security & Privacy*, vol. 10, no. 1, pp. 28–36, 2012.

[77] J. Bonneau, C. Herley, P. C. van Oorschot, and F. Stajano, "Passwords and the evolution of imperfect authentication," *Communications of the ACM*, vol. 58, no. 7, pp. 78–87, 2015.

[78] D. Wang and P. Wang, "On the implications of zipf's law in passwords," in *Computer Security - ESORICS 2016 - 21st European Symposium on Research in Computer Security*, 2016, pp. 111–131.

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

[80] J. Blocki, A. Datta, and J. Bonneau, "Differentially private password frequency lists," in *NDSS 2016.* The Internet Society, Feb. 2016.

[81] W. Yang, N. Li, I. M. Molloy, Y. Park, and S. N. Chari, "Comparing password ranking algorithms on real-world password datasets," in *ESORICS*, 2016, pp. 69–90.

[82] W. Melicher, B. Ur, S. M. Segreti, S. Komanduri, L. Bauer, N. Christin, and L. F. Cranor, "Fast, lean, and accurate: Modeling password guessability using neural networks," in *USENIX Security 2016*, T. Holz and S. Savage, Eds. USENIX Association, Aug. 2016, pp. 175–191.

[83] 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 underground economy," November 2008, retrieved 1/8/2013.

[84] R. B. Miller, "Response time in man-computer conversational transactions," in *Proceedings of the December 9-11, 1968, fall joint computer conference, part I.* ACM, 1968, pp. 267–277.

[85] X. H. Ding Wang, Gaopeng Jian and P. Wang, "Zipf's law in passwords," Cryptology ePrint Archive, Report 2014/631, 2014, http://eprint.iacr.org/2014/631.

[86] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, "A density-based algorithm for discovering clusters in large spatial databases with noise," in *KDD*, 1996, pp. 226–231.

[87] K. Conger, "Yahoo discloses hack of 1 billion accounts," 2016, retrieved 1/13/2017.

[88] T. Hunt, "Have i been pwned?" Jul 2021. [Online]. Available: haveibeenpwned.com

[89] M. Snider and E. Weise, "500 million yahoo accounts breached," http://www.usatoday.com/story/tech/2016/09/22/report-yahoo-may-confirm-massive-data-breach/90824934/, USA Today, 2016, retrieved 5, Dec. 2016.

[90] CynoSurePrime, "How we cracked millions of ashley madison bcrypt hashes efficiently," http://cynosureprime.blogspot.com/2015/09/how-we-cracked-millions-of-ashley.html (retrieved 11/10/2016), 2015.

[91] M. Stockley, "What your hacked account is worth on the dark web," Aug 2016. [Online]. Available: https://nakedsecurity.sophos.com/2016/08/09/what-your-hacked-account-is-worth-on-the-dark-web/

[92] J. Bonneau and S. E. Schechter, "Towards reliable storage of 56-bit secrets in human memory," in *USENIX Security 2014*, K. Fu and J. Jung, Eds. USENIX Association, Aug. 2014, pp. 607–623.

[93] U. Manber, "A simple scheme to make passwords based on one-way functions much harder to crack," *Computers & Security*, vol. 15, no. 2, pp. 171–176, 1996.

[94] J. G. Brainard, A. Juels, B. Kaliski, and M. Szydlo, "A new two-server approach for authentication with short secrets." in *USENIX Security*, vol. 3, 2003, pp. 201–214.

[95] J. Camenisch, A. Lysyanskaya, and G. Neven, "Practical yet universally composable two-server password-authenticated secret sharing," in *Proceedings of the 2012 ACM conference on Computer and Communications Security.* ACM, 2012, pp. 525–536.

[96] A. Everspaugh, R. Chaterjee, S. Scott, A. Juels, and T. Ristenpart, “The pythia prf service,” in *24th USENIX Security Symposium (USENIX Security 15)*, 2015, pp. 547–562.

[97] R. W. F. Lai, C. Egger, D. Schröder, and S. S. M. Chow, “Phoenix: Rebirth of a cryptographic password-hardening service,” in *26th USENIX Security Symposium (USENIX Security 17)*. Vancouver, BC: USENIX Association, 2017, pp. 899–916. [Online]. Available: https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/lai

[98] A. Juels and R. L. Rivest, “Honeywords: Making password-cracking detectable,” in *Proceedings of the 2012 ACM conference on Computer and communications security*. ACM, 2013.

[99] D. Freeman, S. Jain, M. Dürmuth, B. Biggio, and G. Giacinto, “Who are you? A statistical approach to measuring user authenticity,” in *NDSS 2016*. The Internet Society, Feb. 2016.

[100] J. Bonneau and S. Preibusch, “The password thicket: technical and market failures in human authentication on the web,” in *Proceedings of the Ninth Workshop on the Economics of Information Security (WEIS)*, Jun. 2010. [Online]. Available: http://weis2010.econinfosec.org/papers/session3/weis2010_bonneau.pdf

[101] A. Naiakshina, A. Danilova, C. Tiefenau, M. Herzog, S. Dechand, and M. Smith, “"why do developers get password storage wrong",” in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS ’17, 2017, p. (to appear).

[102] D. Wang, Z. Zhang, P. Wang, J. Yan, and X. Huang, “Targeted online password guessing: An underestimated threat,” in *ACM CCS 2016*, E. R. Weippl, S. Katzenbeisser, C. Kruegel, A. C. Myers, and S. Halevi, Eds. ACM Press, Oct. 2016, pp. 1242–1254.

[103] D. Florêncio, C. Herley, and P. C. van Oorschot, “Password portfolios and the finite-effort user: Sustainably managing large numbers of accounts,” in *USENIX Security 2014*, K. Fu and J. Jung, Eds. USENIX Association, Aug. 2014, pp. 575–590.

[104] G. Vranken, “HTTPS Bicycle Attack,” Tech. Rep. [Online]. Available: https://guidovranken.files.wordpress.com/2015/12/https-bicycle-attack.pdf

[105] T. Dierks and E. Rescorla, “The transport layer security (tls) protocol version 1.2,” Internet Requests for Comments, RFC Editor, RFC 5246, August 2008. [Online]. Available: http://www.rfc-editor.org/rfc/rfc5246.txt

[106] P. Kotzias, A. Razaghpanah, J. Amann, K. G. Paterson, N. Vallina-Rodriguez, and J. Caballero, “Coming of age: A longitudinal study of tls deployment,” in *Proceedings of the Internet Measurement Conference 2018*, ser. IMC ’18. New York, NY, USA: ACM, 2018, pp. 415–428. [Online]. Available: http://doi.acm.org/10.1145/3278532.3278568

[107] E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.3," RFC 8446, Aug. 2018. [Online]. Available: https://rfc-editor.org/rfc/rfc8446.txt

[108] Internet Crime Complaint Center, "2015 Internet Crime Report," Tech. Rep., 2015. [Online]. Available: https://pdf.ic3.gov/2015_ic3report.pdf

[109] "Brute force attacks conducted by cyber actors, alert (ta18-086a)," 2019. [Online]. Available: https://www.us-cert.gov/ncas/alerts/TA18-086A

[110] J. Blocki and A. Sridhar, "Client-CASH: Protecting master passwords against offline attacks," in *ASIACCS 16*, X. Chen, X. Wang, and X. Huang, Eds. ACM Press, May / Jun. 2016, pp. 165–176.

[111] S. Brostoff and M. Sasse, ""Ten strikes and you're out": Increasing the number of login attempts can improve password usability," apr 2003. [Online]. Available: http://discovery.ucl.ac.uk/19826/

[112] J. Blocki, M. Blum, and A. Datta, "Naturally rehearsing passwords," in *ASIACRYPT 2013, Part II*, ser. LNCS, K. Sako and P. Sarkar, Eds., vol. 8270. Springer, Heidelberg, Dec. 2013, pp. 361–380.

[113] W. Yang, N. Li, O. Chowdhury, A. Xiong, and R. W. Proctor, "An empirical study of mnemonic sentence-based password generation strategies," in *ACM CCS 2016*, E. R. Weippl, S. Katzenbeisser, C. Kruegel, A. C. Myers, and S. Halevi, Eds. ACM Press, Oct. 2016, pp. 1216–1229.

[114] J. Blocki, S. Komanduri, L. F. Cranor, and A. Datta, "Spaced repetition and mnemonics enable recall of multiple strong passwords," in *NDSS 2015*. The Internet Society, Feb. 2015.

[115] B. Ross, C. Jackson, N. Miyake, D. Boneh, and J. C. Mitchell, "Stronger password authentication using browser extensions," in *USENIX Security 2005*, P. D. McDaniel, Ed. USENIX Association, Jul. / Aug. 2005.

[116] M. J. Wiener, "The full cost of cryptanalytic attacks," *Journal of Cryptology*, vol. 17, no. 2, pp. 105–124, Mar. 2004.

[117] M. Grassl, B. Langenberg, M. Roetteler, and R. Steinwandt, "Applying grover's algorithm to AES: Quantum resource estimates," in *Post-Quantum Cryptography - 7th International Workshop, PQCrypto 2016*, T. Takagi, Ed. Springer, Heidelberg, 2016, pp. 29–43.

[118] B. Langenberg, H. Pham, and R. Steinwandt, "Reducing the cost of implementing the advanced encryption standard as a quantum circuit," *IEEE Transactions on Quantum Engineering*, vol. 1, pp. 1–12, 2020.

[119] X. Bonnetain, M. Naya-Plasencia, and A. Schrottenloher, "Quantum security analysis of aes," *IACR Transactions on Symmetric Cryptology*, vol. 2019, no. 2, pp. 55–93, 2019.

[120] M. Almazrooie, A. Samsudin, R. Abdullah, and K. N. Mutter, "Quantum reversible circuit of aes-128," *Quantum Information Processing*, vol. 17, no. 5, pp. 1–30, 2018.

[121] S. Fluhrer, "Further analysis of a proposed hash-based signature standard," Cryptology ePrint Archive, Report 2017/553, 2017. [Online]. Available: http://eprint.iacr.org/2017/553

[122] R. Bowes, "phpbb breach dataset." [Online]. Available: https://wiki.skullsecurity.org/Passwords

[123] R. Hranický, L. Zobal, O. Rysavý, D. Kolár, and D. Mikus, "Distributed PCFG password cracking," in *ESORICS 2020, Part I*, ser. LNCS, L. Chen, N. Li, K. Liang, and S. A. Schneider, Eds., vol. 12308. Springer, Heidelberg, Sep. 2020, pp. 701–719.

[124] 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.* USENIX Association, 2010, pp. 1–8.

[125] J. Blocki and W. Zhang, "Dalock: Distribution aware password throttling," *arXiv preprint arXiv:2005.09039*, 2020.

[126] Y. Tian, C. Herley, and S. Schechter, "Stopguessing: Using guessed passwords to thwart online guessing," in *Proc. IEEE European Symp. Security and Privacy (EuroS&P 2019)*, 2019, pp. 17–19.

[127] C. Dwork, F. McSherry, K. Nissim, and A. Smith, "Calibrating noise to sensitivity in private data analysis," in *TCC 2006*, ser. LNCS, S. Halevi and T. Rabin, Eds., vol. 3876. Springer, Heidelberg, Mar. 2006, pp. 265–284.

[128] G. Cormode, C. Procopiuc, D. Srivastava, and T. T. Tran, "Differentially private summaries for sparse data," in *Proceedings of the 15th International Conference on Database Theory*, 2012, pp. 299–311.

[129] L. Ren and S. Devadas, "Bandwidth hard functions for ASIC resistance," in *TCC 2017, Part I*, ser. LNCS, Y. Kalai and L. Reyzin, Eds., vol. 10677. Springer, Heidelberg, Nov. 2017, pp. 466–492.

[130] I. J. Good, "The population frequencies of species and the estimation of population parameters," *Biometrika*, vol. 40, no. 3-4, pp. 237–264, 1953.

[131] M. Weir, S. Aggarwal, B. de Medeiros, and B. Glodek, "Password cracking using probabilistic context-free grammars," in *2009 IEEE Symposium on Security and Privacy.* IEEE Computer Society Press, May 2009, pp. 391–405.

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

[133] R. Veras, C. Collins, and J. Thorpe, “On semantic patterns of passwords and their security impact,” in *NDSS 2014*. The Internet Society, Feb. 2014.

[134] C. Castelluccia, M. Dürmuth, and D. Perito, “Adaptive password-strength meters from Markov models,” in *NDSS 2012*. The Internet Society, Feb. 2012.

[135] C. Castelluccia, A. Chaabane, M. Dürmuth, and D. Perito, “When privacy meets security: Leveraging personal information for password cracking,” *arXiv preprint arXiv:1304.6584*, 2013.

[136] J. Ma, W. Yang, M. Luo, and N. Li, “A study of probabilistic password models,” in *2014 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, May 2014, pp. 689–704.

[137] B. Ur, S. M. Segreti, L. Bauer, N. Christin, L. F. Cranor, S. Komanduri, D. Kurilova, M. L. Mazurek, W. Melicher, and R. Shay, “Measuring real-world accuracies and biases in modeling password guessability,” in *USENIX Security 2015*, J. Jung and T. Holz, Eds. USENIX Association, Aug. 2015, pp. 463–481.

[138] N. Hansen, “The cma evolution strategy: A comparing review,” 2006.

[139] C. Audet, S. Le Digabel, and C. Tribes, “NOMAD user guide,” Les cahiers du GERAD, Tech. Rep. G-2009-37, 2009. [Online]. Available: https://www.gerad.ca/nomad/Downloads/userguide.pdf

[140] M. Abramson, C. Audet, G. Couture, J. Dennis, Jr., S. Le Digabel, and C. Tribes, “The NOMAD project,” Software available at https://www.gerad.ca/nomad/. [Online]. Available: https://www.gerad.ca/nomad/

[141] M. Eldred, B. Bichon, B. Adams, and S. Mahadevan, *Overview of reliability analysis and design capabilities in DAKOTA with application to shape optimization of MEMS*, 01 2008, pp. 401–432.

[142] Biteopt. [Online]. Available: https://github.com/avaneev/biteopt

[143] Black box optimization competition. [Online]. Available: https://www.ini.rub.de/PEOPLE/glasmtbl/projects/bbcomp/index.html

[144] B. Pinkas and T. Sander, “Securing passwords against dictionary attacks,” in *ACM CCS 2002*, V. Atluri, Ed. ACM Press, Nov. 2002, pp. 161–170.

[145] “Hackers find new way to bilk eBay users - CNET,” 2019. [Online]. Available: https://www.cnet.com/news/hackers-find-new-way-to-bilk-ebay-users/

[146] M. Motoyama, K. Levchenko, C. Kanich, D. McCoy, G. M. Voelker, and S. Savage, “Re: CAPTCHAs-understanding CAPTCHA-solving services in an economic context,” in *USENIX Security 2010*. USENIX Association, Aug. 2010, pp. 435–462.

[147] C. Percival and S. Josefsson, "The scrypt password-based key derivation function," *IETF Draft URL: http://tools. ietf. org/html/josefsson-scrypt-kdf-00. txt (accessed: 30.11. 2012)*, 2016.

[148] N. Cubrilovic, "Rockyou hack: From bad to worse," Dec 2009. [Online]. Available: https://techcrunch.com/2009/12/14/rockyou-hack-security-myspace-facebook-passwords/

[149] R. Munroe, "Password strength," 2011. [Online]. Available: https://xkcd.com/936/

[150] R. Morris and K. Thompson, "Password security: A case history," *Communications of the ACM*, vol. 22, no. 11, pp. 594–597, 1979. [Online]. Available: http://dl.acm.org/citation.cfm?id=359172

[151] A. Narayanan and V. Shmatikov, "Fast dictionary attacks on passwords using time-space tradeoff," in *Proceedings of ACM CCS*, 2005, Conference Proceedings, pp. 364–372. [Online]. Available: http://dl.acm.org/citation.cfm?id=1102168

[152] J. Yan, A. Blackwell, R. Anderson, and A. Grant, "The memorability and security of passwords: some empirical results," *Technical Report-University Of Cambridge Computer Laboratory*, p. 1, 2000.

[153] Openwall, "John the ripper password cracker," 2006. [Online]. Available: https://www.openwall.com/john/

[154] J. Campbell, W. Ma, and D. Kleeman, "Impact of restrictive composition policy on user password choices," *Behaviour & Information Technology*, vol. 30, no. 3, pp. 379–388, 2011.

[155] 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 *CHI*, 2011, Conference Proceedings, pp. 2595–2604. [Online]. Available: http://dl.acm.org/citation.cfm?id=1979321

[156] R. Shay, S. Komanduri, P. G. Kelley, P. G. Leon, M. L. Mazurek, L. Bauer, N. Christin, and L. F. Cranor, "Encountering stronger password requirements: user attitudes and behaviors," in *Proceedings of the Sixth Symposium on Usable Privacy and Security*, ser. SOUPS '10. New York, NY, USA: ACM, 2010, pp. 2:1–2:20. [Online]. Available: http://doi.acm.org/10.1145/1837110.1837113

[157] J. M. Stanton, K. R. Stam, P. Mastrangelo, and J. Jolton, "Analysis of end user security behaviors," *Comput. Secur.*, vol. 24, no. 2, pp. 124–133, Mar. 2005.

[158] P. G. Inglesant and M. A. Sasse, "The true cost of unusable password policies: Password use in the wild," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ser. CHI '10. New York, NY, USA: ACM, 2010, pp. 383–392. [Online]. Available: http://doi.acm.org/10.1145/1753326.1753384

[159] R. Shay, S. Komanduri, A. L. Durity, P. S. Huh, M. L. Mazurek, S. M. Segreti, B. Ur, L. Bauer, N. Christin, and L. F. Cranor, "Can long passwords be secure and usable?" in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ser. CHI '14. New York, NY, USA: ACM, 2014, pp. 2927–2936. [Online]. Available: http://doi.acm.org/10.1145/2556288.2557377

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

[161] S. Komanduri, R. Shay, L. F. Cranor, C. Herley, and S. Schechter, "Telepathwords: Preventing weak passwords by reading users' minds," in *23rd USENIX Security Symposium (USENIX Security 14)*. San Diego, CA: USENIX Association, Aug. 2014, pp. 591–606. [Online]. Available: https://www.usenix.org/conference/usenixsecurity14/technical-sessions/presentation/komanduri

[162] B. Ur, P. G. Kelley, S. Komanduri, J. Lee, M. Maass, M. Mazurek, T. Passaro, R. Shay, T. Vidas, L. Bauer, N. Christin, and L. F. Cranor, "How does your password measure up? the effect of strength meters on password creation," in *Proceedings of USENIX Security Symposium*, 2012, Conference Proceedings.

[163] X. de Carné de Carnavalet and M. Mannan, "From very weak to very strong: Analyzing password-strength meters," in *Network and Distributed System Security Symposium (NDSS 2014)*. Internet Society, 2014.

[164] R. Shay, S. Komanduri, A. L. Durity, P. S. Huh, M. L. Mazurek, S. M. Segreti, B. Ur, L. Bauer, N. Christin, and L. F. Cranor, "Designing password policies for strength and usability," *ACM Trans. Inf. Syst. Secur.*, vol. 18, no. 4, p. 13, 2016.

[165] M. Steves, D. Chisnell, A. Sasse, K. Krol, M. Theofanos, and H. Wald, "Report: Authentication diary study," National Institute of Standards and Technology (NIST), Tech. Rep. NISTIR 7983, 2014.

[166] D. Florêncio, C. Herley, and P. C. Van Oorschot, "An administrator's guide to Internet password research," in *Proceedings of the 28th USENIX Conference on Large Installation System Administration*, ser. LISA'14, 2014, pp. 35–52.

[167] J. Blocki, S. Komanduri, A. Procaccia, and O. Sheffet, "Optimizing password composition policies," in *Proceedings of the fourteenth ACM conference on Electronic commerce*. ACM, 2013, pp. 105–122.

[168] D. J. Bernstein, "Cache-Timing Attacks on AES," http://cr.yp.to/antiforgery/cachetiming-20050414.pdf.

[169] R. Canetti, S. Halevi, and M. Steiner, *Mitigating Dictionary Attacks on Password-Protected Local Storage*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 160–179. [Online]. Available: http://dx.doi.org/10.1007/11818175_10

[170] J. Blocki, M. Blum, and A. Datta, "Gotcha password hackers!" in *Proceedings of the 2013 ACM workshop on Artificial intelligence and security.* ACM, 2013, pp. 25–34.

[171] J. Blocki and H.-S. Zhou, *Designing Proof of Human-Work Puzzles for Cryptocurrency and Beyond.* Berlin, Heidelberg: Springer Berlin Heidelberg, 2016, pp. 517–546. [Online]. Available: http://dx.doi.org/10.1007/978-3-662-53644-5_20

[172] D. Boneh, H. Corrigan-Gibbs, and S. E. Schechter, "Balloon hashing: A memory-hard function providing provable protection against sequential attacks," in *ASIACRYPT 2016, Part I*, ser. LNCS, J. H. Cheon and T. Takagi, Eds., vol. 10031. Springer, Heidelberg, Dec. 2016, pp. 220–248.

[173] A. Biryukov and D. Khovratovich, "Tradeoff cryptanalysis of memory-hard functions," in *International Conference on the Theory and Application of Cryptology and Information Security.* Springer, 2014, pp. 633–657.

[174] J. Alwen and J. Blocki, "Towards Practical Attacks on Argon2i and Balloon Hashing," in *Proceedings of the 2nd IEEE European Symposium on Security and Privacy (EuroS&P 2017).* IEEE, 2017, pp. 142–157, http://eprint.iacr.org/2016/759.

[175] J. Alwen, J. Blocki, and K. Pietrzak, "Depth-robust graphs and their cumulative memory complexity," in *Advances in Cryptology-EUROCRYPT 2017.* Springer, 2017, p. (to appear), http://eprint.iacr.org/2016/875.

[176] P. Erdos, R. L. Graham, and E. Szemeredi, "On sparse graphs with dense long paths." Stanford, CA, USA, Tech. Rep., 1975.

[177] R. Chatterjee, A. Athayle, D. Akhawe, A. Juels, and T. Ristenpart, "pASSWORD tYPOS and how to correct them securely," in *2016 IEEE Symposium on Security and Privacy.* IEEE Computer Society Press, May 2016, pp. 799–818.

[178] K. Killourhy and R. Maxion, "Comparing anomaly-detection algorithms for keystroke dynamics," *DNS*, 2009.

[179] S. Fahl, M. Harbach, Y. Acar, and M. Smith, "On the ecological validity of a password study," in *Proceedings of the Ninth Symposium on Usable Privacy and Security.* ACM, 2013, p. 13.

[180] R. C. Team, *R: A Language and Environment for Statistical Computing*, R Foundation for Statistical Computing, Vienna, Austria, 2016. [Online]. Available: https://www.R-project.org

[181] J. Blocki and S. Zhou, "On the depth-robustness and cumulative pebbling cost of Argon2i," in *TCC 2017, Part I*, ser. LNCS, Y. Kalai and L. Reyzin, Eds., vol. 10677. Springer, Heidelberg, Nov. 2017, pp. 445–465.

[182] L. G. Valiant, "Graph-theoretic arguments in low-level complexity," in *International Symposium on Mathematical Foundations of Computer Science.* Springer, 1977, pp. 162–176.

[183] T. Lengauer and R. E. Tarjan, "Asymptotically tight bounds on time-space trade-offs in a pebble game," *J. ACM*, vol. 29, no. 4, pp. 1087–1130, Oct. 1982. [Online]. Available: http://doi.acm.org/10.1145/322344.322354

[184] M. A. Simplício Jr., L. C. Almeida, E. R. Andrade, P. Santos, and P. S. L. M. Barreto, "Lyra2: Password hashing scheme with improved security against time-memory trade-offs," Cryptology ePrint Archive, Report 2015/136, 2015, https://eprint.iacr.org/2015/136.

[185] Bitmain, "Antminer s9 specifications." [Online]. Available: https://shop.bitmain.com/promote/antminer_s9i_asic_bitcoin_miner/specification

[186] G. Kedem and Y. Ishihara, "Brute force attack on UNIX passwords with SIMD computer," in *USENIX Security 99*, G. W. Treese, Ed. USENIX Association, Aug. 1999.

[187] N. Valev, "Electricity prices around the world," Sep 2020. [Online]. Available: https://www.globalpetrolprices.com/electricity_prices/

[188] Newegg, Inc, "G.skill aegis 8gb ddr4 sdram listing," Jun 2021. [Online]. Available: https://www.newegg.com/g-skill-8gb-288-pin-ddr4-sdram/p/N82E16820232257?item=N82E16820232257

[189] Amazon, Inc, "Crucial ram 8gb ddr4 listing," Jun 2021. [Online]. Available: https://www.amazon.com/Crucial-DDR4-Desktop-Memory-CT8G4DFRA266/dp/B08C4QF5N4/

[190] Intel, Inc, "Intel power gadget." [Online]. Available: https://software.intel.com/content/www/us/en/develop/articles/intel-power-gadget.html

[191] Bitmain, "Antminer s19 overview." [Online]. Available: https://shop.bitmain.com/release/AntminerS19Pro/overview

[192] Amazon, Inc, "Antminer s19 listing," Jun 2021. [Online]. Available: https://www.amazon.com/gp/product/B08K7CVKDZ