The TypTop System: Personalized Typo-Tolerant Password Checking*

Rahul Chatterjee[1,2], Joanne Woodage[3], Yuval Pnueli[4], Anusha Chowdhury[1], Thomas Ristenpart[2]

[1] Cornell University [2] Cornell Tech [3] Royal Holloway, University of London [4] Technion – Israel Institute of Technology

ABSTRACT

Password checking systems traditionally allow login only if the correct password is submitted. Recent work on typo-tolerant password checking suggests that usability can be improved, with negligible security loss, by allowing a small number of typographical errors. Existing systems, however, can only correct a handful of errors, such as accidentally leaving caps lock on or incorrect capitalization of the first letter in a password. This leaves out numerous kinds of typos made by users, such as transposition errors, substitutions, or capitalization errors elsewhere in a password. Some users therefore receive no benefit from existing typo-tolerance mechanisms.

We introduce personalized typo-tolerant password checking. In our approach, the authentication system learns over time the typos made by a specific user. In experiments using Mechanical Turk, we show that 45% of users would benefit from personalization. We therefore design a system, called TypTop, that securely implements personalized typo-tolerance. Underlying TypTop is a new stateful password-based encryption scheme that can be used to store recent failed login attempts. Our formal analysis shows that security in the face of an attacker that obtains the state of the system reduces to the difficulty of a brute-force dictionary attack against the real password. We implement TypTop for Linux and Mac OS login and report on a proof-of-concept deployment.

1 INTRODUCTION

Passwords remain the predominant means of authenticating users on both computers and the web — however studies show that users persistently pick weak passwords [6, 12, 34]. This phenomena is often ascribed to users selecting easy-to-remember passwords; however a number of studies [17, 18, 32] highlight that strong passwords are also more difficult to type. To increase password usability, some companies [2, 25, 28] allow authentication under a small set of common typos. For example, Facebook permits capitalization errors in the first letter or accidental caps lock errors.

Motivated by this, Chatterjee et al. [8] recently initiated the academic investigation of typo-tolerant password checking. In a 24-hour study at Dropbox, they found that a small set of easy-to-correct typos accounted for over 9.3% of failed login attempts, and 3% of the total users turned away — underscoring the burden that typos represent to both users and the companies that ultimately lose out on user engagement due to them. To increase usability, the authors advocate an approach they call 'relaxed checking'. They establish a handful of the most frequently occurring typos across a user population, and build corrector functions to rectify those particular typos on behalf of the user at the time of authentication (e.g., flipping the case of all letters to correct a caps lock error). The authors show empirically that for a carefully selected set of correctors, the resulting security degradation is minimal.

A limitation of this approach is that checking each correction requires applying a computationally intensive password hashing algorithm; as such the number of typos one may correct is inherently limited by performance constraints. While correcting the five most prevalent typos accounts for 20% of typos made by users [8], this still leaves the majority of password typos uncorrectable. Individual users may be totally neglected, should they frequently make a typo that is rare across the broader population of users. Users who choose complex, strong passwords are likely to fall into this neglected group.

We introduce a new approach to password checking: personalized typo-tolerance. In such a system, the password checking mechanism learns the typos commonly made by each user over time, storing them in a secure manner. After learning frequent typos, the system can check to see if a submitted password is either the one originally registered, or one of the learned variants. By tailoring typo-tolerance to the individual user, we aim to correct a larger set of typos than previously possible, while maintaining strong security guarantees.

Building a personalized typo-tolerant checking system requires care. The system should not begin accepting arbitrary incorrect passwords that are submitted — indeed this would enable potentially malicious users to register arbitrary passwords which allow access to an account. Therefore we need a policy dictating the types of errors that can be added to a cache of allowed typos, and a mechanism to enforce it. We must consider security in the face of remote guessing attacks as well as compromise of password databases, both being threats that frequently arise in practice. This rules out simple schemes in which recent incorrect submissions are simply stored in the clear. Ideally, a scheme would be as secure as a conventional password-based checking system, meaning an attacker must perform as much work to compromise an account as they would have had there been no typo-tolerance.

We overcome these challenges and design a secure personalized typo-tolerant password checking system that we call TypTop. At a high level, the system works as follows. It maintains a set of allowed hashes, corresponding to the registered password and allowed typos of it. The user can successfully login by submitting either the password or an allowed typo. Initially, this set contains only the salted hashes of the registered password and some typos of that password that are considered likely across the population of all users. To personalize, TypTop adapts this set of typos over time by securely storing incorrect submissions encrypted under a public-key for which the associated secret key is, itself, encrypted under the registered password and previously allowed typos. Upon

*This is the full version of the paper published in CCS '17

CCS '17, October 30-November 3, 2017, Dallas, TX, USA
2017. ACM ISBN 978-1-4503-4946-8/17/10…$15.00
https://doi.org/10.1145/3133956.3134000

a subsequent successful login, the recent incorrect submissions can be decrypted and checked to see if they satisfy the policy regarding typos. If so, new salted hashes of the incorrect submissions, now considered as legitimate typos, can be added to the set of allowed hashes. To ensure the set does not grow too large, less frequently observed typos can be evicted. Future login attempts that make one of the frequent typos will be allowed, thereby avoiding the need for the user to retype their password.

Underlying TypTop is therefore a new kind of stateful password-based encryption scheme that ensures the plaintext state for a user can only be unlocked with knowledge of the registered password or one of the policy-checked cached variants of it. We introduce a formal security notion requiring that an attacker learns nothing about a sequence of password logins (including the number of logins, how many variants were entered into the typo cache, or partial information about the passwords) given the state of the password system, unless the attacker can successfully mount a modified form of brute-force guessing attack in which it repeatedly hashes common passwords or typos of them and checks them against the hashes stored within the state.

We prove the security of our construction relative to this notion, and go on to analyze the brute-force guessing game to which we reduce security. We give criteria on password and typo distributions which, if met, mean that the attacker will gain no additional benefit by attempting to guess a stored typo. For such settings, we prove that the optimal strategy is a standard brute-force guessing attack against the registered password as if there were no additional password hashes of typos stored. We show empirically that real-world password and typo distributions meet the required criteria.

To gauge the potential efficacy of TypTop, we conduct a study using Amazon Mechanical Turk (MTurk) [7] in which we ask users to perform repeated logins using a password of their choosing.[1] In this way, we can analyze the types of errors made and the potential benefit of personalization compared to the prior relaxed checking approach with a fixed set of typo correctors. The analysis reveals that 45% of users would benefit from personalization, a 1.5x improvement over the 29% of users that benefit from the top 5 correctors from [8].

We implemented a prototype of TypTop for Unix systems including Linux and Mac OS using the pluggable authentication module (PAM) framework. The prototype enables typo-tolerance for all password-based authentications managed by the operating system. We report on the initial deployment with 25 users, and while further studies will be needed to assess generalizability of our results, they so far indicate that TypTop significantly benefits users that often mistype their passwords in ways not covered by prior correction mechanisms. Our prototype is open-source and publicly available.[2]

2 BACKGROUND AND RELATED WORK

In traditional password-based authentication schemes (PBAS), a user initially registers a user name and a password with the system. The password is stored in some protected form (typically a salted password hash). On subsequent login attempts, the user re-enters their password which is then compared to the stored password; authentication is granted only if these match. A formal definition of PBAS schemes is given in Section 3.

Password distributions and guessing attacks. Measurement studies [6, 12, 24] and password leaks such as [33] show that users frequently pick weak passwords, with a large number of users sharing a relatively small set of passwords at the head of the password distribution. This leaves them vulnerable to guessing attacks (see Section 3).

The reason users persistently pick such weak passwords is often cited as ease of memorability. However studies by Keith et al. [17, 18] indicate that the rate at which typos are made approximately doubles for complex passwords; similarly Mazurek et al. [21] show via a large-scale study that login errors are correlated with stronger passwords, and suggest that users pick weaker passwords due to their ease of typing. Work by Shay et al. [31] finds a similar correlation between length and rate of typo occurrence for CorrectHorseBatteryStaple-type passphrases [26]. For a more detailed discussion of these related works, see [8].

Typo-tolerant password checking. Motivated by the industry practice [2, 25, 28] of allowing a small number of typos (specifically capitalization errors) during authentication, Chatterjee et al. [8] provide the first formal treatment of typo-tolerant password checking for user-selected passwords. With an experiment conducted on MTurk, they show that 20% of typos can be corrected by a small set of corrector functions (e.g., applying caps lock or switching the case of the first letter). They advocate an approach called *relaxed checking*, which allows authentication under a small number of easily correctable typos. Now when a password fails its initial authentication check, a set of (e.g., 5) corrector functions representing common typos are applied to the entered string; the user is allowed to authenticate if any of these corrections matches the registered password. The authors show that relaxed checking with a carefully chosen set of corrector functions can achieve a significant improvement in utility with minimal degradation in security.

While relaxed checking allows for the secure correction of 20% of typos, this still leaves the majority of typos uncorrected. Furthermore, the corrector functions utilized are based on common typo behavior across the population of users, as opposed to that of the individual. In this work, we explore a new approach — personalized typo-tolerance — which allows us to correct a greater proportion of typos by tailoring typo-tolerance to the individual user.

3 PERSONALIZED TYPO TOLERANCE

We introduce *personalized* typo-tolerant password checking which adapts, over time, to correct the specific typos made by an individual user. We begin by defining the abstract components of such a scheme, as well as the utility and the security goals.

Passwords and typos. We let $\mathcal{S}$ denote the set of strings which users may choose as passwords (e.g., the set of ASCII strings up to some maximum length, say ℓ). We let p be a distribution which models user selection of passwords, so $p(w)$ denotes the probability that w is the password chosen by a user, and let $\mathcal{W}$ denote the support of p, which represents the set of user passwords. We let $w_1, w_2, \ldots$ denote the passwords in $\mathcal{W}$ ordered in descending order of probability.

[1] All our study designs were reviewed by our university's IRB.
[2] https://typtop.info/

A typo-tolerant PBAS which allows authentication under *any* string is clearly insecure. As such we need a means to distinguish legitimate user typos from unrelated strings. We do this following the approach of [8]. We first convert both the password w and possible typo $\tilde{w}$ into their key press representation [8], and then compute the Damerau-Levenshtein distance [9, 20] of these representations, which we denote $\mathrm{DL}(w, \tilde{w})$. We view $\tilde{w}$ as a legitimate typo of w if this distance is at most some small fixed parameter δ; here we set $\delta = 2$. We let $\{\tau_w\}$ be a family of distributions over typos for each password w, so $\tau_w(\tilde{w})$ is the probability that the password $w \in \mathcal{W}$ is typed as $\tilde{w} \in \mathcal{S}$. Note that $\tau_w(w)$ denotes the probability that w is typed correctly. Together these components (p, τ) define an *error setting*.

Looking ahead, we will conservatively assume that an attacker has precise knowledge of the underlying error setting when we analyze the security of TypTop with respect to guessing attacks. In practice, TypTop uses a state of the art password strength estimator, zxcvbn [35], to estimate password guessability.

Adaptive checkers. An adaptive password checker Π is a stateful PBAS — that is to say $\Pi = (\mathsf{Reg}, \mathsf{Chk})$ is a pair of algorithm defined as follows:

- Reg is a randomized algorithm which takes as input a password w, and outputs an initial state s_0 for Π.
- Chk is an algorithm (possibly randomized) which takes as input a string $\tilde{w}$ and a state s, and outputs a bit b and an updated state s'. An output $b = 1$ means authentication is granted.

Our definition is analogous to the formalization of standard (non-adaptive) PBAS given in [8]; their definition may be recovered by keeping the state constant across invocations of Chk. We call a non-adaptive PBAS an *exact checker* if it outputs $b = 1$ only if the correct password is entered exactly.

We require that a password checker is *complete*, which is to say that the probability that a user successfully authenticates when he enters his correct password is one. Additionally we desire our typo-tolerant checker to authenticate under as many typos made by a legitimate user as possible (subject to security constraints). We will measure the utility of a typo-tolerant PBAS in terms of the fraction of typos accepted by the password checker across all users. By this measure, the utility of an exact checker is always 0.

Guessing attacks. Guessing attacks against PBAS schemes come in two key flavors: online and offline attacks. In the former, an attacker uses the login API of the system to submit different password guesses in an attempt to impersonate a user. The attacker might target a specific user in what is known as a vertical attack, or may try common passwords against the accounts of many users (known as a horizontal attack). Various countermeasures can be deployed to mitigate these attacks, including locking the account after a number of (e.g., 10) incorrect password submissions, and slowing down responses for login attempts that appear unsafe based on contextual information (for example, those originating from suspicious IP addresses).

In offline attacks we assume that the attacker has compromised the database of stored PBAS states, and attempts to brute-force guess the underlying values. In contrast to online attacks, the number of guesses made by an offline attacker is limited only by the computational power they are willing to expend. We analyze the security of TypTop in the face of both forms of attack in Section 5.

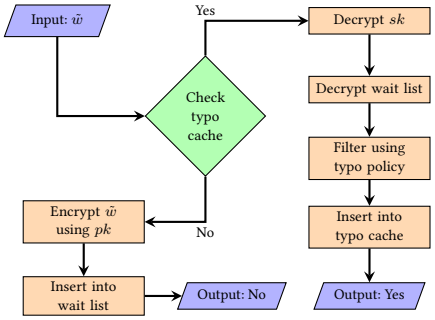


Figure 1: Diagram showing TypTop's approach to personalized typo-tolerant password checking.

4 THE TYPTOP DESIGN

We first give an overview of TypTop, and then detail its components more fully. TypTop uses a *typo cache* and an encrypted *wait list*. The typo cache securely stores the set of strings under which the user is allowed to authenticate; namely their registered password plus a number of previously accepted typos of that password. The wait list is a public-key encryption of recent incorrect password submissions that were not the registered password or one of the typos already in the typo cache. The secret decryption key for the wait list is, in turn, encrypted using the registered password and (separately) under each of the cached typos. When a login attempt's password submission is accepted — either it was the registered password or one of the previously accepted typos — the wait list can be decrypted and processed according to some typo cache policy. The latter defines which incorrect submissions should be allowed into the typo cache. A diagrammatic view of TypTop's Chk procedure is given in Figure 1.

Underlying components. We begin by defining the primitives utilized by TypTop. A public-key encryption (PKE) scheme $\mathsf{PKE} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ is a triple of algorithms. The key generation algorithm $\mathcal{K}$ takes random coins as input and outputs a public / secret key pair $(pk, sk) \leftarrow\!\!\$\ \mathcal{K}$. The randomized encryption algorithm $\mathcal{E}$ takes as input a public key pk and a message $m \in \mathcal{M}_{\mathcal{E}}$ (where $\mathcal{M}_{\mathcal{E}}$ denotes the message space), and outputs a ciphertext $c \leftarrow\!\!\$\ \mathcal{E}_{pk}(m)$. We let $\mathcal{C}_{\mathcal{E}}$ denote the ciphertext space. The deterministic decryption algorithm $\mathcal{D}$ takes as input a secret key sk and a ciphertext c and outputs a message $\tilde{m} \in \mathcal{M}_{\mathcal{E}} \cup \{\perp\}$. We use a PKE scheme with perfect correctness, meaning that the probability an honestly generated ciphertext decrypts to the correct message is one.

A password-based encryption (PBE) scheme $\mathsf{PBE} = (\mathsf{E}, \mathsf{D})$ is a pair of algorithms defined as follows. The randomized encryption algorithm E takes as input a password $w \in \mathcal{W}$ (the set of all allowed passwords) and a message $m \in \mathcal{M}_{\mathsf{E}}$ (the message space), and outputs a ciphertext $c \leftarrow\!\!\$\ \mathsf{E}_w(m)$, where we let $\mathcal{C}_{\mathsf{E}}$ denote the ciphertext space. The deterministic decryption algorithm D takes as input a password w and a ciphertext c and outputs a message $\tilde{m} \in \mathcal{M}_{\mathsf{E}} \cup \{\perp\}$. We require PBE to be perfectly correct. A conventional symmetric encryption scheme is the same as a PBE scheme, except that it assumes uniform bit strings of some length κ as keys.

We let $\mathsf{PBE}[\mathsf{SH},\mathsf{SE}] = (\overline{\mathsf{E}},\overline{\mathsf{D}})$ denote the canonical PBE scheme that works as follows. The scheme utilizes a random oracle with signature $\mathsf{SH}:\{0,1\}^* \to \{0,1\}^\kappa$ and a conventional symmetric encryption scheme $\mathsf{SE} = (\mathsf{E},\mathsf{D})$ that uses keys of length κ bits. Then $\overline{\mathsf{E}}(w,m)$ first chooses a fresh salt $\mathsf{sa} \leftarrow\!\!\$\ \{0,1\}^{\ell_{\mathsf{salt}}}$ for some suitably large ℓ_{salt}, computes $c = \mathsf{E}(\mathsf{SH}(\mathsf{sa} \,\|\, w), m)$, and outputs (sa, c). Decryption works in the obvious way.

Later, we will assume the message space associated to a PKE scheme unambiguously supports passwords of length up to some parameter ℓ (passwords will be unambiguously padded to this max length), a distinguished empty string symbol ε, and the state space of the caching scheme. We assume that encryptions of passwords and the empty string are of equal length. We assume PBE has a message space containing a typical representation of the PKE scheme's secret keys.

Finally, we let $\mathsf{Perm}(t)$ denotes the set of all permutations on $\mathbb{Z}_t$. Looking ahead, we shall set t to be the number of typos stored in the typo cache, and regularly apply a random permutation to randomize the order of elements in the cache. For our scheme, t will be small, making random permutations on $\mathbb{Z}_t$ easily sampleable and representable.

The details. A pseudocode description of TypTop's adaptive typo-tolerant password checking scheme $\Pi = (\mathsf{Reg},\mathsf{Chk})$ appears in Figure 2. TypTop uses a caching scheme to determine which entries in the wait list are integrated into the cache. In the figure we detail a probabilistic least frequently used (PLFU) caching scheme, but TypTop works modularly with other caching schemes as discussed later in this section.

The state of the adaptive checking scheme s consists of a public key pk, the encryption of the caching scheme's (plaintext) state S, a typo cache T, an encrypted wait list W, and an index γ which is a pointer to the next wait list entry that should be used. The typo cache consists of up to t PBE encryptions of the secret key sk corresponding to pk, where t is a parameter of the scheme. The typo cache is initially filled with random ciphertexts, unless otherwise indicated by the caching scheme in use (e.g., one may want to warm up the cache with possible typos as discussed below). The wait list consists of up to ω PKE encryptions of incorrect password submissions. For the wait list we use a simple least-recently entered eviction policy, accomplished by having index γ wrap around. To force a wrap around would require ω incorrect submissions before a correct one, so in practice we can set ω to be equal to a lockout threshold (such as 10). The wait list is initialized with encryptions of the empty string symbol ε, and cleared in the same manner. The index γ is initialized to a random value in $\mathbb{Z}_\omega$.

After every change to the typo cache, a random permutation $\pi \leftarrow\!\!\$\ \mathsf{Perm}(t)$ is used to permute the order of the cached typos. This is to ensure that even if an adversary knows the typos likely to be made by a user, he will not know at which position each typo lies in the cache. This will have ramifications for offline security, making the guessing game harder for particular distributions. See Section 5.

Caching schemes. TypTop maintains a set of cached typos which evolves over time based on the users' login attempts and the caching policy in use. We require that the set of cached typos are distinct, and that the real password is never cached as a typo; this maximizes the number of typos we can tolerate for a given cache size. We abstract out the process of initializing and updating the typo cache via a stateful caching scheme $\mathsf{Cache} = (\mathsf{CacheInit},\mathsf{CacheUpdt})$ defined as follows. The algorithm CacheInit takes as input a password w, and outputs an initial state for the caching scheme S_0 and a set $\mathcal{U}_0$ of typo / index pairs $(\tilde{w}, i)$, indicating that the typo $\tilde{w}$ should be stored at the i^{th} position in the initial cache. The algorithm CacheUpdt takes as input the caching scheme state S and a list $(\tilde{w}_1,\ldots,\tilde{w}_\omega)$ of candidate replacement typos (in our case, drawn from the wait list) plus any other information required to implement the caching policy (e.g., their frequencies). It outputs an updated state S' and a set $\mathcal{U}$ indicating the replacements to be made.

The checker Π is designed so that any caching scheme of choice may be dropped in. The set of caching schemes we consider is given in Figure 3. The simplest is a least recently used (LRU) caching scheme $\mathsf{CacheLRU} = (\mathsf{InitLRU},\mathsf{UpdateLRU})$ which maintains a list of typo cache indices ordered by how recently they were entered by the user; when we update the cache, the last (and least recently used) entry in the cache is evicted and replaced with the most frequently observed entry in the wait list. LRU ignores the frequency with which a user authenticates under a cached typo.

We consider three other strategies that take this frequency into consideration. The simplest scheme is the LFU policy, which performs cache updates by replacing the least frequently used cache typo with the most frequently observed wait list entry. The newly added typo has its frequency set to the number of times it appeared in the wait list.

A potential drawback of this approach is that we replace a cached typo each time we update, and so may inadvertently replace a typo that the user makes reasonably often (and is thus good to keep in the cache) with an anomalous typo from the wait list which they are unlikely to use again. We therefore give a probabilistic-LFU (PLFU) scheme, which performs cache updates as follows. First the frequencies of the least frequently used typo in the cache $\tilde{w}_o$ and the most frequently observed typo in the wait list $\tilde{w}_n$ are compared, and we replace the former with the latter with probability $\nu = f_{\tilde{w}_n}/(f_{\tilde{w}_n} + f_{\tilde{w}_o})$, where f denotes the frequency count of the typo in subscript in the wait list (for $\tilde{w}_n$) or the typo cache (for $\tilde{w}_o$). If such an update occurs, the frequency of the newly cached typo is set to $f_{\tilde{w}_n} + f_{\tilde{w}_o}$. This process is repeated for each of the unique typos in the wait list in descending order of their frequency. This both serves to decrease the probability that a useful cached typo is replaced unnecessarily, and increases the likelihood that typos which are observed repeatedly in low frequencies over different login attempts are cached; we give a detailed discussion in Appendix A.1.

The above schemes require $|\mathsf{S}| \in \mathcal{O}(t)$ space for caching; desirable in our construction since storing more data in the state of Π could negatively impact efficiency. In settings where this is less of a concern (e.g., authentication to personal devices) and we can afford to maintain a larger state, we can employ a most frequently used (MFU) caching policy which records the frequency of *all* valid typos made by a user so far. The t most frequent typos are maintained in the cache.

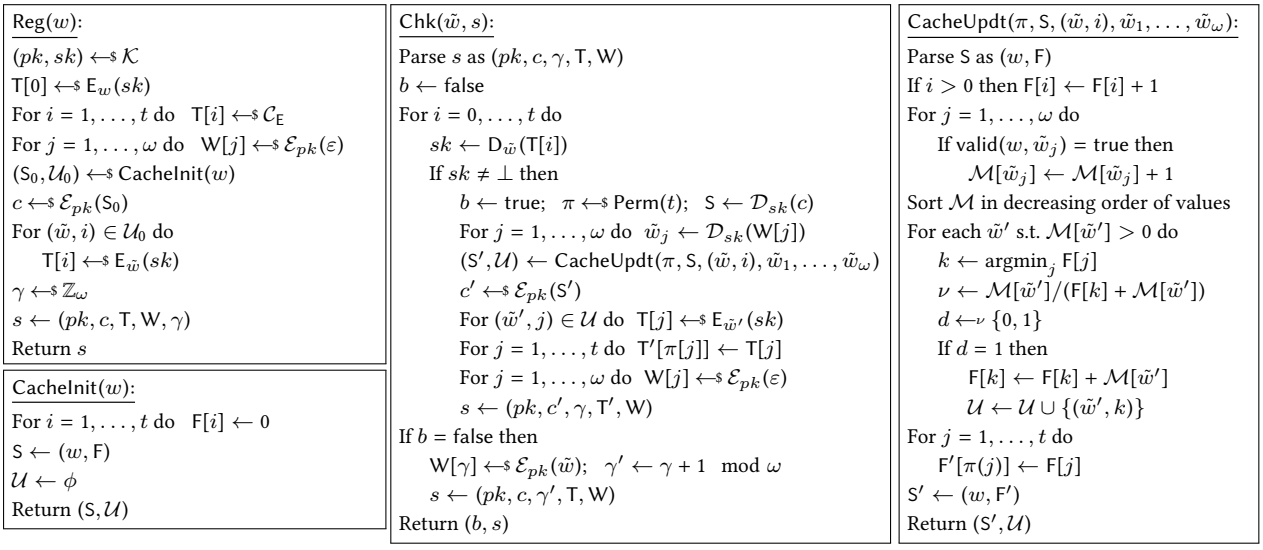


Figure 2: Our adaptive password checking scheme Π = (Reg, Chk) using a modified least-frequently used caching policy. The latter uses a function valid **that checks whether a string should be considered for entry into the typo cache (e.g., checking whether a string lies within an edit distance threshold of the true password).**

Scheme	Replacement Policy
LRU	Replace least recently used typo with $\tilde{w}_n$
LFU	Replace least frequently used typo and associated frequency with $(\tilde{w}_n, f_{\tilde{w}_n})$
PLFU	Replace least frequently used typo $\tilde{w}_o$ and associated frequency with $(\tilde{w}_n, f_{\tilde{w}_n} + f_{\tilde{w}_o})$ with probability $\frac{f_{\tilde{w}_n}}{f_{\tilde{w}_n} + f_{\tilde{w}_o}}$
MFU	Make necessary replacements to ensure t most frequently used typos lie in cache
Best-t	Initialize cache with t most probable typos based on typo model; never replace

Figure 3: Table summarizing the caching schemes considered. Here $\tilde{w}_n$ denotes the wait list typo being considered for inclusion in the cache, f denotes the frequency count of the typo in subscript, and t denotes the cache size.

As a benchmark against which to compare the utility benefit of adaptive checking, we also consider a static caching policy Best-t: fill the cache of a given password w with its t most likely typos according to some typo model, and then never update the cache. Looking ahead, we will build a typo model from measurements of typos made by users.

Admissible typos. As discussed in Section 3, care must be taken when deciding which typos are cached. As such we use a procedure valid to test whether an entry in the wait list should be input to CacheUpdt as a candidate for inclusion. Our policy applies three restrictions. Firstly, we set a threshold d, and only consider a typo $\tilde{w}$ of a password w for inclusion if $\mathrm{DL}(w, \tilde{w}) \leq d$. For TypTop we use $d = 1$ unless stated otherwise, allowing us to capture the caps lock errors, single substitutions, deletions and transpositions which studies indicate account for 46% of typos made by users [8].

Secondly, we wish to avoid including easily guessable typos in the cache which may speed up guessing attacks — for example `Password1#` may be mistyped as `Password1`, but the latter requires only 8 attempts to guess as opposed to nearer 1,000 for the former (as estimated by zxcvbn [35]). As such we impose two password strength checks with associated thresholds m, σ. For a typo $\tilde{w}$ of a password w to be considered admissible, $\tilde{w}$ must be such that both $\mu_{\tilde{w}} \geq m$ and $\mu_{\tilde{w}} \geq \mu_w - \sigma$, where μ denotes the strength estimate of the password / typo in subscript. The first condition ensures that the most easily guessed typos are never cached, while the second prevents the caching of typos significantly more guessable than the real password. We will use $m = 10$ and $\sigma = 3$ unless otherwise specified; a justification for these parameter choices is given in Section 6.1. To estimate guessability, we use zxcvbn due to its accuracy and ease of deployment; one could also use other strength estimators such as those based on neural networks [22].

Warming up caches. For all of the adaptive caching schemes described above, the user must make a typo at least once before it is considered for inclusion in the cache — for example, when the typo cache is "cold" immediately after registration, no typos will be tolerated. As such we consider initializing the typo cache T with probable typos of the registered password; a process we call "warming up" the cache. We build an empirical typo distribution using data collected via an MTurk study (detailed in Section 6.1), and the data released with [8]. We then fill the cache of a given password with its t most likely typos as indicated by the typo distribution, with their frequency counts set to 0. In contrast to relaxed checking, these cached typos are chosen on a per password basis (as opposed to using population-wide corrector functions). We will always warm up the cache unless otherwise specified.

5 SECURITY OF TYPTOP

In this section, we analyze the security of TypTop within the two main threat models for password checking schemes described in Section 3: offline and online attacks.

In the offline setting an attacker gains access to the state of the checker, so we first analyze TypTop from a cryptographic viewpoint, showing that the state does not leak additional information about the user's password and login history. We give a formal secu-

rity notion that captures this requirement, and provide a reduction showing that an attacker obtaining access to the state of TypTop learns nothing about the user's login behavior (including partial information about the password) unless they are able to brute-force guess the password or one of the typos active in the cache. With this in place, we consider the success probability of an attacker in such a brute-force attack. We show that for certain classes of error settings, the maximum advantage of an attacker against TypTop is no greater than that of an optimal attacker against the exact checker (who must brute-force guess the exact password in order to succeed).

We then analyze the online setting, which will be similar to the security analysis of the relaxed checking approach introduced by Chatterjee et al. [8]. The results indicate that security loss in the face of online attacks is minimal, with less security degradation than the prior approach [8].

In both online and offline settings, our analyses are with respect to an attacker who we conservatively assume has precise knowledge of the password distribution. In practice, where precise knowledge is unlikely, security will be even better than our analyses predict.

5.1 Cryptographic Security

Our security notion formalizes the following intuition. Consider an adversary that can obtain the state of a password checking system after registration plus some sequence of login attempts by a user. Then the adversary should not be able to distinguish this real state from one drawn at random from the state space of the checker $\mathcal{S}$, *unless* they are able to brute-force guess one of the passwords or typos allowed by the checking system at the point of compromise.

To this end we introduce some additional notation that will make defining security and our subsequent analysis simpler. For a given error setting (p, τ), we define an associated login transcript generator $\mathcal{T}$ to model a user's sequence of login attempts. Formally a login transcript generator $\mathcal{T}$ is defined to be a randomized algorithm that takes no input and outputs a sequence of passwords and typos which represent a user's selection of their password (the first password in the sequence) and subsequent login attempts, all sampled according to the appropriate distributions. A transcript checker associated to an adaptive password checker $\Pi = (\mathsf{Reg}, \mathsf{Chk})$ (see Section 3) is an algorithm $\mathsf{Checker}[\Pi]$ that takes as input a sequence of passwords and outputs a state value. The canonical such transcript checker, on input $w_0, \tilde{w}_1, \ldots, \tilde{w}_n$, runs $s_0 \leftarrow\!\$\ \mathsf{Reg}(w_0)$, and then computes $s_i \leftarrow\!\$\ \mathsf{Chk}(\tilde{w}_i, s_{i-1})$ iteratively for $1 \le i \le n$. It then outputs the final state s_n.

Let $\Pi = (\mathsf{Reg}, \mathsf{Chk})$ be an adaptive password checker and let $\mathcal{T}$ be a transcript generator. Consider the game OFFDIST of Figure 4. We define the offline distinguishing advantage of an adversary $\mathcal{A}$ against $\Pi, \mathcal{T}$ to be

$$\mathsf{Adv}^{\mathrm{offdist}}_{\Pi,\mathcal{T}}(\mathcal{A}) = 2 \cdot \left| \Pr\left[\mathrm{OFFDIST}^{\mathcal{A}}_{\Pi,\mathcal{T}} \Rightarrow \mathsf{true} \right] - \frac{1}{2} \right|$$

where the probability is over the random coins used in executing the game. We will not provide strict definitions of security (e.g. using asymptotics), but rather measure concretely the advantage of adversaries given certain running times and query budgets.

The security model proposed here coincides with a one-time compromise of the system. A stronger model would perhaps allow adaptive compromises, observing multiple instances of the password checking state over time. We conjecture that Π meets such a definition but leave the analysis to future work.

Preliminaries. Before our analysis, we fix a number of further definitions that will be needed in the proofs.

We implement TypTop with the canonical PBE scheme $\mathsf{PBE}[\mathsf{SH}, \mathsf{SE}] = (\overline{\mathsf{E}}, \overline{\mathsf{D}})$ as described in Section 4 which utilizes a symmetric encryption (SE) scheme SE and random oracle SH; adversaries in security games against this implementation of TypTop are given access to the random oracle accordingly.

We now define two security notions which we will require for the underlying SE scheme. The first is a multi-key real-vs-random security notion for symmetric encryption under chosen plaintext attack. Let $\mathsf{SE} = (\mathsf{E}, \mathsf{D})$ be a SE scheme with associated ciphertext space $\mathcal{C}_{\mathsf{E}}$. The pseudocode description of the security game $\mathrm{MKROR}^{\mathcal{B},t}_{\mathsf{SE}}$ appears in Figure 4. This game tasks the attacker with determining whether it is receiving encryptions of a (chosen) message, or a random ciphertext, in a multi-key setting. We define the distinguishing advantage of an adversary $\mathcal{B}$ as

$$\mathsf{Adv}^{\mathrm{mkror}}_{\mathsf{SE}}(\mathcal{B}, t) = 2 \cdot \left| \Pr\left[\mathrm{MKROR}^{\mathcal{B},t}_{\mathsf{SE}} \Rightarrow \mathsf{true} \right] - \frac{1}{2} \right| .$$

The security game for the familiar single-key real-vs-random security notion, which we denote $\mathrm{SKROR}^{\mathcal{B}}_{\mathsf{SE}}$, is obtained by setting $t = 1$ in the above definition. A straightforward hybrid argument shows that for any symmetric encryption scheme SE and adversary $\mathcal{B}$ in game $\mathrm{MKROR}^{\mathcal{B},t}_{\mathsf{SE}}$ running in time T, there exists an adversary $\mathcal{B}'$ in game $\mathrm{SKROR}^{\mathcal{B}}_{\mathsf{SE}}$ running in time $T' \approx T$ such that $\mathsf{Adv}^{\mathrm{mkror}}_{\mathsf{SE}}(\mathcal{B}, t) \le t \cdot \mathsf{Adv}^{\mathrm{skror}}_{\mathsf{SE}}(\mathcal{B}')$. By $T' \approx T$ (which we will also use in theorem statements below), we mean that the running time of $\mathcal{B}'$ is the same as that of $\mathcal{B}$ plus some qualitatively inconsequential overheads that can be derived from the proof.

We need one additional security property from our SE scheme: that of robustness, which ensures that no computationally efficient adversary can find two keys that both decrypt the same ciphertext. The notion of robustness for PKE schemes was introduced by Abdalla et al. [1], and later extended in [10, 23]. Security notions and constructions for robust SE schemes are given by Farshim et al. [11]. We use a variant of their full robustness notion that is strictly weaker than full robustness, but which suffices for our purposes. Formally, let $\mathrm{ROB}^{\mathcal{R}}_{\mathsf{SE}}$ be the game that works as follows. The adversary $\mathcal{R}$ runs with no inputs and outputs $(\mathsf{k}, \mathsf{k}', m)$, i.e., a pair of keys and a message. The game then computes $c \leftarrow\!\$\ \mathsf{E}(\mathsf{k}, m)$ and $m' \leftarrow \mathsf{D}(\mathsf{k}', c)$. The game outputs true if $\mathsf{k} \ne \mathsf{k}'$ and $m' \ne \perp$. We define the advantage of an adversary $\mathcal{R}$ as $\mathsf{Adv}^{\mathrm{rob}}_{\mathsf{SE}}(\mathcal{R}) = \Pr\left[\mathrm{ROB}^{\mathcal{R}}_{\mathsf{SE}} \Rightarrow \mathsf{true} \right]$.

Finally we require a more standard real-vs-random ciphertext notion of security for a PKE scheme $\mathsf{PKE} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ with associated ciphertext space $\mathcal{C}_{\mathcal{E}}$. The game $\mathrm{ROR}^{\mathcal{C}}_{\mathsf{PKE}}$ (not shown) first generates a key pair $(pk, sk) \leftarrow\!\$\ \mathcal{K}$ and a random bit b. It then runs adversary $\mathcal{C}(pk)$, who is given access to an oracle RoR to which it may query messages m. The oracle computes $c_1 \leftarrow\!\$\ \mathcal{E}(pk, m)$ and $c_0 \leftarrow\!\$\ \mathcal{C}_{\mathcal{E}}$ and returns c_b. Finally $\mathcal{C}$ outputs a bit b', and the game returns $(b = b')$. We define the distinguishing advantage of an adversary $\mathcal{C}$ as

$$\mathsf{Adv}^{\mathrm{ror}}_{\mathsf{PKE}}(\mathcal{C}) = 2 \cdot \left| \Pr\left[\mathrm{ROR}^{\mathcal{C}}_{\mathsf{PKE}} \Rightarrow \mathsf{true} \right] - \frac{1}{2} \right| .$$

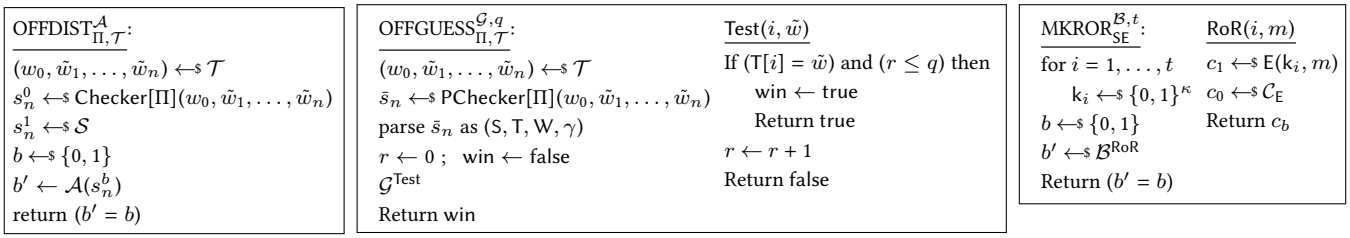


Figure 4: Cryptographic security games for adaptive password checking schemes, offline guessing attacks, and multi-key real-or-random symmetric encryption security.

Offline guessing attacks. With this in place, we now define the eventual target of our reduction: a guessing game in which the adversary obtains an oracle to make guesses against the password and its t cached typos. Observe that for the canonical transcript checker Checker[Π], the eventual entries in the typo cache (and wait list) associated to s_n depend not only on the input transcript $w_0, \tilde{w}_1, \ldots, \tilde{w}_n$ but also on whether a ciphertext in the typo cache is erroneously decrypted to something besides $\perp$ when using the *wrong* password. We can, however, rule out such an event using the robustness of the SE scheme (see definition above).

In order to simplify the subsequent analysis, we define a modified transcript checker PChecker[Π] that evolves the state of Π using only *plaintext* values. Crucially, rather than relying on a successful decryption to determine whether a password / typo lies in the cache, we may now simply compare the input to the plaintext cache values, thereby eliminating the negligible probability of erroneous state updates. The pseudocode for PChecker[Π] is given in Figure 5.

The game OFFGUESS$^{\mathcal{G},q}_{\Pi,\mathcal{T}}$ is given in Figure 4. The guessing advantage of an adversary $\mathcal{G}$ who makes at most q queries to the Test oracle is defined as $\mathsf{Adv}^{\mathsf{offguess}}_{\Pi,\mathcal{T}}(\mathcal{G}, q) = \Pr[\mathsf{OFFGUESS}^{\mathcal{G},q}_{\Pi,\mathcal{T}} \Rightarrow \mathsf{true}]$. The transcript generator and plaintext checker are first used to sample a set of cache values; the adversary succeeds if he can guess any password or typo which lies in the cache. Note that this game requires the adversary to specify which password each guess should be checked against. We measure the complexity of guessing adversaries in terms of the number of Test queries they make.

We assume without loss of generality that all adversaries make legitimate queries in their respective games (i.e., with values inside the appropriate domains, and with an index in the range $[0, t]$ for MKROR and OFFGUESS).

The analysis. Let $\Pi = (\mathsf{Reg}, \mathsf{Chk})$ denote TypTop's password checker, and fix some transcript generator $\mathcal{T}$. Our analysis will show that the $\mathsf{OFFDIST}_{\Pi,\mathcal{T}}$ security of Π reduces to the guessing game $\mathsf{OFFGUESS}_{\Pi,\mathcal{T}}$. We note that this analysis is independent of the specific caching scheme used by TypTop; the effect on security of different such schemes will be surfaced in Section 5.2 when we bound the success probability of an attacker in game $\mathsf{OFFGUESS}_{\Pi,\mathcal{T}}$.

As the first step in our reduction, we introduce PChecker[Π] into game $\mathsf{OFFDIST}_{\Pi,\mathcal{T}}$, via an intermediate game $\overline{\mathsf{OFFDIST}}_{\Pi,\mathcal{T}}$ defined to be identical except we replace Checker[Π] with PChecker[Π], and then use the values in the final plaintext state $\bar{s}_n = (\mathsf{S}, \mathsf{T}, \mathsf{W}, \gamma)$ to compute the final (encrypted) challenge state s_n as specified by the scheme. We bound the transition between the games by

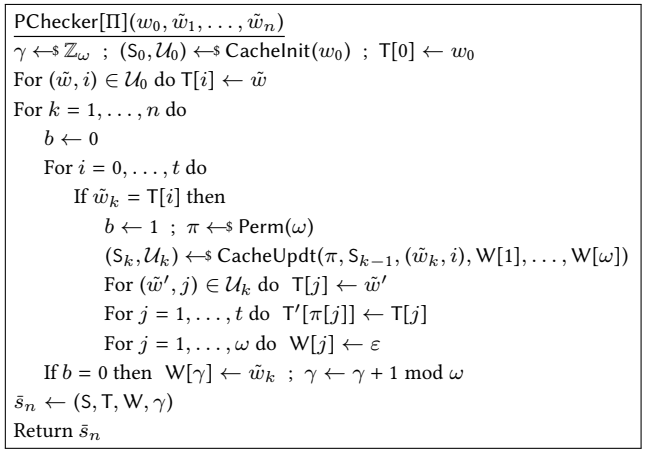


Figure 5: The plaintext transcript checking scheme associated to Π with caching scheme Cache = (CacheInit, CacheUpdt). All entries of tables T and W are initially set to $\perp$ and ε, respectively.

invoking the following lemma, which is implied by a reduction to the robustness of SE. We give the full proof in Appendix A.3.

Lemma 5.1. *Let (p, τ) be an error setting with associated transcript generator $\mathcal{T}$, and let $\Pi = (\mathsf{Reg}, \mathsf{Chk})$ be TypTop's password checker, with associated plaintext checker* PChecker[Π]. *Let Π be implemented using the canonical PBE scheme* $\mathsf{PBE}[\mathsf{SH}, \mathsf{SE}] = (\overline{\mathsf{E}}, \overline{\mathsf{D}})$ *where* SE *is a symmetric encryption scheme and* SH *is a random oracle. Then for any adversary $\mathcal{A}$ running in time T and making at most q queries to* SH*, there exist adversaries $\mathcal{A}'$, $\mathcal{R}$ such that*

$$\mathsf{Adv}^{\mathsf{offdist}}_{\Pi,\mathcal{T}}(\mathcal{A}) \le \mathsf{Adv}^{\overline{\mathsf{offdist}}}_{\Pi,\mathcal{T}}(\mathcal{A}') + 2 \cdot \mathsf{Adv}^{\mathsf{rob}}_{\mathsf{SE}}(\mathcal{R}) + \frac{(t \cdot (n+1)+1)^2}{2^{(\kappa-1)}} ,$$

and, moreover, $\mathcal{A}'$ and $\mathcal{R}$ run in time $T' \approx T$, and $\mathcal{A}'$ makes at most q queries to SH*. Here t denotes the cache size,* SE *takes as keys uniform bit strings of length κ, and n denotes the length of the transcript.*

We now state our main theorem in which we upper bound the advantage of an attacker $\mathcal{A}$ in game $\mathsf{OFFDIST}_{\Pi,\mathcal{T}}$.

Theorem 5.2. *Let (p, τ) be an error setting with associated transcript generator $\mathcal{T}$, and let $\Pi = (\mathsf{Reg}, \mathsf{Chk})$ be TypTop's password checker with associated plaintext checker* PChecker[Π]. *Let Π be implemented using the canonical PBE scheme* $\mathsf{PBE}[\mathsf{SH}, \mathsf{SE}] = (\overline{\mathsf{E}}, \overline{\mathsf{D}})$ *where* SE *is a symmetric encryption scheme and* SH *is a random oracle. Then for any adversary $\mathcal{A}$ running in time T and making at*

most q queries to SH*, there exist adversaries $\mathcal{B}, \mathcal{C}, \mathcal{R}, \mathcal{G}$ such that*

$$\mathsf{Adv}^{\mathrm{offdist}}_{\Pi,\mathcal{T}}(\mathcal{A}) \le \mathsf{Adv}^{\mathrm{offguess}}_{\Pi,\mathcal{T}}(\mathcal{G}, q) + 2 \cdot \mathsf{Adv}^{\mathrm{mkror}}_{\mathsf{SE}}(\mathcal{B}, t) + \frac{(t+1)^2}{2^{\ell_{\mathrm{salt}}}} + 2 \cdot \mathsf{Adv}^{\mathrm{rob}}_{\mathsf{SE}}(\mathcal{R}) + 2 \cdot \mathsf{Adv}^{\mathrm{ror}}_{\mathsf{PKE}}(\mathcal{C}) + \frac{(t \cdot (n+1)+1)^2}{2^{(\kappa-1)}} ,$$

and, moreover, $\mathcal{B}, \mathcal{C}, \mathcal{R}, \mathcal{G}$ run in time $T' \approx T$. Here t denotes the cache size, SE *takes as keys uniform bit strings of length κ, and n denotes the length of the transcript. The salts used to derive keys for the canonical PBE scheme are of length ℓ_{salt}. Adversary $\mathcal{B}$ makes t queries to its encryption oracle, and $\mathcal{C}$ makes $\omega + 1$ queries to its encryption oracle, where ω is the length of the wait list.*

The above theorem shows that the distinguishing advantage of an attacker in game $\mathrm{OFFDIST}_{\Pi,\mathcal{T}}$ is upper bounded by the probability that they can guess either the real password or a cached typo (which comprises the first term of the right hand side of the above equation) plus the remaining terms which, for the appropriate choice of cryptographic components and key sizes, can be assumed to be negligibly small. This implies that an attacker who compromises the state of the adaptive checker in an offline attack learns no information about the underlying password and the login pattern, *unless* they can guess one of the cached values. We will analyze the probability that this occurs in Section 5.2.

We sketch the proof here and defer a detailed treatment to Appendix A.3. We first apply Lemma 5.1 to transition to game $\overline{\mathrm{OFFDIST}}_{\Pi,\mathcal{T}}$ and thereby introduce the plaintext checker. We then argue by a series of game hops, beginning with the cache state s_n as in game $\overline{\mathrm{OFFDIST}}_{\Pi,\mathcal{T}}$ with challenge bit $b = 0$. We then modify the PBE scheme to sample salts without replacement; this will ultimately be used to ensure that the attacker has to submit guesses to distinct cache positions when we reduce to game $\mathrm{OFFGUESS}_{\Pi,\mathcal{T}}$. With this in place, we further set a flag which is set only if the attacker queries one of the cached values to his random oracle, allowing us to eventually reduce to the success probability of $\mathrm{OFFGUESS}_{\Pi,\mathcal{T}}$. Finally we use the real-or-random ciphertext security of the SE and PKE schemes to replace all of the real ciphertexts in the state of the checker with random ciphertexts, thus transforming the state into a random one as per game $\overline{\mathrm{OFFDIST}}_{\Pi,\mathcal{T}}$ with challenge bit $b = 1$.

On the use of PBKDFs. Above in our analysis we have abstracted away the details of the slow hash SH and modeled it simply as a random oracle. This allows accounting for queries to SH as unit cost, which will suffice for our analysis. One can go further, however, replacing SH with a true password-based key derivation function and converting the unit cost to that of computing the hash function (e.g., the cost of c applications of a standard hash function, in the case that SH is replaced by a hash chain construction such as PKCS#5). See for example [5, 16, 27] for a discussion of relevant results.

5.2 Security Against Offline Guessing Attacks

Having reduced the offline security of TypTop to the guessing game OFFGUESS, we now upper bound the success probability of an attacker in this game. Recall that p denotes the distribution of passwords chosen by users, $\mathcal{W}$ denotes the support of p, and τ denotes the family of typo distributions for each password $w \in \mathcal{W}$.

When TypTop is used in a given error setting, the probability that a typo lies in the cache of a given password depends inherently on the caching policy in use. For an error setting and instantiation of TypTop, we let $\tilde{\tau}$ denote the *induced cache inclusion function* where $\tilde{\tau}_w(\tilde{w})$ denotes the probability that $\tilde{w}$ is included in the typo cache of password w. We provide a general security analysis of TypTop in terms of $\tilde{\tau}$, then concretize our analysis by empirically modeling $\tilde{\tau}$ for a real world error setting and a number of caching policies. Letting $T[j]$ denote the distribution of the typo at position j in the cache, the fact that the set of cached typos are distinct and randomly permuted implies that $\Pr[T[j] = \tilde{w} \mid T[0] = w] = \frac{1}{t} \cdot \tilde{\tau}_w(\tilde{w})$, for all $0 < j \le t$.

We would like to establish a class of typo distributions for which adding typo-tolerance via TypTop offers no security degradation over an exact checker. Since an exact checker accepts the correct password only, the analogous guessing game has the adversary attempting to guess a user's password with a budget of q guesses, and we denote the success probability achieved by an optimal attacker as λ_q. It is easy to see that the attacker's best strategy is to guess the q most probable passwords according to the distribution, so it follows that $\lambda_q = \sum_{i=1}^{q} p(w_i)$, where $w_1, w_2, \ldots$ denote the passwords in $\mathcal{W}$ sorted in decreasing order of their probability.

We define the *edge-weight* of a typo $\tilde{w}$ under the induced cache inclusion function $\tilde{\tau}$ to be $b_{\tilde{\tau}}(\tilde{w}) = \sum_{w \in \mathcal{W}} \tilde{\tau}_w(\tilde{w})$. Notice that for a given typo $\tilde{w}$, we have that $\tilde{\tau}_w(\tilde{w}) \in [0, 1]$ for *each* password w, so in theory the edge-weight could be very large. We say that an error setting is *t-sparse* with respect to TypTop with cache size t and a particular caching scheme, if for all $\tilde{w} \in \mathcal{M}$ it holds that $b_{\tilde{\tau}}(\tilde{w}) \le t$. In the following theorem, we show that if an error setting is t-sparse then there is *no speedup* in an optimal offline attack against TypTop as opposed to an optimal offline attack against an exact checker ExChk.

Theorem 5.3. *Let (p, τ) be an error setting with associated transcript generator $\mathcal{T}$, and let $\Pi = (\mathsf{Reg}, \mathsf{Chk})$ be TypTop's password checker with typo cache size t. Then if the error setting is t-sparse with respect to Π, then for any adversary $\mathcal{G}$, it holds that*

$$\mathsf{Adv}^{\mathrm{offguess}}_{\Pi,\mathcal{T}}(\mathcal{G}, q) \le \lambda_q \quad \textit{where} \quad \lambda_q = \sum_{i=1}^{q} p(w_i)$$

The full proof is given in Appendix A.4; we provide a brief sketch here. We begin by splitting the set of guess / index pairs output by $\mathcal{G}$ into two exclusive sets — a set Z_0 consisting of guesses at the real password in cache position $\mathsf{T}[0]$, and a set Z_1 consisting of guesses at the cached typos in positions $\mathsf{T}[j]$ where $0 < j \le t$. The success probability induced by the former set is simply $\sum_{w \in Z_0} p(w)$. The success probability contributed by the latter set is the expectation — over all passwords *not* already accounted for by the guesses at the real password in Z_0 — that one of these cache typo guesses succeeds. We show — via a general result which allows us to succinctly upper bound a certain class of summations in which this latter success probability lies — that the t-sparsity of the error setting implies that the guessing advantage arising from both sets is upper-bounded by λ_q, as required.

Are real world error settings t-sparse? It remains to establish whether real world error settings are t-sparse; if so, then

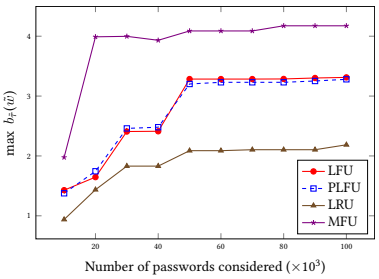


Figure 6: The change in the maximum edge-weight for different number of passwords considered from RockYou leak.

Theorem 5.3 shows that we can enjoy the typo-tolerance offered by TypTop with *no loss* in offline security compared to an exact checker. It is easy to see that not every error setting is t-sparse; e.g., imagine (p, τ) such that all passwords are mistyped to the same typo, with no other typos possible. This one typo will lie in the cache of every password with probability close to 1, greatly degrading security and pushing the maximum edge-weight well above t.

In this section we use simulations to show that real world error settings (p, τ) are indeed t-sparse. We model the password distribution p using password data observed in the RockYou password leak [33], which consists of 14 million unique passwords from 32 million users. We sanitized the Rockyou data by removing passwords longer than 50 characters (as these are unlikely to be human chosen passwords) and shorter than 6 characters (as per common password policy recommendation), and set p equal to the resulting distribution. We model the typo distribution τ using data on user's password typing habits gathered via an MTurk experiment (see Section 6.1), and the data released with [8]. We describe how we built this model in Appendix A.2.

To compute the induced cache inclusion function $\tilde{\tau}$ for (p, τ) and TypTop with a given caching policy, we simulate the associated transcript generator $\mathcal{T}$ using the password distribution p and the typo distribution τ. For each password w, we sample n strings $\tilde{w}_1, \ldots, \tilde{w}_n$ according to τ_w. This captures a user with registered password w who makes a series of login attempts including some typos. We run PChecker (with default parameters) on input $(w, \tilde{w}_1, \ldots, \tilde{w}_n, w)$, where the final correct password entry is included to ensure at least one cache update occurs. We ran the simulation m times for each password w, and set $\tilde{\tau}_w(\tilde{w})$ to be the fraction of these m runs that the typo $\tilde{w}$ lies in the final cache of the password w. We repeat this process for all four caching policies described in Section 4. Via preliminary simulations on a small subset of randomly sampled passwords with different values of m and n, we found that $\tilde{\tau}_w$ changed very little for $n \geq 1000$ and $m \geq 200$; therefore we set $n = 1000$ and $m = 200$ for our full simulation.

Running the simulation for each of the passwords in the RockYou leak would be very slow, so we estimate $\tilde{\tau}_w$ using the k most probable passwords in the RockYou leak for $k \in \{1, \ldots, 10\} \times 10^4$. With this in place, we compute the edge-weight $b_{\tilde{\tau}}(\tilde{w}) = \sum_w \tilde{\tau}_w(\tilde{w})$ for each possible typo $\tilde{w}$, and report the maximum observed edge-weight for each k and caching policy in Figure 6.

For all four caching policies, we found that $\tilde{\tau}_w$ comfortably satisfies the desired 5-sparsity condition (recall that we implement TypTop with cache size $t = 5$) for all $k \leq 10^5$. The largest observed edge-weight was 4.2 for the MFU caching policy, with a maximum edge-weight of only 3.2 for the PLFU policy which we shall ultimately choose for deployment (see Section 6).

Moreover, we find that the maximum edge-weight increases minimally in the range $5 \times 10^4 \leq k \leq 10^5$ for all caching policies considered. This, coupled with the fact that the maximum edge-weights are well below the required threshold of 5.0 for all k considered, indicates that if we were able to perform simulations on the entire password distribution, we would still find the error setting to be 5-sparse as required.

The top 10^5 passwords in the RockYou leak share some structural similarities (e.g., 90% of these passwords contain only letters and numbers, and only 5% are more than 10 characters in length). To check that these similarities are not biasing our results, and to gain further support for our conclusion that the simulated error setting (p, τ) is 5-sparse, we repeat the above experiment using two different sets of passwords to estimate $\tilde{\tau}_w$ and the corresponding edge-weights. The first set consists of k passwords chosen randomly from the support of p, and the second consists of the k most probable passwords when the top one million passwords are excluded from consideration. As before, we consider all values of $k \in \{1, \ldots, 10\} \times 10^4$.

We observed similar trends to those displayed in Figure 6, except that in these two experiments the maximum observed edge-weights are even better (in terms of security) at 2.8 and 3.0 respectively for the MFU caching scheme, and less for all other caching policies. This is likely to be because passwords which are distinct in structure induce more diverse sets of typos. This results in fewer typos being shared between multiple passwords, which in turn decreases their observed edge-weights.

To assess the benefit of the admissible typo policy described in Section 4, we additionally repeated the above experiments with these restrictions removed one at a time. We found that the error setting (p, τ) no longer remains 5-sparse if any of the three restrictions are removed. These simulations emphasize the importance of the admissible typo policy for security.

5.3 Security Against Online Attacks

We briefly discuss TypTop's resistance to online guessing attacks, in which an attacker attempts to impersonate a user via the login API of the system. The main difference between online and offline attacks against TypTop is that in the former, each guess the attacker makes is checked against *every* entry in the cache, whereas in the latter it is only checked against the specific cache slot guessed. We provide full details of these notions and our analyses in Appendix A.5

To estimate the decrease in the online security of TypTop compared to an exact checker, we approximate the success probability of an optimal online attacker using a greedy algorithm similar to that used by Chatterjee et. al. in [8], and data from real world password / typo distributions. We show that security loss is minimal for all caching schemes (namely a loss of 0.2% for the MFU policy, and less than 0.1% for all others, including the PLFU caching scheme which we ultimately choose for deployment). We also describe a simple blacklisting strategy, in which a small subset of the typos most beneficial to an attacker are excluded from the typo cache, and prove that this reduces the security loss to zero.

Another variety of online attack against adaptive checkers considers an adversary who is able to interleave his guesses with correct password submissions by the legitimate user. For TypTop these correct submissions trigger cache updates, so it is possible that one of the attacker's guesses, stored in the wait list, is allowed into the cache.

Recall that we set TypTop's admissible typo policy so only typos within DL distance one of the real password are allowed in the cache. This in itself provides a degree of protection against these interleaving attacks. We additionally propose a simple countermeasure which virtually eliminates them. We can associate an origin tag (such as an IP in the web login setting, or TTY id in an SSH login) to each entry in the wait list; only entries originating from destinations at which the user has successfully authenticated are allowed to enter the cache. This may exclude the occasional typo made by a legitimate user from a new location; we defer a detailed treatment of this attack and countermeasures to future work.

6 EVALUATING UTILITY

In this section, we investigate how successful TypTop is at correcting user's password typos under different parameter settings and ultimately select the best performing ones for real world deployment. To gather data about user's password typing patterns, we conducted an experiment on Amazon Mechanical Turk (MTurk).

We define the *utility* of a typo-tolerant PBAS to be the fraction of typos accepted by the checker taken across the population; formally the utility of a PBAS Π is defined Utility(Π) = # (typos accepted by Π) / # (typos observed). The same utility metric is used by Chatterjee et al. in [8], allowing for a direct comparison of results.

6.1 Data Collection From MTurk

Our study — designed to capture the password typing behavior of a user who first registers a password with a service, and then reenters this password to authenticate himself at subsequent logins — asks an MTurk worker to choose a password and type it repeatedly over a period of time. The design is similar to that of Komanduri et al. in [19], and has two stages: registration and login. Registration consists of a single MTurk HIT (Human Intelligence Task) in which we ask the worker to choose a password that is at least 8 characters long, as if they were registering with an email provider. They are encouraged to choose a password which is strong and distinct from any of their existing passwords, and are informed that they must memorize this password for future logins. As with many registration forms, they must type the password twice. The user then fills in a short survey (1–2 questions) as means of distraction, and then is instructed to attempt to login with their registered password via a login form. The worker has to type the password correctly to be able to submit the HIT. An option for a "forgotten password" link is provided; however this link delays the HIT by 20 seconds, discouraging them from clicking it frequently. We informed workers that they must type their passwords manually and avoid browser auto-fills. We recorded every keypress made inside the password boxes and used heuristics to reject any submissions that did not appear to have been typed in.

After this registration step is complete, we create a sequence of 50 user-specific HITs for the worker, which they alone may view and complete; this allows us to track individual user's typing habits over different login attempts, while keeping their chosen password confidential. Each HIT consists of the same login form used in the final step of the registration stage. The user must complete the login, then answer a few questions on their demographics or password typing behavior. New HITs are created an hour after the submission of the last HIT to prevent workers from completing them back-to-back; we notify the workers when a new HIT is available. The workers are paid $0.05 for each HIT they complete, with an additional bonus of $0.04 for completing every five HITs.

Data cleansing and demographics. Due to various incompatibility issues with rare browser versions and submissions which appeared to have been auto-filled, data from 42 workers had to be discarded. Of the remaining 438 workers, 271 (61.9%) made at least 10 login attempts. In all subsequent analysis, we only consider the data from these 271 workers. Based on the demographics survey, we found that 48% of the 271 workers are males and 52% female; only 35 (13%) of the users are left handed with the rest right handed. 117 (43%) of the users belong to the age group 18–30, 108 (40%) belong to the age group 31–45, 30 (11%) to the age group 46–60, and 16 (6%) are above the age of 60.

6.2 Analysis of Passwords and Typos

All of the passwords chosen by the MTurk workers were unique, with a median and average length of 10 and 10.9 characters respectively, and average estimated zxcvbn strength of 31.5 bits. Most passwords included special characters, numbers and different case letters. Overall the passwords chosen by MTurk workers were stronger than those seen in most password leaks (e.g., passwords in the RockYou leak have a median password length of 7 characters and average zxcvbn entropy of 14.2 bits). While we expect our results to hold for weaker password choices, further studies will need to be performed to confirm this.

The workers made a median of 30 login attempts per person and 8,739 login attempts in total; 491 (5.6%) of these contained at least one incorrect password submission. Within the login attempts, there were a total of 9,440 password submissions; of these 701 (7.4%) were incorrect, with 484 (70%) of these incorrect submissions lying within DL distance 2 of the real password — as per the discussion in Section 3, we classify these 484 incorrect submissions as typos, and therefore eligible for typo-tolerance in our subsequent analysis. A classification of typos into exclusive categories is shown in Figure 7; our study finds similar user typo behavior to that observed by Chatterjee et al. in [8]. Looking ahead, we will calculate the utility of a given implementation of TypTop to be the fraction of these 484 typos accepted by that implementation.

A graph depicting the DL distances of incorrect password submissions is given in Figure 9a (the blue line). In total, 366 separate login attempts (4.2%) required at least one password resubmission due to a typo. We found that 167 users (62%) made at least one typo, with 95 (35%) making at least two typos in two different login attempts.

Typo Category	% of Typos	% of Users
Caps Lock	14	21
Shift first char	4	7
One insertion	12	28
One deletion	12	25
One replacement	31	47
Transposition	4	7
Two insertions	3	5
Two deletions	3	5
Two replacements	10	20
Other	8	16

Figure 7: Categorization of typos observed in the MTurk study. The middle column gives the percentage of observed typos which were of each category. The rightmost column gives the percentage of users who made a typo of that category.

6.3 Simulation Setup

We perform simulations using the data from our MTurk study to evaluate the utility of various combinations of cache sizes, caching schemes, and typo admission policies. In more detail, we consider:

- **Cache size:** While a larger cache allows us to accept more typos and increases utility, this benefit needs to be weighed up against the greater computational power required to process larger caches, and the potential degradation in online security from allowing more typos to authenticate. We consider caches of size t for $t \in \{2, 3, 5\}$ in our simulations.
- **Caching Schemes:** We consider all of the caching schemes discussed in Section 4: LRU, LFU, PLFU, MFU, and Best-t. To see how TypTop compares with the relaxed checking approach of Chatterjee et al., we also run simulations for the relaxed checker using the Top-t corrector functions as per [8], which in order of their efficacy are: flipping the case of all characters, flipping the case of the first character, removing the last character, removing the first character, and applying shift to switch the last digit to a symbol.
- **Typo admission policy**: As discussed in Section 4, admissible typos must satisfy three criteria:
(1) $\mathrm{DL}(w, \tilde{w}) \leq d$; (2) $\mu_{\tilde{w}} > m$; and (3) $\mu_{\tilde{w}} > \mu_w - \sigma$. We investigated all combinations of the values $d \in \{1, 2\}$, $m \in [0, 40]$, and $\sigma \in [0, 9]$.
- **Warming up the cache:** Since warming up the cache allows us to tolerate typos from the very first login, it can only increase utility. Additionally our security simulations in Section 5, which were all performed with warmed caches, indicate that this practice does not negatively impact security. As such we warm caches with the t most probable typos according to our typo model (see Appendix A.2).

For each set of parameter choices in the scope of the above, we simulate each worker's login behavior by replaying their password submissions to the password checker being evaluated. We report the utility for each of these configurations below; recall that the utility of a checker Π is defined to be Utility(Π) = # (typos accepted by Π) / # (typos observed), where both numerator and denominator are taken across all users.

CP	$d = 1$			$d = 2$		
	t=2	t=3	t=5	t=2	t=3	t=5
LFU	18	21	26	19	24	31
PLFU	19	22	27	22	26	32
MFU	18	21	26	19	25	30
LRU	17	21	27	19	25	32
Best-t	16	19	23	16	19	23
Top-t	18	19	22	18	19	22

Figure 8: The utility of different caching policies (CP), cache sizes (t) and edit distance cutoff (d) for admissible typos, when applied to the login transcripts of all MTurk workers who made at least one typo. We impose no guessability restrictions on admissible typos.

6.4 Results

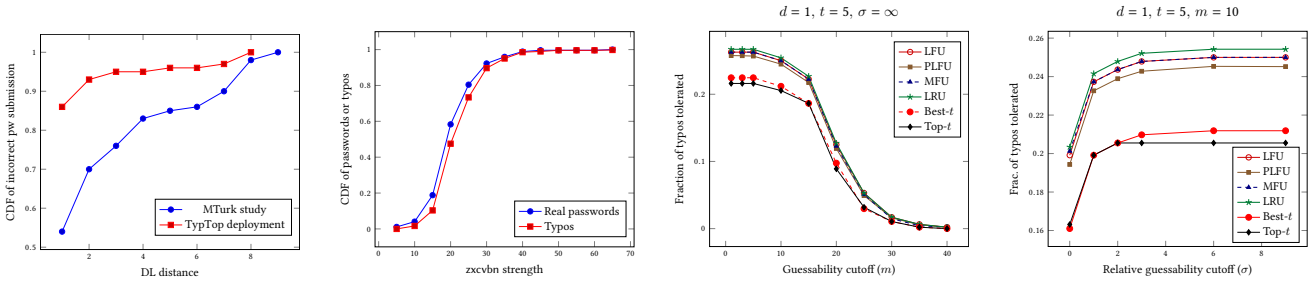
Caching policies and cache sizes. We first compare the efficacy of different caching schemes, cache sizes t, and DL distance thresholds d. For these simulations, we impose no guessability restrictions on admissible typos, setting $m = 0$ and $\sigma = \infty$, as this will maximize the number of admissible typos. The utility of each strategy is shown in Figure 8. We see that larger cache sizes (which allow us to cache and tolerate more typos) correspond to an increase in utility. Similarly increasing the DL distance threshold (and with it the set of typos considered for inclusion in the cache) also increases utility. However, we conjecture that allowing typos with DL distance 2 will degrade security enough to outweigh the utility benefits (e.g., the resulting error settings are less likely to be t-sparse), so we select $t = 5$ and $d = 1$ for further analysis and deployment.

For caching schemes, perhaps unsurprisingly, the utility of relaxed checking with Top-t correctors is very similar to that of the static policy Best-t — the former performs better for $t = 2$, while the latter performs better for $t = 5$. This is because in our unoptimized typo model, the common error of flipping the case of the first character often does not appear among the two most probable typos for a given password, whereas it does get corrected by Top-2 correctors; the resulting typos missed by Best-2 allows Top-2 to outperform it. However for $t = 5$, the benefit of the password-specific typo correction offered by Best-5 emerges, catching typos that the Top-5 correctors (which are chosen based on population-wide analysis) cannot correct. With more data about typos and future refinement of the parameters should overcome this small difference.

Because each individual worker made only a small number of incorrect submissions in our study, the choice of caching policy had little observable impact on utility for cache sizes $t \geq 3$. However the variation is more noticeable for cache size $t = 2$. As anticipated LRU performs less well than the frequency based caching policies. Among the latter set, PLFU performs the best. Surprisingly, we also see that MFU underperforms PLFU; however this could be due to the fact that we did not receive enough incorrect submissions to see the benefit of MFU emerge, and conjecture that in a longer term study, MFU may outperform PLFU.

Due to strong performance for utility and security (as discussed in Section 5), and the cache size restraints, (which make MFU unsuitable for practical purposes), we choose PLFU as the caching policy for deployment, with a cache size of $t = 5$ and DL distance threshold $d = 1$.

Different guessability restrictions. Next we investigate the im-



(a) (b) (c) (d)

Figure 9: (a) CDF of fraction of incorrect password submissions within a given DL distance of the real password. (b) CDFs of zxcvbn strength of real passwords compared to typos in the MTurk study. (c),(d) The change in utility for different absolute guessability thresholds (m) and relative guessability thresholds (σ). The values of other parameters are specified above each chart.

pact of different admissible typo parameters m and σ on utility (recall that for typo $\tilde{w}$ to be considered for inclusion in the cache of password w, it must be the case that $\mu_{\tilde{w}} > m$, and $\mu_{\tilde{w}} > \mu_w - \sigma$). We begin by considering the guessability cutoff parameter $m \in \{0, \ldots, 40\}$. We perform simulations for TypTop implemented with cache size $t = 5$, edit distance threshold $d = 1$, but without any relative guessability cutoff imposed ($\sigma = \infty$). We compute utility for all guessability cutoff values $m \in \{0, \ldots, 40\}$ and all caching policies; the results are shown in Figure 9c. As expected, utility decreases as the guessability cutoff increases and fewer typos are considered for inclusion in the cache. Notably, we see the utility decrease rapidly for $m > 10$. This is because, as shown in Figure 9b, nearly 5% of the observed typos have an estimated security strength of less than 10 bits, indicating that setting $m = 10$ gives the maximum security benefit without significantly degrading utility.

We then investigated the effect on utility of different relative guessability cutoff parameter settings σ. We perform simulations with the same parameter settings as above, except we now fix $m = 10$. We compute utility for all relative guessability cutoff parameters $\sigma \in [0, 9]$ and all caching policies; the results are shown in Figure 9c. For most of the caching policies, we found no significant improvement on utility for $\sigma > 3$. This is to be expected, given that the guessability of passwords and typos are very similar (as shown in Figure 9b). Since increasing the guessability cutoff further increases the possibility that a significantly more guessable string enters the cache with minimal benefit to utility, we choose $\sigma = 3$ to optimize the balance between utility and security.

Users and logins benefited. Among the 167 users who made at least one typo during their submission (recall, here a typo is an incorrect submission within edit distance 2 of the real password), 75 users (44.9%) would benefit by having at least one of their typos accepted by TypTop with the PLFU caching policy and the parameter setting described above. In contrast, only 49 users (29.3%) would receive this benefit from the relaxed checker of [8] implemented with the Top-5 correctors. Of the 366 login attempts containing at least one typo, 106 (29%) of these would require at least one less password resubmission if TypTop were used as the password checker. This saves an average of 5 seconds per login attempt for the users making at least one typo, and an average of 12 seconds for the users who made two typos in two different logins. In our small MTurk study (271 users), we find that TypTop would save 23 person-minutes of login time, and with larger user populations expect to see this time saved grow to several person-months.

We test the null hypothesis that TypTop (with the parameters decided above) does not outperform relaxed checking with Top-t correctors in tolerating typos for a randomly chosen user using a Wilcoxon signed-rank test [36]. We found that we can reject the null hypothesis with p-value < 0.001.

7 A CASE STUDY WITH TYPTOP

Password authenticated personal device logins are a key scenario in which users may benefit from the typo-tolerance offered by TypTop. In this section we discuss how to build TypTop for typo-tolerant device logins in Mac OS X and Linux based systems. We deploy this prototype on 25 volunteers' machines, obtaining data on their password typing behaviors, and report on the performance of TypTop in a real world deployment.

Implementation of TypTop. We build TypTop as a pluggable authentication module (PAM) [30] for Unix based systems. The state of TypTop for each user is stored in a separate file with the same permissions as the `/etc/shadow` file, which is used to store users' password hashes and is readable only by the root. The Chk procedure is implemented in C++ and installed with similar file permissions to `passwd`, a Linux utility tool which is used to update a user's password. Via modifying the PAM configuration files, we set TypTop to be engaged on almost all applications which require password based authentication. TypTop is very simple to install, but requires root privileges.

As detailed in Section 6, we implement TypTop with parameters: {CP=PLFU, t=5, d=1, m=10, σ=3}. There is currently no option to customize the parameters; we plan to introduce restricted control for system admins in the next version.

We tested the computational overhead of TypTop in an Ubuntu 14.04 laptop with Intel Core M processor and 8 GB of memory. The size of the state-file is 13KB. The average turnaround time is 110 milliseconds for a successful authentication (i.e., the correct password or a cached typo is entered), and 250 milliseconds for an incorrect submission. For traditional Ubuntu logins (e.g., with su), login takes less than 1 millisecond.

The main computational overhead incurred by TypTop is due to our use PBKDF2 [16] with 20,000 iterations of SHA-256 for each computation of the hash function SH, while (somewhat surprisingly) a default Ubuntu laptop uses only 1 iteration of SHA-256 for

its hash computation. Current standards [16, 29] recommend using at least 5,000 iterations of SHA-256 to hash passwords.

Chk will always perform the maximum number of hashes for an incorrect submission, which is why we see a longer turn around time for failed authentications. However, this overhead is well below the noticeable limit for users. To avoid timing side channels, we might want to compute the maximum number of hash computations for every login attempt (successful or failed) to make these turnaround times constant.

Pilot deployment of TypTop. To gather data on TypTop's efficacy, we modify the implementation of TypTop slightly to allow confidential logging of users' password typing behavior.

The logging module works as follows. During password registration, two random 16-byte strings are generated and stored in the state of TypTop; the first is used as an HMAC key to compute a unique identifier for each submitted password (or typo), while the second is used as a user-device identifier. With every password submission, we log the HMAC identifier of the submission, along with the DL distance from the real password, the relative guessability, and whether the typo could have been corrected using Top-5 correctors (to form a comparison between TypTop and relaxed checking). This allows us to learn the transcript of password and typo submissions — and thus simulate a run of TypTop — while never learning the actual strings entered. Some of these values (e.g., DL distance) require both the underlying password and the typo to process; we back-fill these values after a successful authentication when the encrypted caching scheme state and wait list are decrypted. To simulate the Best-5 static caching scheme, we also log the identifiers of the five most probable typos of the password.

The log and corresponding user identifier are uploaded to a server via an HTTPS post request every 10 logins, after which the uploaded log is deleted from the user's laptop. The key used to generate the HMAC identifiers is never uploaded, making it impossible to brute-force recover passwords given the information uploaded. Users can disable the logging and / or uploading feature at any time during the study without any effect on the functionality.

Data collection and analysis. We initially advertised our study via two university mailing lists and a number of social media groups. However, we received a lower response rate than anticipated, as users were initially reluctant to run research code as their primary mode of authentication. Therefore, while not optimal, we used snowball sampling [4] to increase participation in our study.

Study participation is anonymous: we do not know the exact set of volunteers who installed TypTop, although we can deduce the number of distinct machines on which TypTop was installed from the logs generated. In subsequent analysis, we assume that each user installed TypTop on only one machine. We collected data from 25 users over a period of 22–100 days (different users have varying data collection periods depending on when they joined the study, and how long they chose to participate for). TypTop was used for a median of 27 days.

We observed a total of 4,563 password submissions during the study, with a median of 103 submissions per user. The average user types their password more than five times a day, and incorrectly 15% of the time. We found that 93% of these incorrect submissions are within DL distance two of the real password and thus are classified as typos. The fraction of incorrect submissions that constitute typos is larger than that observed in the MTurk study (70%); this is probably because MTurk workers created their 'passwords' for the study, and so were more likely to make high DL distance errors due to misremembering them.

In total we observed 501 typos, of which 316 (63%) are tolerated by TypTop. This is significantly higher than the 122 (22%) that would be tolerated by the relaxed checker with Top-5 correctors. However the benefit of typo correction varies across users. We observed 2 users who received an especially great benefit from TypTop. They used TypTop for over 45 days, during which they typo-ed their password 24% of the time — we found TypTop corrected 85% of their typos, whereas the Top-5 corrector functions corrected virtually none. For the remainder of the participants, we found that TypTop and the Top-5 correctors performed roughly the same. We believe this is because personal machines in university settings tend to have very lenient password policies (if any policy at all), meaning users pick simpler passwords whose typos are more likely to be among those corrected by the Top-5 correctors. We expect to see the performance benefit of TypTop emerge in settings where users must pick stronger and less readily correctable passwords (e.g., over 12 characters), and lock / unlock their computers many times a day. We simulated other caching policies on the collected data, and observed roughly the same performance.

These results suggest that personalized typo-tolerance may be very beneficial to users who are especially typo-prone, and that this benefit increases the longer the system is used. The sample size of our initial pilot deployment is too small to allow us to draw more general conclusions at this stage; as such we are planning a study with more participants and which runs for a longer period of time. We will be publishing TypTop as a public, open-source project to facilitate a study with a broader set of participants.

8 CONCLUSION

We introduce the notion of personalized typo-tolerant password checkers, which adapt over time to correct the typos made most frequently by the individual user. We design and build an adaptive password checking scheme called TypTop, which securely caches incorrect password submissions that pass a policy check on what forms of typos to allow. We formalize a cryptographic security notion for such schemes, and show a formal reduction of our scheme's security to the difficulty of brute-force cracking attacks against the registered password or the typos entered into the typo cache. We give simple criteria that, if met, ensure no security loss in offline attack and negligible security loss in online attack, and empirically verify that real world password and typo distributions satisfy this requirement. Simulations conducted with data gathered via a study on Amazon MTurk suggest that TypTop will outperform existing approaches to typo-tolerant password checking, and a small pilot deployment suggests that TypTop can provide a substantial usability benefit to especially typo-prone users.

ACKNOWLEDGMENTS

We thank Sam Scott for helping us with implementation of the first prototype of TypTop, the volunteers for participating in our users

studies, and the anonymous reviewers for their insightful comments. This work was supported in part by NSF grants CNS-1514163 and CNS-1514163, United States Army Research Office (ARO) grant W911NF-16-1-0145, and a generous gift from Microsoft.

A APPENDIX

A.1 Benefits of the PLFU Caching Scheme

We now describe the intuition behind the utility benefits of the probabilistic least frequently used (PLFU) caching scheme over its deterministic counterpart (LFU). Recall that during each PLFU cache update, the cached typo $\tilde{w}_o$ is replaced with the wait listed typo $\tilde{w}_n$ with probability $\nu = f_{\tilde{w}_n}/(f_{\tilde{w}_n} + f_{\tilde{w}_o})$, where f denotes the frequency count of the typo in the subscript. The main goal of the PLFU caching scheme is to let the cache be agile enough to adapt and change, while simultaneously increasing the likelihood that the most useful typos ultimately stay in the cache.

The PLFU scheme has two key benefits over the non-probabilistic LFU scheme. First, it makes it more likely that a typo which is repeated in small amounts over many login attempts (and thus is likely to increase usability if cached) ultimately enters the cache, as opposed to one which appears multiple times in a single (and possibly anomalous) login attempt. For example, if a typo $\tilde{w}_n$ is repeated once across r login attempts in which the least frequently used typo $\tilde{w}_o$ is not entered at all, the probability that $\tilde{w}_n$ will not enter the cache is approximately $\left(1 - 1/(1 + f_{\tilde{w}_o})\right)^r$, which is less than $1/e$ if $r \approx f_{\tilde{w}_o} > 2$. Conversely, if the same typo appears $r \approx f_{\tilde{w}_o}$ times in a single login attempt, the probability that it does not then enter the cache is $\left(1 - r/(f_{\tilde{w}_o} + r)\right) \approx 1/2 > 1/e$.

Second, by setting the frequency count for the newly cached typo to $f_{\tilde{w}_n} + f_{\tilde{w}_o}$, the PLFU scheme increases the eventual stability of the cache, as the increasing frequency counts of the cached typos mean that the probability that they are replaced by a wait listed typo decreases over time. In contrast the LFU caching scheme updates the cache on each login attempt, increasing the chance that useful typos are accidentally evicted from the cache.

A.2 Modeling the Typo Distribution τ_w

In this section, we describe the procedure with which we built the typo model τ used for the simulations on Page 8. We use a supervised training method to learn the typo distribution using the typo data collected in our MTurk study (Section 6.1) and the data released with [8]. Our approach is inspired by that of Houser et al. [14].

A simple way to build the typo model would be to compute the frequency distribution of typos for each password. However, our data set contains only 30,000 typos of 20,000 distinct passwords in total. As such, both the set of passwords on which we have typo data, and the amount of typo data we have for the individual passwords, is too small to build a good frequency based model. Therefore, we make two simplifying assumptions about typographical errors: that a typo of a given character in a password is influenced by the characters very close to it, and that this typo is independent of the characters in the remainder of the string.

Given a list of pairs of passwords and typos, we first align each pair by inserting a special symbol "␣" zero or more times (as required), such that the resulting pair of strings are of the same length, and have the same DL distance as the originals. For example, the password-typo pair (`password`, `pasword`) may be aligned to (`password`, `pa␣sword`). If there are multiple alignments possible, we consider all of them. For each of the aligned password-typo pairs $(w, \tilde{w})$, we take the set of substring pairs $(w_{i:j}, \tilde{w}_{i:j})$ for all $0 \le j \le k$; $0 \le i \le |w| - j$, and compute the frequency distribution of those pairs across all the aligned password-typo pairs. Here $|x|$ denotes the length of the string x; $x_{i:n}$ denotes the sub-string of length n of x beginning at location $i \le |x| - n$; and k is a parameter of the model. We let E denote the frequency distribution of these pairs of strings, and will use it in subsequent steps to compute the typo probability of a given password.

We define an edit as a triplet (i, l, r), where i denotes a location in the string, and l and r are strings of length at most k. An edit (i, l, r) is valid for a password w if $w_{i:|l|} = l$. Transforming a password w by applying a valid edit (i, l, r) means, replacing $w_{i:|l|}$ with the sub-string r. For a given password w we let E_w denote the set of all possible valid edits in the set $\mathbb{Z}_{|w|} \times E$. We assign weights to each edit (i, l, r) in E_w as the frequency of the pair (l, r) according to the frequency distribution E, divided by the number of locations $j \in [0, |w|]$ for which (j, l, r) is a valid edit for w. The weights are then normalized to define a probability distribution P_w over the valid edits of w.

With this in place, the process of sampling a typo of a password w according to the typo model τ_w is reduced to sampling an edit from P_w and applying it to w. Note that there could be multiple edits of a password w which result in the same typo $\tilde{w}$. Thus, $\tau_w(\tilde{w})$ is equal to the sum of the probabilities of all edits that transform w into $\tilde{w}$.

Efficacy of the typo model. We use the average log likelihood — which is a typical measure for evaluating generative models — to gauge the efficacy of our typo model. We compare our model against the naive uniform model, that assigns uniform probabilities to all the typos. Note, a model is better if the average log likelihood value is higher.

We perform a cross validation of our model over five 80:20 train-test splits of the data set of password / typo pairs. For each split, we train our typo model using the training data, and compute the average log likelihood of test samples as the average of $\log \tau_w(\tilde{w})$ taken over all pairs $(w, \tilde{w})$ in the test data. We find that the average log likelihood of the test data according to our model is −7.2 with standard deviation 0.4, which is much better than base uniform model, which obtains an average likelihood of −11. This suggests that our model captures the real world typo distribution fairly well.

A.3 Proofs from Section 5

Proof of Lemma 5.1. We begin by proving Lemma 5.1 which we shall utilize in subsequent analysis.

Proof: We argue by a series of game hops. Let game G_0 denote game $\text{OFFDIST}^{\mathcal{A}}_{\Pi,\mathcal{T}}$ (as defined in Figure 4), so

$$\text{Adv}^{\text{offdist}}_{\Pi,\mathcal{T}}(\mathcal{A}) = 2 \cdot \left| \Pr[\, G_0 \Rightarrow 1 \,] - \frac{1}{2} \right| .$$

Recall that the cache of TypTop stores up to $(t + 1)$ ciphertexts, each of which corresponds to a password-based encryption (using

the canonical PBE scheme PBE[SH, SE]) of the secret key of the PKE scheme sk under the real password and each of the (at most t) cached typos.

We define a new game G_1 which is identical to G_0 except that the keys sampled to encrypt cached ciphertexts are now sampled without replacement. In more detail, when a new typo is cached during the evolution of the challenge state s_n, a salt is chosen $sa \leftarrow\$ \{0,1\}^{\ell_{salt}}$, and the cached ciphertext is computed as $c \leftarrow\$ \mathsf{E}(\mathsf{SH}(sa||\tilde{w}), sk)$. In game G_0 oracle SH responds to fresh queries of this form by returning $k_j \leftarrow\$ \{0,1\}^{\kappa}$, whereas in G_1 oracle SH samples these keys without replacement. In the distinguishing phase of G_1, SH returns to sampling keys with replacement. These games run identically unless two keys sampled during the computation of these cached ciphertexts collide. Since at most $(t \cdot (n+1)+1)$ such keys are sampled while processing a transcript of length n (where the $(t+1)$ term corresponds to the maximum number of keys sampled to encrypt the ciphertexts in the initial cache, and the $t \cdot n$ term arises as processing each of the n typos in the transcript can introduce at most t new ciphertexts in the typo cache), it follows that

$$2 \cdot |\Pr[\,G_0 \Rightarrow 1\,] - \Pr[\,G_1 \Rightarrow 1\,]| \leq \frac{(t \cdot (n+1)+1)^2}{2^{\kappa}} \ .$$

Next we define game G_2 which is identical to G_1 except we replace Checker[Π] with PChecker[Π] and a sequence of statements that encrypt the final typo cache, state and wait list returned by it as specified by the scheme. Notice that these games run identically unless during the process of updating the state we find two distinct keys $k_1 \neq k_2$ such that $\mathsf{D}_{k_2}(\mathsf{E}_{k_1}(sk)) \neq \perp$ where sk denotes the secret key of the PKE scheme which is encrypted under each of the cached typos. As such the fundamental lemma of game playing [3] implies that the gap between game G_1 and G_2 is upper-bounded by the probability that this event occurs. Notice that we can further upper bound this probability by $\mathsf{Adv}^{\mathrm{rob}}_{\mathsf{SE}}(\mathcal{R})$ as follows. Consider an adversary $\mathcal{R}$ in game $\mathrm{ROB}^{\mathcal{R}}_{\mathsf{SE}}$ who simply executes the game G_1, simulating SH by sampling random strings without replacement, and checking if there ever exists a typo cache ciphertext $\mathsf{E}_{k_1}(sk)$ that decrypts under some subsequently sampled $k_2 \neq k_1$ (recall that since G_1 samples without replacement, all keys are distinct). The gap between these two games is upper bounded by the probability that this event occurs, and so the robustness of the encryption scheme implies that

$$2 \cdot |\Pr[\,G_1 \Rightarrow 1\,] - \Pr[\,G_2 \Rightarrow 1\,]| \leq 2 \cdot \mathsf{Adv}^{\mathrm{rob}}_{\mathsf{SE}}(\mathcal{R}) \ .$$

Next we define game G_3 which is identical to G_2 except we return SH to sampling keys with replacement. An analogous argument to that above bounding the probability that two keys collide ensures that the gap between these games is again bounded above by $\frac{(t \cdot (n+1)+1)^2}{2^{\kappa}}$.

Notice that G_3 is identical to game $\overline{\mathrm{OFFDIST}}_{\Pi,\mathcal{T}}$, and so may be perfectly simulated by an attacker $\mathcal{A}'$ in this game. On input challenge state s_n, attacker $\mathcal{A}'$ passes this state to $\mathcal{A}$ in game G_3, simulating $\mathcal{A}$'s random oracle by querying his own oracle SH, and returning the responses. Since $\mathcal{A}'$ makes precisely the same set of oracle queries as $\mathcal{A}$, it follows that if $\mathcal{A}$ makes at most q queries, then $\mathcal{A}'$ does also. At the end of the game $\mathcal{A}'$ outputs whatever bit $\mathcal{A}$ does, and so

$$2 \cdot \left|\Pr[\,G_3 \Rightarrow 1\,] - \frac{1}{2}\right| = \mathsf{Adv}^{\overline{\mathrm{offdist}}}_{\Pi,\mathcal{T}}(\mathcal{A}') \ ,$$

concluding the proof. ∎

Proof of Theorem 5.2. We now provide the full proof of Theorem 5.2.

Proof: Consider an adversary $\mathcal{A}$ in game $\mathrm{OFFDIST}^{\mathcal{A}}_{\Pi,\mathcal{T}}$. Recall that by Lemma 5.1, there exist adversaries $\mathcal{A}'$ and $\mathcal{R}$ both running in time approximately that of $\mathcal{A}$ and where $\mathcal{A}'$ makes the same number of oracle queries as $\mathcal{A}$ such that,

$$\mathsf{Adv}^{\mathrm{offdist}}_{\Pi,\mathcal{T}}(\mathcal{A}) \leq \mathsf{Adv}^{\overline{\mathrm{offdist}}}_{\Pi,\mathcal{T}}(\mathcal{A}') + 2 \cdot \mathsf{Adv}^{\mathrm{rob}}_{\mathsf{SE}}(\mathcal{R}) + \frac{(t \cdot (n+1)+1)^2}{2^{(\kappa-1)}} \ ;$$

so it is sufficient to upper bound the success probability of an attacker $\mathcal{A}'$ in game $\overline{\mathrm{OFFDIST}}^{\mathcal{A}'}_{\Pi,\mathcal{T}}$. We argue by a series of game hops, shown in Figure 10. We begin by defining game G_0, which is identical to game $\overline{\mathrm{OFFDIST}}^{\mathcal{A}'}_{\Pi,\mathcal{T}}$ with $b = 0$. We also set two flags, bad-sa and bad neither of which affect the outcome of the game.

Next we define game G_1 which is identical to G_0 except the salts used by the canonical PBE scheme to compute the ciphertexts in the challenge state are now sampled without replacement. Games G_0 and G_1 run identically unless the flag bad-sa is set to true. Since there are at most $t+1$ salts sampled, the birthday bound and the fundamental lemma of game playing therefore imply that this transition is bounded above by $\frac{(t+1)^2}{2^{\ell_{salt}+1}}$, and so

$$\begin{aligned} |\Pr[\,G_0 \Rightarrow 1\,] - \Pr[\,G_0 \Rightarrow 1\,]| &\leq \Pr[\,\text{bad-sa = true in game } G_1\,] \\ &\leq \frac{(t+1)^2}{2^{\ell_{salt}+1}} \ . \end{aligned}$$

We now define game G_2 which is identical to G_1 except we change the way in which the random oracle SH responds to queries. Now if in the guessing phase $\mathcal{A}$ queries SH on a salt / password pair (sa, w) on which it was queried during the computation of the challenge state s_n, it responds with an independent random string $k \leftarrow\$ \{0,1\}^{\kappa}$ updating its hash table to this new value, as opposed to the previously used value. Accordingly in G_2 the keys k used by the SE encryption scheme are now random and independent of the underlying password. Games G_1 and G_2 run identically unless $\mathcal{A}$ manages to guess and query SH on one of the cached passwords and corresponding salt; an event we mark by setting a flag bad = true. The fundamental lemma of game playing then implies that,

$$|\Pr[\,G_1 \Rightarrow 1\,] - \Pr[\,G_2 \Rightarrow 1\,]| \leq \Pr[\,\text{bad = true in game } G_2\,] \ ,$$

a probability which we shall upper bound in a later game.

Next we define game G_3, which is identical to G_2 except we replace all symmetric encryptions with random ciphertexts. This transition is bounded by a reduction to the MKROR security of SE. Formally, let $\mathcal{B}_1$ be an adversary in game $\mathrm{MKROR}^{\mathcal{B}_1,t}_{\mathsf{SE}}$ with challenge bit b'. Adversary $\mathcal{B}_1$ runs $(w_0, \tilde{w}_1, \ldots, \tilde{w}_n) \leftarrow\$ \mathcal{T}$, followed by $\bar{s}_n \leftarrow\$ \mathsf{PChecker}[\Pi](w_0, \tilde{w}_1, \ldots, \tilde{w}_n)$, and generates a public / secret key pair $(pk, sk) \leftarrow\$ \mathcal{K}$. $\mathcal{B}_1$ then constructs the encrypted state s_n as follows. $\mathcal{B}_1$ first chooses $sa_i \leftarrow\$ \{0,1\}^{\kappa}$ and uses his RoR oracle to compute $c_i = \mathsf{RoR}(i, sk)$ for $i = 0, \ldots, t$, placing the salt / ciphertext pairs in the appropriate positions in the cache. (Recall that from game G_2, the symmetric keys used to create the

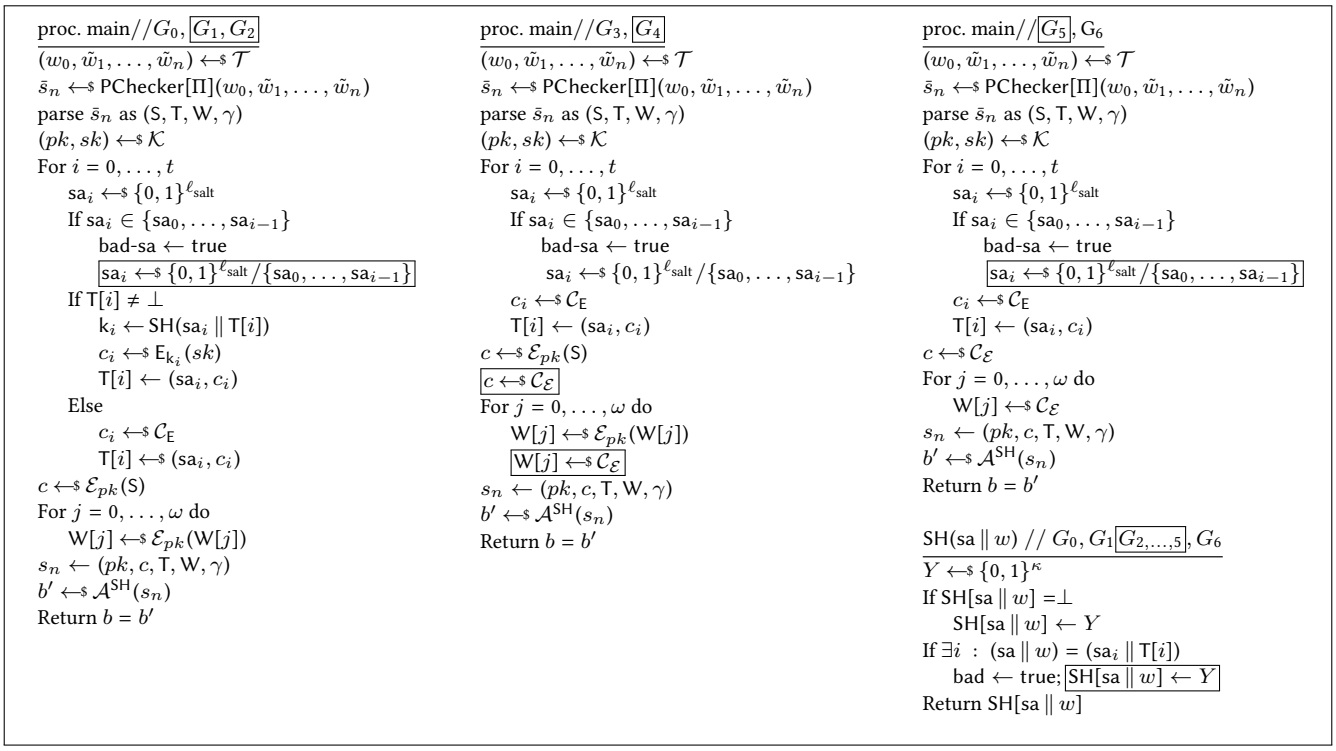


Figure 10: Games used in the proof of Theorem 5.2.

ciphertexts in s_n are random and independent of the salts and underlying passwords, and so identically distributed to those used by $\mathcal{B}_1$'s RoR oracle). Next, $\mathcal{B}_1$ encrypts S and the entries in W under the public key pk, and assembles challenge state s_n accordingly. Finally $\mathcal{B}_1$ passes s_n to $\mathcal{A}'$, simulating queries to SH by returning a random bit string to each fresh query, and at the end of the game outputs whatever bit $\mathcal{A}'$ does. Notice that if $b' = 1$ and $\mathcal{B}_1$ is receiving real encryptions from the RoR oracle then this perfectly simulates G_2, and if $b' = 1$ this perfectly simulates G_3. It follows that,

$$\begin{aligned}|\Pr[\,\mathrm{G}_2 \Rightarrow 1\,] - \Pr[\,\mathrm{G}_3 \Rightarrow 1\,]| &= |\Pr\left[\,\mathcal{B}_1 \Rightarrow 1 \mid b' = 0\,\right] - \Pr\left[\,\mathcal{B}_1 \Rightarrow 1 \mid b' = 1\,\right]| \\ &\leq \mathsf{Adv}^{\mathrm{mkror}}_{\mathsf{SE}}(\mathcal{B}_1, t).\end{aligned}$$

We can similarly show that the probability that bad is set in G_3 is close to the probability that it is set in G_2 via a separate reduction to the MKROR security of SE. Formally let $\mathcal{B}_2$ be an adversary in game $\mathrm{MKROR}^{\mathcal{B}_2}_{\mathsf{SE},t}$. $\mathcal{B}_2$ constructs the simulated state s_n as described above, using its RoR oracle to compute the symmetric encryptions in the state. However now when $\mathcal{B}_2$ passes s_n to $\mathcal{A}'$, it watches the queries $\mathcal{A}'$ makes to SH. If there exists a query $(\mathsf{sa}_i || \tilde{w}_i)$ where $i \in [0, t]$ such that $\tilde{w}_i$ is equal to the typo corresponding to position i in the cache (in which case the flag bad $\leftarrow$ true), $\mathcal{B}_2$ outputs 1; else it returns 0. By the same argument made above, it follows that,

$$\begin{aligned}\Pr[\,\mathsf{bad} = \mathsf{true} \text{ in } \mathrm{G}_2\,] &\leq \Pr[\,\mathsf{bad} = \mathsf{true} \text{ in } \mathrm{G}_3\,] \\ &\quad + |\Pr\left[\,\mathcal{B}_2 \Rightarrow 1 \mid b' = 0\,\right] - \Pr\left[\,\mathcal{B}_2 \Rightarrow 1 \mid b' = 1\,\right]| \\ &\leq \Pr[\,\mathsf{bad} = \mathsf{true} \text{ in } \mathrm{G}_3\,] + \mathsf{Adv}^{\mathrm{mkror}}_{\mathsf{SE}}(\mathcal{B}_2, t)\ .\end{aligned}$$

We may now define a third adversary $\mathcal{B}$ in game $\mathrm{MKROR}^{\mathcal{B},t}_{\mathsf{SE}}$ who flips a bit and depending on the outcome runs either $\mathcal{B}_1$ or $\mathcal{B}_2$ and then outputs the same bit as that adversary. It is easy to see that $\mathsf{Adv}^{\mathrm{mkror}}_{\mathsf{SE}}(\mathcal{B}, t) = \frac{1}{2} \cdot (\mathsf{Adv}^{\mathrm{mkror}}_{\mathsf{SE}}(\mathcal{B}_1, t) + \mathsf{Adv}^{\mathrm{mkror}}_{\mathsf{SE}}(\mathcal{B}_2, t))$, and so it follows that

$$\begin{aligned}&|\Pr[\,\mathrm{G}_2 \Rightarrow 1\,] - \Pr[\,\mathrm{G}_3 \Rightarrow 1\,]| + \Pr[\,\mathsf{bad} = \mathsf{true} \text{ in } \mathrm{G}_2\,] \\ &\leq \frac{1}{2} \cdot \left(\mathsf{Adv}^{\mathrm{mkror}}_{\mathsf{SE}}(\mathcal{B}_1, t) + \mathsf{Adv}^{\mathrm{mkror}}_{\mathsf{SE}}(\mathcal{B}_2, t)\right) + \Pr[\,\mathsf{bad} = \mathsf{true} \text{ in } \mathrm{G}_3\,] \\ &\leq 2 \cdot \mathsf{Adv}^{\mathrm{mkror}}_{\mathsf{SE}}(\mathcal{B}, t) + \Pr[\,\mathsf{bad} = \mathsf{true} \text{ in } \mathrm{G}_3\,]\ .\end{aligned}$$

Now we can define a game G_4 which replaces all encryptions under the public-key encryption scheme PKE with random ciphertexts, where this transition is bounded by a reduction to the ROR security of PKE. Formally we can define an adversary $\mathcal{C}_1$ in game $\mathrm{ROR}^{\mathcal{C}_1}_{\mathsf{PKE}}$ with challenge bit b' who proceeds as follows: on input pk, $\mathcal{C}_1$ first runs $(w_1, \tilde{w}_1, \ldots, \tilde{w}_n) \leftarrow\$ \mathcal{T}$, $\bar{s}_n \leftarrow\$ \mathsf{PChecker}[\Pi](w, \tilde{w}_1, \ldots, \tilde{w}_n)$. $\mathcal{C}_1$ submits S and the elements in W to its RoR oracle, chooses random symmetric ciphertexts and salts, and assembles the final state including the public key pk it was given as part of its challenge. $\mathcal{C}_1$ then passes s_n to $\mathcal{A}'$, simulating queries to SH in the natural way, and at the end of the game outputs whatever bit $\mathcal{A}'$ does. Notice that if $b' = 0$ then $\mathcal{C}$ receives real encryptions and this perfectly simulates G_3; otherwise it perfectly simulates G_4. It follows that,

$$\begin{aligned}|\Pr[\,\mathrm{G}_3 \Rightarrow 1\,] - \Pr[\,\mathrm{G}_4 \Rightarrow 1\,]| &= |\Pr\left[\,\mathcal{C}_1 \Rightarrow 1 \mid b' = 0\,\right] - \Pr\left[\,\mathcal{C}_1 \Rightarrow 1 \mid b' = 1\,\right]| \\ &\leq \mathsf{Adv}^{\mathrm{ror}}_{\mathsf{PKE}}(\mathcal{C}_1)\ .\end{aligned}$$

Furthermore, an analogous argument to that made above implies

that we can construct an adversary $\mathcal{C}_2$ in the $\text{ROR}^{\mathcal{C}_2}_{\text{PKE}}$ game against PKE who simulates the final state s_n using its RoR oracle, passes s_n to $\mathcal{A}'$ and outputs 1 if and only if $\mathcal{A}'$ sets the flag bad by guessing one of the cached passwords. It follows that,

$$\Pr[\text{ bad = true in } G_3] \leq \Pr[\text{ bad = true in } G_4] + \text{Adv}^{\text{ror}}_{\text{PKE}}(\mathcal{C}_2) .$$

With this in place, we may again define an adversary $\mathcal{C}$ in the $\text{ROR}^{\mathcal{C}}_{\text{PKE}}$ game against PKEwho randomly chooses to run either $\mathcal{C}_1$ or $\mathcal{C}_2$, and outputs the same bit that they do. It follows that

$$\begin{aligned}|\Pr[G_3 \Rightarrow 1] - \Pr[G_4 \Rightarrow 1]| + \Pr[\text{ bad = true in } G_3] \\ \leq \frac{1}{2} \cdot \left(\text{Adv}^{\text{ror}}_{\text{PKE}}(\mathcal{C}_1) + \text{Adv}^{\text{ror}}_{\text{PKE}}(\mathcal{C}_2) \right) + \Pr[\text{ bad = true in } G_4] \\ \leq 2 \cdot \text{Adv}^{\text{ror}}_{\text{PKE}}(\mathcal{C}) + \Pr[\text{ bad = true in } G_4] .\end{aligned}$$

Notice that in game G_4 all values in the state s_n given to $\mathcal{A}'$ are random and independent of $\mathcal{T}$, and so the state s_n may be perfectly simulated by an adversary $\mathcal{G}$ in game $\text{OFFGUESS}^{\mathcal{G},q}_{\Pi,\mathcal{T}}$; we shall now use a reduction to this game to bound the probability that bad is set in game G_4. $\mathcal{G}$ generates a public / secret key pair $(pk, sk) \leftarrow\!\!\$\ \mathcal{K}$, assembles the remainder of s_n by choosing the appropriate random components, and passes s_n to $\mathcal{A}'$. Now each time $\mathcal{A}'$ makes a new query $(\text{sa}||\tilde{w})$ to SH such that $\text{sa} \in \{\text{sa}_0, \ldots, \text{sa}_t\}$, $\mathcal{G}$ returns a fresh random string to $\mathcal{A}'$, and submits a query of the form $(i, \tilde{w})$ to its Test oracle. Since by construction all salts are distinct, it follows that if $\mathcal{A}$ makes q queries to SH then $\mathcal{G}$ makes at most q queries to his Test oracle also. Therefore,

$$\begin{aligned}\Pr[\text{ bad = true in game } G_4] = \Pr\left[\text{OFFGUESS}^{\mathcal{G},q}_{\Pi,\mathcal{T}} \Rightarrow 1 \right] \\ \leq \text{Adv}^{\text{offguess}}_{\Pi,\mathcal{T}}(\mathcal{G}, q) .\end{aligned}$$

In game G_5 we return SH to responding truthfully to oracle queries. Since these values are no longer set during the construction of challenge state s_n, this does not affect the outcome of the game and so $|\Pr[G_4 \Rightarrow 1] = \Pr[G_5 \Rightarrow 1]|$.

Finally in game G_6 we return to sampling salts without replacement. An identical argument to that made previously implies that,

$$|\Pr[G_5 \Rightarrow 1] - \Pr[G_6 \Rightarrow 1]| \leq \frac{(t+1)^2}{2^{\ell_{\text{salt}}+1}} .$$

Now G_6 is identical to game $\overline{\text{OFFDIST}}^{\mathcal{A}',q}_{\Pi,\mathcal{T}}$ with challenge bit $b = 1$. Putting all this together, we conclude that,

$$\begin{aligned}\text{Adv}^{\text{offdist}}_{\Pi,\mathcal{T}}(\mathcal{A}) \leq \text{Adv}^{\text{offguess}}_{\Pi,\mathcal{T}}(\mathcal{G}, q) + 2 \cdot \text{Adv}^{\text{mkror}}_{\text{SE}}(\mathcal{B}, t) + \frac{(t+1)^2}{2^{\ell_{\text{salt}}}} \\ + 2 \cdot \text{Adv}^{\text{rob}}_{\text{SE}}(\mathcal{R}) + 2 \cdot \text{Adv}^{\text{ror}}_{\text{PKE}}(\mathcal{C}) + \frac{(t \cdot (n+1) + 1)^2}{2^{(\kappa-1)}} .\end{aligned}$$

A.4 Proof of Theorem 5.3

Before giving the full proof of Theorem 5.3, we begin by proving a useful lemma which we will use in subsequent analysis.

Lemma A.1. *Let $\{p_1, \ldots, p_n\}$ and $\{x_1, \ldots, x_n\}$ be sequences of numbers such that each $p_i, x_i \in [0, 1]$ and $p_1 \geq p_2 \geq \cdots \geq p_n$. Then*

$$\sum_{i=1}^{n} p_i \cdot x_i \leq \sum_{i=1}^{r} p_i \text{ where } r = \left\lceil \sum_{i=1}^{n} x_i \right\rceil .$$

Proof: Since $p_1 \geq \cdots \geq p_n$, the rearrangement inequality [13] implies that $\sum_{i=1}^{n} p_i \cdot x_i$ is maximized when the x_i are such that $x_1 \geq x_2 \geq \cdots \geq x_n$, so to upper bound this sum, we reorder them so this is the case. Notice that since $r = \left\lceil \sum_{i=1}^{n} x_i \right\rceil \geq \sum_{i=1}^{r} x_i + \sum_{i=r+1}^{n} x_i$, this implies that,

$$r - \sum_{i=1}^{r} x_i \geq \sum_{i=r+1}^{n} x_i \Rightarrow \sum_{i=1}^{r} (1 - x_i) \geq \sum_{i=r+1}^{n} x_i .$$

With this in place, it follows that

$$\sum_{i=1}^{r} p_i \cdot (1 - x_i) \geq p_r \cdot \sum_{i=1}^{r} (1 - x_i) \geq p_r \cdot \sum_{i=r+1}^{n} x_i \geq \sum_{i=r+1}^{n} p_i \cdot x_i .$$

The first inequality follows since $p_1 \geq \cdots \geq p_r$. The second inequality since $\sum_{i=1}^{r} (1 - x_i) \geq \sum_{i=r+1}^{n} x_i$. The final inequality follows since $p_r \geq \cdots \geq p_n$. Finally rearranging yields,

$$\sum_{i=1}^{r} p_i \geq \sum_{i=1}^{n} p_i \cdot x_i ,$$

as required. ∎

We now proceed to the proof of Theorem 5.3.

Proof: We wish to upper bound the maximum advantage of an attacker $\mathcal{G}$ who makes at most q queries to the Test oracle in game $\text{OFFGUESS}^{\mathcal{G},q}_{\Pi,\mathcal{T}}$. Let the sequence of guesses made by $\mathcal{G}$ be denoted $G = \{(\tilde{w}_1, j_1), (\tilde{w}_2, j_2), \ldots, (\tilde{w}_q, j_q)\}$, and recall that if $j_i = 0$ then this corresponds to a guess at the value of the real password; otherwise the guess represents a guess at the typo stored at position $0 < j_i \leq t$ in the cache. Without loss of generality, we may assume that $\mathcal{G}$ never repeats a query, since this would decrease his success probability. We split the guesses into two sets Z_0 and Z_1 where,

$$Z_0 = \{(\tilde{w}_i, j_i) \mid j_i = 0\} \text{ and } Z_1 = \{(\tilde{w}_i, j_i) \mid 0 < j_i \leq t\} .$$

We let $q_0 = |Z_0|$ and $q_1 = |Z_1|$, so $q_1 \leq q - q_0$. We let $T[j]$ denote the distribution of the typo at the j^{th} position in the cache. Notice that the adversary $\mathcal{G}$ will succeed if either the real password $T[0]$ lies in the set Z_0, or $T[j_i] = \tilde{w}_i$ for some $(\tilde{w}_i, j_i) \in Z_1$. It follows that

$$\begin{aligned}\text{Adv}^{\text{offguess}}_{\Pi,\mathcal{T}}(\mathcal{G}, q) &= \Pr[T[0] \in Z_0 \ \vee\ \exists(\tilde{w}_i, j_i) \in Z_1 \ :\ T[j_i] = \tilde{w}_i] \\ &= \Pr[T[0] \in Z_0] + \Pr[T[0] \notin Z_0 \wedge \exists(\tilde{w}_i, j_i) \in Z_1 \ :\ T[j_i] = \tilde{w}_i] .\end{aligned}$$

We may rewrite the above expression

$$\begin{aligned}&\text{Adv}^{\text{offguess}}_{\Pi,\mathcal{T}}(\mathcal{G}, q) - \Pr[T[0] \in Z_0] \\ &= \sum_{w \in \mathcal{W} \setminus Z_0} \Pr[\exists(\tilde{w}_i, j_i) \in Z_1 \ :\ T[j_i] = \tilde{w}_i \mid T[0] = w] \cdot \Pr[T[0] = w] \\ &= \sum_{w \in \mathcal{W} \setminus Z_0} \Pr\Big[\bigvee_{(\tilde{w}_i, j_i) \in Z_1} T[j_i] = \tilde{w}_i \mid T[0] = w \Big] \cdot \Pr[T[0] = w] .\end{aligned}$$

For each $w \in \mathcal{W}$, $\Pr\left[\bigvee_{(\tilde{w}_i, j_i)} T[j_i] = \tilde{w}_i \mid T[0] = w \right] \in [0, 1]$, and so an application of Lemma A.1 implies that

$$\begin{aligned}\text{Adv}^{\text{offguess}}_{\Pi,\mathcal{T}}(\mathcal{G}, q) &\leq \sum_{w \in Z_0} \Pr[T[0] = w] + \sum_{w \in Z_1^*} \Pr[T[0] = w] \\ &\leq \sum_{i=1}^{q_0 + \lceil q_1' \rceil} \Pr[T[0] = w_i] ,\end{aligned}$$

where Z_1^* is the set of the q_1' heaviest passwords in $\mathcal{W} \setminus Z_0$ and,

$$q_1' = \left\lceil \sum_{w \in \mathcal{W} \setminus Z_0} \Pr\Big[\bigvee_{(\tilde{w}_i, j_i) \in Z_1} T[j_i] = \tilde{w}_i \mid T[0] = w \Big] \right\rceil .$$

We now upper bound q_1'. Since by assumption the error setting is t-sparse, it holds that $b_{\tilde{\tau}}(\tilde{w}) \leq t$ for all $\tilde{w} \in \mathcal{M}$. It follows that

$$\begin{aligned} q_1' &\leq \sum_{w \in \mathcal{W} \setminus Z_0} \Pr\Big[\bigvee_{(\tilde{w}_i, j_i) \in Z_1} T[j_i] = \tilde{w}_i \mid T[0] = w \Big] \\ &\leq \sum_{w \in \mathcal{W}} \Pr\Big[\bigvee_{(\tilde{w}_i, j_i) \in Z_1} T[j_i] = \tilde{w}_i \mid T[0] = w \Big] \\ &\leq \sum_{w \in \mathcal{W}} \sum_{(\tilde{w}_i, j_i) \in Z_1} \Pr[T[j_i] = \tilde{w}_i \mid T[0] = w] \\ &= \sum_{w \in \mathcal{W}} \sum_{(\tilde{w}_i, j_i) \in Z_1} \frac{1}{t} \cdot \tilde{\tau}_w(\tilde{w}_i) \\ &= \sum_{(\tilde{w}_i, j_i) \in Z_1} \frac{1}{t} \cdot b_{\tilde{\tau}}(\tilde{w}_i) \quad \leq q - q_0 \ . \end{aligned}$$

The first inequality follows since $\mathcal{W} \setminus Z_0 \subseteq \mathcal{W}$ for any Z_0. The second inequality follows by taking a union bound over the points in Z_1. The next equality follows because the typo cache elements are distinct and randomly permuted, so $\Pr[T[j_i] = \tilde{w}_i \mid T[0] = w] = \frac{1}{t} \cdot \tilde{\tau}_w(\tilde{w})$. The next equality follows since by definition $b_{\tilde{\tau}}(\tilde{w}_i) = \sum_{w \in \mathcal{W}} \tilde{\tau}_w(\tilde{w}_i)$. The final inequality follows since $b_{\tilde{\tau}}(\tilde{w}_i) \leq t$ for all $\tilde{w}_i$, and there are at most $q - q_0$ guesses in Z_1. Putting this all together implies that,

$$q_0 + q_1' \leq q_0 + (q - q_0) = q \ ,$$

and we conclude that

$$\mathsf{Adv}^{\mathsf{offguess}}_{\Pi,\mathcal{T}}(\mathcal{G}, q) \leq \sum_{i=1}^{q} p(w_i) \ . \qquad \blacksquare$$

A.5 Online Security

Following from the discussion in Section 5.3, we now detail the security analysis of TypTop in the online setting.

We define online security via the game ONGUESS depicted in Figure 11, adapting the corresponding notion formulated by Chatterjee et al. in [8] to the adaptive checking setting, with the advantage defined

$$\mathsf{Adv}^{\mathsf{onguess}}_{\Pi,\mathcal{T}}(\mathcal{A}, q) = \Pr\Big[\mathrm{ONGUESS}^{\mathcal{A},q}_{\Pi,\mathcal{T}} \Rightarrow \mathsf{true} \Big] .$$

We sample a password and login transcript via the transcript generator $\mathcal{T}$ and evolve the state of the adaptive checker accordingly. The attacker is given access to an oracle Test to which he may submit guesses; he succeeds if he makes a guess which is accepted by the checking algorithm Chk. The game is parameterized by q representing the number of Test queries $\mathcal{A}$ is allowed; this reflects the standard online attack countermeasure of locking an account after a certain number of incorrect guesses.

The analysis. Following the similar discussion in Section 5, we first define a game $\overline{\mathrm{ONGUESS}}$ analogous to $\overline{\mathrm{OFFGUESS}}$, in which the final cache state is generated via the plaintext checker PChecker, and the advantage is defined as

$$\mathsf{Adv}^{\overline{\mathsf{onguess}}}_{\Pi,\mathcal{T}}(\mathcal{A}, q) = \Pr\Big[\overline{\mathrm{ONGUESS}}^{\mathcal{A},q}_{\Pi,\mathcal{T}} \Rightarrow \mathsf{true} \Big] .$$

In Lemma A.2 we bound the difference between the two games for TypTop in terms of the robustness of the underlying SE scheme SE.

Lemma A.2. *Let (p, τ) be an error setting with associated transcript generator $\mathcal{T}$, and let Π = (Reg, Chk) be TypTop's password checker with associated plaintext checker* PChecker[Π]. *Let Π be implemented using the canonical PBE scheme* PBE[SH, SE] = $(\overline{\mathsf{E}}, \overline{\mathsf{D}})$ *where* SE *is a symmetric encryption scheme and* SH *is a random oracle. Then for any adversary $\mathcal{A}$ running in time T, there exist adversaries $\mathcal{A}'$, $\mathcal{R}$ such that*

$$\begin{aligned} \mathsf{Adv}^{\mathsf{onguess}}_{\Pi,\mathcal{T}}(\mathcal{A}, q) &\leq \mathsf{Adv}^{\overline{\mathsf{onguess}}}_{\Pi,\mathcal{T}}(\mathcal{A}', q) \\ &\quad + \mathsf{Adv}^{\mathrm{rob}}_{\mathsf{SE}}(\mathcal{R}) + \frac{(t \cdot (n + 1 + q) + 1 + q)^2}{2^{\kappa}} \ , \end{aligned}$$

and, moreover, $\mathcal{A}'$ runs in time $T' \approx T$. Here t denotes the size of the cache, n denotes the length of the transcript output by $\mathcal{T}$ and SE *has key space $\{0,1\}^{\kappa}$.*

Proof: We argue by a series of game hops. Let game G_0 be equivalent to game $\mathrm{ONGUESS}_{\Pi,\mathcal{T}}$, so

$$\mathsf{Adv}^{\mathsf{onguess}}_{\Pi,\mathcal{T}}(\mathcal{A}, q) = \Pr[\, \mathrm{G}_0 \Rightarrow 1 \,] \ .$$

Let game G_1 be identical to G_0 except that the keys used to compute the cached ciphertexts in state s_n, and those used for trial decryptions in response to Test queries made by $\mathcal{A}$ in the guessing stage of the game while win = false, are sampled without replacement. These games run identically unless two of the keys sampled during these phases collide. There are at most $(t \cdot (n + 1) + 1)$ such keys sampled while computing the cached ciphertexts for a transcript of length n, and at most $q \cdot (t + 1)$ keys sampled during the guessing phase (reflecting the $(t + 1)$ trial decryptions performed by Chk for each of the q Test queries made by $\mathcal{A}$). Notice that since cache updates only occur in the guessing phase if $\mathcal{A}$ guesses a string which is accepted by Chk, the cache will never update while win = false. It follows that

$$|\Pr[\, \mathrm{G}_0 \Rightarrow 1 \,] - \Pr[\, \mathrm{G}_1 \Rightarrow 1 \,]| \leq \frac{(t \cdot (n + 1 + q) + 1 + q)^2}{2^{\kappa+1}} \ .$$

Next we define game G_2 which is identical to G_1 except that we replace Checker[Π] with PChecker and redefine Test to perform comparisons on the plaintext typo cache output by PChecker. Since the adversary in these games never sees the internal state of the checker, not encrypting the values which lie in this state does not change the adversary's view of the game; rather the two run identically unless during the process of updating the state and the adversary's subsequent queries to Test we find two distinct keys $\mathsf{k}_1 \neq \mathsf{k}_2$ such that $\mathsf{D}_{\mathsf{k}_2}(\mathsf{E}_{\mathsf{k}_1}(sk)) \neq \perp$ where sk denotes the secret key of the PKE scheme which is encrypted under each of the cached typos. Thus the fundamental lemma of game playing implies that the gap between game G_1 and G_2 is upper bounded by the probability that this event occurs. Consider an adversary $\mathcal{R}$ in game $\mathrm{ROB}^{\mathcal{R}}_{\mathsf{SE}}$ who simply executes the game G_1, simulating SH by sampling random strings without replacement, and checking if there ever exists a typo cache ciphertext $\mathsf{E}_{\mathsf{k}_1}(sk)$ that decrypts under some subsequently sampled $\mathsf{k}_2 \neq \mathsf{k}_1$ (recall that since G_1

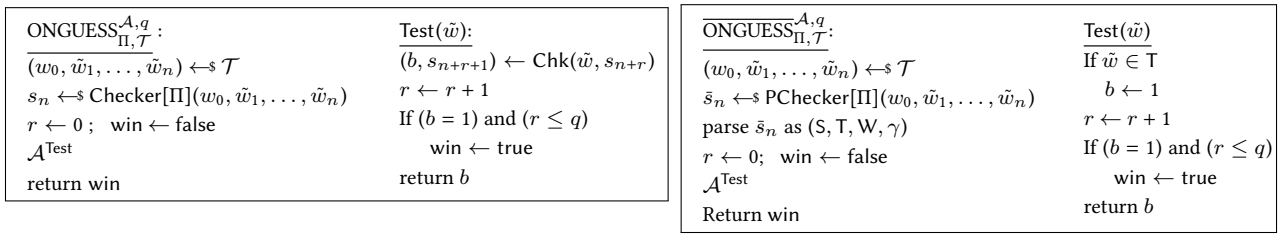


Figure 11: Security games for online attacks.

samples without replacement, all keys are distinct). The robustness of the encryption scheme implies that the probability that this event occurs, and thus the gap between games G_1 and G_2 is upper-bounded by $\text{Adv}^{\text{rob}}_{\text{SE}}(\mathcal{R})$

$$|\Pr[\,G_1 \Rightarrow 1\,] - \Pr[\,G_2 \Rightarrow 1\,]| \leq \text{Adv}^{\text{rob}}_{\text{SE}}(\mathcal{R})\ .$$

Next we define game G_3 which is identical to G_2 except we return SH to sampling keys with replacement. An analogous argument to that above bounding the probability that two keys collide ensures that

$$|\Pr[\,G_2 \Rightarrow 1\,] - \Pr[\,G_3 \Rightarrow 1\,]| \leq \frac{(t \cdot (n+1+q)+1+q)^2}{2^{\kappa+1}}\ .$$

Notice that G_3 is identical to $\overline{\text{ONGUESS}}^{\mathcal{A}}_{\Pi,\mathcal{T}}$, and so can be perfectly simulated by an adversary $\mathcal{A}'$ in this game. $\mathcal{A}'$ simulates $\mathcal{A}$'s Test oracle by submitting $\mathcal{A}$'s queries to his own oracle, and returning the responses. Since $\mathcal{A}'$ makes precisely the same set of queries as $\mathcal{A}$, it follows that if $\mathcal{A}$ makes at most q queries, then $\mathcal{A}'$ does also. Since both games are identically distributed, it follows that

$$\text{Adv}^{\overline{\text{onguess}}}_{\Pi,\mathcal{T}}(\mathcal{A}', q) = \Pr[\,G_3 \Rightarrow 1\,]\ ,$$

concluding the proof. ∎

Online guessing advantage. It remains to bound $\text{Adv}^{\overline{\text{onguess}}}_{\Pi,\mathcal{T}}(\mathcal{A}, q)$. The key difference between the online guessing game and its offline counterpart is that in the former each guess is tested for equality against each of the $t+1$ positions in the cache, whereas in the latter a guess is only checked against the specific slot to which it was guessed.

We reduce the online guessing game — in which the attacker's goal is to find q guesses that maximizes its success probability — to a weighted maximum coverage problem, and use an approximate greedy algorithm to compute the attacker's advantage. We can then bound the advantage of the optimal attacker using the classic result of [15]. However, due to the complex dependencies of the cached elements, we could not show the reduction in the other direction: that is to say, reduce an NP-complete problem to that of finding the optimal set of guesses in the online guessing game. We strongly believe that this problem is NP-hard but leave the detailed reduction as an open problem.

Approximation via greedy algorithm. Recall that the maximum coverage problem is defined as follows. Given n subsets S_i from a universe U, the goal is to find k subsets that cover the maximum number of elements. In the weighted version of this problem, every element in U is weighted, and the goal is to maximize the sum total weight of the covered elements.

We reduce the online guessing game $\overline{\text{ONGUESS}}$ for a particular error setting (p, τ) and plaintext checker PChecker[Π] to a weighted maximum coverage problem as follows. We define the universe U to be the set of all possible cache-tuples, where a cache-tuple consists of a password $w \in \mathcal{W}$ followed by at most t distinct and alphabetically sorted typos $\tilde{w}_i \in \mathcal{S}$. The weight of a given cache-tuple is defined to be the probability that this tuple lies in the cache of the state $\bar{s}_n$ that the attacker guesses against in game $\overline{\text{ONGUESS}}$. For each password or typo $\tilde{w}$, we define $S_{\tilde{w}} \subseteq U$ to be the set of all cache-tuples that contain $\tilde{w}$. Given all such subsets, the attacker's goal is to find q subsets so that the sum total of the covered cached-tuples is maximized.

With this reduction in place, we can apply the greedy approximation algorithm for finding the weighted maximum coverage.

Empirical analysis. We wish to compute the advantage of an adversary in the online guessing game for real world error settings. While it is easy to describe the reduction to a weighted maximum coverage problem, generating the universe of cache-tuples and the corresponding subsets for large numbers of passwords and typos is computationally very expensive. For example, there could be more than a billion cache-tuples for a password with 100 typos and cache size $t = 5$, and finding all such cache-tuples for a large number of passwords seems infeasible.

We therefore perform the simulation on a subset of k passwords from RockYou in the following way. For each real password w, we sample m typos from τ_w with replacement, run the plaintext checker PChecker[Π] on the sampled list, and record the final cache-tuple. We repeat this process n times for each password, and record all the unique cache-tuples with their weight set to $p(w) \cdot f/n$, where f is the number of times the cache-tuple was observed. We set the universe U to be the set of all cache-tuples we collected in the above experiment, and for each string $\tilde{w}$, subset $S_{\tilde{w}}$ is defined as the set of all cache-tuples from U which contain $\tilde{w}$. The attacker's goal is to find a set of q strings $\tilde{w}$ such that the cumulative weight of the elements covered by their subsets is maximized.

The greedy algorithm to find the weighted maximum cover works as follows: find the subset $S_{\tilde{w}^*}$ that has the highest cumulative weight, add the corresponding string $\tilde{w}^*$ to the list of guesses, remove all occurrences of cache-tuples in $S_{\tilde{w}^*}$ from other subsets, and repeat until q guesses are found or all subsets are empty.

We wish to compute the security loss incurred by using TypTop compared to an exact checker. Recall that λ_q denotes the success probability of an optimal attack against an exact checker with a budget of q guesses, and that $\lambda_q = \sum_{i=1}^{q} p(w_i)$. We define the security loss of a checker Π over the exact checker to be

$$\Delta_q = \text{Adv}^{\overline{\text{onguess}}}_{\Pi,\mathcal{T}}(\mathcal{A}, q) - \lambda_q\ .$$

Using k most frequent passwords from RockYou, we ran the above simulation with $m = 200$, $n = 500$. We chose $k = 10^5$, and for each caching policy we compute the greedy attacker's advantage for $q = 100$. For all caching policies the security loss Δ_q is minimal, with a maximum security loss of $\Delta_{100} = 0.001$ for the MFU caching policy, and less than 0.0006 for all other caching policies. We also tried sampling passwords randomly from the support of the password distribution, and taking the k most frequent passwords in RockYou after ignoring the first million passwords. The security loss is even less in such samples as we observed in the offline scenario in Section 5.2. To see the effect of n on the final security loss, we also ran the experiment with $n = 1000$ for the PLFU caching strategy. We found negligible change in the security loss.

By the result of Hochbaum [15], we know that the output of the greedy algorithm is no less than $1 - 1/e$ times that of the optimal algorithm. If we include this adjustment into the output of our greedy approximation algorithm, we get $\Delta_q \leq \frac{e}{e-1} \cdot \mathsf{Adv}^{\overline{\text{onguess}}}_{\Pi,\mathcal{T}}(\mathcal{A}, q) - \lambda_q$. Therefore, the security loss due to TypTop is at most $\Delta_{100} \leq 1.582 \times 0.0456 - 0.045 = 0.027$.

This bound is pessimistic. It assumes the attacker has precise knowledge of the typo distribution. Moreover, the final bound is looser if the greedy approximation results are closer to the optimal—which we believe to be the case.

We might be able to use a blacklisting strategy similar to the one proposed in [8] to further reduce the security loss. A naive blacklisting approach would be to block a set of 'risky' typos (that is to say those which allow an attacker to achieve too great an advantage) from entering the typo-cache. However to decide which typos to blacklist, we need an accurate measure of the cache inclusion function, which will itself change each time a new typo is blacklisted, significantly complicating the analysis of this approach. We leave a detailed treatment of blacklisting strategies for future work.

REFERENCES

[1] Michel Abdalla, Mihir Bellare, and Gregory Neven. 2010. Robust encryption. *Journal of Cryptology* (2010), 1–44.

[2] S. Antilla. 2015. Vanguard group fires whistleblower who told thestreet about flaws in customer security. (2015).

[3] Mihir Bellare and Phillip Rogaway. 2006. Code-based game-playing proofs and the security of triple encryption. In *Advances in Cryptology–EUROCRYPT*, Vol. 4004. 10.

[4] Patrick Biernacki and Dan Waldorf. 1981. Snowball sampling: Problems and techniques of chain referral sampling. *Sociological methods & research* 10, 2 (1981), 141–163.

[5] Alex Biryukov, D Dinu, and D Khovratovich. 2015. *Argon and argon2: password hashing scheme.* Technical Report. Technical report.

[6] Joseph Bonneau. 2012. The science of guessing: analyzing an anonymized corpus of 70 million passwords. In *IEEE Symposium on Security and Privacy (SP)*. IEEE, 538–552.

[7] Michael Buhrmester, Tracy Kwang, and Samuel D Gosling. 2011. Amazon's Mechanical Turk a new source of inexpensive, yet high-quality, data? *Perspectives on psychological science* 6, 1 (2011), 3–5.

[8] Rahul Chatterjee, Anish Athalye, Devdatta Akhawe, Ari Juels, and Thomas Ristenpart. 2016. pASSWORD tYPOS and How to Correct Them Securely. *IEEE Symposium on Security and Privacy* (may 2016). Full version of the paper can be found at the authors' website.

[9] Fred J Damerau. 1964. A technique for computer detection and correction of spelling errors. *Commun. ACM* 7, 3 (1964), 171–176.

[10] Pooya Farshim, Benoît Libert, Kenneth G Paterson, Elizabeth A Quaglia, and others. 2013. Robust Encryption, Revisited.. In *Public Key Cryptography*, Vol. 7778. Springer, 352–368.

[11] Pooya Farshim, Claudio Orlandi, and Razvan Rosie. 2017. Security of Symmetric Primitives under Incorrect Usage of Keys. *IACR Transactions on Symmetric Cryptology* 2017, 1 (2017), 449–473.

[12] Dinei Florencio and Cormac Herley. 2007. A Large-scale Study of Web Password Habits. In *Proceedings of the 16th International Conference on World Wide Web (WWW '07)*. ACM, New York, NY, USA, 657–666. https://doi.org/10.1145/1242572.1242661

[13] Godfrey Harold Hardy, John Edensor Littlewood, and George Pólya. 1952. *Inequalities*. Cambridge university press.

[14] Andreas W Hauser and Klaus U Schulz. 2007. Unsupervised learning of edit distance weights for retrieving historical spelling variations. In *Proceedings of the First Workshop on Finite-State Techniques and Approximate Search*. 1–6.

[15] Dorit S Hochbaum. 1996. Approximating covering and packing problems: set cover, vertex cover, independent set, and related problems. In *Approximation algorithms for NP-hard problems*. PWS Publishing Co., 94–143.

[16] Burt Kaliski. 2000. PKCS #5: Password-based cryptography specification version 2.0. (2000). RFC 2289.

[17] Mark Keith, Benjamin Shao, and Paul Steinbart. 2009. A behavioral analysis of passphrase design and effectiveness. *Journal of the Association for Information Systems* 10, 2 (2009), 2.

[18] Mark Keith, Benjamin Shao, and Paul John Steinbart. 2007. The usability of passphrases for authentication: An empirical field study. *International journal of human-computer studies* 65, 1 (2007), 17–28.

[19] Saranga Komanduri, Richard Shay, Patrick Gage Kelley, Michelle L. Mazurek, Lujo Bauer, Nicolas Christin, Lorrie Faith Cranor, and Serge Egelman. 2011. Of passwords and people: measuring the effect of password-composition policies. In *CHI*.

[20] Vladimir I Levenshtein. 1966. Binary codes capable of correcting deletions, insertions, and reversals. In *Soviet physics doklady*, Vol. 10. 707–710.

[21] Michelle L Mazurek, Saranga Komanduri, Timothy Vidas, Lujo Bauer, Nicolas Christin, Lorrie Faith Cranor, Patrick Gage Kelley, Richard Shay, and Blase Ur. 2013. Measuring password guessability for an entire university. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*. ACM, 173–186.

[22] William Melicher, Blase Ur, Sean M Segreti, Saranga Komanduri, Lujo Bauer, Nicolas Christin, and Lorrie Faith Cranor. Fast, lean and accurate: Modeling password guessability using neural networks.

[23] Payman Mohassel. 2010. A closer look at anonymity and robustness in encryption schemes. In *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 501–518.

[24] R. Morris and K. Thompson. 1979. Password security: a case history. *Commun. ACM* 22, 11 (Nov. 1979), 594–597. https://doi.org/10.1145/359168.359172

[25] Alec Muffet. 2015. Facebook: Password Hashing & Authentication. Presentation at Real World Crypto. (2015).

[26] Randall Munroe. 2015. Password Strength. https://xkcd.com/936/. (2015). Accessed: 2015-11-13.

[27] Colin Percival and Simon Josefsson. 2015. The scrypt Password-Based Key Derivation Function. (2015).

[28] Emil Protalinski. 2015. Facebook passwords are not case sensitive. http://www.zdnet.com/article/facebook-passwords-are-not-case-sensitive-update/. (2015). Accessed: 2015-11-12.

[29] Kenneth Raeburn. 2005. Advanced encryption standard (AES) encryption for Kerberos 5. (2005).

[30] Vipin Samar. 1996. Unified login with pluggable authentication modules (PAM). In *Proceedings of the 3rd ACM conference on Computer and communications security*. ACM, 1–10.

[31] Richard Shay, Patrick Gage Kelley, Saranga Komanduri, Michelle L Mazurek, Blase Ur, Timothy Vidas, Lujo Bauer, Nicolas Christin, and Lorrie Faith Cranor. 2012. Correct horse battery staple: Exploring the usability of system-assigned passphrases. In *Proceedings of the Eighth Symposium on Usable Privacy and Security*. ACM, 7.

[32] Richard Shay, Saranga Komanduri, Adam L Durity, Phillip Seyoung Huh, Michelle L Mazurek, Sean M Segreti, Blase Ur, Lujo Bauer, Nicolas Christin, and Lorrie Faith Cranor. 2014. Can long passwords be secure and usable?. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 2927–2936.

[33] M. Siegler. 14 Dec. 2009. One of the 32 million with a RockYou account? you may want to change all your passwords. like now. *TechCrunch* (14 Dec. 2009).

[34] Blase Ur, Fumiko Noma, Jonathan Bees, Sean M Segreti, Richard Shay, Lujo Bauer, Nicolas Christin, and Lorrie Faith Cranor. 2015. "I Added '!' at the End to Make It Secure": Observing Password Creation in the Lab. In *Eleventh Symposium On Usable Privacy and Security (SOUPS 2015)*. 123–140.

[35] Dan Lowe Wheeler. 2016. zxcvbn: Low-budget password strength estimation. In *Proc. USENIX Security*.

[36] Frank Wilcoxon, SK Katti, and Roberta A Wilcox. 1963. *Critical values and probability levels for the Wilcoxon rank sum test and the Wilcoxon signed rank test.* American Cyanamid Company.