Detecting Stuffing of a User's Credentials at Her Own Accounts

Ke Coby Wang and Michael K. Reiter, *University of North Carolina at Chapel Hill*

https://www.usenix.org/conference/usenixsecurity20/presentation/wang

This paper is included in the Proceedings of the
29th USENIX Security Symposium.

August 12–14, 2020

978-1-939133-17-5

Open access to the Proceedings of the
29th USENIX Security Symposium
is sponsored by USENIX.

Detecting Stuffing of a User's Credentials at Her Own Accounts

Ke Coby Wang
Department of Computer Science
University of North Carolina at Chapel Hill

Michael K. Reiter
Department of Computer Science
University of North Carolina at Chapel Hill

Abstract

We propose a framework by which websites can coordinate to detect credential stuffing on individual user accounts. Our detection algorithm teases apart normal login behavior (involving password reuse, entering correct passwords into the wrong sites, etc.) from credential stuffing, by leveraging modern anomaly detection and carefully tracking suspicious logins. Websites coordinate using a novel private membership-test protocol, thereby ensuring that information about passwords is not leaked; this protocol is highly scalable, partly due to its use of cuckoo filters, and is more secure than similarly scalable alternatives in an important measure that we define. We use probabilistic model checking to estimate our credential-stuffing detection accuracy across a range of operating points. These methods might be of independent interest for their novel application of formal methods to estimate the usability impacts of our design. We show that even a minimal-infrastructure deployment of our framework should already support the combined login load experienced by the airline, hotel, retail, and consumer banking industries in the U.S.

1 Introduction

In the past decade, massive numbers of website account credentials have been compromised via password database breaches, phishing, and keylogging. According to a report by Shape Security [58],[1] 2.3 billion credentials were reported compromised in 2017 alone. Such compromised username-password pairs place those users' *other* accounts in jeopardy, since people tend to reuse their passwords across different websites (e.g., [13, 31, 38, 49, 59]). As such, automatically attempting leaked username-password pairs at a wide array of sites compromises vast numbers of accounts, a type of attack termed *credential stuffing*. Credential stuffing is now a dominant method of account takeover [58] and is remarkably commonplace; e.g., Akamai estimates it observed 30 billion credential-stuffing attempts in 2018 [1]. Credential stuffing imposes *actual* losses estimated at $300M, $400M, $1.7B and $6B on the hotel, airline, consumer banking, and retail industries, respectively, per year [58, Table 2]. A survey of 538 IT security practitioners who are responsible for the security of their companies' websites revealed a total annualized cost of credential stuffing across their organizations, excluding fraud, of $3.85M, owing to costs of prevention, detection, and remediation; downtime; and customer churn [51, Tables 1–3].

Despite the prominence of credential stuffing, users are remarkably resistant to taking steps to defend themselves against it. Thomas et al. [64] report that less than 3.1% of users who suffer account hijacks enable two-factor authentication after recovering their accounts. Users are similarly resistant to stopping password reuse even despite specific warnings when doing so, leading Golla et al. to conclude that "notifications alone appear insufficient in solving password reuse" [33]. And though password managers would seem to enable users to more easily avoid password reuse, users are reluctant to adopt them. In a 2016 survey of 1040 American adults, only 12% reported ever using password management software, and only 3% said this is the password technique they rely on most [60]. In a 2019 Google/Harris Poll survey of 3000 U.S. adults, still only 24% reported using a password manager [34].

Conceding that the factors that enable credential stuffing to succeed today are likely to persist for the foreseeable future, we propose a framework by which websites could cooperate to detect active credential-stuffing attacks on a per-user basis. Developing such a framework is not straightforward, in part because the exact behaviors that such a framework should detect are difficult to define. Anecdotally, users sometimes engage in behaviors that might appear quite similar to a credential-stuffing attack, e.g., submitting the same small handful of passwords to multiple sites in the course of logging into each, if she is unsure of which password she set at which site. A framework to detect credential stuffing on a user's accounts will need to tease apart behaviors that the user might normally undertake from actual credential abuse.

[1] We recognize that this and other reports produced by companies that market credential-stuffing defenses might exaggerate the risks or costs of credential stuffing. We are unaware of more objective sources with which to corroborate or refute their claims, however.

To do so, our framework leverages the following technique. Anomaly detection systems (ADS) now exist by which a site can differentiate login attempts by the legitimate user from those by attackers, even sophisticated ones, with moderately good accuracy, using features *other* than the password entered (e.g., [29]). A site in our framework leverages this capability to track *suspicious* login attempts locally, namely abnormal attempts (per the ADS) using an incorrect password or, for sites requiring second-factor authentication for abnormal login attempts, such attempts for which the second-factor authentication fails (even if the password is correct). Then, our framework enables a site (the *requester*) receiving a login attempt that it deems abnormal to query other sites (the *responders*) where this user has accounts, to determine the number of them at which this same password has been submitted in suspicious login attempts. If this number is larger than a threshold, then the requester deems this login attempt to be credential abuse—even if the password is correct.

Of course, such an approach raises concerns. First, it risks false detections, and lacking datasets of how legitimate users submit login attempts—both correct and incorrect ones—across their many accounts, the false detection rate seems hopeless to evaluate. Second, measuring the true detection rate of this scheme would require knowledge of how attackers conduct credential-stuffing attacks today (again, we are aware of no such datasets) and, more importantly, how attackers would respond if our framework were deployed by a collection of websites. Finally, since both the requester's query and a responder's suspicious-password set will contain sensitive passwords, supporting these queries has the potential to leak sensitive data to the requester or responder.

We address these concerns as follows. To estimate the true and false detection rates of our design, we formulate experiments in the form of Markov decision processes (MDPs), in which the adversary's choices in the experiment determine a probability of the adversary achieving a specified goal in our framework. In the true-detection-rate MDP, the adversary corresponds to a credential stuffer, and we leverage probabilistic model checking to calculate the true detection rate for the *best* adversarial strategy, yielding what we believe is a conservative estimate of our true detection rate in practice. The false-detection-rate MDP casts the "adversary" as the legitimate user who knows *how* she chooses her passwords (i.e., the distribution) but who cannot recall which one she set at which website. Again, we allow the "adversary" (forgetful user) arbitrary flexibility to submit login attempts, toward the "goal" of ensuring that she will be detected as a credential stuffer when eventually entering her correct password at a designated website. We use probabilistic model checking to find the *best* strategy for this "adversary", which we believe serves as a conservative estimate of our false-detection rate.

To protect passwords while allowing queries to suspicious-password sets, we develop a new private membership-test (PMT) protocol that ensures that responders do not learn the requester's query or the protocol result (no matter how they misbehave) and that limits the information about the responder's suspicious-password set that is leaked to the requester. We quantify the suspicious-password-set leakage in terms of a measure we call *extraction complexity*, which informally is the number of protocol runs a responder can tolerate before succumbing to an *offline* attack on its set. We show that our protocol improves over previous communication-efficient PMT protocols substantially in this measure.

Finally, we present an implementation of our framework by which a requester leverages a *directory* to contact responders where a user holds accounts. We evaluate performance of our design in two privacy contexts, one where the directory is trusted to hide the requester (i.e., where the user is currently active) from responders, and one where it is not and so the requester contacts the directory using Tor [20]. We show that even with just one directory machine, various configurations of our design can already support the typical login load experienced by the airline, hotel, retail, and consumer banking industries in the U.S., combined.

To summarize, our contributions are as follows:

- We develop a framework by which websites can coordinate to detect active credential stuffing on a user's accounts, and we estimate the true and false detection rates of this algorithm using probabilistic model checking (Secs. 3–4).
- We instantiate this framework with a new PMT protocol that ensures security against a malicious requester or responder, including improving on other communication-efficient designs in a security measure (*extraction complexity*) that is important in our context (Sec. 5).
- We report the performance of an implementation of our framework under two privacy configurations, in experiments ranging up to 256 responders (Sec. 6). Our results indicate that even with minimal infrastructure, our design should scale to accommodate real login loads experienced by major sectors of the U.S. economy.

2 Related Work

Interfering with password reuse A user's reuse of the same passwords across her accounts is the impetus for credential stuffing. Password reuse is widespread (e.g., [7, 13, 38, 49, 53, 59, 67]) and is very resistant to warnings to avoid it—even reactive warnings triggered by a detected reuse [33]. Most closely related to our work is a recent proposal by which websites could coordinate to actively interfere with a user's attempt to reuse the same or similar passwords across those sites [69]. While we borrow ingredients of this design (see Sec. 3.1 and Sec. 4), our work targets credential stuffing directly, *without* interfering with a user's password reuse across sites or assuming that it does not occur. These different goals lead to a fundamentally different design, requiring novel underlying cryptographic protocols (Sec. 5) and wholly novel detection algorithms (Sec. 3).

Detecting user selection of compromised or popular passwords It is now common (and recommended [35]) for sites to cross-reference account passwords against known-leaked passwords, either for their own users (e.g., [10]) or as a service for others (e.g., https://haveibeenpwned.com). Thomas et al. [65] and Li et al. [43] proposed improvements to these types of services that leak less information to or from the service. Pal et al. [48] developed personalized password strength meters that warn users when selecting passwords similar to ones previously compromised, particularly their own. More distantly related are services that track password popularity and enable a website to detect if one of its users selects a popular password [45, 56]. In contrast to these works, our work detects credential stuffing of a user's password before its compromise is reported (which today takes an *average* of 15 months [58]) and irrespective of its popularity.

Discovering compromised accounts Several techniques have been proposed to discover compromised accounts. For example, to detect the breach of its password database, a site might list several site-generated *honey passwords* in the database alongside the valid password for each account [5, 24, 39]. Any submission of a honey password in a login attempt then discloses the breach of the password database. Similarly, *honey accounts* for a user can be set up at websites where she does not have an account, specifically for detecting any attempt to log into them with the password of one of her actual accounts [17]. Both of these can be used in conjunction with our framework but do not supersede it, as attackers holding a user's correct password for one account (e.g., as obtained from phishing the user) can use it to attempt logins at the user's actual accounts at other websites without either of these techniques detecting it. This is precisely the type of attack that we seek to detect here.

Detecting guessing attacks Herley and Schechter [37] provide an algorithm by which a large-volume website can estimate the likelihood that a login is part of a guessing attack, based on the assumption that these malicious logins are a small fraction of total logins. They point out that this assumption "is of course not true for an attacker exploiting password re-use or other non-guessing approach", which is our interest here. Schechter et al. [57] suggest features for distinguishing benign login attempts from guessing attacks, though these features should not be characteristic of credential stuffing.

3 Detecting Credential Stuffing

In this section, we present our framework to detect credential stuffing on a user's accounts. Our framework detects credential stuffing on a per-user basis, and so is agnostic to whether the user is the only one being subjected to credential stuffing (e.g., after one of her passwords was phished) or whether she is one of many (e.g., after a password database breach).

3.1 Assumptions

Account identifiers Our framework assumes the ability to associate the accounts of the same user across different websites—or more specifically, to do so at least as well as a credential-stuffing attacker could. When user accounts are tied to email addresses confirmed during account creation, this can generally be done, even despite the email-address variations for a single email account that are supported by email service providers [69]. As such, we will generally refer to a user's account identifier a as being the same across multiple websites. We also assume the ability of one website at which a user has an account to contact (perhaps anonymously) other websites where the user has an account. In our design, this ability is supported using a logical *directory* service, as will be detailed in Sec. 4.

Password management *Nothing in our framework requires that a site store passwords in the clear or in a reversible fashion*, and most existing best practices (e.g., using expensive hash functions to compute password hashes [3, 61]) can be applied within our scheme. In particular, while in Sec. 3.2 we will use $s.\mathsf{pwd}[a]$ to denote the correct password for account a at site s, we stress that s need not (and should not) store this password explicitly.

We do make one concession in password-management best practices, however, similarly to some other defenses (e.g., [32, 69]). In our algorithm, each site s will maintain a set $s.\mathsf{susp}[a]$ of hashes of passwords submitted in *suspicious* login attempts on account a. (This will be detailed in Sec. 3.2.) For one site s (the *requester*) to query whether a hash value e is present in $s'.\mathsf{susp}[a]$ at another site s' (the *responder*), it is necessary that any value that s uses to salt e be the same as the value that s' uses to salt the elements of $s'.\mathsf{susp}[a]$. To do so, the salt corresponding to identifier a could be generated deterministically from a or generated randomly and distributed to a site s by the directory when s registers as a responder for a (see Sec. 4.2). Below, we elide these details and simply write "$\pi \in s'.\mathsf{susp}[a]$" to denote the membership of a hash e in the set $s'.\mathsf{susp}[a]$, where e uniquely identifies the password π from which it was computed but that also incorporates this salting.

While less secure than per-site, per-account salting, an attacker's precomputation (before breaching s') to aid his search for the passwords whose hashes are contained in $s'.\mathsf{susp}[a]$ would need to be repeated per identifier a. Moreover, the need for consistent salting per account a across sites pertains only to the storage and querying of $s.\mathsf{susp}[a]$ sets, and not to the hashing of $s.\mathsf{pwd}[a]$ for the purposes of checking the correctness of a password entered in a login attempt. That is, login attempts can be checked against a hash of $s.\mathsf{pwd}[a]$ that

is salted with a per-site, per-account salt value. As we will discuss in Sec. 3.2, while a hash of $s.\mathsf{pwd}[a]$ might end up in $s.\mathsf{susp}[a]$ in certain cases, we do not expect this to be the common case. Consequently, we believe that resorting to consistent salting across sites for the management of $s.\mathsf{susp}[a]$ sets per account a is a small concession to make.

Anomaly detection The key assumption we make about sites that participate in our design is that each one conducts anomaly detection on the login attempts to its site, based on locally available features such as the time, client IP address, useragent string, etc. For a 10% false-detection rate, Freeman et al. [29] report a 99% true detection rate for attacker logins to an account a from the country from which the user for account a normally logs in (a so-called *researching* attacker). Also for a 10% false-detection rate, they report a 74% true-detection rate for the most advanced attackers they consider, who also initiate login attempts with the same useragent string as the legitimate user (a so-called *phishing* attacker).

Here we treat each site's anomaly-detection system (ADS) as a block-box that takes as input a group of login features and classifies the login as either normal or abnormal. We assume that the ADS can be parameterized (e.g., with a threshold) to tune its true- and false-detection rates, where a "detection" means an abnormal classification. Our credential-stuffing detection algorithm will leverage two parameter settings (e.g., thresholds) for the ADS, yielding for each login attempt L an ADS output of the form $\langle \mathsf{abnormal}^{\mathsf{col}}(L), \mathsf{abnormal}^{\mathsf{cnt}}(L)\rangle$, a pair of boolean values. The "col" and "cnt" qualifiers denote the "collecting" and "counting" phases of our algorithm, respectively, which will be explained in Sec. 3.2. The flexibility provided by allowing different ADS parameter settings in the two phases of our algorithm is important to permit algorithm tuning; see Sec. 3.3. Because $\mathsf{abnormal}^{\mathsf{col}}(L)$ and $\mathsf{abnormal}^{\mathsf{cnt}}(L)$ are derived from a common ADS, these indicators are not independent. We denote their (false-, true-) detection rates as $(\mathsf{FDR}^{\mathsf{col}}_{\mathsf{ads}}, \mathsf{TDR}^{\mathsf{col}}_{\mathsf{ads}})$ and $(\mathsf{FDR}^{\mathsf{cnt}}_{\mathsf{ads}}, \mathsf{TDR}^{\mathsf{cnt}}_{\mathsf{ads}})$, respectively, and assume that either $\mathsf{TDR}^{\mathsf{col}}_{\mathsf{ads}} \geq \mathsf{TDR}^{\mathsf{cnt}}_{\mathsf{ads}}$ and $\mathsf{FDR}^{\mathsf{col}}_{\mathsf{ads}} \geq \mathsf{FDR}^{\mathsf{cnt}}_{\mathsf{ads}}$ or $\mathsf{TDR}^{\mathsf{cnt}}_{\mathsf{ads}} \geq \mathsf{TDR}^{\mathsf{col}}_{\mathsf{ads}}$ and $\mathsf{FDR}^{\mathsf{cnt}}_{\mathsf{ads}} \geq \mathsf{FDR}^{\mathsf{col}}_{\mathsf{ads}}$, as otherwise one parameter setting would be strictly better than the other. When the login L is clear from context, we will generally elide it and simply denote the ADS output as $\langle \mathsf{abnormal}^{\mathsf{col}}, \mathsf{abnormal}^{\mathsf{cnt}}\rangle$.

Threat model We specify our credential-stuffing detection algorithm in Sec. 3.2 assuming sites that cooperate to detect credential stuffing against a user's accounts at those sites. That is, the attacker can submit login attempts at participating sites, possibly using passwords it stole, but cannot participate as a site in our framework. In Secs. 4–5, however, we address the potential for *malicious* participating sites. In particular, we address user login privacy against participating sites in Sec. 4.1 and the risk of denial-of-service attacks by participating sites in Sec. 4.2. Finally, we address user account security despite misbehavior of participating sites in Sec. 5. We make no effort to address sites that misbehave so as to reduce true detections of our framework, since these sites could equally well do so by simply not participating. The directory, introduced in Sec. 4, is trusted to not conduct denials-of-service and to help defend against them (see Sec. 4.2), but is not trusted for security of sites' user accounts.

3.2 Algorithm

To detect credential stuffing, each website processes each login attempt in two *phases*, called the *collecting phase* and the *counting phase*. To support these phases, each site s maintains a set $s.\mathsf{susp}[a]$ of (salted hashes of) passwords used in "suspicious" login attempts to account a at site s, as discussed below. Site s assembles $s.\mathsf{susp}[a]$ in the *collecting phases* of local login attempts to account a, and queries the $s'.\mathsf{susp}[a]$ sets at other sites s' in the *counting phases* of logins to a at s. These queries are performed using *private membership tests* (PMTs), which hide s's query when acting as a requester, and hide the contents of $s.\mathsf{susp}[a]$ when acting as a responder in the protocol. We defer details of the PMT protocol to Sec. 5.

Our algorithm begins when a site receives a local login attempt. The site submits the login features for classification by its ADS, yielding a classification $\langle \mathsf{abnormal}^{\mathsf{col}}, \mathsf{abnormal}^{\mathsf{cnt}}\rangle$. The site then performs the *collecting phase* and the *counting phase*, in that order.

Collecting phase In the *collecting phase* of a login attempt to account a at site s, if $\mathsf{abnormal}^{\mathsf{col}} = \mathit{true}$, then s applies one of the following two rules as appropriate, where π is the submitted password:

SUSP	If s does not support second-factor authentication for a, then s adds π to $s.\mathsf{susp}[a]$ if the password is incorrect (i.e., $\pi \neq s.\mathsf{pwd}[a]$).
SUSP$^+$	If s supports second-factor authentication for a, then s adds π to $s.\mathsf{susp}[a]$. If $\pi = s.\mathsf{pwd}[a]$, then it is subsequently removed from $s.\mathsf{susp}[a]$ only once a second-factor challenge issued by s to the owner of account a is completed successfully.

Counting phase In the *counting phase* of a login attempt to account a at site s, if $\mathsf{abnormal}^{\mathsf{cnt}} = \mathit{true}$ and if the submitted password π is correct for account a (i.e., $\pi = s.\mathsf{pwd}[a]$), then s performs the role of the requester using password π in PMTs. In each of these PMTs, another site s' where account a exists performs the role of the responder with set $s'.\mathsf{susp}[a]$; i.e., s' interacts with s to allow s to learn whether $\pi \in s'.\mathsf{susp}[a]$. (In Sec. 4 we will discuss how s contacts each s'.) Site s then detects credential stuffing if $|\{s' \mid s' \neq s \wedge \pi \in s'.\mathsf{susp}[a]\}| \geq w$, for a specified *attack width* w.

Again, the *collecting phase* is performed first, or more precisely, s begins its *counting phase* only once any addition to $s.\mathsf{susp}[a]$ in the preceding *collecting phase* is complete. Similarly, upon receiving a PMT query from s, site s' defers responding until any additions to $s'.\mathsf{susp}[a]$ in ongoing *collecting phases* for the same account a are completed locally.

Note that if SUSP would add π to $s.\mathsf{susp}[a]$, then SUSP$^+$ would, as well. In addition, SUSP$^+$ allows even $s.\mathsf{pwd}[a]$ to be added to $s.\mathsf{susp}[a]$ if it is submitted in a login that is deemed abnormal but the resulting second-factor challenge is not completed successfully. We do not expect this to be the norm, however: A rough estimate assuming $\mathsf{FDR}^{\mathsf{col}}_{\mathsf{ads}} = 0.10$ and a second-factor failure rate by the correct user of 0.12 (e.g., see [21]) is that a legitimate login attempt with the correct password at a site s supporting SUSP$^+$ leaves that password in $s.\mathsf{susp}[a]$ with probability only 0.012.

When evaluating SUSP$^+$, we assume that an attacker is unable to complete the second-factor challenge (which is generally true [21]), but that for usability purposes, the site invokes the second-factor challenge only on logins for which $\mathsf{abnormal}^{\mathsf{col}} = \mathit{true}$. Some sites s can maintain $s.\mathsf{susp}[a]$ according to SUSP while others use SUSP$^+$. In our evaluations in Sec. 3.3, we will consider the impact of different balances of sites that use SUSP versus SUSP$^+$.

SUSP and SUSP$^+$ indicate when s should *add* a password to $s.\mathsf{susp}[a]$, but not when s should *remove* a password from it. One approach would be for s to remove a password from $s.\mathsf{susp}[a]$ if that password is not used in an attempted login to account a for a specified *expiration time*. Provided that s's login interface rate-limits login attempts on a (as is recommended [35]), an upper bound on the capacity of s's set $s.\mathsf{susp}[a]$ can be ensured. For example, if s permits 100 failed login attempts on a single account in any 30-day period [8, Section 8.2.3], and if each password expires from $s.\mathsf{susp}[a]$ in 30 days since its last use in a login attempt, then $|s.\mathsf{susp}[a]|$ will never exceed 101. Such a delay should allow ample time for our framework to detect even a moderately aggressive credential-stuffing attack, or conversely should dramatically slow a credential-stuffing attack if it is to go undetected.

Finally, when adding a password π to $s.\mathsf{susp}[a]$, s may reduce π to a canonical form, e.g., converting capital letters at selected positions to lowercase, or converting a specific digit to a digit wildcard. Provided that the rules for this canonicalization are employed by both requesters and responders, our framework can then detect stuffing of some passwords *similar* to that chosen by this user at another site ("credential tweaking"). Of course, s could also explicitly add selected passwords similar to π, but at the cost of increasing $|s.\mathsf{susp}[a]|$. We do not consider these extensions further here.

3.3 Effectiveness

We now estimate the false- and true-detection rates for the algorithm in Sec. 3.2 across a range of parameter settings. In doing so, we seek to demonstrate that our algorithm can be effective in detecting credential stuffing without imposing significantly on legitimate users. We stop short of recommending a specific course of action when a site detects credential stuffing via our algorithm, though we will discuss alternatives at the end of this section.

Evaluating false- and true-detection rates empirically would require datasets that are unavailable to us. To evaluate false detections empirically, we would presumably need datasets that shed light on how users both set passwords across websites and then try (and sometimes fail) to log into websites using them. To evaluate true detections, we would need datasets of recorded credential-stuffing campaigns, along with the correct and guessed passwords and (for sites supporting SUSP$^+$) the results of second-factor challenges.

In the absence of any foreseeable way of obtaining such datasets, we instead perform an evaluation using probabilistic model checking. The tool we used to perform probabilistic model checking is Prism [42], which supports automated analysis of Markov decision processes (MDP). Each MDP we design models an actor interacting with a specific account a, who is either the legitimate user of a or an attacker, across multiple websites. To do so, we specify the actor as a set of *states* and possible *actions*. When in a state, the actor can choose from among these actions nondeterministically; the chosen action determines a probability distribution on the state to which the actor then transitions. These state transitions satisfy the *Markov property*: informally, the probability of next transitioning to a specific state depends only on the current state and the actor's chosen action. Prism exhaustively searches all decisions an actor can make to maximize the probability of the actor succeeding in its goal.

Below, we assume that the legitimate user's password choices across websites are represented by a probability distribution D_a; i.e., $D_a(\pi)$ is the probability with which the user selects π as its password for any given site. We abuse notation slightly and also use D_a to denote the set of passwords with non-zero probability. For example, we write $\pi \in D_a$ to indicate that $D_a(\pi) > 0$; $|D_a|$ to denote the number of passwords π for which $\pi \in D_a$; and $\pi \xleftarrow{\$} D_a$ to denote the selection of a password from distribution D_a and its assignment to π. As some prior works [4, 68], we model D_a as a Zipf distribution with parameter $\lambda_a \geq 0$, so that the user chooses her k-th most probable password ($1 \leq k \leq |D_a|$) independently with probability $(1/k^{\lambda_a})/(1/1^{\lambda_a} + 1/2^{\lambda_a} + \ldots + 1/|D_a|^{\lambda_a})$.

The MDPs below need to synthetically model the distribution of $\langle \mathsf{abnormal}^{\mathsf{col}}, \mathsf{abnormal}^{\mathsf{cnt}} \rangle$ pairs for login attempts or sessions thereof, similar to their distribution in practice (notably, lacking independence). To do so for specified detection rates $\mathsf{DR}^{\mathsf{col}}_{\mathsf{ads}}$ and $\mathsf{DR}^{\mathsf{cnt}}_{\mathsf{ads}}$, let $hi \in \{\mathsf{col}, \mathsf{cnt}\}$ and $lo \in \{\mathsf{col}, \mathsf{cnt}\}$ be such that $\mathsf{DR}^{hi}_{\mathsf{ads}}$ and $\mathsf{DR}^{lo}_{\mathsf{ads}}$ are the larger and smaller of

$\mathsf{DR}^{\mathsf{col}}_{\mathsf{ads}}$ and $\mathsf{DR}^{\mathsf{cnt}}_{\mathsf{ads}}$, respectively. Then, we let

$$\mathbb{P}\left(\mathsf{abnormal}^{hi} = true\right) = \mathsf{DR}^{hi}_{\mathsf{ads}}$$

$$\mathbb{P}\left(\mathsf{abnormal}^{lo} = true \;\middle|\; \mathsf{abnormal}^{hi} = true\right) = \mathsf{DR}^{lo}_{\mathsf{ads}}/\mathsf{DR}^{hi}_{\mathsf{ads}}$$

$$\mathbb{P}\left(\mathsf{abnormal}^{lo} = true \;\middle|\; \mathsf{abnormal}^{hi} = false\right) = 0$$

We denote selection of $\langle \mathsf{abnormal}^{\mathsf{col}}, \mathsf{abnormal}^{\mathsf{cnt}} \rangle$ according to this distribution in the experiment descriptions below by the notation $\langle \mathsf{abnormal}^{\mathsf{col}}, \mathsf{abnormal}^{\mathsf{cnt}} \rangle \xleftarrow{\$} \mathsf{ads}(\mathsf{DR}^{\mathsf{col}}_{\mathsf{ads}}, \mathsf{DR}^{\mathsf{cnt}}_{\mathsf{ads}})$.

3.3.1 Estimating the false detection rate

False detections can arise in our framework; indeed, even the entry of the *correct* password for an account at a website by the legitimate user might trigger a credential-stuffing detection if the user erroneously submitted the same password to other websites (that use SUSP), or even if correctly but without completing a second-factor challenge from those sites (that use SUSP$^+$). Here we leverage probabilistic model checking to conservatively estimate the probability with which a user induces a false detection.

We express the process by which a user might do so as a MDP in which the legitimate user is represented by two parties, to whom we refer here as "Dr. Jekyll" ($\mathcal{J}$) and "Mr. Hyde" ($\mathcal{H}$).[2] Informally, the user's $\mathcal{H}$ persona knows the distribution from which the user previously set passwords at various websites, but does not remember which password the user set at which site. $\mathcal{H}$ attempts a number of logins before turning control over to the $\mathcal{J}$ persona, who is challenged to log into another website, for which he does remember the password. Still, $\mathcal{J}$'s entry of the correct password might be detected as possible credential stuffing, depending on the actions of $\mathcal{H}$ before him. In a Jekyll-Hyde experiment, then, we say that $\mathcal{H}$ wins (and $\mathcal{J}$ loses) if $\mathcal{J}$'s login attempt is (falsely) detected as credential stuffing, and otherwise $\mathcal{H}$ loses (and $\mathcal{J}$ wins).

Since both $\mathcal{J}$ and $\mathcal{H}$ represent the legitimate user, we assume both can complete any second-factor challenges that a website issues. Under this assumption, there is no difference between SUSP and SUSP$^+$, and so we do not differentiate sites supporting SUSP$^+$ from those supporting SUSP in Jekyll-Hyde experiments. Since users who forget their passwords presumably tend to attempt multiple password guesses in the same login session (and so from the same platform and location) until finding the right one, $\mathcal{H}$ is classified once by the ADS at site s for all login attempts there. Finally, we forbid $\mathcal{H}$ from attempting logins at a site s after he has already submitted the correct password to s (see step 2) to preclude him from trivially winning. After all, once $\mathcal{H}$ has "recalled" the correct password for s, he could artificially add extra passwords to $s.\mathsf{susp}[a]$ by "attempting" logins with them, thereby unreasonably inflating his chances of winning.

[2] In their namesake novella [62], Mr. Hyde and Dr. Jekyll are the evil and good personae, respectively, of the same person.

More precisely, a Jekyll-Hyde experiment takes as input an account identifier a, the distribution D_a, a number of responders n_a, an integer w, and probabilities $\mathsf{FDR}^{\mathsf{col}}_{\mathsf{ads}}$ and $\mathsf{FDR}^{\mathsf{cnt}}_{\mathsf{ads}}$, and proceeds as follows:

(1) Sites s_i, for $0 \le i \le n_a$, are initialized as follows:
- A password $s_i.\mathsf{pwd}[a] \xleftarrow{\$} D_a$ is selected independently for account a at website s_i.
- The suspicious password set is cleared: $s_i.\mathsf{susp}[a] \leftarrow \emptyset$.
- To model the classification of $\mathcal{H}$ by the ADS at s_i, a boolean $s_i.\mathsf{collectionFlag}$ is set to $\mathsf{abnormal}^{\mathsf{col}}$ where $\langle \mathsf{abnormal}^{\mathsf{col}}, \mathsf{abnormal}^{\mathsf{cnt}} \rangle \xleftarrow{\$} \mathsf{ads}(\mathsf{FDR}^{\mathsf{col}}_{\mathsf{ads}}, \mathsf{FDR}^{\mathsf{cnt}}_{\mathsf{ads}})$.

(2) $\mathcal{H}$ is given the experiment inputs and performs login attempts on a at any of $s_1, \ldots, s_{n_a}$, provided that if $\mathcal{H}$ submits the correct password $s_i.\mathsf{pwd}[a]$ in a login attempt at s_i, then this is $\mathcal{H}$'s last login attempt at s_i. Each incorrect login attempt at s_i adds the attempted password to $s_i.\mathsf{susp}[a]$ if and only if $s_i.\mathsf{collectionFlag}$ is *true*.

(3) Once $\mathcal{H}$ is done, $\mathcal{J}$ logs into s_0 using the correct password $s_0.\mathsf{pwd}[a]$. If $\mathsf{abnormal}^{\mathsf{cnt}} = true$ for $\langle \mathsf{abnormal}^{\mathsf{col}}, \mathsf{abnormal}^{\mathsf{cnt}} \rangle \xleftarrow{\$} \mathsf{ads}(\mathsf{FDR}^{\mathsf{col}}_{\mathsf{ads}}, \mathsf{FDR}^{\mathsf{cnt}}_{\mathsf{ads}})$ and if $\left|\left\{s_i \;\middle|\; s_0.\mathsf{pwd}[a] \in s_i.\mathsf{susp}[a] \wedge 1 \le i \le n_a\right\}\right| \ge w$, then $\mathcal{H}$ wins. Otherwise, $\mathcal{J}$ wins.

We define $\mathsf{FDR}_{\mathsf{csd}}$ ("csd" denotes "credential-stuffing detection") to be the probability with which $\mathcal{H}$ wins and so $\mathcal{J}$ loses, under an optimal strategy for $\mathcal{H}$. We believe that $\mathsf{FDR}_{\mathsf{csd}}$ is a very conservative estimate on the false detection rate of our framework in practice, in that it reflects the worst case behavior (in terms of usability) of $\mathcal{H}$. Moreover, by testing $\left|\left\{s_i \;\middle|\; s_0.\mathsf{pwd}[a] \in s_i.\mathsf{susp}[a] \wedge 1 \le i \le n_a\right\}\right| \ge w$ only for $\mathcal{J}$, i.e., after $\mathcal{H}$ has filled the suspicious password sets of sites as much as possible, our false-detection estimates are even more conservative (notably, ignoring $\mathcal{H}$'s logins where he went undetected).

Consider an example with $n_a = 2$ sites (s_1 and s_2), $|D_a| = 2$ passwords (π_1 and π_2), and $w = 1$. Consider the following sequence of choices by $\mathcal{H}$: First, $\mathcal{H}$ attempts π_1 at s_2, which is correct with probability $D_a(\pi_1)$ and incorrect with probability $1 - D_a(\pi_1) = D_a(\pi_2)$. In addition, this login attempt is detected as abnormal if $s_2.\mathsf{collectionFlag} = true$, which occurs with probability $\mathsf{FDR}^{\mathsf{col}}_{\mathsf{ads}}$. Suppose that π_1 is incorrect and $s_2.\mathsf{collectionFlag} = true$, and so $s_2.\mathsf{susp}[a]$ now contains π_1. Then suppose $\mathcal{H}$ attempts password π_2 at s_1, but that this guess is incorrect and $\mathcal{H}$'s login is not deemed abnormal. Finally, suppose that $\mathcal{J}$ gains control and submits the correct password π_1 to s_0, but that his attempt is detected as abnormal (with probability $\mathsf{FDR}^{\mathsf{cnt}}_{\mathsf{ads}}$). Because $s_0.\mathsf{pwd}[a]$ (= π_1) is in at least $w = 1$ of the suspicious sets at other sites, i.e., $s_2.\mathsf{susp}[a]$, $\mathcal{H}$ wins. $\mathcal{H}$'s choices induce this sequence of events with probability $\left[D_a(\pi_2) \cdot \mathsf{FDR}^{\mathsf{col}}_{\mathsf{ads}}\right] \cdot \left[D_a(\pi_1) \cdot (1 - \mathsf{FDR}^{\mathsf{col}}_{\mathsf{ads}})\right] \cdot \left[D_a(\pi_1) \cdot \mathsf{FDR}^{\mathsf{cnt}}_{\mathsf{ads}}\right]$, and $\mathsf{FDR}_{\mathsf{csd}}$ is computed by exhaustively considering all possible choices by $\mathcal{H}$ and all event sequences.

3.3.2 Estimating the true detection rate

To evaluate the true-detection rate of our credential-stuffing algorithm, we use a different type of MDP, in which a credential-stuffing attacker $\mathcal{C}$ is given a "leaked" password $\pi_{\text{leaked}} \xleftarrow{\$} D_a$ and allowed to attempt logins using it at sites $s_1, \ldots, s_{n_a}$ where a has accounts. The attacker knows which sites have second-factor authentication enabled for abnormal logins to a, as specified by a set $\mathsf{has2FA}_a \subseteq \{s_1, \ldots, s_{n_a}\}$. Each site $s_i \in \mathsf{has2FA}_a$ therefore uses SUSP$^+$ to manage $s_i.\mathsf{susp}[a]$, and we assume that $\mathcal{C}$ cannot pass a second-factor challenge for a. Other sites use SUSP. We also allow the attacker knowledge of the true-detection rates $\mathsf{TDR}^{\mathsf{col}}_{\mathsf{ads}}$ and $\mathsf{TDR}^{\mathsf{cnt}}_{\mathsf{ads}}$ of sites' ADS.

As such, our true-detection experiment takes as input an account identifier a, the distribution D_a, the number of responders n_a, the set $\mathsf{has2FA}_a$, an integer w, and probabilities $\mathsf{TDR}^{\mathsf{col}}_{\mathsf{ads}}$ and $\mathsf{TDR}^{\mathsf{cnt}}_{\mathsf{ads}}$, and proceeds as follows:

(1) Sites s_i, for $1 \le i \le n_a$, are initialized as follows:

- A password $s_i.\mathsf{pwd}[a] \xleftarrow{\$} D_a$ is selected independently for account a at website s_i.
- The suspicious password set is cleared: $s_i.\mathsf{susp}[a] \leftarrow \emptyset$.
- A boolean $s_i.\mathsf{attemptedFlag}$ is initialized to *false*.

(2) $\mathcal{C}$ is given a, $\pi_{\text{leaked}} \xleftarrow{\$} D_a$, $\mathsf{has2FA}_a$, $\mathsf{TDR}^{\mathsf{col}}_{\mathsf{ads}}$, $\mathsf{TDR}^{\mathsf{cnt}}_{\mathsf{ads}}$, w, and the opportunity to perform one login attempt using π_{leaked} at each of $s_1, \ldots, s_{n_a}$. On $\mathcal{C}$'s l-th login attempt ($l = 1, 2, \ldots$), let s_{i_l} denote the site at which this attempt occurs. Then:

- Set $\langle \mathsf{abnormal}^{\mathsf{col}}, \mathsf{abnormal}^{\mathsf{cnt}} \rangle \xleftarrow{\$} \mathsf{ads}(\mathsf{TDR}^{\mathsf{col}}_{\mathsf{ads}}, \mathsf{TDR}^{\mathsf{cnt}}_{\mathsf{ads}})$ and $s_{i_l}.\mathsf{collectionFlag} \leftarrow \mathsf{abnormal}^{\mathsf{col}}$.
- s_{i_l} adds π_{leaked} to $s_{i_l}.\mathsf{susp}[a]$ if $\mathsf{abnormal}^{\mathsf{col}} = \mathit{true}$ and either $\pi_{\text{leaked}} \neq s_{i_l}.\mathsf{pwd}[a]$ (per SUSP) or, if $s_{i_l} \in \mathsf{has2FA}_a$, even if $\pi_{\text{leaked}} = s_{i_l}.\mathsf{pwd}[a]$ (per SUSP$^+$, since $\mathcal{C}$ cannot pass second-factor authentication).
- If $l > w$, then
  - $s_{i_l}.\mathsf{attemptedFlag} \leftarrow \mathit{true}$
  - $s_{i_l}.\mathsf{detectedFlag} \leftarrow \mathit{true}$ if $\mathsf{abnormal}^{\mathsf{cnt}}$ and, at this point, $|\{s_{i'} \mid \pi_{\text{leaked}} \in s_{i'}.\mathsf{susp}[a] \wedge i' \neq i_l\}| \ge w$. Otherwise, $s_{i_l}.\mathsf{detectedFlag} \leftarrow \mathit{false}$.

When the experiment is finished, define

$$\mathsf{accessed} = \left\{ s_i \;\middle|\; \begin{array}{l} s_i.\mathsf{attemptedFlag} \wedge \pi_{\text{leaked}} = s_i.\mathsf{pwd}[a] \;\wedge \\ (s_i.\mathsf{collectionFlag} \Rightarrow s_i \notin \mathsf{has2FA}_a) \end{array} \right\}$$

$$\mathsf{detected} = \left\{ s_i \;\middle|\; s_i \in \mathsf{accessed} \wedge s_i.\mathsf{detectedFlag} \right\}$$

Then, we define $\mathsf{TDR}_{\mathsf{csd}} = \frac{\mathbb{E}(|\mathsf{detected}|)}{\mathbb{E}(|\mathsf{accessed}|)}$ where this ratio is computed using the adversary's optimal strategy for minimizing $\mathbb{E}(|\mathsf{detected}|)$ among all strategies that maximize $\mathbb{E}(|\mathsf{accessed}|)$. The condition $l > w$ in the last bullet of step (2) limits accessed and detected to include only sites at which the attacker succeeded or was detected, respectively, starting with the $(w+1)$-th login attempt, since by design, our algorithm cannot detect w or fewer credential-stuffing login attempts. As such, $\mathsf{TDR}_{\mathsf{csd}}$ is best interpreted as the true-detection rate for attacks of width greater than w. We expect that $\mathsf{TDR}_{\mathsf{csd}}$ is very conservative as an estimate of the true detection rate in practice, since it is computed using the best possible strategy for $\mathcal{C}$, equipped with perfect knowledge of parameters he would not generally have.

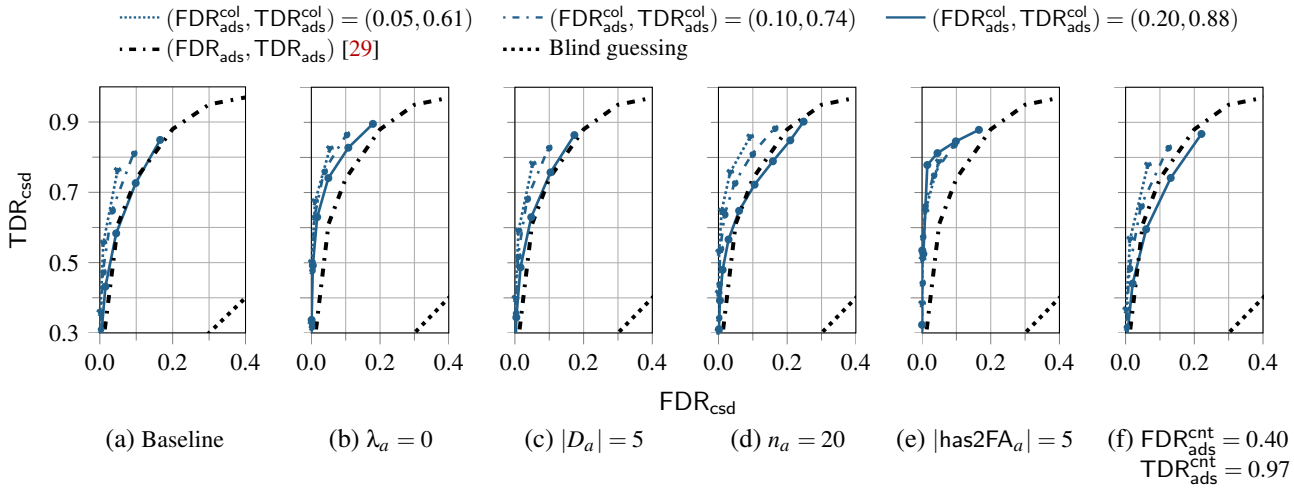
Consider an example with $n_a = 2$ sites (s_1 and s_2), neither of which support second-factor authentication for a; $|D_a| = 2$ passwords (π_1 and π_2); and $w = 1$. Suppose $\mathcal{C}$ is given π_1 (with probability $D_a(\pi_1)$) as the "leaked" password. $\mathcal{C}$ picks one site, say s_2, and tries to log in with π_1. With probability $D_a(\pi_2) \cdot \mathsf{TDR}^{\mathsf{col}}_{\mathsf{ads}}$, $\mathcal{C}$ fails at s_2 with π_1 being added to $s_2.\mathsf{susp}[a]$. In this event, suppose $\mathcal{C}$ then submits π_1 to s_1, where π_1 is correct (with probability $D_a(\pi_1)$) and so s_1 is added to accessed, but where this login attempt is detected as abnormal in s_1's *counting phase* (with probability $\mathsf{TDR}^{\mathsf{cnt}}_{\mathsf{ads}}$). Since π_1 appears in $w = 1$ of the suspicious sets at other sites (i.e., in $s_2.\mathsf{susp}[a]$), s_1 is added to detected. $\mathcal{C}$'s choices induce these events with probability $D_a(\pi_1) \cdot [D_a(\pi_2) \cdot \mathsf{TDR}^{\mathsf{col}}_{\mathsf{ads}}] \cdot [D_a(\pi_1) \cdot \mathsf{TDR}^{\mathsf{cnt}}_{\mathsf{ads}}]$. $\mathsf{TDR}_{\mathsf{csd}}$ is then computed by considering all possible choices by $\mathcal{C}$ and all event sequences.

3.3.3 Trading off $\mathsf{TDR}_{\mathsf{csd}}$ and $\mathsf{FDR}_{\mathsf{csd}}$

Given the above MDPs and resulting $\mathsf{TDR}_{\mathsf{csd}}$ and $\mathsf{FDR}_{\mathsf{csd}}$ measures, we now explore how they vary together as w is varied, for fixed $(\mathsf{FDR}^{\mathsf{col}}_{\mathsf{ads}}, \mathsf{TDR}^{\mathsf{col}}_{\mathsf{ads}})$ and $(\mathsf{FDR}^{\mathsf{cnt}}_{\mathsf{ads}}, \mathsf{TDR}^{\mathsf{cnt}}_{\mathsf{ads}})$ pairs and parameters λ_a, $|D_a|$, n_a, and $|\mathsf{has2FA}_a|$. The $(\mathsf{FDR}^{\mathsf{col}}_{\mathsf{ads}}, \mathsf{TDR}^{\mathsf{col}}_{\mathsf{ads}})$ and $(\mathsf{FDR}^{\mathsf{cnt}}_{\mathsf{ads}}, \mathsf{TDR}^{\mathsf{cnt}}_{\mathsf{ads}})$ pairs we consider were drawn by inspection from ROC curves published by Freeman et al. [29, Fig. 4b] for their ADS, for two categories of attackers: a *researching* attacker who issues login attempts from the legitimate user's country (presumably after researching that user), and a *phishing* attacker who issues login attempts both from the legitimate user's country and presenting the same useragent string as the legitimate user would (presumably after phishing the user). In particular, phishing attackers are the most powerful attackers considered by Freeman et al. The curves labeled $(\mathsf{FDR}_{\mathsf{ads}}, \mathsf{TDR}_{\mathsf{ads}})$ in Fig. 1 and Fig. 2 depict the ROC curves reported by Freeman et al. for *phishing* and *researching* attackers, respectively.

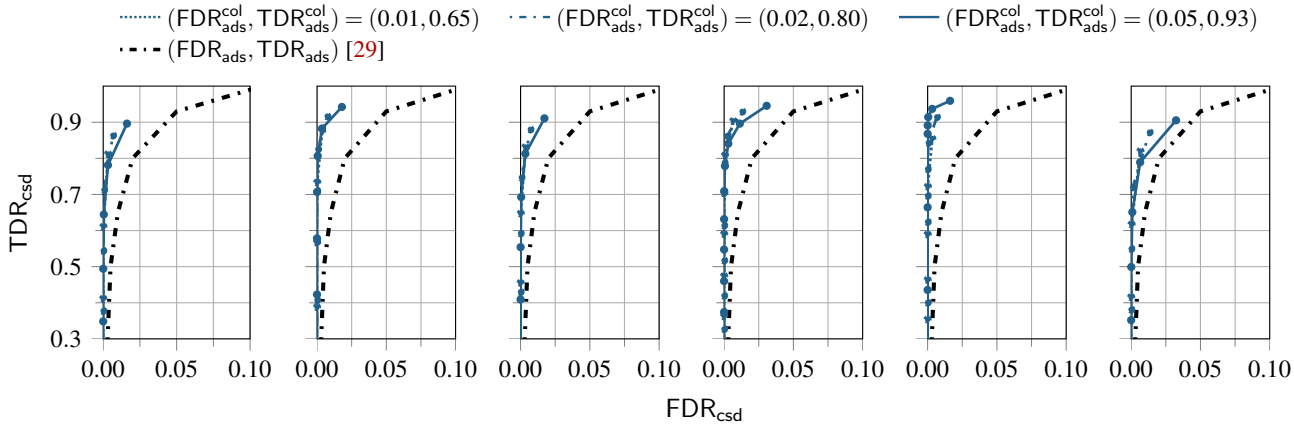
Fig. 1 shows representative ROC curves for a *phishing* attacker, and Fig. 2 shows curves for a *researching* attacker. "Baseline" configurations, detailed in each figure's caption, are shown in Fig. 1a and Fig. 2a. Each figure to the right of the baseline shows the effects of strengthening security in one parameter, starting from the baseline. So, for example, starting from the baseline, Fig. 1b shows the effects of users choosing passwords more uniformly (by changing $\lambda_a = 1$ to $\lambda_a = 0$). Similarly, Fig. 1c shows the effects, again starting from the baseline, of a user leveraging five passwords versus only four (i.e., by changing $|D_a| = 4$ to $|D_a| = 5$).

These ROC curves suggest that our credential-stuffing detector can be highly effective in detecting credential stuffing without impinging substantially on usability. Notably, our detector is more effective than simply using a state-of-the-art



(a) Baseline (b) $\lambda_a = 0$ (c) $|D_a| = 5$ (d) $n_a = 20$ (e) $|\mathsf{has2FA}_a| = 5$ (f) $\mathsf{FDR}^{\mathsf{cnt}}_{\mathsf{ads}} = 0.40$, $\mathsf{TDR}^{\mathsf{cnt}}_{\mathsf{ads}} = 0.97$

Figure 1: *Phishing* attacker. Baseline: $|D_a| = 4$, $\lambda_a = 1$, $n_a = 10$, $|\mathsf{has2FA}_a| = 0$, $(\mathsf{FDR}^{\mathsf{cnt}}_{\mathsf{ads}}, \mathsf{TDR}^{\mathsf{cnt}}_{\mathsf{ads}}) = (0.30, 0.95)$.



(a) Baseline (b) $\lambda_a = 0$ (c) $|D_a| = 5$ (d) $n_a = 20$ (e) $|\mathsf{has2FA}_a| = 5$ (f) $\mathsf{FDR}^{\mathsf{cnt}}_{\mathsf{ads}} = 0.20$, $\mathsf{TDR}^{\mathsf{cnt}}_{\mathsf{ads}} = 1.00$

Figure 2: *Researching* attacker. Baseline: $|D_a| = 4$, $\lambda_a = 1$, $n_a = 10$, $|\mathsf{has2FA}_a| = 0$, $(\mathsf{FDR}^{\mathsf{cnt}}_{\mathsf{ads}}, \mathsf{TDR}^{\mathsf{cnt}}_{\mathsf{ads}}) = (0.10, 0.99)$.

ADS [29] for a wide range of parameter settings.

Choosing a good operating point for our design depends on how a credential-stuffing detection is treated at the detecting site. An aggressive response such as locking the account pending a password reset (performed after two-factor authentication if deployed, or a different intervention if not) would favor keeping $\mathsf{FDR}_{\mathsf{csd}}$ small, e.g., $\mathsf{FDR}_{\mathsf{csd}} < 0.05$. A less aggressive response, such as invoking two-factor authentication on every login attempt until the password is reset, might allow a higher $\mathsf{FDR}_{\mathsf{csd}}$, e.g., $0.05 \leq \mathsf{FDR}_{\mathsf{csd}} < 0.10$. Simply warning the user might permit an even higher $\mathsf{FDR}_{\mathsf{csd}}$.

4 The Directory

Our framework in Sec. 3 requires one website to run PMT protocols as a requester with other sites where the same user has accounts. This capability is similar to that implemented in previous work [69] using a *directory* that stores, per account identifier a, an address (possibly a pseudonym) to contact each site where the account a exists. Assuming a one-round PMT protocol (as in Sec. 5), the directory receives a PMT query from a requester for an account a and forwards a copy of this query to each site with the same account. The directory then receives every responder's reply, permutes them randomly, and forwards the responses back to the requester in a batch. By shuffling the responses, the directory ensures that the requester learns only the number of responders that returned *true* (respectively, *false*), not which ones, for good measure. (The directory learns nothing about the private inputs to/outputs from the PMT protocol by requesters and responders.) Since our goal here is not to innovate in the design of scalable directory services—itself a topic with a long history, with many deployments that far surpass our needs

here, e.g., [19, 46]—we largely adopt this design in our implementation (see Sec. 6.1). Below we address two concerns about such a directory specifically in our context, however, namely the potentials for privacy risks and denials of service.

4.1 Privacy

Among the design goals adopted in previous work [69] is hiding the identity of the requester from the responders and the identity of each responder from other responders and the requester. The purpose of doing so is hiding where the user has accounts, a property termed "account location privacy". To this end, the requester and responders either trust the directory to hide their identities (as an anonymizing proxy, cf., [6, 30]) or communicate with the directory using Tor [20].

Unfortunately, account location privacy is impossible in our framework against an active attacker: an attacker can attempt a login on account a at a site s with a truly random password π, and if a exists at s and the attempt is deemed abnormal, π will be added to $s.\mathsf{susp}[a]$ under SUSP (or SUSP$^+$). The attacker can then attempt to use π in the PMT protocol as a requester, thereby learning whether some responder returns a *true* result; if so, then apparently a exists at s. This attack is of academic interest only, since in practice, an attacker could equally easily determine whether a exists at s by simply trying to establish account a at s; most sites will inform the attacker if a already exists. Still, our framework only further renders irrelevant any attempts to hide where the user has accounts.

We thus settle for a weaker notion of privacy here, namely hiding the identity of the requester only, which will at least hide the site at which the user is presently logging in. As such, while in our design the requester still communicates to the directory using Tor if it does not trust the directory to protect its identity, there is no point in the responders doing so; the responders receive requests directly from the directory and respond directly to it. We refer to the model in which requesters contact the directory directly as TLP ("trusted for login privacy"), and the model in which requesters contact the directory using Tor as ULP ("untrusted for login privacy").

4.2 Denials of Service

Like any critical service (cf., DNS), the directory should employ state-of-the-art defenses against blunt denial-of-service (DoS) attempts (e.g., request overloading). If the directory succumbs to such a DoS, then detecting credential stuffing will not be possible while the directory is offline, and a site will incur a delay awaiting a timeout on the directory for any login attempt with the correct password but for which $\mathsf{abnormal}^{\mathsf{cnt}} = \mathit{true}$. If the directory is responsible for providing the salt for an account to each site having that account (see Sec. 3.1), then a site s with a newly created account a will also be delayed in populating its $s.\mathsf{susp}[a]$ set until the directory recovers.

Our main concern here is whether the directory introduces DoS risks based on its particular functionality. One such DoS risk is associated with the process by which a website s informs the directory that the user with identifier a has registered an account at s and so s should now be consulted as a responder for a in the framework of Sec. 3. The risk lies primarily in malicious actors falsifying such registrations, e.g., potentially registering millions of sites per identifier a.

In our envisioned method of deploying our framework, this risk can be managed. For example, in Sec. 6.4, we evaluate the scalability of our design to support the U.S. airline, hotel, retail, and consumer banking industries. For a deployment by these industries, the websites permitted to register as a responder for an account a can be limited to approved members of these industry consortia. The directory can then limit each approved member to at most one such registration for a. In doing so, the directory can enforce a limit on the number of site registrations per account a. Moreover, owing to the security guarantees of our framework (specifically, see Sec. 5.7), a website has no motivation to register for an account a superfluously, since it learns nothing as a responder in the protocol (except that the user for a is active at some website).

That said, if further limiting the registrations for account a is desirable, then the directory can leverage the online presence of the user when creating account a at site s to confirm the request for s to register as a responder for a at the directory. For example, the directory can send a confirmation email to the email address a, asking her to confirm that she created an account at s. The registration attempt at the directory is then deferred until the user confirms it.

Not only do we contend that the directory is not particularly vulnerable to DoS, but it can also help in mitigating other DoS risks of our framework:

- *Defending requesters*: The primary DoS threat to a requester is the possibility that some responders always return a PMT protocol result indicating membership holds, increasing $\mathsf{FDR}_{\mathsf{csd}}$ accordingly. However, the directory can "audit" responders by issuing queries as a requester itself with a truly random password, which should garner a *false* result from every responder. Any responder whose response generates a *true* result is detected as misbehaving.
- *Defending responders*: Permitting PMT queries against $s.\mathsf{susp}[a]$ sets raises the possibility that an attacker will perform queries repeatedly to discover the contents of those sets. (In particular, recall that SUSP$^+$ permits $s.\mathsf{pwd}[a]$ to be added to $s.\mathsf{susp}[a]$.) A responder thus should rate-limit PMT queries, just as it would regular login attempts, to stem such online dictionary attacks. However, for accounts experiencing an unusually high rate of queries, the directory can pose CAPTCHAs [66] back to the requesters as a precondition to forwarding their queries to responders. In this way, the limited PMT budgets of responders can be allocated preferentially to requesters with real users, preventing bots from starving those requesters.

5 Privately Testing Set Membership

An ingredient of our framework in Sec. 3 is a protocol by which a requester s, having received password $\pi = s.\mathsf{pwd}[a]$ in a login attempt for account a, inquires with a responder s' to determine whether $\pi \in s'.\mathsf{susp}[a]$. Because $\pi = s.\mathsf{pwd}[a]$, it is important that the protocol not disclose π to s'. Moreover, since $s'.\mathsf{susp}[a]$ might contain $s'.\mathsf{pwd}[a]$ (see Sec. 3.2) or passwords similar to it, the protocol should not divulge $s.\mathsf{susp}[a]$ to s. This specification is met by a *private membership test* (PMT) protocol.

5.1 The Need for a New Protocol

Several PMT protocols have been proposed (e.g., [44, 47, 52, 63, 69]). In addition, PMT protocols are closely related to private set-intersection (PSI; surveyed by Pinkas et al. [50]) and private set-intersection cardinality protocols (PSI-CA; e.g., [14, 15, 18, 22, 41]). In particular, having the requester in a PSI/PSI-CA protocol prove in zero knowledge that its input is a set of size one yields a PMT protocol.

Considering the additional requirements of our framework in Secs. 3–4 somewhat narrows the options for implementing our PMT, however. First, because our threat model permits the requester or responder to misbehave arbitrarily, we require a protocol that accommodates the malicious behavior of either party while still protecting the privacy of each party's input to the protocol. Second, minimizing rounds of communication in the protocol is critical for the scalability of our framework, since these rounds (each with a different website as responder) will traverse wide-area links and—in the ULP model (see Sec. 4.1)—an anonymous communication channel, which will add even more overhead to each round. For the same reason, we wish to leverage bandwidth-efficient protocols to the extent possible, and because responders may need to respond to significant numbers of PMT queries (as we will analyze in Sec. 6.4), computational efficiency for the responder is a secondary but still important concern.

To our knowledge, among PSI protocols that are secure against malicious behaviors (e.g., [12, 16, 27, 28, 36, 40, 43, 54, 55, 65]), only those of De Cristofaro et al. [16] and of Thomas et al. [65] and Li et al. [43] execute in one round. However, the responses in these protocols are of size $O(\ell)$ ciphertexts for a set of size ℓ. While there are several one-round PSI-CA protocols (e.g., [14, 15, 18, 22]), we are aware of none that address malicious parties (without introducing a trusted third party, cf., [15, 18]).

One strategy to improve performance has been to weaken security in quantified ways against malicious parties. For example, for an integer $\chi \leq \ell$, Thomas et al. [65] and Li et al. [43] explored protocols in which the requester leaks $\log_2 \chi$ bits of the requester's input, in exchange for reducing the response size to $O(\ell/\chi)$ ciphertexts. However, a protocol that gains efficiency by leaking information only in the other direction (from responder to requester) is arguably more appropriate for our context, since the requester s invokes the protocol with the correct password, i.e., $s.\mathsf{pwd}[a]$. Ramezanian et al. [52] and Wang & Reiter [69] proposed protocols whereby the responder learns nothing about the requester's element, but the requester learns more information about the responder's set than just the truth of its membership query. Specifically, in the Ramezanian et al. protocol [52], the responder leaks its set to the requester over $O(\ell/\chi)$ responses, each of $O(\chi \log_2 \frac{1}{p})$ bits in size, where p is a tunable false positive rate for the membership test. The Wang & Reiter protocol [69] leaks the responder's set to a malicious requester over $O(\ell \log_2 \frac{1}{p})$ responses, each of size only one ciphertext. The protocol that we propose here also allows a malicious requester to learn the responder's set faster than the ideal—but only after $\Omega(\frac{1}{p})$ responses, much better than the Ramezanian et al. and Wang & Reiter protocols. (Below we term this measure the "extraction complexity" of the protocol, and justify this claim in Sec. 5.6.) The request and response sizes of our protocol are only $O(\ell/\chi)$ and $O(\chi)$ ciphertexts, respectively.

5.2 Partially Homomorphic Encryption

Our protocol builds on a partially homomorphic encryption scheme $\mathcal{E} = \langle \mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}, +_{[\cdot]} \rangle$ with these algorithms:

- Gen is a randomized algorithm that on input 1^κ outputs a public-key/private-key pair $\langle pk, sk \rangle \leftarrow \mathsf{Gen}(1^\kappa)$. The value of pk identifies a prime r for which the *plaintext space* for encrypting with pk is the finite field $\langle \mathbb{Z}_r, +, \times \rangle$ where $+$ and $\times$ are addition and multiplication modulo r, respectively. For clarity below, we denote the additive identity by $\mathbf{0}$, the multiplicative identity by $\mathbf{1}$, and the additive inverse of $m \in \mathbb{Z}_r$ by $-m$. pk also determines a *ciphertext space* $C_{pk} = \bigcup_{m \in \mathbb{Z}_r} C_{pk}(m)$, where $C_{pk}(m)$ denotes the ciphertexts for plaintext $m \in \mathbb{Z}_r$.
- Enc is a randomized algorithm that on input public key pk and a plaintext $m \in \mathbb{Z}_r$, outputs a ciphertext $c \leftarrow \mathsf{Enc}_{pk}(m)$ chosen uniformly at random from $C_{pk}(m)$.
- $+_{[\cdot]}$ is a randomized algorithm that, on input a public key pk and ciphertexts $c_1 \in C_{pk}(m_1)$ and $c_2 \in C_{pk}(m_2)$, outputs a ciphertext $c \leftarrow c_1 +_{pk} c_2$ chosen uniformly at random from $C_{pk}(m_1 + m_2)$.
- isZero is a deterministic algorithm that on input a private key sk and ciphertext $c \in C_{pk}$, outputs a boolean $z \leftarrow \mathsf{isZero}_{sk}(c)$ where $z = \mathit{true}$ iff $c \in C_{pk}(\mathbf{0})$.

Note that our protocol does not require an efficient decryption capability. Indeed, the instantiation of this scheme that we leverage, described in App. A, does not support one—though it does support an efficient isZero calculation.

5.3 Additional Operators

To express our protocol, it will be convenient to define a few additional operators involving ciphertexts. These additional

operators can all be expressed using the operators given in Sec. 5.2, and so require no new functionality from the cryptosystem. Below, "$Y \stackrel{d}{=} Y'$" denotes that random variables Y and Y' are distributed identically; "$\mathbf{Z} \in (Z)^{\alpha\times\alpha'}$" means that $\mathbf{Z}$ is an α-row, α'-column matrix of elements in the set Z; and "$(\mathbf{Z})_{i,j}$" denotes the row-i, column-j element of the matrix $\mathbf{Z}$.

- $\sum_{pk}$ denotes summing a sequence using $+_{pk}$, i.e.,

$$\sum_{k=1}^{z}{}_{pk}\ c_k \stackrel{d}{=} c_1 +_{pk} c_2 +_{pk} \ldots +_{pk} c_z$$

- If $\mathbf{C} \in (C_{pk})^{\alpha\times\alpha'}$ and $\mathbf{C}' \in (C_{pk})^{\alpha\times\alpha'}$, then $\mathbf{C} +_{pk} \mathbf{C}' \in (C_{pk})^{\alpha\times\alpha'}$ is the result of component-wise addition using $+_{pk}$, i.e., so that

$$\left(\mathbf{C} +_{pk} \mathbf{C}'\right)_{i,j} \stackrel{d}{=} (\mathbf{C})_{i,j} +_{pk} \left(\mathbf{C}'\right)_{i,j}$$

- If $\mathbf{M} \in (\mathbb{Z}_r)^{\alpha\times\alpha'}$ and $\mathbf{C} \in (C_{pk})^{\alpha\times\alpha'}$, then $\mathbf{M} \circ_{pk} \mathbf{C} \in (C_{pk})^{\alpha\times\alpha'}$ is the result of Hadamard (i.e., component-wise) "scalar multiplication" using repeated application of $+_{pk}$, i.e., so that

$$\left(\mathbf{M} \circ_{pk} \mathbf{C}\right)_{i,j} \stackrel{d}{=} \sum_{k=1}^{(\mathbf{M})_{i,j}}{}_{pk} \ (\mathbf{C})_{i,j}$$

- If $\mathbf{M} \in (\mathbb{Z}_r)^{\alpha\times\alpha'}$ and $\mathbf{C} \in (C_{pk})^{\alpha'\times\alpha''}$, then $\mathbf{M} *_{pk} \mathbf{C} \in (C_{pk})^{\alpha\times\alpha''}$ is the result of standard matrix multiplication using $+_{pk}$ and "scalar multiplication" using repeated application of $+_{pk}$, i.e., so that

$$\left(\mathbf{M} *_{pk} \mathbf{C}\right)_{i,j} \stackrel{d}{=} \sum_{k=1}^{\alpha'}{}_{pk} \sum_{k'=1}^{(\mathbf{M})_{i,k}}{}_{pk} \ (\mathbf{C})_{k,j}$$

5.4 Cuckoo Filters

Our PMT protocol, called CUCKOO-PMT, uses a cuckoo filter [25] as an underlying building block. A cuckoo filter is a set representation that supports insertion and deletion of elements, as well as testing membership. The cuckoo filter uses a "fingerprint" function $\mathsf{fprint} : \{0,1\}^* \to F$ and a hash function $\mathsf{hash} : \{0,1\}^* \to [\beta]$, where for an integer z, the notation "$[z]$" denotes $\{1,\ldots,z\}$, and where β is a number of "buckets". We require that $F \subset \mathbb{Z}_r \setminus \{\mathbf{0}\}$ for any r determined by $\langle pk, sk\rangle \leftarrow \mathsf{Gen}(1^\kappa)$, and that members of F can be distinguished from members of $\mathbb{Z}_r \setminus F$ using a public predicate. (For example, defining F to be the odd elements of $\mathbb{Z}_r$ would suffice.) For an integer bucket "capacity" χ, the cuckoo filter data structure is a χ-row, β-column matrix $\mathbf{X}$ of elements in $\mathbb{Z}_r$, i.e., $\mathbf{X} \in (\mathbb{Z}_r)^{\chi\times\beta}$. Then, the cuckoo filter contains the element e if and only if there exists $i \in [\chi]$ such that either

$$(\mathbf{X})_{i,\mathsf{hash}(e)} = \mathsf{fprint}(e) \qquad \text{or} \qquad (1)$$

$$(\mathbf{X})_{i,\mathsf{hash}(e)\oplus\mathsf{hash}(\mathsf{fprint}(e))} = \mathsf{fprint}(e) \qquad (2)$$

Cuckoo filters permit false positives (membership tests that return *true* for elements not previously added or already removed) with a probability p that, for fixed χ, can be decreased by increasing the size of F [25].

5.5 Protocol Description

Our protocol is illustrated in Fig. 3, where the steps performed by the requester R with input e are shown on the left in lines r1–r7 (in addition to sending message m1), and the steps performed by the responder S with cuckoo filter $\mathbf{X}$ are shown on the right in lines s1–s4 (in addition to sending message m2). The protocol returns *true* to R if e is in the cuckoo filter $\mathbf{X}$ and *false* otherwise.

In our protocol, R creates a β-row, 2-column matrix $\mathbf{Q}$ of ciphertexts, where the first column contains a ciphertext of $\mathbf{1}$ in row $\mathsf{hash}(e)$ and ciphertexts of $\mathbf{0}$ in other rows, and where the second column contains a ciphertext of $\mathbf{1}$ in row $\mathsf{hash}(e) \oplus \mathsf{hash}(\mathsf{fprint}(e))$ and ciphertexts of $\mathbf{0}$ in others (line r5). The requester also generates a ciphertext f of $-\mathsf{fprint}(e)$ (line r2), and sends this ciphertext and the matrix $\mathbf{Q}$ to S, along with the public key pk (message m1). After checking in line s1 that $f \in C_{pk}$, and $\mathbf{Q} \in (C_{pk})^{\beta\times 2}$ (and that pk is well-formed, which is left implicit in Fig. 3), S generates a matrix $\mathbf{F} \in (C_{pk})^{\chi\times 2}$ having a copy of f in each component (line s2) and a matrix $\mathbf{M} \in (\mathbb{Z}_r)^{\chi\times 2}$ of random elements of $\mathbb{Z}_r \setminus \{\mathbf{0}\}$ (line s3). S then forms the response matrix $\mathbf{R} \leftarrow \mathbf{M} \circ_{pk} \left(\left(\mathbf{X} *_{pk} \mathbf{Q}\right) +_{pk} \mathbf{F}\right)$, which is best understood component-wise: $(\mathbf{R})_{i,j}$ is a ciphertext of a random element of $\mathbb{Z}_r \setminus \{\mathbf{0}\}$ if $\left(\left(\mathbf{X} *_{pk} \mathbf{Q}\right) +_{pk} \mathbf{F}\right)_{i,j}$ is a ciphertext of anything other than $\mathbf{0}$, since $(\mathbf{M})_{i,j}$ is chosen at random from $\mathbb{Z}_r \setminus \{\mathbf{0}\}$. Moreover, $\left(\left(\mathbf{X} *_{pk} \mathbf{Q}\right) +_{pk} \mathbf{F}\right)_{i,j}$ is an encryption of $\mathbf{0}$ iff $\left(\mathbf{X} *_{pk} \mathbf{Q}\right)_{i,j}$ is a ciphertext of $\mathsf{fprint}(e)$, since $(\mathbf{F})_{i,j}$ is a ciphertext of $-\mathsf{fprint}(e)$. And $\left(\mathbf{X} *_{pk} \mathbf{Q}\right)_{i,j}$ is a ciphertext of $\mathsf{fprint}(e)$ iff either (1) holds (since $(\mathbf{Q})_{\mathsf{hash}(e),1}$ is an encryption of $\mathbf{1}$) or (2) holds (since $(\mathbf{Q})_{\mathsf{hash}(e)\oplus\mathsf{hash}(\mathsf{fprint}(e)),2}$ is an encryption of $\mathbf{1}$). So, if R and S behave correctly, the protocol returns *true* to R iff e is an element of the cuckoo filter $\mathbf{X}$.

For (an artificially small) example, suppose $\beta = 3$, $\chi = 1$, and that the requester R queries the membership of element e such that $i_1 = \mathsf{hash}(e) = 3$ and $i_2 = \mathsf{hash}(e) \oplus \mathsf{hash}(\mathsf{fprint}(e)) = 2$. The responder S generates a cuckoo filter $\mathbf{X} \in (\mathbb{Z}_r)^{1\times 3}$ based on its input set. Here we assume e is in S's set, as indicated by $(\mathbf{X})_{1,3} = \mathsf{fprint}(e)$. Then,

$$\begin{aligned}
&\mathbf{X} *_{pk} \mathbf{Q} \\
&\stackrel{d}{=} \begin{bmatrix} m_1 & m_2 & \mathsf{fprint}(e) \end{bmatrix} *_{pk} \begin{bmatrix} \mathsf{Enc}_{pk}(\mathbf{0}) & \mathsf{Enc}_{pk}(\mathbf{0}) \\ \mathsf{Enc}_{pk}(\mathbf{0}) & \mathsf{Enc}_{pk}(\mathbf{1}) \\ \mathsf{Enc}_{pk}(\mathbf{1}) & \mathsf{Enc}_{pk}(\mathbf{0}) \end{bmatrix} \\
&\stackrel{d}{=} \begin{bmatrix} \mathsf{Enc}_{pk}(\mathsf{fprint}(e)) & \mathsf{Enc}_{pk}(m_2) \end{bmatrix}
\end{aligned}$$

where m_1 and m_2 are elements of $\mathbb{Z}_r$ that, barring a collision in the output of fprint, are not equal to $\mathsf{fprint}(e)$. So,

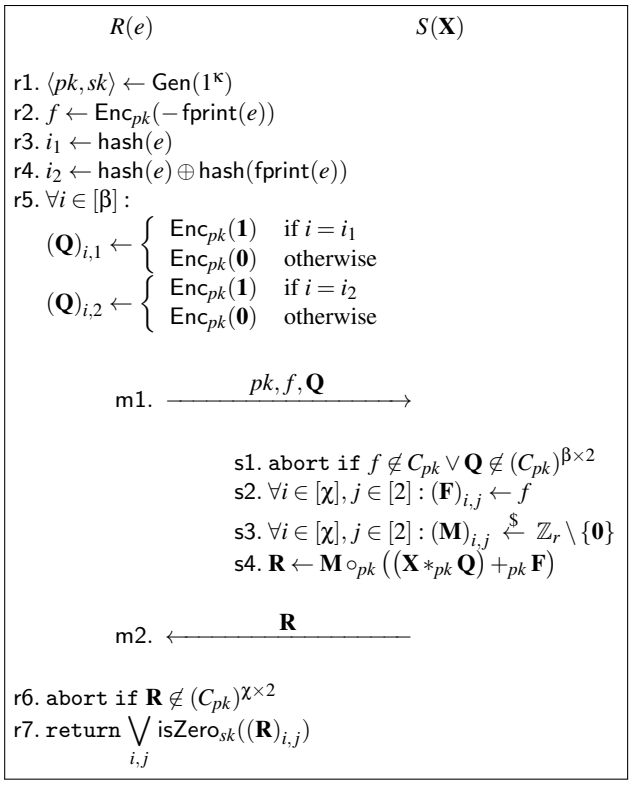


Figure 3: CUCKOO-PMT with requester R and responder S, described in Sec. 5. Matrix types are: $\mathbf{Q} \in (C_{pk})^{\beta\times 2}$; $\mathbf{X} \in (\mathbb{Z}_r)^{\chi\times\beta}$; $\mathbf{F} \in (C_{pk})^{\chi\times 2}$; $\mathbf{M} \in (\mathbb{Z}_r)^{\chi\times 2}$; and $\mathbf{R} \in (C_{pk})^{\chi\times 2}$.

$(\mathbf{X} *_{pk} \mathbf{Q}) +_{pk} \mathbf{F} \stackrel{d}{=} \left[\begin{array}{cc} \mathsf{Enc}_{pk}(\mathbf{0}) & \mathsf{Enc}_{pk}(m_3) \end{array}\right]$ where $m_3 = m_2 - \mathsf{fprint}(e) \neq \mathbf{0}$, again assuming $m_2 \neq \mathsf{fprint}(e)$, and thus $\mathbf{M} \circ_{pk} \left((\mathbf{X} *_{pk} \mathbf{Q}) +_{pk} \mathbf{F}\right) \stackrel{d}{=} \left[\begin{array}{cc} \mathsf{Enc}_{pk}(\mathbf{0}) & \mathsf{Enc}_{pk}(m_4) \end{array}\right]$, where m_4 is distributed uniformly in $\mathbb{Z}_r \setminus \{\mathbf{0}\}$.

5.6 Security Against a Malicious Requester

If the responder follows the protocol, then the *only* information encoded in each $(\mathbf{R})_{i,j}$ is $\mathsf{isZero}_{sk}((\mathbf{R})_{i,j})$, as a corollary of the following two propositions.

Proposition 1. *If the responder follows the protocol, then* $\mathbb{P}\left((\mathbf{R})_{i,j} \in C_{pk}(m) \;\middle|\; (\mathbf{R})_{i,j} \notin C_{pk}(\mathbf{0})\right) = \frac{1}{r-1}$ *for any* $i \in [\chi]$, $j \in [2]$, *and* $m \in \mathbb{Z}_r \setminus \{\mathbf{0}\}$.

Proof. $\left((\mathbf{X} *_{pk} \mathbf{Q}) +_{pk} \mathbf{F}\right)_{i,j} \in C_{pk}$, since by line s1, $f \in C_{pk}$ and $\mathbf{Q} \in (C_{pk})^{\beta\times 2}$. Moreover, $\left((\mathbf{X} *_{pk} \mathbf{Q}) +_{pk} \mathbf{F}\right)_{i,j} \notin C_{pk}(\mathbf{0})$ since $(\mathbf{R})_{i,j} \notin C_{pk}(\mathbf{0})$ by assumption. Since $(\mathbf{M})_{i,j}$ is drawn uniformly from $\mathbb{Z}_r \setminus \{\mathbf{0}\}$ (line s3), the plaintext of $(\mathbf{R})_{i,j}$ is uniformly distributed in $\mathbb{Z}_r \setminus \{\mathbf{0}\}$. □

Proposition 2. *If the responder follows the protocol, then* $\mathbb{P}\left((\mathbf{R})_{i,j} = c \;\middle|\; (\mathbf{R})_{i,j} \in C_{pk}(m)\right) = \frac{1}{|C_{pk}(m)|}$ *for any* $i \in [\chi]$, $j \in [2]$, $m \in \mathbb{Z}_r$, *and* $c \in C_{pk}(m)$.

Proof. This is immediate since $+_{pk}$ ensures that for $c_1 \in C_{pk}(m_1)$ and $c_2 \in C_{pk}(m_2)$, $c_1 +_{pk} c_2$ outputs a ciphertext c chosen uniformly at random from $C_{pk}(m_1 + m_2)$. □

Prop. 1 and Prop. 2 are also true for the protocol of Wang & Reiter [69] (henceforth called BLOOM-PMT), in that each protocol execution leaks to the requester only one yes/no answer about the responder's set representation, regardless of the actions of the requester. A critical distinction exists between our protocol and BLOOM-PMT, however, in that BLOOM-PMT permits a malicious requester to craft queries so that the yes/no answer can be expected to carry a (full) bit of information to the requester about the responder's set. We capture this information leakage using *extraction complexity*, which is the expected number of queries for a malicious requester to extract the responder's set representation, thereby enabling the requester to conduct offline attacks on the set. More precisely, for a fixed responder set Z, the *extraction complexity* of a PMT protocol is the expected number of protocol runs required for a malicious requester to extract enough information from an honest responder to locally determine $e \stackrel{?}{\in} Z$ for any e with the same accuracy as the PMT provides.

BLOOM-PMT enables a malicious requester to learn any single bit in the Bloom-filter representation of the responder's set. Since to accommodate a set of size ℓ with false-positive rate of p for membership tests, a Bloom filter uses $O(\ell \log_2 \frac{1}{p})$ bits, this is the extraction complexity for BLOOM-PMT; after this many queries, the malicious requester knows enough to conduct an *offline* attack on set members. In contrast:

Proposition 3. *The extraction complexity of* CUCKOO-PMT *is* $\Omega(\frac{1}{p})$.

Proof. Suppose the responder behaves according to the protocol, and for each $i \in [\beta]$, $j \in [2]$, denote by $m_{i,j} \in \mathbb{Z}_r$ the plaintext such that $(\mathbf{Q})_{i,j} \in C_{pk}(m_{i,j})$. Similarly, denote by $m_f \in \mathbb{Z}_r$ the plaintext such that $f \in C_{pk}(m_f)$. A corollary of Props. 1–2 is then that in one PMT response, the requester learns only the result

$$\left(\sum_{k=1}^{\beta} \left((\mathbf{X})_{i,k} \times m_{k,j}\right)\right) + m_f \stackrel{?}{\equiv}_r \mathbf{0} \tag{3}$$

for each $i \in [\chi]$, $j \in [2]$, i.e., a total of 2χ linear congruence-mod-r tests, where the $m_{k,j}$ and m_f values are chosen by the requester. Even if $\mathbf{X}$ represents a set consisting of only a single element e chosen so that $\mathsf{fprint}(e)$ is uniformly distributed in F, confirming the presence of $\mathsf{fprint}(e)$ in $\mathbf{X}$ requires, in expectation, testing $|F|/2$ linear congruences and so performing $|F|/4\chi$ PMT queries. Since $|F|/2\chi \geq 1/p$ to retain the false-positive rate p [25, Section 5.1], CUCKOO-PMT has an extraction complexity of $\Omega(\frac{1}{p})$ queries. □

The lower bound in Prop. 3 is very coarse, in that it applies even for a cuckoo filter $\mathbf{X}$ storing a single element—not to

mention one storing many. Moreover, there are a number of measures that can make extraction even more difficult for a malicious requester at minimal expense to the responder.

- The responder can permute each column of $\mathbf{X}$ independently after each execution of CUCKOO-PMT, since the query matrix $\mathbf{Q}$ produced by a correct requester will select the same elements from $\mathbf{X}$ regardless of this permuting. Interpreting the results of multiple malicious PMT queries will become more difficult, however.
- The responder can select any $(\mathbf{X})_{i,j} \notin F$ uniformly at random from $\mathbb{Z}_r \setminus F$, ensuring that any linear test (3) involving $(\mathbf{X})_{i,j}$ (i.e., for which $m_{i,j} \neq \mathbf{0}$, using the notation in the proof of Prop. 3) succeeds with probability only $\frac{1}{|\mathbb{Z}_r \setminus F|}$.
- The responder can randomly permute the elements of $\mathbf{R}$ before returning it, since the result computed by a correct requester will be the same (line r7). In doing so, the requester is deprived of knowing which of its linear tests (3) were satisfied (if any were).

5.7 Security Against a Malicious Responder

We now prove security for the requester against a malicious responder. To do so, we define a malicious responder to be a triple $B = \langle B_1, B_2, B_3 \rangle$ of algorithms that participates in the experiment $\mathsf{Expt}^{\text{PMT-}b}_{\text{CUCKOO-PMT}}$ defined as follows:

$$
\begin{array}{l}
\texttt{Experiment } \mathsf{Expt}^{\text{PMT-}b}_{\text{CUCKOO-PMT}}(\langle B_1, B_2, B_3 \rangle) \\
\quad \langle e_0, e_1, \phi_1 \rangle \leftarrow B_1() \\
\quad \langle \langle pk, f, \mathbf{Q} \rangle, sk \rangle \leftarrow R_{\text{r1-r5}}(e_b) \\
\quad \langle \mathbf{R}, \phi_2 \rangle \leftarrow B_2(\langle pk, f, \mathbf{Q} \rangle, \phi_1) \\
\quad b' \leftarrow R_{\text{r6-r7}}(sk, pk, \mathbf{R}) \\
\quad b'' \leftarrow B_3(\phi_2, b') \\
\quad \texttt{return } b''
\end{array}
$$

In this experiment, $R_{\text{r1-r5}}$ denotes steps r1–r5 in Fig. 3, producing the message m1 and the private key sk. Similarly, $R_{\text{r6-r7}}$ denotes steps r6–r7. In the experiment, B_1 chooses two elements e_0, e_1, and b determines which of the two that is used in the experiment. B_2 is given message m1 and produces the response matrix $\mathbf{R}$. Finally, B_3 is given the final result b' of the protocol from line r7, and outputs a bit b''. Note that though Fig. 3 does not disclose R's result explicitly to S, we allow it to be disclosed to S (i.e., B_3) in this analysis, to permit CUCKOO-PMT to be used in other contexts (e.g., [69]). We define the responder-adversary advantage as

$$
\begin{aligned}
\mathsf{Adv}^{\text{PMT}}_{\text{CUCKOO-PMT}}(B) &= \mathbb{P}\left(\mathsf{Expt}^{\text{PMT-0}}_{\text{CUCKOO-PMT}}(B) = 0\right) \\
&\quad - \mathbb{P}\left(\mathsf{Expt}^{\text{PMT-1}}_{\text{CUCKOO-PMT}}(B) = 0\right)
\end{aligned}
$$

$$
\mathsf{Adv}^{\text{PMT}}_{\text{CUCKOO-PMT}}(t) = \max_B \mathsf{Adv}^{\text{PMT}}_{\text{CUCKOO-PMT}}(B)
$$

where the maximum is taken over all adversaries B that run in time t. Intuitively, $\mathsf{Adv}^{\text{PMT}}_{\text{CUCKOO-PMT}}(t)$ captures the ability of any adversary running in time t to differentiate which of two passwords of its choice the requester uses to run the protocol.

We reduce security of CUCKOO-PMT against a responder adversary B to IND-CPA security [2, Definition 5.8] of the encryption $\mathcal{E}$. The IND-CPA experiment $\mathsf{Expt}^{\text{CPA-}\hat{b}}_{\mathcal{E}}$ is

$$
\begin{array}{l}
\texttt{Experiment } \mathsf{Expt}^{\text{CPA-}\hat{b}}_{\mathcal{E}}(A) \\
\quad \langle pk, sk \rangle \leftarrow \mathsf{Gen}(1^\kappa) \\
\quad \check{b} \leftarrow A^{\mathsf{Enc}_{pk}(\text{LR}(\cdot,\cdot,\hat{b}))}(pk) \\
\quad \texttt{return } \check{b}
\end{array}
$$

Here, the IND-CPA adversary A is given access to a "left-or-right" oracle $\mathsf{Enc}_{pk}(\text{LR}(\cdot,\cdot,\hat{b}))$ that takes two plaintexts $m_0, m_1 \in \mathbb{Z}_r$ as input and returns $\mathsf{Enc}_{pk}(m_{\hat{b}})$. Finally, A returns a bit $\check{b}$, which the experiment returns. We define

$$
\begin{aligned}
\mathsf{Adv}^{\text{CPA}}_{\mathcal{E}}(A) &= \mathbb{P}\left(\mathsf{Expt}^{\text{CPA-0}}_{\mathcal{E}}(A) = 0\right) - \mathbb{P}\left(\mathsf{Expt}^{\text{CPA-1}}_{\mathcal{E}}(A) = 0\right) \\
\mathsf{Adv}^{\text{CPA}}_{\mathcal{E}}(t,q) &= \max_A \mathsf{Adv}^{\text{CPA}}_{\mathcal{E}}(A)
\end{aligned}
$$

where the maximum is taken over all IND-CPA adversaries A running in time t and making up to q oracle queries.

Proposition 4. $\mathsf{Adv}^{\text{PMT}}_{\text{CUCKOO-PMT}}(t) \leq 2\mathsf{Adv}^{\text{CPA}}_{\mathcal{E}}(t', q)$ *for* $q = 2\beta + 1$ *and some* $t' \leq 2t$.

Proof. Given a responder adversary $B = \langle B_1, B_2, B_3 \rangle$, we construct an IND-CPA adversary A as follows. A first invokes B_1 to obtain e_0 and e_1. Let m_{0k} denote the k-th plaintext that R would encrypt in an execution of the protocol on e_0, and similarly let m_{1k} denote the k-th plaintext that R would encrypt in an execution of the protocol on e_1. Then, A simulates R exactly, except using its oracle to obtain the k-th ciphertext $c_k \leftarrow \mathsf{Enc}_{pk}(\text{LR}(m_{0k}, m_{1k}, \hat{b}))$. Note that because A does not have sk, it cannot compute b' as R would, and so it chooses $b' \xleftarrow{\$} \{0,1\}$ randomly and provides it to B_3. When B_3 outputs b'', A copies this bit as its output $\check{b}$.

The value b' provided to B_3 is correct, i.e., $b' = \bigvee_{i,j} \mathsf{isZero}_{sk}((\mathbf{R})_{i,j})$, with probability $1/2$. In this case, the simulation provided by A to B is perfectly indistinguishable from a real execution, and so

$$
\begin{aligned}
&\mathbb{P}\left(\mathsf{Expt}^{\text{CPA-0}}_{\mathcal{E}}(A) = 0\right) = \mathbb{P}\left(\mathsf{Expt}^{\text{PMT-0}}_{\text{CUCKOO-PMT}}(B) = 0\right) \quad \text{and} \\
&\mathbb{P}\left(\mathsf{Expt}^{\text{CPA-1}}_{\mathcal{E}}(A) = 0\right) = \mathbb{P}\left(\mathsf{Expt}^{\text{PMT-1}}_{\text{CUCKOO-PMT}}(B) = 0\right)
\end{aligned}
$$

As such, $\mathsf{Adv}^{\text{CPA}}_{\mathcal{E}}(A) \geq \frac{1}{2}\mathsf{Adv}^{\text{PMT}}_{\text{CUCKOO-PMT}}(B)$. A makes $q = 2\beta + 1$ queries to construct $\mathbf{Q}$ and f, and consumes time at most $2t$ due to the time needed to construct both m_{0k} and m_{1k} for each k. □

6 Performance and Scalability

In this section, we describe an implementation of our framework and evaluate its performance and scalability. The goals of our evaluation are:

- To demonstrate the performance of our framework with varying parameters that could be used for real-world scenarios, e.g., different numbers of participating websites for different users, and various sizes of suspicious password sets maintained at a responder;
- To explore the potential performance degradation brought by adopting Tor to ensure the requester's login privacy when the directory is untrusted in this sense (i.e., ULP, as discussed in Sec. 4); and
- To evaluate the scalability of our prototype and to interpret its scalability in a real-world context.

6.1 Implementation

Here, we give the salient details of our prototype implementation of our framework.

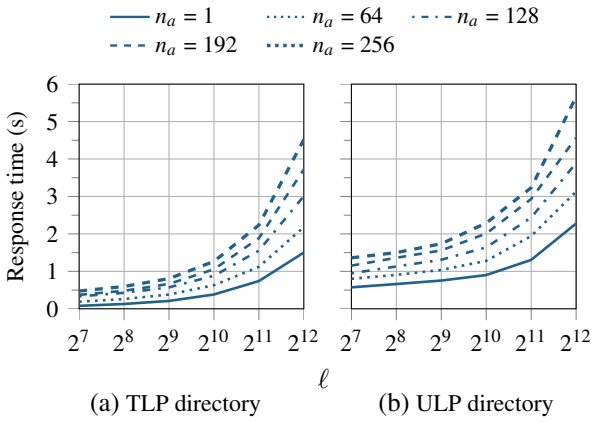
The PMT implementation We implemented CUCKOO-PMT (Sec. 5) in Go. We instantiated the exponential Elgamal cryptosystem (App. A) on a prime-order elliptic curve group, secp256r1 (NIST P-256) [9,26], which ensures approximately 128-bit symmetric encryption security or 3072-bit RSA security. For the cuckoo filter, we chose the bucket size $\chi = 16$, which permits an occupancy of 98%. That is, to accommodate a set of size ℓ, we need to build a cuckoo filter with capacity at least $\ell/0.98$.

We leveraged precomputation on the requester side for line r1 and line r5 in Fig. 3. Specifically, the requester can precompute $\langle pk, sk \rangle$, $2\beta - 2$ ciphertexts of $\mathbf{0}$, and two ciphertexts of $\mathbf{1}$ such that the online part of the computation in line r5 is simply to assemble the matrix $\mathbf{Q}$.

The directory We implemented the directory in Go, leveraging multi-threading to support parallel message processing. For each PMT query, the directory shuffles the intended responders' addresses before forwarding the requester's query to those responders and shuffles all responses from responders before returning them back to the requester. The first shuffling is to avoid evaluation bias due to the networking or computation differences among multiple responders. The second shuffling further weakens the linkability between responses and source responders, as an extra layer of security protection against a malicious requester (see Sec. 4).

6.2 Experimental Setup

We set up one requester, one directory, and up to 256 responders. The requester and the directory ran on two machines in our department, both with 2.67GHz × 8 physical cores, 72GiB RAM, and Ubuntu 18.04 x86_64. The 256 responders were split evenly across eight Amazon EC2 instances in the Eastern North American region, each with 3.2GHz × 32 physical cores, 256GiB RAM and Ubuntu 18.04 x86_64.



(a) TLP directory (b) ULP directory

Figure 4: Response time with varying ℓ and n_a

Each responder was limited to one physical core, and had its own exclusive data files, processes, and network sockets.

To test scenarios where the directory is trusted for login privacy (TLP) and where it is not (ULP), we set up two different types of communication channels between the requester and the directory. For the TLP directory, the requester and the directory communicated directly. For the ULP directory, we set up a private Tor network, through which the requester communicated with the directory via a newly built two-hop (i.e., with two Tor nodes) circuit for each new query to hide its identity from the directory. These two Tor nodes were chosen by Tor's default selection algorithm from eight Tor nodes running in eight different Amazon datacenters in North America and Europe. In both the TLP and ULP cases, the directory communicated with responders directly.

Each reported datapoint is the average of 50 runs. The relative standard deviations of each datapoint for the TLP directory and ULP directory scenarios were less than 4% and 8%, respectively.

6.3 Response Time

We first report the results of our response-time evaluation experiments for our implementation and setup above. In these experiments, the requester issued one CUCKOO-PMT query via the directory to n_a responders, and awaited the n_a responses from the responders. The response time is the duration observed by the requester between starting to generate a PMT query (after precomputation) and receiving all responses and outputting the result. Fig. 4a shows the response time when directory is trusted for login privacy (TLP), while Fig. 4b shows the response time when the directory is untrusted for login privacy (ULP). As mentioned in Sec. 6.2, in the former case the requester directly connected to the directory, while in the latter case, the requester communicated with the directory via a Tor circuit. Tor circuit setup is included in the response-time measurement. In both cases, the directory had

direct connections with all responders, with no Tor circuits involved.

The main takeaway from Fig. 4 is that when the capacity ℓ of the suspicious password set at each responder was relatively small, say $\ell \leq 2^9$, the response time was less than 1s with a TLP directory and less than 2s with a ULP directory, even for users with a large number of web accounts, say $n_a = 256$. Since the average user has far fewer accounts ($n_a \approx 26$ [49]), and since modern password-management recommendations would allow suspicious-password sets to be capped at a size $\ell < 2^7$ (see Sec. 3.2), we can expect the response time for an isolated request to be less than 1s even in the ULP directory scenario.

6.4 Scalability

To evaluate the scalability of our framework, we measured the maximum qualifying response rate that our prototype can achieve. Here, a *qualifying* response is one for which the response time falls within a certain allowance, which we specified as 5s in the TLP directory case and 8s in the ULP directory case. For each query, n_a responders were chosen uniformly at random from all 256 responders. To produce a conservative ULP estimate, we required the requester to communicate with the ULP directory via a new Tor circuit for each new query, to account for the potential scalability degradation brought by building Tor connections.

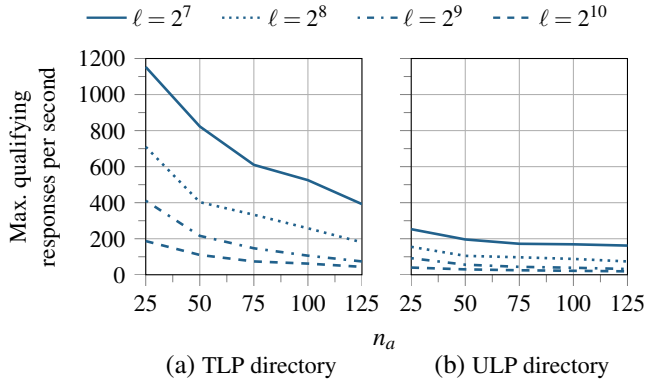


Figure 5: Maximum qualifying responses per second

The results of these tests are shown in Fig. 5. To put these numbers in context, consider that there are ≈ 369.4M credential-stuffing login attempts per day for the four U.S. industries listed in Table 1. According to the reported success rates of credential stuffing, 0.83M of these login attempts are with correct passwords. Moreover, there are ≈ 187.6M legitimate login attempts per day in these industries; for a conservative estimate here, we assume that they all provide the correct passwords. With the baseline *phishing* ADS configuration used in Sec. 3.3, i.e., $(\mathsf{FDR}^{\mathsf{cnt}}_{\mathsf{ads}}, \mathsf{TDR}^{\mathsf{cnt}}_{\mathsf{ads}}) = (0.30, 0.95)$, there would be 57.07M ($= 187.6\text{M} \times 0.3 + 0.83\text{M} \times 0.95$) login attempts per day that induce PMT queries or, in other words, about 660 PMT queries per second. With the baseline *researching* ADS configuration used in Sec. 3.3, i.e., $(\mathsf{FDR}^{\mathsf{cnt}}_{\mathsf{ads}}, \mathsf{TDR}^{\mathsf{cnt}}_{\mathsf{ads}}) = (0.10, 0.99)$, an analogous calculation suggests 19.58M PMT queries per day or 227 per second. Our experiments suggest that our prototype could achieve these throughputs with just one directory server for a range of configurations. For example, configured for *phishing* attackers, our TLP directory should support the requisite throughput when $\ell < 2^7$ for up to $n_a \approx 69$ responders. Configured for *researching* attackers, even the ULP configuration could support the expected throughput when $\ell < 2^7$ for up to $n_a \approx 36$ responders, and the TLP configuration would support the needed throughput when $\ell < 2^7$ for n_a as large as $n_a = 125$.

Industry	Credential-stuffing login attempts per day [58, Tables 3–6]	Proportion that succeed [58, Tables 3–6]	Proportion of all login attempts [58, Fig. 13]
Airline	1.4M	1.00%	60%
Hotel	4.3M	1.00%	44%
Retail	131.5M	0.50%	91%
Consumer banking	232.2M	0.05%	58%

Table 1: Credential-stuffing estimates for U.S. industries

Though already encouraging, these results leveraged only one requester machine and one directory machine, and concentrated all PMT queries to be served by (a randomly chosen subset of size n_a of) the same 256 responders, each allocated only a single CPU core. Responder CPU was the bottleneck in the TLP experiments. Tor was the bottleneck in the ULP experiments, being the only difference from the TLP experiments. With a more dispersed query pattern launched from more requesters, with more capable responders, and with a distributed directory, our design could scale even further.

7 Conclusion

In this paper we have proposed a novel framework by which websites can coordinate to detect active credential-stuffing attacks on individual accounts. Our framework accommodates the tendencies of human users to reuse passwords, to enter their passwords into incorrect sites, etc., while still providing good detection accuracy across a range of operating points. The framework is built on a new private membership-test protocol that scales better than previous alternatives and/or ensures a higher *extraction complexity*, which captures the ability of a requester in the protocol to extract enough information to search elements of the set offline. Using probabilistic model checking applied to novel experiments designed to capture both usability and security, we quantified the benefits of

our framework. Finally, we showed through empirical results with our prototype implementation that our design should scale easily to accommodate the login load of large sectors of the U.S. economy, for example.

Acknowledgments We are grateful to the anonymous reviewers and to our shepherd, Prof. Stephen Checkoway, for their constructive feedback.

References

[1] Akamai. State of the internet/security: Credential stuffing – attacks and economies. https://www.akamai.com/us/en/multimedia/documents/state-of-the-internet/soti-security-credential-stuffing-attacks-and-economies-report-2019.pdf, 2019.

[2] M. Bellare and P. Rogaway. Introduction to modern cryptography. https://web.cs.ucdavis.edu/~rogaway/classes/227/spring05/book/main.pdf, 2005.

[3] A. Biryukov, D. Dinu, and D. Khovratovich. Argon2: New generation of memory-hard functions for password hashing and other applications. In *1st IEEE Euro S&P*, March 2016.

[4] J. Blocki, B. Harsha, and S. Zhou. On the economics of offline password cracking. In *39th IEEE S&P*, May 2018.

[5] H. Bojinov, E. Bursztein, X. Boyen, and D. Boneh. Kamouflage: Loss-resistant password management. In *ESORICS*, volume 6345 of *LNCS*, September 2010.

[6] J. Boyan. The Anonymizer: Protecting user privacy on the web. *Computer-Mediated Communication Magazine*, 4(9), September 1997.

[7] A. S. Brown, E. Bracken, S. Zoccoli, and K. Douglas. Generating and remembering passwords. *Applied Cognitive Psychology*, 18(6), 2004.

[8] W. E. Burr et al. Electronic Authentication Guideline. https://doi.org/10.6028/NIST.SP.800-63-2, August 2013. NIST Special Publication 800-63-2.

[9] Certicom Research. SEC 2: Recommended elliptic curve domain parameters. http://www.secg.org/SEC2-Ver-1.0.pdf, 2000. Standards for Efficient Cryptography.

[10] K. Collins. Facebook buys black market passwords to keep your account safe. https://www.cnet.com/news/facebook-chief-security-officer-alex-stamos-web-summit-lisbon-hackers/, November 9 2016.

[11] R. Cramer, R. Gennaro, and B. Schoenmakers. A secure and optimally efficient multi-authority election scheme. In *EUROCRYPT '97*, volume 1233 of *LNCS*, pages 103–118, 1997.

[12] D. Dachman-Soled, T. Malkin, M. Raykova, and M. Yung. Efficient robust private set intersection. In *7th ACNS*, volume 5536 of *LNCS*, 2009.

[13] A. Das, J. Bonneau, M. Caesar, N. Borisov, and X. Wang. The tangled web of password reuse. In *ISOC NDSS*, 2014.

[14] A. Davidson and C. Cid. An efficient toolkit for computing private set operations. In *22nd ACISP*, volume 10343 of *LNCS*, July 2017.

[15] E. De Cristofaro, P. Gasti, and G. Tsudik. Fast and private computation of cardinality of set intersection and union. In *11th CANS*, volume 7712 of *LNCS*, 2012.

[16] E. De Cristofaro, J. Kim, and G. Tsudik. Linear-complexity private set intersection protocols secure in malicious model. In *ASIACRYPT*, volume 6477 of *LNCS*, 2010.

[17] J. DeBlasio, S. Savage, G. M. Voelker, and A. C. Snoeren. Tripwire: Inferring internet site compromise. In *17th IMC*, November 2017.

[18] S. K. Debnath and R. Dutta. Secure and efficient private set intersection cardinality using Bloom filter. In *18th ISC*, volume 9290 of *LNCS*, September 2015.

[19] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels. Dynamo: Amazon's highly available key-value store. In *21st ACM SOSP*, 2007.

[20] R. Dingledine, N. Mathewson, and P. Syverson. Tor: The second-generation onion router. In *13th USENIX Security*, August 2004.

[21] P. Doerfler, K. Thomas, M. Marincenko, J. Ranieri, Y. Jiang, A. Moscicki, and D. McCoy. Evaluating login challenges as a defense against account takeover. In *28th WWW*, May 2019.

[22] R. Egert, M. Fischlin, D. Gens, S. Jacob, M. Senker, and J. Tillmanns. Privately computing set-union and set-intersection cardinality via Bloom filters. In *20th ACISP*, volume 9144 of *LNCS*, 2015.

[23] T. ElGamal. A public-key cryptosystem and a signature scheme based on discrete logarithms. *IEEE TOIT*, 31(4), 1985.

[24] I. Erguler. Achieving flatness: Selecting the honeywords from existing user passwords. *IEEE TPDS*, 13(2), 2016.

[25] B. Fan, D. G. Andersen, M. Kaminsky, and M. D. Mitzenmacher. Cuckoo filter: Practically better than Bloom. In *10th ACM CoNEXT*, pages 75–88, 2014.

[26] Federal Information Processing Standards (FIPS) 186-4, Digital Signature Standard (DSS). http://dx.doi.org/10.6028/nist.fips.186-4, July 2013. National Institute of Standards and Technology (NIST).

[27] M. J. Freedman, C. Hazay, K. Nissim, and B. Pinkas. Efficient set intersection with simulation-based security. *J. Cryptology*, 29(1), 2016.

[28] M. J. Freedman, K. Nissim, and B. Pinkas. Efficient private matching and set intersection. In *EUROCRYPT 2004*, volume 3027 of *LNCS*, pages 1–19, May 2004.

[29] D. Freeman, S. Jain, M. Dürmuth, B. Biggio, and G. Giacinto. Who are you? A statistical approach to measuring user authenticity. In *23rd ISOC NDSS*, February 2016.

[30] E. Gabber, P. B. Gibbons, D. M. Kristol, Y. Matias, and A. Mayer. Consistent, yet anonymous, web access with LPWA. *CACM*, 42(2), February 1999.

[31] X. Gao, Y. Yang, C. Liu, C. Mitropoulos, J. Lindqvist, and A. Oulasvirta. Forgetting of passwords: Ecological theory and data. In *27th USENIX Security*, August 2018.

[32] C. Gentry, P. MacKenzie, and Z. Ramzan. A method for making password-based key exchange resilient to server compromise. In *CRYPTO*, volume 4117 of *LNCS*, pages 142–159, 2006.

[33] M. Golla, M. Wei, J. Hainline, L. Filipe, M. Dürmuth, E. Redmiles, and B. Ur. 'What was that site doing with my Facebook password?' Designing password-reuse notifications. In *25th ACM CCS*, October 2018.

[34] Google/Harris Poll. Online security survey. http://services.google.com/fh/files/blogs/google_security_infographic.pdf, February 2019.

[35] P. A. Grassi et al. Digital Identity Guidelines: Authentication and Lifecycle Management. https://doi.org/10.6028/NIST.SP.800-63b, June 2017. NIST Special Publication 800-63B.

[36] C. Hazay and K. Nissim. Efficient set operations in the presence of malicious adversaries. *J. Cryptology*, 25(3), July 2012.

[37] C. Herley and S. Schechter. Distinguishing attacks from legitimate authentication traffic at scale. In *26th ISOC NDSS*, February 2019.

[38] I. Ion, R. Reeder, and S. Consolvo. '... no one can hack my mind': Comparing expert and non-expert security practices. In *SOUPS*, 2015.

[39] A. Juels and R. L. Rivest. Honeywords: Making password-cracking detectable. In *ACM CCS*, 2013.

[40] S. Kamara, P. Mohassel, M. Raykova, and S. Sadeghian. Scaling private set intersection to billion-element sets. In *18th Financial Crypto*, volume 8437 of *LNCS*, March 2014.

[41] L. Kissner and D. Song. Privacy-preserving set operations. In *CRYPTO*, volume 3621 of *LNCS*, August 2005.

[42] M. Kwiatkowska, G. Norman, and D. Parker. PRISM 4.0: Verification of probabilistic real-time systems. In *CAV*, volume 6806 of *LNCS*, 2011.

[43] L. Li, B. Pal, J. Ali, N. Sullivan, R. Chatterjee, and T. Ristenpart. Protocols for checking compromised credentials. In *26th ACM CCS*, November 2019.

[44] T. Meskanen, J. Liu, S. Ramezanian, and V. Niemi. Private membership test for Bloom filters. In *IEEE Trustcom/BigDataSE/ISPA*, August 2015.

[45] M. Naor, B. Pinkas, and E. Ronen. How to (not) share a password: Privacy preserving protocols for finding heavy hitters with adversarial behavior. In *26th ACM CCS*, November 2019.

[46] S. A. Noghabi, S. Subramanian, P. Narayanan, S. Narayanan, G. Holla, M. Zadeh, T. Li, I. Gupta, and R. H. Campbell. Ambry: LinkedIn's scalable geo-distributed object store. In *ACM SIGMOD*, June 2016.

[47] R. Nojima and Y. Kadobayashi. Cryptographically secure Bloom-filters. *Trans. Data Privacy*, 2(2), August 2009.

[48] B. Pal, T. Daniel, R. Chatterjee, and T. Ristenpart. Beyond credential stuffing: Password similarity models using neural networks. In *40th IEEE S&P*, May 2019.

[49] S. Pearman, J. Thomas, P. E. Naeini, H. Habib, L. Bauer, N. Christin, L. F. Cranor, S. Egelman, and A. Forget. Let's go in for a closer look: Observing passwords in their natural habitat. In *24th ACM CCS*, October 2017.

[50] B. Pinkas, T. Schneider, and M. Zohner. Scalable private set intersection based on OT extension. *ACM TOPS*, 21(2), 2018.

[51] Ponemon Institute LLC. The cost of credential stuffing: Asia-Pacific. https://www.akamai.com/us/en/multimedia/documents/white-paper/the-cost-of-credential-stuffing-asia-pacific.pdf, June 2018.

[52] S. Ramezanian, T. Meskanen, M. Naderpour, and V. Niemi. Private membership test protocol with low communication complexity. In *11th NSS*, volume 10394 of *LNCS*, August 2017.

[53] S. Riley. Password security: What users know and what they actually do. *Usability News*, 8(1), 2006.

[54] P. Rindal and M. Rosulek. Improved private set intersection against malicious adversaries. In *EUROCRYPT 2017*, volume 10210 of *LNCS*, pages 235–259, 2017.

[55] P. Rindal and M. Rosulek. Malicious-secure private set intersection via dual execution. In *24th ACM CCS*, October 2017.

[56] S. Schechter, C. Herley, and M. Mitzenmacher. Popularity *is* everything: A new approach to protecting passwords from statistical-guessing attacks. In *5th USENIX HotSec*, August 2010.

[57] S. Schechter, Y. Tian, and C. Herley. StopGuessing: Using guessed passwords to thwart online guessing. In *4th IEEE Euro S&P*, June 2019.

[58] Shape Security. 2018 credential spill report. https://info.shapesecurity.com/rs/935-ZAM-778/images/Shape_Credential_Spill_Report_2018.pdf, 2018.

[59] R. Shay, S. Komanduri, P. G. Kelley, P. G. Leon, M. L. Mazurek, L. Bauer, N. Christin, and L. F. Cranor. Encountering stronger password requirements: user attitudes and behaviors. In *SOUPS*, 2010.

[60] A. Smith. Americans and cybersecurity. https://www.pewinternet.org/2017/01/26/americans-and-cybersecurity/, January 2017.

[61] E. H. Spafford. OPUS: Preventing weak password choices. *Computers & Security*, 11(3), 1992.

[62] R. L. Stevenson. Strange case of Dr. Jekyll and Mr. Hyde. In M. A. Danahay, editor, *Strange Case of Dr. Jekyll and Mr. Hyde*. Broadview Press, 3rd edition, April 14 2015.

[63] S. Tamrakar, J. Liu, A. Paverd, J. Ekberg, B. Pinkas, and N. Asokan. The circle game: Scalable private membership test using trusted hardware. In *ACM ASIACCS*, 2017.

[64] K. Thomas, F. Li, A. Zand, J. Barrett, J. Ranieri, L. Invernizzi, Y. Markov, O. Comanescu, V. Eranti, A. Moscicki, D. Margolis, V. Paxson, and E. Bursztein. Data breaches, phishing, or malware? understanding the risks of stolen credentials. In *24th ACM CCS*, 2017.

[65] K. Thomas, J. Pullman, K. Yeo, A. Raghunathan, P. G. Kelley, L. Invernizzi, B. Benko, T. Pietraszek, S. Patel, D. Boneh, and E. Bursztein. Protecting accounts from credential stuffing with password breach alerting. In *28th USENIX Security*, August 2019.

[66] L. von Ahn, M. Blum, and J. Langford. Telling humans and computers apart automatically. *CACM*, 47(2):57–60, February 2004.

[67] C. Wang, S. T. K. Jan, H. Hu, D. Bossart, and G. Wang. The next domino to fall: Empirical analysis of user passwords across online services. In *8th CODASPY*, March 2018.

[68] D. Wang, H. Cheng, P. Wang, X. Huang, and G. Jian. Zipf's law in passwords. *IEEE TIFS*, 12(11):2776–2791, November 2017.

[69] K. C. Wang and M. K. Reiter. How to end password reuse on the web. In *26th ISOC NDSS*, February 2019.

A Exponential ElGamal Encryption

A cryptosystem that can be used to instantiate the specification of Sec. 5.2 is a variant of ElGamal encryption [23] commonly referred to as "exponential ElGamal" and implemented as follows (see, e.g., [11]). It uses an algorithm $\mathsf{ElGamalInit}$ that, on input 1^κ, outputs a multiplicative abelian group G of order r for a κ-bit prime r.

- $\mathsf{Gen}(1^\kappa)$ generates $G \leftarrow \mathsf{ElGamalInit}(1^\kappa)$; selects $u \xleftarrow{\$} \mathbb{Z}_r$; and returns a private key $sk = \langle u \rangle$ and public key $pk = \langle G, g, U \rangle$, where g is a generator of G, and $U \leftarrow g^u$.
- $\mathsf{Enc}_{\langle G,g,U \rangle}(m)$ returns $\langle V, W \rangle$ where $V \leftarrow g^v$, $v \xleftarrow{\$} \mathbb{Z}_r$, and $W \leftarrow g^m U^v$.
- $\langle V_1, W_1 \rangle +_{\langle G,g,U \rangle} \langle V_2, W_2 \rangle$ returns $\langle V_1 V_2 g^y, W_1 W_2 U^y \rangle$ for $y \xleftarrow{\$} \mathbb{Z}_r$ if $\{V_1, W_1, V_2, W_2\} \subseteq G$ and returns $\perp$ otherwise.
- $\mathsf{isZero}_{\langle u \rangle}(\langle V, W \rangle)$ returns *true* if $\{V, W\} \subseteq G$ and $W = V^u$, and returns *false* otherwise.

To use this cryptosystem in CUCKOO-PMT, it is necessary to test for ciphertext validity (line s1 and line r6). The next proposition shows that it suffices to test membership in G.

Proposition 5. *For the exponential ElGamal cryptosystem,* $C_{\langle G,g,U \rangle} = G \times G$.

Proof. $C_{\langle G,g,U \rangle} \subseteq G \times G$ follows by construction, and so we focus on proving $G \times G \subseteq C_{\langle G,g,U \rangle}$. For any $\langle V, W \rangle \in G \times G$, consider the group element WV^{-u} where $g^u = U$. Since g is a generator of G, there is a plaintext $m \in \mathbb{Z}_r$ such that $g^m = WV^{-u}$ and so $\langle V, W \rangle \in C_{\langle G,g,U \rangle}(m)$. □