How to (not) share a password: Privacy preserving protocols for finding heavy hitters with adversarial behavior

Moni Naor[*], Benny Pinkas[**], and Eyal Ronen(✉)[***]

Abstract. Bad choices of passwords were and are a pervasive problem. Users choosing weak passwords do not only compromise themselves, but the whole ecosystem. E.g, common and default passwords in IoT devices were exploited by hackers to create botnets and mount severe attacks on large Internet services, such as the Mirai botnet DDoS attack.
We present a method to help protect the Internet from such large scale attacks. Our method enables a server to identify popular passwords (heavy hitters), and publish a list of over-popular passwords that must be avoided. This filter ensures that no single password can be used to compromise a large percentage of the users. The list is dynamic and can be changed as new users are added or when current users change their passwords. We apply maliciously secure two-party computation and differential privacy to protect the users' password privacy. Our solution does not require extra hardware or cost, and is transparent to the user.
Our private heavy hitters construction is secure even against a malicious coalition of devices which tries to manipulate the protocol to hide the popularity of some password that the attacker is exploiting. It also ensures differential privacy under continual observation of the blacklist as it changes over time.
As a reality check we conducted three tests: computed the guarantees that the system provides wrt a few publicly available databases, ran full simulations on those databases, and implemented and analyzed a proof-of-concept on an IoT device.
Our construction can also be used in other settings to privately learn heavy hitters in the presence of an active malicious adversary. E.g., learning the most popular sites accessed by the Tor network.
Keywords: differential privacy, heavy hitters, passwords, secure computation, malicious model.

1 Introduction

We show a novel solution to the problem of privately learning heavy hitters in the presence of an active malicious adversary. Although we highlight several compelling use cases for our solution (see Section 1.2), we were particularly motivated by the problem of identifying and blacklisting popular passwords.

1.1 Passwords

The first use of a password in the modern sense was in 1961 in MIT's CTSS, and they are still ubiquitous today. It is well-known that users tend to choose very simple and predictable passwords,

[*] Department of Computer Science and Applied Mathematics, Weizmann Institute of Science Israel, Rehovot 76100, Israel. Supported in part by grant 950/16 from the Israel Science Foundation. Incumbent of the Judith Kleeman Professorial Chair. Email: `moni.naor@weizmann.ac.il`

[**] Department of Computer Science, Bar-Ilan University, Ramat-Gan, Israel. Supported by the BIU Center for Research in Applied Cryptography and Cyber Security in conjunction with the Israel National Cyber Bureau in the Prime Ministers Office, and by a grant from the Israel Science Foundation (no. 1018/16). Incumbent of the Alter Family Chair in Cyber Security. Email: `benny@pinkas.net`

[***] School of Computer Science, Tel Aviv University, Tel Aviv 76100, Israel. Supported in part by grant from the Israel Science Foundation (no. 1523/14). Member of CPIIS. Email: `er@eyalro.net`

with very little min-entropy. The use and reuse of easy to guess passwords is one of the top security threats for the internet [CC19]. Moreover, using an insecure password does not only endanger the user but also endangers other users. For example, using a compromised account to send spam mail or to perform a denial-of-service (DoS) attack. Although their demise has been announced many times (most notably in Bill Gates' famous 2004 speech), passwords are here to stay for the foreseeable future, especially in the IoT world.

Motivating Example: Web Cameras and the Mirai Attack Web cameras allow users to connect and watch live video streams over the Internet. Dedicated search engines such as Shodan [Wikb] and the recent Mirai attack, demonstrate how such devices can be found and hacked using default or popular passwords. In the Mirai attack, a huge number of compromised web-based cameras were used to mount a distributed DoS (DDoS) attack, sending a total of over 1Tbps and taking down large DNS servers [Wika]. This attack motivated many security experts to demand *liability* from the manufactures of such insecure products [Sch16].

Many IoT devices today (especially low-cost products) use passwords as a cheap and easy to use authentication solution. Alternative solutions (such as two-factor authentication) may increase cost and arguably hurt the usability of the system and are hardly used even in security-oriented services (e.g. less than 10% of Google's users [Tho18]). We need a cheap and user-friendly way to make the usage of passwords more secure not just for the single user, but to protect the Internet from such large scale attacks.

Many of the compromised cameras in the Mirai attack used a factory default password. There are some straightforward measures to prevent the usage of weak default passwords. The most basic of which is forcing a password change before the first use of their product. However, the response of many users might be to choose a simple password such as "123456". Such popular or easy passwords may be used in an amplified dictionary attack.[1]

Blacklisting Popular Passwords How to cause users to choose safe passwords is an open problem. For example, the National Institute of Standards and Technology (NIST) have recently changed their longstanding recommendation to require users to choose passwords including uppercase letters, symbols, and numerals in passwords [GGF17]. There are two types of weak passwords: One type is based on the user's private information (e.g a child's name and birthday). Such a password is vulnerable only to a targeted attack against the user. The second type is a popular password used by many different users. This type is much more dangerous, as the attacker does not need to know any auxiliary private data about the user, and can mount a large scale attack.

A promising approach is to blacklist any popular password, thus preventing any password from being used by more than a small τ fraction of the users. Although this might **not protect some individual users** from an attack, it will help **protect the whole ecosystem** from large scale attacks. In order to blacklist "popular" passwords a service provider needs to know the *current* popular passwords, which is hard for several reasons:

· User populations in different languages or locations (e.g. US vs. France) might use different popular passwords. Users in a small university town might choose their passwords based on the local sports team or mascot. Attackers can target their password guesses to the location or other

[1] A law approved in California in 2018 [32718], requires that connected devices either use a unique preprogrammed password or force the user to choose a new password before first access. We assume that users will prefer to choose the passwords and that they are likely to choose unsafe ones.

public properties of the victims. Therefore, the system must identify popular passwords along different locations (or other dimensions such as age, language, etc.). [2]

- When a system prevents the usage of certain popular passwords, other passwords become popular and should be identified as such (e.g. "password"$\rightarrow$ "Password"$\rightarrow$"Password1" $\rightarrow$"P@ssword!")
- There might be password trends that are based on trending topics (for example, more users today (2019) will prefer "wonderwoman" to "superman" as their password, not to mention "covfefe").

The simplest solution is for all users to send their passwords to a server, which will, in turn, identify and reject popular passwords. However, in that case, the server needs to save a list of all passwords or at the very least *unsalted* hashes of these passwords, which are vulnerable to a dictionary attack. This compromises the user's password privacy in case the server is breached, or in case of a subpoena. This also jeopardizes other sites, as users tend to reuse passwords between different sites and services.

Can publishing a password blacklist help attackers? The simple answer is that one should compare publishing the list of popular passwords to revealing a new bug or security weakness. Note that a returning user who enters his/her password can be alerted to the fact and asked to change it. Thus, though publishing the blacklist may put some users at risk, it helps protect the majority of users and the ecosystem from a large scale attack.

1.2 Use Cases

For concreteness, most of the discussion in this paper focuses on passwords for IoT devices. However, we also mention here other promising applications for the solutions suggested in this paper.

IoT service providers Our solution can be used to protect IoT devices against Mirai-like attacks, especially as IoT users tend to choose weak passwords and the dangers of large scale attacks are proven. The solution is very suitable to the IoT world, as it is low-cost and easy to implement. Moreover, it does not require the users to reveal their passwords to the IoT manufacturer who might not be trusted.

On-device user authentication Many devices or apps require users to authenticate themselves using a short PIN code or pattern that are kept *on the device* and are not sent to a service provider (e.g. unlocking smartphones, or in applications such as Snapchat's "My Eyes Only" secure cloud storage [Yun17]). Those PIN codes or patterns typically have less entropy than regular passwords.

Our solution lets service providers identify popular passwords while preserving the privacy and security of individual users (see exact definition in Section 2.1).

Tor network statistics The Tor network aims to provide better privacy and security for Internet users. To allow for a better understanding of users' needs and to advocate for better privacy, the Tor network provides researchers with statistics about its operation [LMD]. However, such statistics gathering must protect the users' privacy and be resilient to malicious users trying to influence the results. For example, we would like to enable researchers to *privately* learn the most popular websites visited by Tor users. A malicious repressive regime may want to delegitimize Tor, by making human rights sites seem less popular and drug trafficking sites more popular.

[2] Wang et al. [WCW+17] argued that it is not possible to accurately approximate different passwords databases using a *single* distribution.

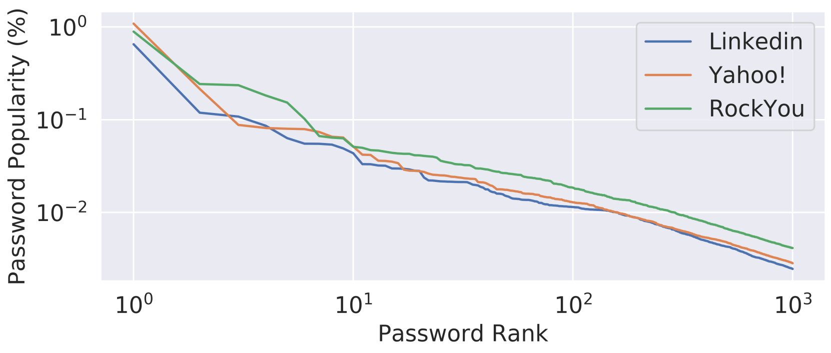

Fig. 1. Popular Password Distribution

Dynamic password statistics Our solution can be used by large service providers such as Google and Facebook to identify overly popular passwords among *specific subsets* of their user population. This procedure can be applied to specific targeted populations (e.g. by language, age range, favorite sports team, etc.) and be used to monitor passwords over time, again while protecting the individual users and without storing sensitive data such as unsalted hashes of the users' passwords.

1.3 Password Statistics Gathering: Desiderata

Our goal is to identify popular passwords while maintaining privacy. Such a system must satisfy several seemingly contradicting goals.

- The center (server) must learn the approximate distribution of the currently popular passwords, but without requiring the devices to divulge their passwords (even if the server is malicious).
- The center should publish, essentially to the world, some information about these passwords, but without this allowing a coalition of devices to learn anything (useful) about other devices' passwords. **This assurance must hold even after repeated publications of lists of popular passwords by different servers**.
- It is not realistic to run a secure multi-party protocol involving more than a single device, and any protocol between the server and a single device must be very lightweight.
- We only assume the existence of a single (untrusted) server and do not rely on an assumption that data sent outside the device is divided between several non-colluding parties.
- Even a coordinated **coalition of malicious devices** must not be able to affect the statistics in a significant manner, except in the obvious way: choosing their passwords. This is true both for **undercounting** the popular passwords and for **overcounting** (by more than one password per device in the coalition). The protocol should handle dynamic password changes and early stopping by the devices.

Communication Model: We work in the local model. We assume that the server executes the protocol with each device separately. The server maintains a state and communicates with each device at a different time over a **secure and encrypted channel**. There are no synchrony assumptions, and the communication between the server and the devices may be executed concurrently.

1.4 Our Contributions

To identify and blacklist popular passwords we propose a general scheme for privately identifying heavy hitters. It is the first such scheme which is secure against malicious adversaries and ensures differential privacy even under the continual observation of the dynamically changing heavy hitters' list. We define the privacy requirements of the passwords via a password guessing game and show its tight relationship to differential privacy. We bound the error probability both in the semi-honest and malicious settings, and give concrete examples for the possible parameters, calculated using a simulation. We describe two instantiations of the secure protocols for this task (both secure in the malicious setting). We show run times of a proof-of-concept (PoC) implemented on a sample platform, and describe the results of simulating the application of our private blacklisting scheme to three actual password databases.

We examined three lists of frequencies of passwords that were leaked, of users of Linkedin, Yahoo! and RockYou [BH19,Bon12,Wik19] which contain 174M, 70M, and 33M passwords respectively. Our theoretical analysis for an *arbitrary distribution* guarantees the blacklisting of the top 5, 3, and 5 passwords of these lists, respectively, and altogether blacklisting the passwords used by 1%, 1.4% and 1.4% of the users. The *Zipfian nature* of the password distribution (see Figure 1 and [WCW+17]) means that the popularity of the t most popular password is exponentially decreasing in t. This implies two things: we only care about blacklisting the top passwords (since they are much more popular than the other passwords), and we can do better than what our theorems guarantee. Indeed our simulation in Section 7.2 shows that we are very close to the "ideal password blacklisting process", one that gets the passwords of all users and blacklists the real top passwords. Our scheme performs closely to that ideal blacklisting process even when blacklisting the top 25 passwords. See Figure 3 for a comparison with the ideal system.

For the formal definition of our scheme's correctness and privacy properties see Section 2.4

1.5 Background

Differential Privacy (DP) Differential Privacy is a natural tool to discuss the limit on the amount of information that is learned about a user's password; see Dwork and Roth [DR14] for a survey of the field. Roughly speaking, a mechanism that outputs some aggregate statistic on a collection of users is differentially private if for any user, whether its data contributed to the aggregate statistic or not, cannot be determined *even a posteriori*. A little bit more precisely, it is ϵ-differentially private if for every possible outcome and every two adjacent inputs (differing in a single entry) the ratio between the probabilities of obtaining the output is bounded by e^{ϵ}. Two common techniques of achieving DP are adding *Laplacian noise* and randomized response. In the *randomized response* technique, to learn the prevalence of a property one asks responders to give a noisy answer regarding it. This affects the aggregate answer in a manner that allows retrieving, at least approximately, the real answer. One important property of differential privacy is that it is preserved under *post-processing*. Also note the posteriori nature of DP: the privacy is protected for any individual even if chosen after the release of the data (which would be false for a scheme that say samples a subset and releases all information about it).

Secure Computation: Secure two-party computation allows two mutually distrustful parties, each holding an input, to compute a function of their joint inputs without leaking any other information about their inputs.

Related Work The problem of finding the *heavy hitters* (i.e. popular items) of a collection of items in a differential private manner has been addressed in several settings, including when the input is given as a stream (see [DNP$^+$10,DNPR10,CSS11,CLSX12,HKR12,DNRR15]).

Blocki et al. [BDB16] showed an efficient differential private mechanism to securely publish a password frequency list of Yahoo!'s users. However, their mechanism requires the full frequency list as an input and does not solve the problem of how to learn it privately.

Bassily and Smith [BS15] have an efficient protocol in the local model (similar to our communication model) for finding heavy hitters, that is differentially private. Bassily et al. [BNST17] have recently provided an improved protocol wrt to the communication and computational complexity and in particular showed a technique for domain reduction on the set of items. Chan et al. [CSS] have given a lower bound of $\Omega(\sqrt{n})$ on the approximation accuracy in the local DP model (see Theorem 9.3 [Vad17]), which corresponds to our communication model.

However, none of these work in the adversarial setting with malicious users (e.g. trying to make a popular item disappear - undercounting see Section 3.4).

The RAPPOR project at Google [EPK14] is a good example of an application of local differential privacy to gather statistics about end-users (e.g. their homepages). It does not withstand malicious users who collude to make a homepage seem less popular.

Moran and Naor [MN06] considered *randomized response protocols* when some of the responders are trying to bias the result (see also Ambainis et al. [AJL]). This is the analogue of a malicious set of users who are trying to overcount or undercount the heavy hitters. They related the issue to oblivious transfer protocols.

One work that combines a privacy-preserving statistics collection scheme robust against malicious manipulation is that of Mani and Sherr [MS17]. They also proposed a histogram-based privacy-preserving statistics collection scheme robust against malicious manipulation in the context of a Tor network. However, their communication complexity is linear in the number of bins, compared to logarithmic in our protocol (for our example of 2^{16} bins they require 122 MB per client instead of 2 MB in our case). They also required non-colluding mix servers for privacy, while in our protocol the user does not need to trust anyone.

Schechter et al. [SHM10] proposed a method (data structure) to securely publish a list of popular passwords, with similar motivation to ours. However, they require users to reveal their passwords to the server and do not offer DP for the user upon password publication.

1.6 Paper Structure

Section 2 describes the security definitions and requirements. Section 3 explains the basic scheme, the usage of DP to protect the device, and the possible attacks on the system and required secure functionality in the malicious setting. Section 4 discusses how the server generates the list of popular hash values and proves the correctness of our scheme.

The next two (sections 5 and 6) describe two different methods for securely computing the required functionality in the malicious setting, based on garbled circuits, and the assumption on the intractability of quadratic residuosity (QR), respectively. The garbled circuit solution is more efficient both in run time and in bandwidth. On the other hand, it requires an interactive protocol. The QR based protocol demands more resources but has a non-interactive version.

Section 7 describes a PoC implementation of the QR-based protocol and simulation results of learning popular passwords. Section 9 discusses the results and raises open questions. Appendix A has a summary of frequently asked questions about the scheme.

2 Overview of Security Definitions

We start with describing a password security game (Section 2.1) that is used to properly define the privacy requirement regarding password protection (Section 2.2) and then go on with correctness requirements (Section 2.3).

2.1 The Password Security Game

There are two main parameters for a password guessing attack: one is L, the number of trials that the attacker is allowed to make. This number might be restricted either by computational power (e.g. brute-forcing a hashed password list) or by the number of allowed trials (e.g. an online account might be locked after 3 login attempts with a wrong password). The other parameter is γ, the probability of success.

The password security game: To analyze the effect that the proposed heavy hitters gathering system has on password security we consider the password security game $\mathbf{PGame}(L)$: we think of an attacker $\mathcal{A}$ that has some info about the system. Its move against a user (device) $\boldsymbol{D}$ is to publish a list of L words which are the attacker's guesses of the password of $\boldsymbol{D}$. If the password is one of the L words then the attacker wins. The attacker wishes L to be as small as possible and the probability of winning to be as large as possible.

Our definition of privacy will say that the attacker does not gain much in terms of the parameters of the attack as a result of witnessing the execution of the system.

Length/success probability trade-off: In general, there is a trade-off between the length L and the probability of success γ: if an attacker $\mathcal{A}$ can win $\mathbf{PGame}(L)$ with probability γ, then for any $0 \leq \rho \leq 1$ there is an attacker $\mathcal{A}'$ with the same information who can win $\mathbf{PGame}(\rho L)$ with probability $\rho \cdot \gamma$. This is true, since $\mathcal{A}'$ can imitate $\mathcal{A}$ and then simply sample ρL of the words in the list. If the list contains the password (which happens with probability γ), then the probability of successfully sampling it is ρ.

The effect of releasing one bit: Given any a priori information about the password, releasing a single bit function (even one chosen adversarially) about the password can increase the success probability by a factor of at most 2 (which is equivalent to making the list twice as long). More formally, for any Boolean function f, and for any attacker $\mathcal{A}$ that has some information about the system and also obtains $f(\mathit{password})$, and then plays $\mathbf{PGame}(L)$, there exists attacker $\mathcal{A}'$ that receives the same information about the system but not the value of $f(\mathit{password})$ and has at least the same probability of success in $\mathbf{PGame}(2L)$ (where the attacker generates a list that is twice as long). The argument is simple: run $\mathcal{A}$ twice, with the two possibilities for the value of $f(\mathit{password})$, and then concatenate the resulting lists.

Note that this sort of protection on its own might not be sufficient wrt system-wide considerations. In particular, the problem is that the adversary may decide on whom to prey *after* the fact, namely after the information is gathered and some leakage occurred. Consider the case where for each user there 10 possible passwords and the adversary guesses at random for each user a candidate password c and a single bit function f_c that isolates c from the rest of the bunch (this could even be a linear function). Now for each user, after receiving $f_c(\mathit{password})$ the adversary learns whether the candidate c is correct or not. For roughly 1/10th of the users, the adversary learns the password and can win the game $\mathbf{PGame}(1)$ with probability 1, without leaving a trace of this wrt the rest of the users. As we shall see, this is addressed by differential privacy (DP).

Differential privacy: An important issue is the relationship between the password security game and DP. Consider an adversary $\mathcal{A}$ that obtained ϵ-differentially private information regarding a password, and compare its probability of success to that of $\mathcal{A}'$ that did not receive this information at all. Then for any list length L, the success probability of $\mathcal{A}$ in $\mathbf{PGame}(L)$ is at most e^{ϵ} times the probability of $\mathcal{A}'$ in $\mathbf{PGame}(L)$. This follows from the immunity to post-processing of differential privacy. This holds similarly to $(\epsilon, \delta)-$DP, where δ should be added to the probability of success.

Looking at the above example of picking which devices to attack after the protocol, we see that the posteriori property of differential privacy prevents this case: the adversary cannot choose a bunch of users where he will win the password game $\mathbf{PGame}(L)$ with much higher probability. I.e. if the adversary received ϵ-DP information about many passwords, whenever the adversary chooses a device to attack the probability of success is at most e^{ϵ} times the probability of success of $\mathcal{A}'$.

2.2 Privacy and Password Security

The privacy requirements are that the chances of an adversary in winning the above Password Game with regards to a device (i.e. its password) do not increase by much if the individual device participates in the protocol (or participates with a fake-password). This should remain true against an adversary controlling a coalition of devices. Furthermore, this should remain true even a posteriori, that is when the adversary decides which devices to attack following the execution of the protocol[3].

We consider an adversary $\mathcal{A}$ that has a lot of power: it chooses a collection of distributions of passwords for the devices, witnesses the protocol (in the sense that it sees the communication with the devices and perhaps controls a subset of them) and then chooses one of the devices (not under its control) as the challenge device, receives the passwords of the remaining devices and participates in a Password Guessing game against the challenge device with parameter L. We want to say that this $\mathcal{A}$ does not succeed in the game much better than a player that did not participate in the protocol. We define the requirement by two parameters a and b related to whether the server cooperates with the adversary or not (these parameters should ideally be very close to 1).

Coalition without the server: For any adversary $\mathcal{A}$ as above, controlling a coalition of devices that does not include the server and after the execution of the protocol choosing another device as a challenge in the password game $\mathbf{PGame}(L)$ (and receiving the passwords of all other devices) there exists an $\mathcal{A}'$ that simply gets the passwords of all the devices except for that of the challenge device, but does not witness the execution of the protocol, so that A's probability of winning the game is at most a multiplicative factor a larger compared to that of $\mathcal{A}'$.

Coalition with the server: When the server joins the coalition (and may even behave maliciously in the protocol) then the probability of winning the game $\mathbf{PGame}(L)$ against the chosen device increases by at most a multiplicative factor of b compared to that of $\mathcal{A}'$ that simply gets the passwords of all the devices except the challenge device but does not witness the execution of the protocol.

[3] So, for instance, a scheme where some devices are sampled and full information about their passwords is given would be very bad in this respect.

2.3 Correctness Requirement

The correctness requirement is defined via the functionality of a trusted third party (TTP). The functionality is probabilistic and receives passwords one by one and at the end of the process releases the list of heavy hitters. Functionality 1 describes the desired functionality in terms of correctness.

The parameters of the system are τ, the threshold, and δ, an allowed tolerance. The TTP should return the correct label of the heavy hitter even in the presence of a malicious coalition of devices. Namely, for parameters τ and δ, the protocol must ensure that passwords with a frequency of at least $\tau(1+\delta)$ are added to the heavy hitters list, whereas passwords with a frequency of less than $\tau(1-\delta)$ are not added to this list (where $\tau, \delta, \tau(1+\delta) \in (0,1)$). There are also tolerable parameters for the probabilities of false negatives p_{FN} and false positives p_{FP}.

We think of the false negative probability threshold p_{FN} as being negligible, and therefore any frequent password in the above sense is labeled as such with probability at least $1 - p_{FN}$, where the probability is over the random coins of the server and the participants. On the other hand, the false positive probability threshold p_{FP} is not necessarily negligible: in particular, if we hash the passwords to a relatively small range with 16, 24 or 32 bits, then an 'innocent' password may simply hit a heavy hitter. If we have that for any password the probability of being disallowed is p_{FP}, this might not be so bad, since all it means that the human user will need to choose a new one. But what we do not want to happen is the case where the user has in his or her mind a sequence of potential passwords and *all* of them are falsely labeled as heavy hitters (so the user cannot successfully choose a password). Instead, we add the requirement that any large enough collection of passwords (of size at least M), where none of them is a heavy hitter, are not *simultaneously* labeled as such, except with negligible probability p_{total} (which is another parameter).

Functionality 1 (Protocol's Correctness requirement)

· Input:
- All users send their passwords to the trusted party (TTP).

· Blacklist generation:
- For each password *pass* $\in \{0,1\}^*$ with frequency of at least $\tau \cdot (1+\delta)$ we have that *pass* is added to the blacklist with probability at least $(1 - p_{FN})$.
- For each password *pass* $\in \{0,1\}^*$ with frequency of at most $\tau \cdot (1-\delta)$ we have that *pass* is added to the blacklist with probability at most p_{FP}.
- For any subset P of passwords of size at least M where all of passwords in P have frequency at most $\tau \cdot (1-\delta)$ we have that the probability that all passwords in P are added to the blacklist is at most p_{total}.
- A password with frequency between $\tau \cdot (1+\delta)$ and $\tau \cdot (1-\delta)$ may be added to the list arbitrarily.

· Output:
- The TTP sends to server the passwords in the blacklist.

The threshold τ is dynamically chosen as the smallest possible value satisfying the required error probability p_{FN}, with the current number of users N_C.

2.4 Correctness and Privacy Achieved

The protocols described in this work achieve correctness with parameters p_{FN}, p_{FP} and p_{total} as described above in the sense that the parameters can be set as a function of the number of participants and the privacy requirement. See Section 4.5.

As to privacy, any coalition that does not include the server can only increase its success probability in the Password Guessing Game $\mathbf{PGame}(L)$ by at most an $a = e^{\epsilon_{DP}}$ factor. In Section 4.3 we derive the value ϵ_{DP} from the scheme parameters. We achieve this by making sure that the password is protected in a differentially private manner; therefore the adversary cannot pick a device whose password was compromised. When the coalition includes the server this increase in success should be bounded by b. If the server learns a single bit regarding the password, as is the case in our protocols, then to make sure that b is indeed the bound we may use a simple randomized response technique by flipping this bit at random with probability $\ln 2/(b-1)$. The password is then protected in a differentially private manner and the corresponding consequences in the Password Guessing game.

"Everlasting" security and privacy: The proposed system should handle prolonged maintenance of these properties, that is the system lives for a long time and tries to learn the current list of heavy hitters. We will modify the requirements with some minor relaxations. The server needs to add ephemeral noise to each publication of the blacklist to protect the user's privacy and prevent leakage of data that is required for the protocol's secure functionality. Therefore, with regards to the published blacklist, we will allow a password that should appear in this list to be omitted in a single publication with probability $pEph_{FN}$, and for a password that should not appear in the list to appear in it with probability $pEph_{FP}$.

3 Overview of the Scheme

We start by explaining our domain reduction to hash values. We then describe the basic construction and provide details on how to fulfill properties such as differential privacy and security against malicious behavior. A rough outline of the basic scheme is that blacklisting the popular passwords can be reduced to learning a single bit via the Fourier coefficients of the distribution on the passwords' hash values. This is done by the server sending a random vector to the device, who responds with the inner product of the vector and the password's hash value. To overcome malicious behavior we need to perform this operation securely, without letting the user learn the server's random vector.

3.1 Password Domain Reduction

As we assume the password domain to be unbounded, we use a system-wide hash function H to map arbitrary length passwords to an ℓ-bit output (a typical output length will be $\ell = 16, 24$ or 32). Our scheme will find and publish a blacklist of heavy hitters *hash values*. A user's device can check if the hash value of a newly chosen password is in the blacklist. In that case, the user will be asked to choose a different password.

A collision in the hash value of a user's password and some "popular" password will cause a "false positive", where the user will be asked to change its password even though it is not popular. This happens with a low probability, $O(2^{-\ell}/\tau)$ where τ is the threshold frequency for a heavy hitter (see Section 4.3). We can tolerate these events since the only consequence of a collision is that a user will be asked to choose another password. We will therefore analyze the general problem of finding heavy hitters in ℓ-bit hash values.

3.2 The Basic Semi-honest Scheme

The scheme works in the following way each time a new device is added or change password:

1. Device j maps its password to a secret ℓ-bit value $v_j = H(\text{pass}_j)$.

2. The device receives from the server a uniformly distributed random ℓ-bit value r_j. The device sends back the one bit value of the inner product of v_j and r_j over $GF[2]$, denoted as $\langle v_j, r_j \rangle$.
3. The server keeps a table $T[x]$ of 2^ℓ counters, corresponding to all possible ℓ-bit values x (initialized to zero on system setup).
4. For every value of x if $\langle x, r_j \rangle = \langle v_j, r_j \rangle$ the corresponding counter is incremented by one, otherwise it is decreased by one. Equality holds for exactly half of the values, that we call the "correlated" values.

We denote the total number of unique users that ran the protocol as N_C, and p is the frequency of the hash value x. The expected numbers of increments and decrements are $N_C(p + (1-p)/2)$ and $N_C(1-p)/2$ respectively. The expected value of the counter is $E(T[x]) = pN_C$.

For a threshold frequency τ, the server simply publishes all x values such as $T[x] > \tau N_C$. Each device j can now check if $H(\text{pass}_j)$ is in the published hash values list. If it is, the device asks the user to change the password.

Note that to allow *password changes*, the server needs to save r_j and $\langle v_j, r_j \rangle$. If a device wants to change its password, the server will first reverse the change to the counters due to the old password before applying the new changes.

Running Time: This procedure requires 2^ℓ operations per update. This can be optimized using Goldreich-Levin's list decoding (see [Sud00]). However, the actual run time is insignificant for suitable values of ℓ such as 16, 24 or 32.

3.3 User Differential Privacy

The scheme leaks only one bit of information about the user's password. Although it seems that leaking one bit of entropy about a password would not affect its security by much (since passwords should have good min-entropy to begin with), in some cases even one bit might be too much. There are two different privacy concerns from the user's point of view:

1. Privacy from the server – Although some information must be leaked to the server for the scheme to work, users may want to have some differential privacy guarantees on the single bit they send to the server.
2. Privacy from third parties – Although a user may be willing to leak some information about his password to the server, we want to assure that this information does not leak to any coalition of users viewing the popular hash values list that is published by the server. This issue is amplified if the user participates in schemes for discovering popular passwords with several different services, and each of these services publishes a list of popular passwords.

Protection from the Server: Pure Differential Privacy by Applying Randomized Response The device can use a randomized response to reduce the amount of information that is leaked to the server. Namely, after hashing its password to v_j, the device decides with probability ϵ_r to choose a uniformly distributed random value v'_j, and send $\langle v'_j, r_j \rangle$ instead of $\langle v_j, r_j \rangle$. It holds that:

$$\Pr(\langle v'_j, r_j \rangle = \langle v_j, r_j \rangle \mid v'_j \neq v_j) = 1/2 \tag{1}$$

$$\Pr(v'_j = v_j) = 2^{-\ell} \tag{2}$$

From equations (1) and (2) we get that the probability of the server learning the correct bit $\langle v_j, r_j \rangle$ is $p_c = 1 - \epsilon_r(1 - 2^{-\ell})/2$. From this we can conclude that this procedure provides the device with pure DP, with $\epsilon \approx \ln(2/\epsilon_r - 1)$.

Protecting Privacy from other users: (ϵ_n, δ) Differential Privacy by Applying Laplacian Noise To ensure users that they have ϵ_n differential privacy from any coalition of users, the server can add independent Laplacian noise $Lap(1/\epsilon_n)$ to each table entry before deciding which hash values to blacklist (adding a device's password can change the value of each entry by at most 1).

We comment that this procedure might not be sufficient. The device can affect the publication probability of several passwords in the list. Moreover, the server needs to periodically republish its current hash value list. As we need to generate new noise each time we publish, an attacker can average the results over time and learn information about a single user's password. Even given this observation, we still retain DP as long as the number of hash value list publications is not very large. Dwork et al. [DRV10] have shown the advanced composition property that enables to get $O(\sqrt{k \log(1/\delta')} \cdot \epsilon_n, \delta')$-differential privacy for k publications. This means that the penalty we get in privacy is proportional to the square root of the number of publications, see Section 4.3.

3.4 The Malicious Setting

In a semi-honest model, the naive protocol suggested above is sufficient: the server sends r_j to the device and receives $\langle v_j, r_j \rangle$ (perhaps after the user applies the randomized response technique). The server cannot learn more than a single bit, and the device does not have anything to learn. However, in the malicious setting, the parties have different possible behaviors:

A Malicious Server The user/server protocol must ensure that the server does not learn more than a single bit about the password. This protects each user individually. However, on a system-wide scale, a malicious server can tweak the hash value list that is published. One option is to publish a very large list of popular passwords and cause a large number of false positives. This can be done to cause a DoS attack or to try and influence the users to choose from a smaller set of popular passwords. However, if τ is the popularity threshold, then the server can publish at most $O(1/\tau)$ popular passwords. Moreover, the server has no incentive to do a DoS attack on its own system.

Another option is to create a targeted malicious blacklist for each device. This attack can be prevented by publicly publishing the blacklist to all users.

A Malicious Device In the setting that we consider, it is very likely that some devices were compromised (perhaps, by utilizing their weak passwords) and are now controlled by an adversary. A coalition of malicious devices may try to influence the statistics gathered by the server in two ways. It can apply an *overcount attack* to make unpopular passwords seem more popular or apply an *undercount attack* to make popular passwords seem unpopular. Note that in our scheme, a single device can change the value of a counter by at most ± 1.

Undercount attack: This might be the most severe form of attack: A coalition of devices can try to "hide" some popular passwords, and cause "false negatives" to be able to continue exploiting these passwords. This attack may result in a larger fraction of the users using the same (weak) password and being susceptible to attacks. Assume that a corrupt device wants to cause an undercount of a popular password *pass*. Lets assume that $v_p = H(pass)$. The expected contribution to the counter $T[v_p]$ by a device that did not choose v_p is 0. However, by choosing v s.t. $\langle v, r_j\rangle = -\langle v_p, r_j\rangle$ the contribution is always -1. In this way, a β fraction of malicious users out of N_C can undercount a popular passwords counter by βN_C. This can cause the counter value to go below the threshold, and remove the hash value from the published list.

Overcount attack: A coalition of devices can try to make many passwords that have been chosen by less than a fraction τ of the users to seem more popular. This will result in a large number of "false positives". This attack is significant only if the attacker can apply it on a large fraction of the passwords, and reduce the total min-entropy. However, this attack is feasible only on a small number of passwords simultaneously and will have a negligible effect on the min-entropy. Moreover, the solution we present for the undercount attack is also effective against the overcount attack. Therefore we will focus only on the more severe undercount attack.

A Malicious Third party A third party might try to comprise the protocol or the published blacklist using a Man in the Middle attack on the connection. This is not possible, as we assume an encrypted and authenticated channel between the server and device. Moreover, the server will sign the publicly published blacklist including a timestamp (e.g. using HTTPS-protected website).

3.5 The Required Secure Functionality

To protect against malicious undercount attacks we need to prevent the possibility of sending an output bit that is anti-correlated to any value v (namely $1 - \langle v, r_{s,j}\rangle$) with probability greater than 1/2. We want to only allow the device to choose some value x and send $\langle x, r_{s,j}\rangle$. In Functionality 2 we define a functionality in the ideal model (where a trusted party is assumed to exist) which provides the desired privacy and correctness properties. The actual execution of the protocol (with no trusted party) must securely compute this functionality.

Functionality 2 (Ideal inner-product)

· Input:
- The server sends to the TTP an ℓ-bit input $r_{s,j}$.
- The device sends to the TTP an ℓ-bit input v.

· Output:
- The TTP sends to server the inner-product value $\langle v, r_{s,j}\rangle$.
- The device learns no output.

The definition implies that a device cannot learn $r_{s,j}$ before providing its input. The device is allowed to randomize its response, by choosing a random input v. As v is independent of $r_{s,j}$, the result of the inner-product is random.

In Sections 5 and 6 we show two secure protocols which implement Functionality 2 (or a small variant). The protocols are secure against malicious devices.

4 Popular Hash List Correctness

We describe how to generate and publish the blacklist of popular hash values using the inner-product results received from the devices. We prove the correctness by providing upper bounds on the probability of false negative and positive.

The basic scheme is secure against semi-honest devices. On the other hand, a coalition of malicious devices can force an *undercount* to a *specific* popular hash value rather simply and break the correctness (see Section 3.4). However, due to our secure functionality, a malicious device with no knowledge about the secret r_s vector, has no better strategy to undercount a hash value v than choosing a random hash value $v' \neq v$. This reduces the case to the semi-honest setting. We will prove the correctness of the scheme under this assumption. We also show how to calculate the minimum possible threshold τ_{min} preserving correctness as a function of the number of devices.

We also analyze correctness under the much stronger "Malicious Campaign" setting. In this setting, a malicious coalition of all of the devices tries to learn information about the secret r_s vectors (which are fixed per device in this setting) over several years to mount an undercount attack. This is done by adaptive runs of the protocol and learning information about the r_s vectors using leakage from the repeated publications of the blacklist. Using this information the coalition tries to undercount the hash value chosen by the small fraction of honest devices that were added before the last publication. Although this model might not seem realistic, it allows us to prove correctness in an information theoretical worst case. This allows us to calculate an upper bound on the slightly larger threshold value τ'_{min} that is required to preserve the correctness requirement in any setting.

4.1 Notation

We use the following notation in the analysis:

1. N_C - the total number of devices participating in the protocol.
2. $N_H(v)$ - the total number of devices that are currently using any password such that $v = H(pass)$.
3. $T[v]$ - the value stored at the counter table with index v.
4. $\alpha(v)$ - the server's approximation of the number of votes for a hash value v.
5. ϵ_r - the probability for randomizing a device's response (for DP).
6. $\mathbb{1}(x)$ - the unit step function.

All probability calculations are taken over the possible values of the r^s vectors used by the server and the devices' possible randomized responses.

4.2 Estimation of Popular Hash Values

The server's approximation to the number of devices that are currently using a password that hashes to a value v, given the current value of $T[v]$, is defined as:

$$\alpha(v) = (T[v] - N_C \epsilon_r 2^{-\ell})/(1 - \epsilon_r) \tag{3}$$

Lemma 1. *If a fraction p of the devices choose a password which hashes to a value v, then the expected value of $\alpha(v)$ is pN_C.*

The proof of this lemma appears in Appendix E.

4.3 False Negative and Positive Bounds

We want to identify hash values which are used by more than a fraction τ of the devices. Namely, identify any hash value v for which $N_H(v) > \tau N_C$. As we can only learn an approximation of $N_H(v)$, we relax our requirement, to that of identifying any hash value for which $N_H(v) > \tau N_C(1+\delta)$. We will also allow error on the other side – namely allow the server to blacklist hash values which are only used by more than $\tau N_C(1-\delta)$ devices (but no hash value which is used by fewer devices should be declared as popular).

We assume the malicious setting using our secure functionality.

Estimating the false negative probability: We define as a "false negative" the event that at a given time, a specific hash value v was over the upper threshold, but the approximation $\alpha(v)$ was below the threshold. Namely,

$$N_H(v) > \tau N_C(1+\delta) \quad \wedge \quad \alpha(v) < \tau N_C$$

Lemma 2. *The probability of a false negative event, p_{FN}, is bounded by:*

$$p_{FN} \leq 2\exp(-N_C(\tau\delta(1-\epsilon_r))^2/2)$$

Estimating the false positive probability: We define a false positive as the mirror case of a false negative, namely

$$N_H(v) < \tau N_C(1-\delta) \quad \wedge \quad \alpha(v) > \tau N_C$$

Lemma 3. *The probability for a false positive p_{FP} is bounded by:*

$$p_{FP} \leq \exp(-N_C(\tau\delta(1-\epsilon_r))^2/2)$$

Lemmata 2 and 3 are proved in Appendix E.

Note that as was described in Section 3.1 there is a larger chance of false positive on the original passwords due to collision in the hash values. This probability is calculated in Appendix E.

Dynamic Threshold $\boldsymbol{\tau}$ As N_C increases we can get a better approximation for the hash values distribution (namely, can use smaller τ values). The server can dynamically change τ as a function of N_C and its chosen bound on p_{FN}. Using Lemma 2 we propose that the minimal threshold τ_{min} will be bounded such that the following constraint holds for some constant C:

$$C < N_C(\tau_{min}\delta(1-\epsilon_r))^2/2 \implies \tau_{min} > \sqrt{\frac{2C}{N_C}} \cdot \frac{1}{\delta(1-\epsilon_r)}$$

This will assure a very low probability of false negatives even after publishing the hash list a polynomial number of times. For example, in a system with ten million users ($N_C = 10^7$), $\delta = 0.8$, $\epsilon_r = 0.25$ and $C = 7$, we get that $\tau_{min} = 0.002$. We can also use a numerical approach to calculate p_{FN} and find a lower more accurate τ_{min}.

Laplacian Noise We use Laplacian noise to get (ϵ_{DP}, δ') differential privacy from third parties. Before each publication, the server adds to each counter $T[x]$ independent noise $\chi_L \sim Lap(1/\epsilon_n)$, and blacklists a hash value x only if $T[x] + \chi_L^x > \tau N_c$.

Remember that the maximal number of possibly published counters in each publication is $1/(\tau(1-\delta))$. So we get that for k repeated publications of the blacklist we can get (ϵ_{DP}, δ') differential privacy where $\epsilon_{DP} = \sqrt{k/(\tau(1-\delta)) * \ln 1/\delta'} \cdot \epsilon_n$. Using a τ value slightly larger than τ_{min} will allow us to add the required Laplacian noise while still keeping the value of $pEph_{FN}$ negligible.

4.4 Bounds for a "Malicious Campaign"

In a malicious setting, a coalition of devices might run an undercount attack, trying to cause a false negative wrt an actual popular password. The system, however, uses a secure inner-product protocol that is resilient to malicious devices (see Sections 5 and 6). In a *"one-shot"* scenario, where the attacker has no information about the secret r_s vectors, we get the same probability p_{FN} as in the semi-honest case (best strategy for the coalition: choose random hashes).

The publication of the list of popular hash values leaks a small amount of information about the secret r_s values. This information might be used in an undercount attack to cause false negatives with higher probability. For example, a device might learn from a change in the published hash list that its chosen hash value is correlated with a popular hash value. Therefore the device chooses another hash value in the hope that the result will be anti-correlated.

The server can add independent random noise to the counter table before each publication of the blacklist. In this way, the server reduces the amount of leaked data. We conjecture that even a small amount of added noise (e.g. the Laplacian noise used for DP) will render the leaked data unusable for the attacker. However, we provide a full information-theoretical worst-case analysis.

We assume that for the entire life span of the system, a malicious coalition of all devices found an optimal adaptive strategy to cause each publication to leak the maximum possible (in an information-theoretic sense) number of bits about the secret r_s vectors. Moreover, we assume that this coalition has an optimal strategy to use this *noisy* information to improve their undercount probability by the maximal theoretically possible factor. In Part B of the Appendix we prove the scheme's correctness under those assumptions.

4.5 Security and Correctness Analysis

We need to show that the desired privacy property expressed by the password game of Section 2.1 and 2.2 indeed holds, and show that the correctness property of Section 2.3 holds as well. The privacy property follows from the randomized response giving us ϵ differential privacy protection against a malicious server, and from the (ϵ_{DP}, δ') differential privacy protection given to each password regarding any malicious coalition that does not include the server, as shown in Section 4.3.

As for correctness, in Section 4.3 we show how to bound p_{FN}, and how to dynamically (i.e. as the number of participants increases) choose the parameters to ensure this requirement. Figure 2 shows the τ_{min} values for $\delta = 0.8$, $\epsilon_r = 0.25$ and $C = 7$. Parts B.4 and B.5 of the supplementary material show the same bounds in the worst case "Malicious Campaign"setting. A suitable choice of slightly larger $\tau' > \tau_{min}$ and ϵ_n will give us the required $pEph_{FN}$ and $pEph_{FP}$.

Regarding p_{total}, we prove in Appendix E that the only event that should worry us is the occurrence of M collisions to popular hash values under the hash function. This happens with a very small probability of $(2^{-\ell} \,/\, \tau(1-\delta))^M$.

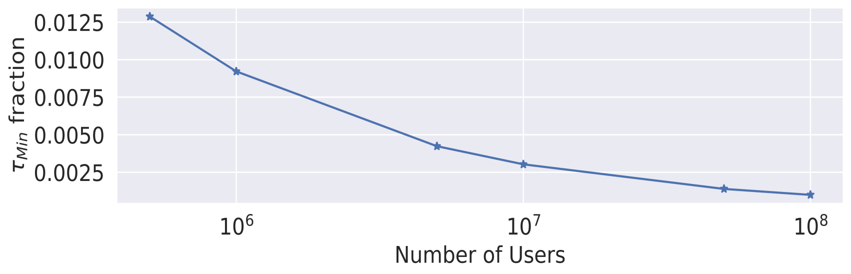

Fig. 2. τ_{min} as a function of the number of users

5 Garbled Circuits Based Inner Product Protocol

Generic protocols for secure computation, such as Yao's garbled circuit protocol, can be used for securely computing any function. The Binary circuit computing the inner-product of ℓ-bit values is very small, and is composed of only ℓ AND gates and $\ell - 1$ exclusive-or gates. The secure computation of this circuit is extremely efficient.

The main challenge in using garbled circuits in our application is ensuring security against malicious behavior. The most common approach for achieving security against malicious adversaries is the cut-and-choose paradigm (see, e.g. [LP07]), but this paradigm requires computing at least $\Omega(s)$ copies of the circuit to reduce the cheating probability to 2^{-s} [4].

Fortunately, our setting enables a much more efficient implementation. This is based on two features existing in this setting:

1. The device does not receive any output from the computation.
2. The server is allowed to learn a single bit about the input of the device (this feature is a byproduct of learning the inner-product, but we can also use it for designing a more efficient two-party protocol).

In our implementation, the server is the constructor of the circuit, and will also learn the output of the computation. We make a minor change to Yao's protocol to enable the device to verify that the circuit outputs only a single bit. The device cannot, however, know which function is computed by the circuit (the device can verify the circuit's topology, but not the function computed). The server can therefore use the circuit to learn other one-bit functions of the input of the device. This is still fine by our security requirements, but we need to re-define the ideal functionality to model this computation. The modified functionality appears in Functionality 3.

The protocol we use is based on the observation that running the basic Yao protocol, which is only guaranteed to provide security against semi-honest adversaries, is actually also secure against malicious circuit evaluators (this was also used in the dual execution paradigm [MF06], and follow up work [KMRR15,RR16]).

The parties run Yao's semi-honest protocol, with two modifications:

1. The oblivious transfers are secure against malicious adversaries. They can be implemented, e.g. using the protocols of [PVW08,CO15], whose overhead consists of only a few exponentiations.

[4] We are only interested in running a single computation, and therefore there is no benefit in using protocols which reduce the amortized overhead of cut-and-choose over many computations of the same function, as in [HKK$^+$14,LR15].

Functionality 3 (Yao's protocol modified inner-product)

· Input:
- The server sends to the TTP the following inputs:
  - ∗ An ℓ-bit input r^s_j.
  - ∗ A circuit computing a function F which receives two ℓ-bit inputs and outputs a single bit.
- The device sends to the TTP an ℓ-bit input v.

· Output:
- The output of the server is $F(h, r^s_j)$.
- The device learns no output.

2. The server provides in advance the output of a collision-resistant hash function applied to the two possible garbled outputs of the circuit. The device verifies, before sending the garbled output to the server, that the garbled output hashes to one of these values. (This guarantees that the circuit can have only two valid outputs.)

We assume that the reader is familiar with Yao's garbled circuit protocol. The protocol we suggest includes the following steps:

1. The server prepares a garbling of a circuit computing the inner product of two ℓ-bit inputs. In addition, it applies the hash function to the two garbled values of the output wire of the circuit and records the results.
2. The parties run ℓ invocations of 1-out-of-2 oblivious transfer, one for each of the ℓ input wires corresponding to the device's input. The server is the sender, and in each OT the device learns one of the two garbled values of the corresponding input wire. The oblivious transfers are implemented using an oblivious transfer protocol that is secure against malicious adversaries.
3. The server sends to the device (1) the garbled tables of all gates, (2) the garbled values corresponding to the input wires of the server, and (3) the hash values of the two possible garbled values of the single output wire (sent in random order).
4. The device evaluates the circuit's garbled output value, and compares the hash of this value to the two hashes of the output wire sent by the server. If there is a match then it sends the garbled value to the server.
5. The server receives the garbled output value and translates it to the 0/1 value of the inner product.

Overhead. We note that the circuit size is very small ($2\ell - 1$ gates, where ℓ is typically equal to 16, 24 or 32), whereas current implementations of Yao's protocol can process millions of gates per second. The run times of each oblivious transfer is at most a few milliseconds (see, e.g., [CO15]).

Security analysis of the protocol The full security analysis of the protocol can be found in Appendix C.

6 Quadratic Residuosity Based Inner Product Protocol

The goal of the protocol is to implement the secure functionality described in Figure 2 and calculate the inner product $\langle \boldsymbol{V_s}, \boldsymbol{R_s} \rangle$ between a device's ($\boldsymbol{D}$) input $\boldsymbol{V_s}$ and a server's ($\boldsymbol{S}$) input $\boldsymbol{R_s}$ in a malicious setting. This functionality requires that $\boldsymbol{S}$ only learns the result, while $\boldsymbol{D}$ learns nothing and is not able to deviate from the protocol.

We use the following notation regarding a composite number $N = p \cdot q$:

1. $\left(\frac{a}{N}\right)$ is the Jacobi symbol of a with respect to N (which is easy to calculate).

2. QR_N (nQR_N) is the set of all numbers that are (non) quadratic residues modulo N, and have a Jacobi symbol 1.
3. $T_{QR_N}(a)$ tests the quadratic residuosity of a in respect to N. The output is $\perp$ if $\left(\frac{a}{N}\right) \neq 1$. Else the output is 0 if $a \in QR_N$ and 1 otherwise. The quadratic residuosity assumption implies that computing this function requires the knowledge of p and q.
4. The notation $\overset{R}{\leftarrow}$ denotes that an item is chosen uniformly at random from a specific domain. In particular,
   - $a \overset{R}{\leftarrow} QR_N$ (nQR_N) denotes that the value a is chosen uniformly at random from QR_N (nQR_N).
   - $\pi_a \overset{R}{\leftarrow} \pi\{\ell\}$ denotes that the permutation π_a is chosen uniformly at random from the family of permutations on ℓ items.
   - $\boldsymbol{x} \overset{R}{\leftarrow} \{0,1\}^\ell$ denotes that the value x is chosen uniformly at random from $\{0,1\}^\ell$.
5. Capital letters such as $\boldsymbol{V_s}$ denote vectors of numbers in $\{0,1\}$, lowercase letters such as $\boldsymbol{r_p}$ denote vectors of numbers in $\mathbb{Z}_N$.

The protocol is based on the *intractability of the quadratic residuosity (QR) assumption.* Under this assumption it is hard to distinguish whether $x \overset{R}{\leftarrow} QR_N$ or $x \overset{R}{\leftarrow} nQR_N$ where $\left(\frac{x}{N}\right) = 1$ (with non-negligible advantage) without knowledge of the factorization of N.

We use this assumption for encrypting vector $\boldsymbol{R_s}$ of the server $\boldsymbol{S}$ in a similar way to the famous Goldwasser-Micali public encryption scheme [GM84]. In particular, we use the *homomorphic properties* of this scheme – i.e. if $x, y \in QR_N$ and $a, b \in nQR_N$ then $xy, ab \in QR_N$ and $ax \in nQR_N$. This also allows us to 'blind' an encryption: we create a new encryption which is decrypted to the same bit but cannot be connected to the original one by multiplying with a random square.

6.1 Protocol - Semi-honest Version

We will start by describing the semi-honest version of the protocol. For each device $\boldsymbol{D}$, the server $\boldsymbol{S}$ generates a unique public and secret key pair $SK = (p, q), PK = N = p \cdot q$ (p and q are primes and $|N|$ depends on a security parameter k_1). $\boldsymbol{S}$ then encodes $\boldsymbol{R_s}$ as a public vector $\boldsymbol{r_p} \in \mathbb{Z}_N^\ell$ that is sent to $\boldsymbol{D}$. Each number $r_{p,i}$ is a public-key encryption of the corresponding bit $R_{s,i}$:

- 0 is encoded as a QR: if $R_{s,i} = 0$: $r_{p,i} \overset{R}{\leftarrow} QR_N$.
- 1 is encoded as a nQR: if $R_{s,i} = 1$: $r_{p,i} \overset{R}{\leftarrow} nQR_N$.

$\boldsymbol{D}$ calculates the product of the numbers in $\boldsymbol{r_p}$ corresponding to the 1 bits in $\boldsymbol{V_s}$, and blinds the result by multiplying it with a random QR:

$$e = d^2 \cdot \prod_{i=1}^{\ell} (r_{p,i})^{V_{s,i}} \quad \text{where} \quad d \overset{R}{\leftarrow} \mathbb{Z}_N$$

After receiving e from $\boldsymbol{D}$, $\boldsymbol{S}$ learns the result of the inner product:

$$result = T_{QR_N}(e)$$

Under the QR, assumption $\boldsymbol{D}$ does not learn anything (in the semantic security sense) about $\boldsymbol{R_s}$. Due to the homomorphic properties of the encryption, a product of encoded bits gives an encoding for the xor of the bits. We get that e encodes the inner product result.

The blinding by d^2 does not change the result (it is equivalent to an exclusive-or with zero). However, because d is chosen uniformly at random in QR_N, then e is also distributed uniformly at random inside its class (either QR_N or nQR_N). This ensures that the $\boldsymbol{S}$ only learns the one-bit result of the inner product.

Insecurity in the malicious setting? The attack we wish to prevent is anti-correlation with a certain password (or passwords). Applying this attack is easy given a member of nQR_N, since multiplying the answer with an nQR_N number flips the result[5]. However, under the QR assumption, we do not know if it is easy or hard to find a single number x such that $x \in nQR_N$ (with probability better than 1/2). By assuming that finding $x \in nQR_N$ for certain N's is indeed hard we do not need to change the semi-honest protocol to handle the malicious case, but we take the conservative path of not adding new assumptions.

6.2 Malicious Setting: Naive Implementation

The basic approach for handling the malicious case is to add a zero-knowledge proof by the device $\boldsymbol{D}$ to prevent malleability attacks. We begin the exposition with a naive implementation that is secure against a malicious $\boldsymbol{D}$, but allows $\boldsymbol{S}$ to learn more than one bit of information about $\boldsymbol{V_s}$.

$\boldsymbol{D}$ creates a vector $\boldsymbol{r_*}$ that is a permuted and "blinded" version of the vector $\boldsymbol{r_p}$ that it received from the server. That is, the device blinds each value in $\boldsymbol{r_p}$ by multiplying it with a random QR_N and randomly permutes the resulting vector. $\boldsymbol{D}$ then sends to $\boldsymbol{S}$ the list of indexes of the numbers in $\boldsymbol{r_*}$ that correspond to the 1 bits in $\boldsymbol{D}$'s vector. $\boldsymbol{S}$ can then compute the multiplication of these numbers.

Since $\boldsymbol{S}$ must not learn how many bits equal to 1 in $\boldsymbol{D}$'s vector, we pad $\boldsymbol{r_p}$ with ℓ numbers that are in QR_N. This allows $\boldsymbol{D}$ to always send a product of exactly ℓ numbers regardless of the Hamming weight of its vector $\boldsymbol{V_s}$.

$\boldsymbol{D}$ proves to $\boldsymbol{S}$ that $\boldsymbol{r_*}$ is indeed a blind permutation of $\boldsymbol{r_p}$. This is done by generating a vector $\boldsymbol{r_j}$ that is a blinded permutations of $\boldsymbol{r_*}$, and proving to $\boldsymbol{S}$ that $\boldsymbol{D}$ knows how to "open" this vector either to $\boldsymbol{r_*}$ or to $\boldsymbol{r_p}$ (this is in the style of the zero-knowledge proof of Hamiltonicity). A cheating $\boldsymbol{D}$ is caught with probability 1/2. $\boldsymbol{D}$ must therefore use k_2 vectors $\boldsymbol{r_j}$, and then the probability of $\boldsymbol{D}$ to deviate from the protocol is 2^{-k_2}.

Device setup Phase $\boldsymbol{D}$ generates and saves the following data.

1. $\boldsymbol{r} = \boldsymbol{r_p}||\boldsymbol{pad}^2$ where $\boldsymbol{pad} \overset{R}{\leftarrow} \mathbb{Z}_N^{\ell}$. We pad $\boldsymbol{r_p}$ with ℓ numbers in QR_N.
2. $\boldsymbol{V} = \boldsymbol{V_s}||\overline{\boldsymbol{V_s}}$ where $\overline{V_{s,i}} = 1 - V_{s,i}$.
3. $\pi_* \overset{R}{\leftarrow} \pi\{2\ell\}$, $\boldsymbol{d_*} \overset{R}{\leftarrow} \mathbb{Z}_N^{2\ell}$. $\boldsymbol{D}$ generates a random permutation π_* on 2ℓ items, while $\boldsymbol{d_*}$ are 2ℓ random numbers.
4. $\boldsymbol{r_*} = \pi_*(r \cdot (\boldsymbol{d_*})^2)$, $\boldsymbol{V_*} = \pi_*(\boldsymbol{V})$. $\boldsymbol{r_*}$ is a blinded random permutation of $\boldsymbol{r}$, and $\boldsymbol{V_*}$ is the same permutation of $\boldsymbol{V}$.
5. $e = \prod_{i=1}^{2\ell}(r_i(d_{*,i})^2)^{V_i} = \prod_{i=1}^{2\ell}(r_{*,i})^{V_{*,i}}$. e is the product of the blinded items in $\boldsymbol{r}$ corresponding to the 1 bits in $\boldsymbol{V}$. The bit that is encoded is the result of the inner product.
6. For $j = 1 \ldots k_2$:

[5] That is, if e is the appropriate answer for the password that a malicious $\boldsymbol{D}$ wishes to undercount, then by sending $e' = e \cdot x$ the count is decreased

(a) $\pi_j \overset{R}{\leftarrow} \pi\{2\ell\}$, $\boldsymbol{d_j} \overset{R}{\leftarrow} \mathbb{Z}_N^{2\ell}$. $\boldsymbol{D}$ generates a random permutation π_* on 2ℓ items, while $\boldsymbol{d_*}$ are 2ℓ random numbers.

(b) $\boldsymbol{r_j} = \pi_j(\boldsymbol{r_*} \cdot (\boldsymbol{d_j})^2)$ is a blinded random permutation of $\boldsymbol{r_*}$

7. $\boldsymbol{D}$ sends $(\boldsymbol{r_*}, \boldsymbol{V_*}, \boldsymbol{pad})$ to $\boldsymbol{S}$.

Interactive proof phase The server first computes the result.

1. $\boldsymbol{S}$ calculates $\boldsymbol{r} = \boldsymbol{r_p}||\boldsymbol{pad}^2$ using the vector $\boldsymbol{pad}$ it received from $\boldsymbol{D}$ and $\boldsymbol{r_p}$ that it stored locally.
2. $\boldsymbol{S}$ calculates $e = \prod_{i=1}^{2\ell}(r_{*,i})^{V_{*,i}}$. Namely, e is the product of the items in $\boldsymbol{r_*}$ corresponding to the 1 bits in $\boldsymbol{V_*}$.
3. $\boldsymbol{S}$ calculates $result = T_{QR_N}(e)$, to retrieve the inner product.

Then the server verifies the result using an interactive proof for $j = 1, \ldots, k_2$:

1. $\boldsymbol{D}$ sends $\boldsymbol{r_j}$ to $\boldsymbol{S}$
2. $\boldsymbol{S}$ sends $b_j \overset{R}{\leftarrow} 0, 1$ to $\boldsymbol{D}$
3. if $b_j = 0$
   (a) $\boldsymbol{D}$ sends to $\boldsymbol{S}$ $(\pi_{*,j} = \pi_j\pi_*, \boldsymbol{d_{*,j}} = \pi_j(\pi_*(\boldsymbol{d_*}) \cdot \boldsymbol{d_j})$, opening the blinded permutation from $\boldsymbol{r}$ to $\boldsymbol{r_j}$.
   (b) $\boldsymbol{S}$ verifies that $\boldsymbol{r_j} = \pi_{*,j}(r) \cdot (\boldsymbol{d_{*,j}})^2$.
4. else
   (a) $\boldsymbol{D}$ sends to $\boldsymbol{S}$ $(\pi_j, \boldsymbol{d_j})$, opening the blinded permutation from $\boldsymbol{r_*}$ to $\boldsymbol{r_j}$.
   (b) $\boldsymbol{S}$ verifies that $\boldsymbol{r_j} = \pi_j(\boldsymbol{r_*} \cdot \boldsymbol{d_j}^2)$.

Information leakage in the naive implementation $\boldsymbol{S}$ receives a blinded version of the numbers used in the inner product calculation (the numbers in $\boldsymbol{r_*}$ corresponding to the '1' bits in $\boldsymbol{V_*}$). However, recall that $\boldsymbol{S}$ can find the sign of $T_{QR_N}(x)$ for any x. This lets it learn the exact number of 1 bits in the device's vector $\boldsymbol{V_s}$ that correspond to 1 bits in the server's vector $\boldsymbol{R_s}$. For example, $\boldsymbol{S}$ can choose $\boldsymbol{R_s}$ to be all '1' and use it to learn the Hamming weight of $\boldsymbol{V_s}$.

6.3 The Full Protocol

The main difference between the full and naive protocols is that instead of sending the vector $\boldsymbol{r_*}$, $\boldsymbol{D}$ sends a vector $\boldsymbol{s_*}$ containing the *squares* of the entries of $\boldsymbol{r_*}$, and also the indexes of the entries in $\boldsymbol{r_*}$ which correspond to $\boldsymbol{D}$'s 1 bits, and the result e of multiplying these entries. As all the numbers in $\boldsymbol{s_*}$ are in QR_N, the vector that $\boldsymbol{D}$ sends is indistinguishable from a random vector in $QR_N^{2\ell}$ and leaks no information. Nonetheless, the server can verify that e is indeed the square root of the multiplication of the ℓ elements in $\boldsymbol{s_*}$ which correspond to 1 bits. We prove that under some restriction on the generation of N the soundness of the modified protocol holds. The complete protocol is described in Appendix F.

Non-interactive proof The interactive part of the protocol can be converted into a non-interactive proof. In addition, the results of $\boldsymbol{S}$ setup can be stored on $\boldsymbol{D}$, and this allows the implemented protocol to include only a single message from $\boldsymbol{D}$ to $\boldsymbol{S}$.

The conversion is done using the Fiat-Shamir heuristic [FS86] in the (non-programmable) random oracle model.

Completeness, Soundness and Zero-Knowledge Proof The full proofs can be found in Appendix F.

6.4 Implementation Considerations

Reusing $\boldsymbol{r_p}$ The vector $\boldsymbol{r_p}$ can be re-used for repeated runs of the protocol, without any effect on the security from $\boldsymbol{S}$'s viewpoint. From $\boldsymbol{D}$'s side, re-using $\boldsymbol{r_p}$ helps by limiting the number of password bits that can be leaked to one, even if the same password is chosen multiple times (e.g. if $\boldsymbol{D}$ was reset, or the user changed back to an old password).

As $\boldsymbol{r_p}$ is fixed, it can be stored on $\boldsymbol{D}$ in production and not sent over the network. This saves the first message in the protocol.

Interactive vs. Non-Interactive As was mentioned above, the protocol can be turned into a non-interactive protocol, allowing $\boldsymbol{D}$ to prepare and send a single message. $\boldsymbol{S}$ can then verify and add the result to the gathered statistics with minimal interaction.

This advantage comes with a price. In the interactive setting k_2 is a parameter of the probability of deviating from the protocol $p(k_2) = 2^{-k_2}$. For all practical usages setting k_2 to be between 10 and 30 should be more than sufficient. However, in the non-interactive proof k_2 is a parameter of the amount of preprocessing work necessary for deviating from the protocol ($O(2^{k_2})$), and we will usually require a much larger value of k_2 (e.g. 80 or 128).

Although the non-interactive protocol is simpler, using a larger value of k_2 is more suitable for IoT devices with sufficient memory and processing power, such as web cameras. Low-performance or low-power devices should use the interactive protocol.

Communication complexity We calculate the number of bits that are sent in the protocol. As we assume $\boldsymbol{S}$ generates a fixed $\boldsymbol{r_p}$ per $\boldsymbol{D}$, we will only consider the size of $\boldsymbol{D}$'s proof (and neglect the k_2 bits sent by $\boldsymbol{S}$).

The size of the representation of each permutation is $2\ell \cdot |2\ell| = 2\ell \cdot (\log \ell + 1)$. The size of each vector is $|(\boldsymbol{s_*}, e, \boldsymbol{V_*}, \boldsymbol{pad})| = k_1 \cdot (2\ell + 1 + \ell) + 2\ell \approx 3\ell k_1$, $|(\boldsymbol{s_j}, \pi^{j*}, \boldsymbol{blind_j})| = |(\boldsymbol{s_j}, \pi^j, \boldsymbol{d_j})| = 4k_1\ell + 2\ell \cdot (\log \ell + 1) \approx 4\ell k_1$.

The total communication complexity is approximately $4\ell k_1 k_2$. For specific parameters of the non interactive protocol, $\ell = 16, k_1 = 2048, K_2 = 128$, it is about 2 MB, and for the interactive protocol with $k_2 = 20$ about 312 KB.

7 Proof of Concept Implementation

7.1 QR Protocol PoC

We implemented a proof-of-concept (PoC) of our maliciously secure QR protocol in python, showing that even an unoptimized implementation is sufficiently efficient for many IoT devices. To simulate an IoT device, we ran the device-side code on a Raspberry Pi 3 Model B [ras], and the server code on an Intel i7-7500U 2.7GHz CPU.

We use a modulus N of size 2048 bits and $\ell = 16$. We measure processing time of the protocol with $k_2 = 20$ (interactive version) and $k_2 = 128$ (non-interactive version).

The device required 2.8 seconds of run time and 13.5 seconds of preprocessing for $k_2 = 128$, and 0.4 seconds of run time and 2.1 seconds of preprocessing for $k_2 = 20$. The server required about 0.5 sec to verify the $k_2 = 128$ non-interactive proof, and about 3msec to update the counters.

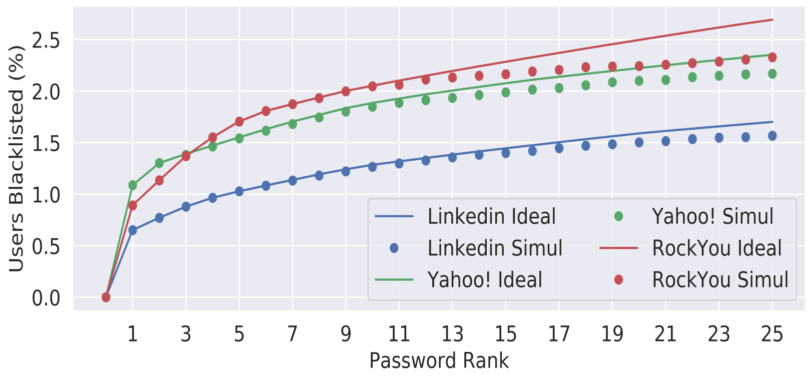

Fig. 3. A Comparison of the numbers of users blacklisted by our scheme and by an ideal blacklisting process.

7.2 Popular Hash List Simulation

We simulated our password blacklisting scheme using three lists of frequencies of passwords that were leaked, of users of Linkedin, Yahoo! and RockYou [BH19,Bon12,Wik19], which were of sizes of 174M, 70M, and 33M passwords respectively.

We ran the simulation 150 times for each database. In each run of the simulation, the passwords were hashed to random 16-bit values, and the protocol was simulated between each "user" and the server. For supporting local differential privacy, each user's answer was randomized with probability $\epsilon_v = 0.25$.

We compare the success of blacklisting passwords using our scheme to the success of an "ideal" blacklisting of passwords which has the entire list of passwords of all users. We assume that the server decides on a threshold t and blacklists the top t popular passwords. We measure the percentage of users whose password is blacklisted by the server. For our simulation, we take the median result among the 150 runs of the simulation.

Figure 3 shows the percentage of users whose password is blacklisted, as a function of the number t of top popular passwords that are blacklisted. The results of our simulation are compared to those of an "ideal blacklister" which blacklists the actual t most popular passwords. Note that due to the Zipf distribution of passwords, the utility of blacklisting each additional password is sharply decreasing. Therefore it is only needed to examine the effect of blacklisting a relatively small number of passwords.

When blacklisting up to the $t = 8$ most popular passwords, the results of the simulation are identical to the ideal blacklisting. When blacklisting more passwords, the results of the simulation are very close to the ideal run. For example, when blacklisting the top $t = 25$ passwords, applying the simulation to the Linkedin database blocked the passwords of 92% of the users whose passwords were blacklisted by the ideal blacklister. The simulation for the smaller RockYou database blacklisted the passwords of 86% of the users that were blocked by the ideal simulation.

8 Discussion and Open Questions

Tradeoff between the QR and garbled circuit solutions: The garbled circuit solution is more efficient both in run time and in bandwidth. On the other hand, it requires an interactive protocol and generating a new r value for each password change. The QR based protocol demands more resources but has a non-interactive version that only requires the device to prepare and send a single message to the server reusing the same r.

Implementation for the Tor Network: We believe our protocol can be useful for private statistics gathering in the Tor network. This requires working with the Tor project for choosing the best use case and adjusting and implementing the protocol in that setting.

Open question – is cryptography needed? We described a protocol for the semi-honest setting which does not require any (complexity based) cryptographic primitives. (Namely, the server sends r to the device, which sends back the result of the inner-product.) This protocol is secure even if the server is malicious. However, to guarantee security against a coalition of malicious devices, our protocol instantiations use Oblivious Transfer (OT) or public-key cryptography. The interesting question is whether protecting against an undercount attack implies the existence of OT or other cryptographic primitives. It is an open problem to either prove this claim or show an alternative protocol.

Open question – how effective is the data leakage? In the "Malicious Campaign" setting, we treated the data leakage as allowing the adversary to mount an attack with a success probability that is linear in the number of bits leaked. It is unclear if this approach can indeed be exploited. Hence the parameters of the system may be improved. Is it possible to argue that the system behaves better than our analysis?

References

32718. California Senate Bill No. 327. Bill no. 327 information privacy: connected devices, 2018.

AJL. Andris Ambainis, Markus Jakobsson, and Helger Lipmaa. Cryptographic randomized response techniques. In *Public Key Cryptography - PKC 2004*.

BDB16. Jeremiah Blocki, Anupam Datta, and Joseph Bonneau. Differentially private password frequency lists. In *NDSS*, 2016.

BH19. Jeremiah Blocki and Ben Harsha. LinkedIn Password Frequency Corpus. 2019.

BNST17. Raef Bassily, Kobbi Nissim, Uri Stemmer, and Abhradeep Guha Thakurta. Practical locally private heavy hitters. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*, pages 2285–2293, 2017.

Bon12. Joseph Bonneau. The science of guessing: Analyzing an anonymized corpus of 70 million passwords. In *IEEE SP*. IEEE Computer Society, 2012.

BS15. Raef Bassily and Adam Smith. Local, private, efficient protocols for succinct histograms. In *Proc. of 47th STOC*, pages 127–135. ACM, 2015.

CC19. National Cyber and Security Centre. Passwords, passwords everywhere, 2019.

CLSX12. T.-H. Hubert Chan, Mingfei Li, Elaine Shi, and Wenchang Xu. Differentially private continual monitoring of heavy hitters from distributed streams. In *PETS*, 2012.

CO15. Tung Chou and Claudio Orlandi. The simplest protocol for oblivious transfer. In *LATINCRYPT*, 2015.

CSS. T.-H. Hubert Chan, Elaine Shi, and Dawn Song. Optimal lower bound for differentially private multi-party aggregation. In *Algorithms - ESA 2012*.

CSS11. T.-H. Hubert Chan, Elaine Shi, and Dawn Song. Private and continual release of statistics. *ACM Trans. Inf. Syst. Secur.*, 14(3), 2011.

DNP$^+$10. Cynthia Dwork, Moni Naor, Toniann Pitassi, Guy N. Rothblum, and Sergey Yekhanin. Pan-private streaming algorithms. In *Innovations in Computer Science - ICS.*, 2010.

DNPR10. Cynthia Dwork, Moni Naor, Toniann Pitassi, and Guy N. Rothblum. Differential privacy under continual observation. In *STOC*, 2010.

DNRR15. Cynthia Dwork, Moni Naor, Omer Reingold, and Guy N. Rothblum. Pure differential privacy for rectangle queries via private partitions. In *ASIACRYPT*, 2015.

DR14. Cynthia Dwork and Aaron Roth. The algorithmic foundations of differential privacy. *Foundations and Trends in Theoretical Computer Science*, 9(3-4):211–407, 2014.

DRV10. C. Dwork, G. N. Rothblum, and S. Vadhan. Boosting and differential privacy. In *FOCS*, 2010.

EPK14. Úlfar Erlingsson, Vasyl Pihur, and Aleksandra Korolova. RAPPOR: randomized aggregatable privacy-preserving ordinal response. In *ACM CCS (2014)*, 2014.

FS86. Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *CRYPTO*, 1986.

GGF17. Paul A Grassi, M Garcia, and J Fenton. DRAFT NIST special publication 800-63-3 digital identity guidelines. 2017.

GM84. Shafi Goldwasser and Silvio Micali. Probabilistic encryption. *Journal of computer and system sciences*, 28(2):270–299, 1984.

HKK$^+$14. Yan Huang, Jonathan Katz, Vladimir Kolesnikov, Ranjit Kumaresan, and Alex J. Malozemoff. Amortizing garbled circuits. In *CRYPTO*, 2014.

HKR12. Justin Hsu, Sanjeev Khanna, and Aaron Roth. Distributed private heavy hitters. In *ICALP*, 2012.

Hoe63. Wassily Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American statistical association*, 1963.

KMRR15. Vladimir Kolesnikov, Payman Mohassel, Ben Riva, and Mike Rosulek. Richer efficiency/security trade-offs in 2pc. In Yevgeniy Dodis and Jesper Buus Nielsen, editors, *TCC*, volume 9014 of *Lecture Notes in Computer Science*. Springer, 2015.

LMD. Karsten Loesing, Steven J. Murdoch, and Roger Dingledine. A case study on measuring statistical data in the Tor anonymity network. In *WECSR (2010)*.

LP07. Yehuda Lindell and Benny Pinkas. An efficient protocol for secure two-party computation in the presence of malicious adversaries. In *EUROCRYPT*, 2007.

LP09. Yehuda Lindell and Benny Pinkas. A proof of security of yao's protocol for two-party computation. *J. Cryptology*, 22(2), 2009.

LR15. Yehuda Lindell and Ben Riva. Blazing fast 2pc in the offline/online setting with security for malicious adversaries. In Indrajit Ray, Ninghui Li, and Christopher Kruegel, editors, *ACM CCS*, 2015.

MF06. Payman Mohassel and Matthew K. Franklin. Efficiency tradeoffs for malicious two-party computation. In Moti Yung, Yevgeniy Dodis, Aggelos Kiayias, and Tal Malkin, editors, *Public Key Cryptography - PKC 2006*, volume 3958 of *Lecture Notes in Computer Science*. Springer, 2006.

MN06. Tal Moran and Moni Naor. Polling with physical envelopes: A rigorous analysis of a human-centric protocol. In *EUROCRYPT*, 2006.

MS17. Akshaya Mani and Micah Sherr. Historϵ: Differentially Private and Robust Statistics Collection for Tor. In *(NDSS)*, 2017.

PVW08. Chris Peikert, Vinod Vaikuntanathan, and Brent Waters. A framework for efficient and composable oblivious transfer. In *CRYPTO*, 2008.

ras. Raspberry pi 3 model b. [Online; accessed 17-May-2017].

RR16. Peter Rindal and Mike Rosulek. Faster malicious 2-party secure computation with online/offline dual execution. In *USENIX Security 16*, pages 297–314, Austin, TX, 2016.

Sch16. Bruce Schneier. Your WiFi-connected thermostat can take down the whole internet. we need new regulations - the washington post. 2016.

SHM10. Stuart Schechter, Cormac Herley, and Michael Mitzenmacher. Popularity is everything: A new approach to protecting passwords from statistical-guessing attacks. In *USENIX conference on Hot topics in security*, 2010.

Sud00. Madhu Sudan. List decoding: Algorithms and applications. In *IFIP International Conference on Theoretical Computer Science*, 2000.

Tho18. Iain Thomson. Who's using 2fa? sweet fa. less than 10two-factor authentication, 2018.

Vad17. Salil Vadhan. *The Complexity of Differential Privacy*, pages 347–450. Springer International Publishing, Cham, 2017.

WCW$^+$17. Ding Wang, Haibo Cheng, Ping Wang, Xinyi Huang, and Gaopeng Jian. Zipf's law in passwords. *IEEE Trans. Information Forensics and Security*, 12(11):2776–2791, 2017.

Wika. Wikipedia. Mirai (malware). [Online; accessed 12-May-2019].

Wikb. Wikipedia. Shodan. [Online; accessed 19-May-2017].

Wik19. Wikipedia contributors. Rockyou — Wikipedia, the free encyclopedia, 2019. [Online; accessed 15-August-2019].

Yun17. Moti Yung. Snapchat's "my eyes only" secure cloud storage protocol, 2017.

A Frequently Asked Questions

Isn't the password distribution already known?
As we explain in Section 1.1, the password distribution can change over time, or between different populations.

Can publishing the blacklist put users at risk?
As we explain in Section 1.1, this is similar to publishing new code vulnerabilities, as the only way to help most of the users and to protect the ecosystem. Moreover, as we prevent any single password from becoming too popular, this will actually limit the attackers' ability to exploit this information.

Can we use some PIR, PSI or other mechanisms to protect the blacklist of passwords?
Unfortunately no. In our analysis we assume there is a large colluding coalition, and each device is allowed to test if its password is blacklisted or not. If this coalition is large then it might be able to test the entire hash domain and find all blacklisted values regardless of the protection mechanism.

What about the users that are already using a blacklisted password?
As we note in Section 1.1, the device can alert the user if its current password is added to the blacklist. However, this results in a security trade-off, as the change may reveal a lot of information about the user's previous password.

Leakage: why leaking one bit of information is fine?
The password game defined in Section 2.1 shows what is the effect of releasing one bit of information. Moreover, this section shows how differential privacy can be used to reduce this effect.

Is it practical to implement this protocol?
Yes! In section 7 we describe the PoC we implemented both for the device and the server sides and for participial parameters of the scheme. A relatively weak server requires ≈ 0.5 seconds to run the whole non-interactive proof verification and to update the counters. A low resources Raspberry Pi 3 Model B can prepare the proof in less than 15 seconds.

What size of the domain can be used in the real world?
The domain size does not affect the security of the protocol in any way. It is just a trade-off between communication complexity and performance on one hand, and password rejection false positive rate on the other hand. Our PoC was done with $\ell = 16$, which allows for good performance and a low false positive rate of $2^{-16}/\tau$.

Your protocol does not defend against unique weak passwords
Yes, but the main goal is to protect against large scale attacks. Exploiting unique weak password requires a targeted attack against a specific user. See Section 1.1 for more details.

What is the main difference between the QR and the garbled circuits based solutions?
See Section 9 for the tradeoff between the two solutions.

Does using a new r value for each password protect against the leakage that leads to the "Malicious Campaign" scenario?
Unfortunately no. If there is any leakage on the result of the protocol, the attacker can just run the protocol multiple times until it gets the required results. On the other hand, keeping r fixed allows for a simpler and safer protocol (see Section 6.4).

How do you set the threshold for τ for popular passwords
The threshold τ is chosen dynamically to be the smallest possible value satisfying the required error probability p_{FN}, with the current number of users N_C. See Section 4.3 for more details.

How many of the top passwords can you blacklist?

Our theoretical analysis for an *arbitrary distribution* guarantees only the blacklisting of a few very popular passwords. However, the *Zipfian nature* of the password distribution means that we only care about blacklisting the top passwords as they are much more valuable to an attacker. Moreover, using simulation on real-world data we show that our scheme performs closely to that ideal blacklisting process even when blacklisting the top 25 passwords. See Section 1.4 and Section 7.2 for more details.

Can someone modify the blacklist and create a DoS type of attack?

We assume that the published blacklist is authenticated (e.g. in an https protected website). As the list is publicly available it is easy to make sure all devices receive the same blacklist, and the device can reject the blacklist if its size is unreasonably large (e.g. more than some fraction of the domain), see Section 3.4.

B Bounds for the "Malicious Campaign" Setting

In this section we will bound $pMalCam_{FN}$, the false negative probability in the "Malicious Campaign"setting. We also show how the server can dynamically control the scheme's parameters to achieve a target $pMalCam_{FN}$.

We therefore discuss in this section how the server can add noise to the published list of popular hash values, in order to reduce the probability of a successful undercount attack. Note that we assume that the r^s *value is reused between different runs* of the protocol with the same device, see Section D.1.

B.1 "Malicious Campaign"Setting Information Leakage

We consider the worst case scenario, where all of the devices are malicious and try to collude to learn information about the different secret r^s_j values used by the server. We want to bound the amount of bits leaked on a single publication of the statistics. As all of the devices are colluding, all the chosen hash values are known (the devices know $N_H(v)$ for every v), and so all the information leaked is on the r_j values.

A hash value v is added to the published list if $\mathbb{1}(\alpha(v) - \tau N_C) = 1$. The maximum entropy for a single bin happens when $E(\alpha(v) = \tau N_C)$ and then the entropy is $H(\mathbb{1}(\alpha(v) - \tau N_C)) = 1$. As the malicious devices do not have to randomize their responses, this happens when exactly $\tau N_C(1-\epsilon_r)$ devices choose v. Due to our secure inner product protocol the devices cannot control the part of $\alpha(v)$ that is a binomial distribution:

$$\chi_B \sim bin(n = N_C(1 - \tau(1-\epsilon_r)), p = 1/2) \approx bin(n = N_C, p = 1/2)$$
$$\alpha(v) \approx \tau N_C + 2(\chi_B - N_C/2)/(1 - \epsilon_r)$$

The devices control all of the chosen v values, but χ_B is randomized due to the secret r_j vectors. Any bit of information on the value of χ_B can be translated to information on r_j. We will like to bound the amount of information leaked.

B.2 The Effect of the Laplacian Noise on False and Positive Negatives

We consider the addition of Laplacian noise with $1/\epsilon_n = \tau N_C \delta / C$ where C is a small constant (e.g. 2). In that case we can view the noisy approximation α_N as:

$$\alpha_N(v) = \alpha(v) + \chi_L = N_H(v) + \chi_B + \chi_L$$

where χ_B is the binomial noise due to the devices that did not choose v and the randomized response, and χ_L is the added Laplacian noise.

The ephemeral Laplacian noise added by the server can cause false and positive negatives, with a relatively low but non-negligible probability. By choosing a threshold value $\tau > \tau_{min}$ we get smaller binomial noise compared to $\delta\tau N_C$, and allow for the extra noise. Moreover, the server generates new ephemeral noise for each publication in the hash list. So even if a false negative will happen with low probability, the expected number of such events is small and they will have a small effect on the users. For example if once every 10 weeks a popular password will not appear on the list, not many new users will choose it. In contrast, a false negative event caused by the binomial noise that was described before will be maintained in all the next publications.

B.3 Bounding the Information Leakage

As we have demonstrated, all bins with $N_H(v) > \tau N_C(1+\delta)$ have negligible probability of not being published. Therefore, the fact that they are in the list leaks no information. The mirror case is with regards to all bins such that $N_H(v) < \tau N_C(1-\delta)$. The fact that they are not in the list also leaks no information. We get that a bin can only leak information if:

$$\tau N_C(1-\delta) < N_H(v) < \tau N_C(1+\delta)$$

In the “Malicious Campaign”setting, the devices do not have to randomize their response and the maximum number of such bins is:

$$1/(\tau(1-\delta)(1-\epsilon_r))$$

As the colluding devices know $N_H(v)$, the only information they can gather is on the value of χ_B, which is the sum of binomial noises. To bound the leakage we want to bound the mutual information.

$$\begin{aligned}
I(\chi_B; \mathbb{1}(\alpha_N(v) - \tau N_C)) &= H(\mathbb{1}(\alpha_N(v) - \tau N_C)) \\
&\quad - H(\mathbb{1}(\alpha_N(v) - \tau N_C)|\chi_B) \\
&\leq 1 - H(\mathbb{1}(\alpha_N(v) - \tau N_C)|\chi_B) \\
&= 1 - H(\mathbb{1}(\chi_B + \chi_L)|\chi_B)
\end{aligned}$$

We define $b(t)$ as the maximum possible number of bits leaked at time t with the current values of $\tau(t)$:

$$b(t) = \frac{(1 - H(\mathbb{1}(\chi_B(t) + \chi_L(t))|\chi_B(t))}{\tau(t)(1-\delta)(1-\epsilon_r)}$$

As N_C increases we can add larger Laplacian noise χ_L. As χ_L gets larger $H(\mathbb{1}(\chi_B(t)+\chi_L(t))|\chi_B(t))$ tends to 1 and $b(t)$ tends to zero.

B.4 Bounding the Probability of an Undercount Attack

In Lemma 2 we bounded the probability of a false negative p_{FN} in the semi-honest setting. We can view all the possible choices of hashes by the attacker's devices as a search space, where the attacker goal is find such a hashes that cause an undercount attack. Without any auxiliary data on the r_j values, an attacker best strategy is to randomly sample from this space, and test if the undercount attack succeed. This will happen at a probability of p_{FN}.

Every bit of information on r_j can help the attacker ignore half the search space or increase the success probability by a factor of 2. We define $B(t) = \sum b(t_i)$ the total bits of information leaked up to time t. So the attacker can use $B(t)$ to increase is success probability by $2^{B(t)}$. We get an upper bound for the new probability of a false negative with malicious devices:

$$\begin{aligned} pMalCam_{FN} &= p_{FN} \cdot 2^{B(t)} \leq 2\exp(-N_C(\tau\delta(1-\epsilon_r))^2/2) \cdot 2^{B(t)} \\ &= 2\exp(-N_C(\tau\delta(1-\epsilon_r))^2/2 + B(t) \cdot \ln(2)) \\ &\approx 2\exp((1.38B(t) - N_C(\tau\delta(1-\epsilon_r))^2)/2) \end{aligned}$$

This Probability is greatly affected by two parameters N_C and τ. As N_C increases $pMalCam_{FN}$ decreases exponentially. Moreover, as N_C increases we can use larger Laplacian noise, reduce the data leakage and so $B(t)$. As τ decreases $pMalCam_{FN}$ decreases exponentially. Moreover $B(t) \propto 1/\tau$.

B.5 Dynamic Publication Frequency

Similarly to the analysis in Section 4.3, we want to allow the server to dynamically change the τ and the publication frequency. We define LS as the planned lifespan of the system. This might be the life expectancy of the devices or of the service (e.g. the manufacture can decide that he will stop support of the web cameras after 10 years).

For correctness in the "Malicious Campaign"setting, we require that for any time t, $pMalCam_{FN} \leq \exp(-C)$ for some constant C. We do this by defining a dynamic threshold and publication frequency that depend on the amount of previously leaked information $B(t)$, the remaining lifespan of the system $LS - t$, and the number of devices that have run the protocol so far[6]

$$\begin{aligned} \ln(pMalCam_{FN}) \approx (1.38L(t) - N_C(t)(\tau(t)\delta(1-\epsilon_r))^2)/2 \leq -C \\ N_C(t)(\tau(t)\delta(1-\epsilon_r))^2 - 1.38L(t) \geq 2C \end{aligned}$$

For any time t, the worst case scenario is that no more devices will choose a hash and $N_C(LS) = N_C(t)$. In that case we want to choose a publication frequency $freq(t)$ and threshold $\tau(t)$ such that even at the end of the lifespan of the system the information leaked will be bounded:

$$\begin{aligned} L(LS) &= L(t) + (LS - t)freq(t)l(t) && (4) \\ 2C &\leq N_C(LS)(\tau(LS)\delta(1-\epsilon_r))^2 - 1.38L(LS) && (5) \\ &= N_C(t)(\tau(t)\delta(1-\epsilon_r))^2 - 1.38(L(t) + (LS-t)\mathit{freq}(t)l(t)) \end{aligned}$$

Equation 4 gives an upper bound for the amount of information leaked in the rest of the lifespan if the statistics are published with frequency $\mathit{freq}(t)$ and threshold $\tau(t)$. Equation 5 gives the required ratio needed to prevent "Malicious Campaign"undercount attacks.

As $N_C(t)$ increases over time the server can increase the frequency of publication or decrease the threshold.

[6] Devices can change their hash but not remove themselves from the statistics, so $N_C(t)$ is a monotonic increasing function of time t

C Security analysis of the Garbled Circuits Protocol

Security against a malicious device. The device plays the role of the evaluator (receiver) in the protocol. It is well known that Yao's protocol is secure against a malicious evaluator [MF06]. The device sends an output to the server, but since the server verifies that this output is a garbled value of the output wire the device has no option but to evaluate the circuit and send the garbled output value to the server. Formally, security can be proven in a straightforward way by assuming that the oblivious transfers are secure and using the simulation proof presented by Lindell and Pinkas [LP09].

Security against a malicious server. A malicious server can use an arbitrary circuit in the protocol. The main feature ensuring security against the server is that the device is only willing to send to the server an output whose hash value matches one of two outputs of the hash function. Namely, there are only two possible outputs which the device might send to the server.

A selective failure attack. In addition, a malicious server can apply a selective failure attack, and prepare a corrupt circuit whose evaluation fails for a subset of the inputs of the device. In this case, the server can identify that the device's input is in this subset. In total, the server can learn whether the device's input is in one of three subsets (the preimages of 0, the preimages of 1, and the values for which evaluation fails). In other words, the server can learn at most $\log_2 3 = 1.58$ bits of information about the password.

We note, however, that this failure of the protocol is an illegitimate outcome. When this happens, the device can complain about the behavior of the server. (If the protocol further requires the server to sign the messages that it sends, then the device has a proof that the sender misbehaved).

A simulation argument for the security of the construction works with a slightly modified function used in the ideal model, which also considers the selective failure attack. The server's input to the trusted party is a value r_j^s, and a function $\hat{F}$, which has three possible outputs: 0, 1, or "computation failure". The simulator receives the server's input and output. The simulator can simulate the server's input to the OTs, and can easily generate the garbled tables. It then simulates the answer that the server receives based on the output of $\hat{F}$ given by the trusted party.

D Design Choices

D.1 Reusing r vectors between runs

A malicious adversary can undercount a popular hash and achieve a higher p_{FN} due to information leaked on the secret r from publishing the blacklist. An obvious approach to avoid this attack is to use a new r vector for each run of the protocol. However, this only complicates the attack but does not stop it. On the other hand, using a fixed r helps simplifying the protocol and block any targeted adaptive attack by the server.

The main idea of the attack, is that a device can use other colluding devices to learn if his vote is anti correlated or not to the popular hash, without rerunning the protocol. Lets assume that the device knows that a hash h_p will become popular in the future, and whats to start undercounting it. Colluding devices will vote for h_p over time, until they reach the threshold τ. This is done by checking over time that the probability of the hash to be published is equal to half. After that the device votes for some value $h_a \neq h_p$ and checks if the probability for the hash to appear decreases - meaning that he is anti correlated. Although this attack seems to be impractical due to large Laplacian noise added, a more sophisticated attack might still be possible.

If we do not reuse the same r, even a semi-honest server might learn extra information about a password using auxiliary data. Lets assume the user switch back to an old password he used before. If the server knows that it is the same password, by using different r he can possibly learn more than one bit.

Moreover, in the OR based protocol, the server can save the public $\boldsymbol{r_p}$ vector on the device in production time. This saves us the first message in the protocol. In the Non-Interactive version, the protocol is reduced to sending one message from the device to the server.

E Proofs

We prove here Lemmata 1, 2 and 3 of Section 4.

Lemma 1. If a fraction p of the devices choose a password which hashes to a value v, then the expected value of $\alpha(v)$ is pN_C.

Proof. Let us first calculate the expected value of counter $T[v]$ given the number of devices whose password is hashed to v. The expected contribution of any value $v_j \neq v$ to $T[v]$ is 0 and for $v_j = v$ it is 1. We calculate the expected number of devices that chose the value v. There are three ways for a device to choose a password $pass$, and get a hash v.

1. $v = H(pass)$ and the device is not randomizing the response.
2. $v \neq H(pass)$, the device randomizes the response, and as a result gets the value v.
3. $v = H(pass)$, the device randomizes the response, but gets the same v again.

Therefore,

$$\begin{aligned} E(T[v] \mid N_H(v) = pN_C) &= N_C \cdot (p(1-\epsilon_r+\epsilon_r 2^{-\ell}) + (1-p)\epsilon_r 2^{-\ell}) \\ &= N_C \cdot (p(1-\epsilon_r) + \epsilon_r 2^{-\ell}) \end{aligned} \tag{6}$$

From (3) and (6) we get that the expected value of $\alpha(v)$ is:

$$E(\alpha(v)|N_H(v) = pN_C) = pN_C$$

Lemma 2. The probability for a false negative p_{FN} is bounded by $p_{FN} \leq 2\exp(-N_C(\tau\delta(1-\epsilon_r))^2/2)$.

Proof. The worst case is $N_H(v) = \tau N_C(1+\delta)$

$\alpha(v)$ can be viewed as the sum of N_C independent random variables bounded by the interval $[-1/(1-\epsilon_r), 1/(1-\epsilon_r)]$ and $E(\alpha(v)|N_H(v) = pN_C) = pN_C$. Using Hoeffding's inequality (Theorem 2) [Hoe63] we can show that:

$$\begin{aligned} p_{FN} &= \Pr(\alpha(v) \leq \tau N_C | N_H(v) = \tau N_C(1+\delta)) \\ &= \Pr(\alpha(v) - \tau N_C(1+\delta) \leq -\tau N_C\delta | N_H(v) = \tau N_C(1+\delta)) \\ &= \Pr(\alpha(v) - E(\alpha(v)) \leq -\tau N_C\delta | N_H(v) = \tau N_C(1+\delta)) \\ &\leq \Pr(|\alpha(v) - E(\alpha(v))| \geq \tau N_C\delta | N_H(v) = \tau N_C(1+\delta)) \\ &\leq 2\exp(-\frac{2(\tau N_C\delta(1-\epsilon_r))^2}{4N_C}) = 2\exp(-N_C(\tau\delta(1-\epsilon_r))^2/2) \end{aligned}$$

There are at most $\frac{1}{\tau(1+\delta)}$ bins where $N_H(v) \geq \tau N_C(1+\delta)$ (bins where a false negative can occur). The expected number of FN events N_{FN} is: $E(N_{FN}) \leq 2\exp(-N_C(\tau\delta(1-\epsilon_r))^2/2) \;/\; \tau(1+\delta)$.

Lemma 3. The probability for a false positive p_{FP} is bounded by $p_{FP} \leq 2\exp(-N_C(\tau\delta(1-\epsilon_r))^2/2)$

Proof. In a similar manner to Equation E we can show that

$$\begin{aligned} p_{FP} &= \Pr(\alpha(v) \geq \tau N_C | N_H(v) = \tau N_C(1-\delta)) \\ &= \Pr(\alpha(v) - \tau N_C(1-\delta) \geq \tau N_C\delta | N_H(v) = \tau N_C(1-\delta)) \\ &= \Pr(\alpha(v) - E(\alpha(v)) \geq \tau N_C\delta | N_H(v) = \tau N_C(1-\delta)) \\ &\leq \exp(-\frac{2(\tau N_C\delta(1-\epsilon_r))^2}{4N_C}) = \exp(-N_C(\tau\delta(1-\epsilon_r))^2/2) \end{aligned}$$

There are 2^ℓ bins, so the expected number of FP events N_{FP} is: $E(N_{FP}) \leq 2^\ell \exp(-N_C(\tau\delta(1-\epsilon_r))^2/2)$.

Note that as long as $E(N_{FP}) \ll 1/(\tau(1+\delta))$ the probability for a false positive on a **password** is dominated by the probability of a collision with a "popular" hash value. As there are at most $1/(\tau(1+\delta)$ such hash values, p_{PassFP} the probability for a password false positive is: $p_{\text{PassFP}} \leq 2^{-\ell} \,/\, \tau(1-\delta)$.

F Complete Protocol and Proofs for the QR Protocol

F.1 Complete Protocol

We shall now give a full description of the complete protocol:

1. $\boldsymbol{S}$ global setup – initial setup of the public and private parameters of the protocol. This step can be run once by $\boldsymbol{S}$ with regards to all $\boldsymbol{D}$s.
2. $\boldsymbol{S}$ instance setup – should be run once by $\boldsymbol{S}$ for each $\boldsymbol{D}$.
3. $\boldsymbol{D}$ instance setup – should be run by $\boldsymbol{D}$ every time a new $\boldsymbol{V_s}$ is chosen. Calculates the inner product.
4. The interactive protocol – a description of the run of the interactive protocol.

Server global setup $\boldsymbol{S}$ runs a setup algorithm $\mathit{GlobalSetup}(1_1^k)$, taking security parameter k_1. It generates an RSA key pair $(PK; SK)$ where $SK = (p, q)$ and $PK = (N = pq)$, p and q are distinct prime numbers congruent to 1 mod 4 (this implies that $-1 \in QR_N$, as is required for proving soundness).

Additionally $\boldsymbol{S}$ publishes a proof that N is a semiprime ($N = p^a q^b$ where p and q are primes). This is required for the zero-knowledge proof. Such a proof can be found in Appendix G.

Server instance setup For each $\boldsymbol{D}$, $\boldsymbol{S}$ runs $\mathit{ServerSetup}(PK, SK, \boldsymbol{R_s}, \ell)$, taking the private and public keys generated by $\mathit{GlobalSetup}$, and a secret vector $\boldsymbol{R_s} \in \{0,1\}^\ell$. $\boldsymbol{S}$ encodes $\boldsymbol{R_s}$ in a public vector $\boldsymbol{r_p} \in \mathbb{Z}_N^\ell$ that is sent to $\boldsymbol{D}$. As in the semi-honest version, each number in $r_{p,i}$ is a public encryption of the corresponding bit $R_{s,i}$. $\boldsymbol{S}$ sends $\boldsymbol{r_p}$ to $\boldsymbol{D}$.

Device instance setup $\boldsymbol{D}$ runs $\mathit{DeviceSetup}(1^{k_2}, \ell, PK, \boldsymbol{r_p}, \boldsymbol{V_s})$ taking a security parameter k_2, ℓ, PK, $\boldsymbol{r_p}$ and a private vector $\boldsymbol{V_s} \in \{0,1\}^\ell$.

First, $\boldsymbol{D}$ checks that for any number $x \in \boldsymbol{r_p}$, the Jacobi symbol $\left(\frac{x}{N}\right)$ is 1. Then $\boldsymbol{D}$ generates and saves the following data:

1. $\boldsymbol{r} = \boldsymbol{r_p}||\boldsymbol{pad}^2$ where $\boldsymbol{pad} \overset{R}{\leftarrow} \mathbb{Z}_N^{\ell}$. We pad $\boldsymbol{r_p}$ with ℓ numbers in QR_N.
2. $\boldsymbol{V} = \boldsymbol{V_s}||\overline{\boldsymbol{V_s}}$ where $\overline{V_{s,i}} = 1 - V_{s,i}$.
3. $\pi_* \overset{R}{\leftarrow} \pi\{2\ell\}$, $\boldsymbol{d_*} \overset{R}{\leftarrow} \mathbb{Z}_N^{2\ell}$. Namely π_* is a random permutation on 2ℓ items, and $\boldsymbol{d_*}$ are 2ℓ random numbers.
4. $\boldsymbol{r_*} = \pi_*(r \cdot (\boldsymbol{d_*})^2)$, $\boldsymbol{V_*} = \pi_*(\boldsymbol{V})$. Namely, $\boldsymbol{r_*}$ is a blinded random permutation of $\boldsymbol{r}$, and $\boldsymbol{V_*}$ is the same permutation of $\boldsymbol{V}$.
5. $\boldsymbol{s_*} = (\boldsymbol{r_*})^2$.
6. $e = \prod_{i=1}^{2\ell}(r_i(d_{*,i})^2)^{V_i} = \prod_{i=1}^{2\ell}(r_{*,i})^{V_{*,i}}$. That is, e is the product of the blinded items in $\boldsymbol{r}$ corresponding to the 1 bits in $\boldsymbol{V}$, and therefore encodes the inner product result.

$\boldsymbol{D}$ sends $(\boldsymbol{s_*}, e, \boldsymbol{V_*}, \boldsymbol{pad})$ to $\boldsymbol{S}$. Notice that this time $\boldsymbol{S}$ can only calculate e^2 by itself, and therefore it is also required to send e.

Inner product calculation and verification $\boldsymbol{D}$ and $\boldsymbol{S}$ run the following protocol. If any step or calculation results in $\perp$, or a verification fails, then $\boldsymbol{S}$ outputs reject. Otherwise $\boldsymbol{S}$ outputs accept and can use the value $result = \langle \boldsymbol{V_s}, \boldsymbol{R_s} \rangle$.

1. $\boldsymbol{S}$ calculates e^2, and verifies that $e^2 = \prod_{i=1}^{2\ell}(s_{*,i})^{V_{*,i}}$. $\boldsymbol{S}$ verify that e^2 is indeed equal to the product of numbers in $\boldsymbol{s_*}$.
2. $\boldsymbol{S}$ calculates $\boldsymbol{r_{sqr}} = (\boldsymbol{r_p}||\boldsymbol{pad}^2)^2$, the **square** of $\boldsymbol{r}$ using $\boldsymbol{pad}$ it received from $\boldsymbol{D}$ and $\boldsymbol{r_p}$ that it stored locally.
3. For $j = 1, \ldots, k_2$:
   (a) $\pi_j \overset{R}{\leftarrow} \pi\{2\ell\}$, $\boldsymbol{d_j} \overset{R}{\leftarrow} \mathbb{Z}_N^{2\ell}$.
   (b) $\boldsymbol{D}$ generates $\pi_j \overset{R}{\leftarrow} \pi\{2\ell\}$, $\boldsymbol{d_j} \overset{R}{\leftarrow} \mathbb{Z}_N^{2\ell}$ (Namely a random permutation on 2ℓ items, and 2ℓ random numbers).
   (c) $\boldsymbol{D}$ sends $\boldsymbol{s_j} = \pi_j(\boldsymbol{s_*} \cdot (\boldsymbol{d_j})^4)$ to $\boldsymbol{S}$. $\boldsymbol{s_j}$ is a blinded random permutation of $\boldsymbol{s_*}$.
   (d) $\boldsymbol{S}$ sends $b_j \overset{R}{\leftarrow} 0, 1$ to $\boldsymbol{D}$.
   (e) if $b_j = 0$
      i. $\boldsymbol{D}$ sends to $\boldsymbol{S}$ the values $(\pi_{*,j} = \pi_j\pi_*, \boldsymbol{d_{*,j}} = \pi_j(\pi_*(\boldsymbol{d_*}) \cdot \boldsymbol{d_j})$, opening the blinded permutation from $\boldsymbol{r_{sqr}}$ to $\boldsymbol{s_j}$.
      ii. $\boldsymbol{S}$ verifies that $\boldsymbol{s_j} = \pi_{*,j}(\boldsymbol{r_{sqr}}) \cdot (\boldsymbol{d_{*,j}})^4$.
   (f) else
      i. $\boldsymbol{D}$ sends to $\boldsymbol{S}$ the values $(\pi_j, \boldsymbol{d_j})$, opening the blinded permutation from $\boldsymbol{s_*}$ to $\boldsymbol{s_j}$.
      ii. $\boldsymbol{S}$ verifies that $\boldsymbol{s_j} = \pi_j(\boldsymbol{s_*} \cdot \boldsymbol{d_j}^4)$.
4. $\boldsymbol{S}$ calculates $result = T_{QR_N}(e)$, to retrieve the value of the inner product.

F.2 Completeness

It is easy to see that if both $\boldsymbol{S}$ and $\boldsymbol{D}$ follow the protocol then the protocol ends successfully as the following statements hold:

$$T_{QR_N}(e) = \langle \boldsymbol{V_s}, \boldsymbol{R_s} \rangle$$
$$e^2 = \prod_{i=1}^{2\ell}(s_{*,i})^{v_{*,i}}$$
$$\forall j, \boldsymbol{s_j} = \pi_j(\boldsymbol{s_*} \cdot \boldsymbol{d_j}^2)$$
$$\forall j, \boldsymbol{s_j} = \pi_{*,j}(\boldsymbol{r_{sqr}}) \cdot \boldsymbol{blind_j}^2$$

F.3 Soundness

Theorem 4. *For any (possibly not efficiently computable) adversarial algorithm* $\boldsymbol{A_D}$, *if* $T_{QR_N}(e) \neq \langle \boldsymbol{V_s}, \boldsymbol{R_s} \rangle$ *then*

$$\Pr_{b \xleftarrow{R} \{0,1\}^{k_2}} [\boldsymbol{S} \ = accept] \leq 2^{-k_2}]$$

Theorem 4 if proved using the following two lemmas.

Lemma 4. *For any (possibly not efficiently computable) adversarial algorithm* $\boldsymbol{A_D}$, *and for any step* j *in the interactive proof stage,* $\Pr_{b_j \xleftarrow{R} 0,1} [\boldsymbol{S} \ \neq reject] \geq \frac{1}{2}$, *only if* $\boldsymbol{A_D}$ *knows* $(\boldsymbol{d_*}, \pi_*)$ *such that* $\boldsymbol{r_*} = (\pi_*(\boldsymbol{r} \cdot (\boldsymbol{d_*})^2))$ *and* $\boldsymbol{s_*} = (\boldsymbol{r_*})^2$.

Proof. $\Pr_{b_j \xleftarrow{R} 0,1} [\boldsymbol{S} \ \neq reject] \geq \frac{1}{2}$ only if $\boldsymbol{A_D}$ knows $(\pi_j, \boldsymbol{d_j}, \pi_{*,j}, \boldsymbol{d_{*,j}})$ such that:

$$\boldsymbol{s_j} = \pi_j(\boldsymbol{s_*} \cdot \boldsymbol{d_j}^4) \tag{7}$$

$$\boldsymbol{s_j} = \pi_{*,j}(\boldsymbol{r}^2) \cdot (\boldsymbol{d_{*,j}})^4 \tag{8}$$

From 7 $\boldsymbol{A_D}$ can calculate

$$\boldsymbol{s_*} = \pi_j^{-1}(\boldsymbol{s_j}) \cdot \boldsymbol{d_j}^{-4}$$

From 7 + 8 $\boldsymbol{A_D}$ can calculate

$$\boldsymbol{s_*} = \pi_j^{-1}(\pi_{*,j}(\boldsymbol{r}^2) \cdot (\boldsymbol{d_{*,j}})^4) \cdot \boldsymbol{d_j}^{-4} = \pi_j^{-1}\pi_{*,j}(\boldsymbol{r}^2 \cdot (\pi_{*,j}^{-1}((\boldsymbol{d_{*,j}}) \cdot \pi_j(\boldsymbol{d_j}^{-1})))^4)$$

We denote $\pi_* = \pi_j^{-1}\pi_{*,j}$ $\boldsymbol{d_*} = (\pi_{*,j}^{-1}((\boldsymbol{d_{*,j}}) \cdot \pi_j(\boldsymbol{d_j}^{-1}))$

$$\boldsymbol{r_*} = \pi_*(\boldsymbol{r} \cdot (\boldsymbol{d_*})^2), \quad \boldsymbol{s_*} = \pi_*(\boldsymbol{r} \cdot (\boldsymbol{d_*})^2)^2 = (\boldsymbol{r_*})^2 \tag{9}$$

And $\boldsymbol{A_D}$ can calculate $(\pi_*, \boldsymbol{d_*})$.

Lemma 5. *For any (possibly not efficiently computable) adversarial algorithm* $\boldsymbol{A_D}$ *that knows* $(\pi_*, \boldsymbol{d_*})$ *such that* $\boldsymbol{s_*} = \pi_*(\boldsymbol{r} \cdot (\boldsymbol{d_*})^2)^2$, *it holds that finding* e *such that* $T_{QR_N}(e) \neq \langle \boldsymbol{V_s}, \boldsymbol{R_s} \rangle$ *and* $\boldsymbol{S} \ \neq reject$ *is equivalent to factoring* N.

Proof. $\boldsymbol{S} \ = rejects$ unless

$$e^2 = \prod_{i=1}^{2\ell} (\boldsymbol{s_{*,i}})^{v_{*,i}} \tag{10}$$

From 9 + 10 we get that e' is a root of e^2.

$$(e')^2 = (\prod_{i=1}^{2\ell} (\boldsymbol{r_{*,i}})^{v_{*,i}})^2 = \prod_{i=1}^{2\ell} ((\boldsymbol{r_{*,i}})^2)^{v_{*,i}} = \prod_{i=1}^{2\ell} (\boldsymbol{s_{*,i}})^{v_{*,i}} = e^2$$

As $-1 \in QR_N$ we get that $T_{QR_N}(e) = T_{QR_N}(-e)$. If $e = \pm e'$ then as $T_{QR_N}(e) = T_{QR_N}(-e) = T_{QR_N}(e')$ and we get $T_{QR_N}(e) = \langle \boldsymbol{V_s}, \boldsymbol{R_s} \rangle$.

If $T_{QR_N}(e') \neq \langle \boldsymbol{V_s}, \boldsymbol{R_s} \rangle$ than $e \neq \pm e'$ and $\boldsymbol{A_D}$ can calculate all 4 roots of N, and that is equivalent to factoring.

F.4 Zero-Knowledge Proof

Privacy of $\boldsymbol{R_s}$ The encoding of $\boldsymbol{R_s}$ into $\boldsymbol{r_p}$ is a semantically secure encryption [GM84], and does not reveal any information on $\boldsymbol{R_s}$ to any PPT algorithm.

Privacy of $\boldsymbol{V_s}$ There is a simulator $\boldsymbol{S_D}$ that given $\langle \boldsymbol{V_s}, \boldsymbol{R_s} \rangle, \boldsymbol{R_s}$ can simulate $\boldsymbol{D}$. $\boldsymbol{S_D}$ chooses UAR $\boldsymbol{V'}$ such that $result = \langle \boldsymbol{V'}, \boldsymbol{R_s} \rangle = \langle \boldsymbol{V_s}, \boldsymbol{R_s} \rangle$. It then runs the protocol as $\boldsymbol{D}$ with $\boldsymbol{V'}$ as the input.

The distribution of all values is the same between $\boldsymbol{D}$ and $\boldsymbol{S_D}$: All values of $\boldsymbol{s_*}, \boldsymbol{s_j}$ are independently and uniformly distributed in QR_N. e is uniformly distributed either in QR_N or in nQR_N depending only on $result$ (as N is a semiprime, see Appendix G). Both $\pi_{*,j}$ and π_j are independently and uniformly distributed random permutations (conditioned on the fact that for any j only one of the permutations is revealed), and $\boldsymbol{V_*}$ is also a uniformly distributed random value with Hamming weight ℓ.

G Proving N is Semi-Prime

The blinding in our protocol is done by multiplying a given value x with a random uniformly chosen $d \in QR_N$. It is easy to see that if $x \in QR_N$ then $x \cdot d$ is a uniformly random number in QR_N. However if $x \in nQR_N$ and $|nQR_N| > |QR_N|$ then $x \cdot d$ will not be uniform in nQR_N. In that case, blinding done by the device might by ineffective.

It holds that $|QR_N| = |nQR_N|$ if $N = p^a q^b$ where p, q are primes and a, b are positive integers. In that case $\Pr_{x \in \mathbb{Z}_N}(x \in QR_N) = 1/4$. In any other case ($N$ is the product of 3 or more powers of primes) $\Pr_{x \in \mathbb{Z}_N}(x \in QR_N) \leq 1/8$. If we can prove to the device that with high probability $\Pr_{x \in \mathbb{Z}_N}(x \in QR_N) > 1/8$ then this implies $N = p^a q^b$. This can be in the (non programmable) random oracle model in the following way. We generate $1, \dots, k_3$ random numbers using a strong Hash function. The server (knowing the factorization of N) publishes a single root for all the numbers that are in QR_N.

For $i = 1, \dots, k_3$:

1. $v_i = Hash(N||i) \mod N$.
2. if $v_i \in QRn$:
   (a) $a, -a, b, -b = \sqrt{v} \mod N$
   (b) Publish (i, a)

Any verifier can check that $v_i = Hash(N||i) = a_i^2 \mod N \in QRn$.

We define $N_{HQR} = |v_i \in QR_N|$ and $p_{QR} = \Pr_{x \in \mathbb{Z}_N}(x \in QR_N)$.

N_{HQR} can be viewed as the sum of k_3 independent random variables bounded by the interval $[0, 1]$, and $E(N_{HQR}|pr = p_{QR}) = k_3 \cdot p_{QR}$. Using Hoeffding's inequality [Hoe63] we can show that:

$$\begin{aligned}
\Pr(N_{HQR} \geq k_3/4|p_{QR} \leq 1/8) &\leq \Pr(N_{HQR} \geq k_3/4|p_{QR} = 1/8) \\
&= \Pr(N_{HQR} - k_3/8 \geq k_3/8|p_{QR} = 1/8) \\
&= \Pr(N_{HQR} - E(N_{HQR}) \geq k_3/8|p_{QR} = 1/8) \\
&\leq \exp(-2\frac{k_3^2}{8^2 k_3}) = \exp(-\frac{k_3}{32})
\end{aligned}$$

For $k_3 > 128 \cdot 32/1.44 \approx 2850$ the probability of this event is smaller than 2^{-128}. However if we generate N correctly then $\Pr(N_{HQR} \geq k_3/4) = 1/2$. If for our chosen N, $N_{HQR} < k_3/4$ then we just try to generate another N until the condition is satisfied.

This non-interactive proof reveals to the verifier numbers that are in nQR (all the unpublished numbers with Jacobi Symbol 1) that might be difficult to learn. However this information cannot be used by the device in our protocol. If needed this proof can be turned to one not revealing such information by using the same blind and permute scheme we used in the protocol.