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

Moni Naor[*] Benny Pinkas[†] Eyal Ronen(✉)[‡]

January 2, 2018

Abstract

Bad choices of passwords were and are a pervasive problem. Most password alternatives (such as two-factor authentication) may increase cost and arguably hurt the usability of the system. This is of special significance for low cost IoT devices.

Users choosing weak passwords do not only compromise themselves, but the whole eco system. For example, 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. We enable 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 comprise 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.

The construction is secure even against a malicious coalition of devices which tries to manipulate the protocol in order to hide the popularity of some password that the attacker is exploiting. We show a proof-of-concept implementation and analyze its performance.

Our construction can also be used in any setting where one would desire to privately learn heavy hitters in the presence of an active malicious adversary. For example, learning the most popular sites accessed by the TOR network.

Keywords: passwords, secure computation, differential privacy, heavy hitters, malicious model.

1 Introduction

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. Email: `benny@pinkas.net`

[‡]Department of Computer Science and Applied Mathematics, Weizmann Institute of Science Israel, Rehovot 76100, Israel. Email: `eyal.ronen@weizmann.ac.il`

with very little min-entropy[1]. The usage of popular and easy to guess passwords is one of the top security threats for users. Moreover, using an insecure password does not only endanger the user, but also endangers other users. A compromised account can be used for attacking the rest of the Internet. 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), they are here to stay for the foreseeable future, especially in the IoT world.

Web Cameras and the Mirai Attack

An example that demonstrates our setting is the case of web cameras. Web cameras allow their users to connect to the camera over the Internet, and watch the live video stream. Dedicated search engines such as Shodan [Wik17b] and the recent Mirai attack, demonstrate how hackers can find such devices, and hack them using a default or popular password. In the Mirai attack a huge number of compromised web-based cameras were used to mount a very large scale distributed DoS (DDoS) attack, sending a total of over 1Tbps and taking down large DNS servers [Wik17a].

Consequently, the Mirai attack motivated many well known security experts to demand *liability* from the manufactures of such insecure products [Sch]. However, many IoT devices today (especially low cost products) use passwords, as this is a cheap solution that is easy to use. Alternative solutions (such as two factor authentication) may increase cost and arguably hurt the usability of the system. 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 that manufacturers can take 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. Blacklisting them will greatly improve the situation but requires the server to know the current popular password distribution, which is hard for several reasons:

1. When a system prevents the usage of certain popular passwords, other passwords become popular and therefore need to be identified and prevented (for example, a popular password might change along the following sequence that is caused by changing password policies "password"$\rightarrow$ "Password"$\rightarrow$"Password1"$\rightarrow$"Password1!", etc.)

2. User populations in different locations (e.g. in the US vs. China) might use different popular passwords. Attackers can target their password guesses to the location of victims. Therefore, the system must identify popular passwords along different locations (and possibly other different dimensions such as age, etc.)

3. There might be passwords trends that are based on trending topics (for example, more users today (2017) will prefer "wonderwoman" to "superman" as their password then in the past).

[1]For example, according to a report by a company for secure password management, the 25 most frequent passwords, including passwords such as "123456", "password" and "qwerty", account for over 50% of over 10M passwords that were analyzed. https://blog.keepersecurity.com/2017/01/13/most-common-passwords-of-2016-research-study

In order to apply *effective* blacklisting, the server must learn the currently popular passwords. The simplest solution is for all users to send their passwords to the server, which will in turn identify and reject popular passwords. The major drawback of this solution is that the server learns all passwords (or hashes of these passwords, which are vulnerable to a dictionary attack). Many users will not want to send their passwords to a somewhat untrusted manufacturer or service provider, that has no other reason to know their passwords. Users may not want the manufacturer to maintain control over their cameras or to learn passwords which might also be used for other purposes (more then 80% of users reuse passwords between different sites and services[2]).

Tor network statistics

Tor network aims to provide better privacy and security for users in the Internet. To understand users' needs and to advocate for better privacy, the Tor network provide researchers with statistics about its operation [LMD10]. However, such statistic gathering is very limited as we need to protect the users' privacy. For example, we would like to enable researchers to privately learn the most popular websites visited by Tor users. A group of malicious users, might want to influence the results. For example, a repressive regime may want to delegitimize Tor, by making human rights sites seem less popular and drug trafficking sites more popular. For simplicity we will focus on passwords in the rest of the paper, but most of the solutions and requirements are easily translated to the Tor network setting.

1.1 Desiderata: Requirements from a Password Statistics Gathering Protocol

Our goal is to be able to collect the popular passwords while maintaining their privacy. We want to come up with a system satisfying several seemingly contradicting goals. We want the center (server) to learn the popular passwords, but without requiring the devices to divulge their passwords. Furthermore, we want the center to publish, essentially to the world, some information about these passwords, but without this allowing a coalition of devices to learn anything (useful) about the other devices' passwords.

What is perhaps unique about this paper is that the requirements must hold even in the face of a malicious (rather than semi-honest) and coordinated behavior of a coalition of devices that are under the control of the adversary. Namely, it is very likely that the goal of these devices is to perform an *undercount* attack which will reduce the perceived popularity of some popular passwords. This attack can keep these passwords under the radar, thus preventing them from being blacklisted and allowing them to become more popular. The attacker can then exploit the growing number of devices that use these passwords.

Let us reiterate the importance of this requirement: It is "legitimate" for a device that is controlled by the adversary to report an arbitrary password (rather than the true weak password that was used to break into the device). Such behavior increments the popularity count of some other passwords and does not change the count of the weak password. However, the protocol must prevent the adversary from sending data which decrements the popularity count of the weak password. As we shall see, this is the major technical challenge facing our construction.

A feature which helps the design of a relevant system is that the application we have in mind cares mostly about the most popular passwords and does not need very accurate estimates of the

[2]https://keepersecurity.com/assets/pdf/Keeper-Mobile-Survey-Infographic.pdf

popularity of the heavy hitters (distinguishing whether a password appears at %4 or %5 of the devices is not of the highest priority, as opposed to distinguishing between %0.1 and %5).

To summarize, any system for gathering statistics about password popularity must meet the following desiderata:

Heavy Hitters: The server must be able to learn the *approximate distribution* of the currently heavily used passwords, and publish this list to all.

Password not leaked to server or server gaining little information about the password: Both honest and adversarial servers should learn little information regarding the password. Since the goal of the password is to be hard to guess we can bound this information in bits and require that they learn, say, at most one bit of information about each device's password. This information cannot increase the probability of guessing the password by much.

Resiliency to a malicious server: In addition to a malicious server not learning much about any password, we do not want it to be able to perform a DoS attack on the system (not allowing any password). On the other hand, we do not protect the integrity of the published password list. To prevent a targeted attack against specific devices we require that the list will be publicly published.

Password privacy: The privacy of a single device's password must be preserved against any third party, even after repeated publications of the list. Here even leaking a bit may be problematic, since passwords could be related. Furthermore, device owners might opt out of the system (for example, by giving faulty information instead of their actual password) if information about their password is leaked to other users.

Functionality: Devices should be able to choose any password (with a very low false positive rate) and *repeatedly change* it over time.

Resiliency to malicious coalitions: Even a 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 overcounting (by more than one password per device in the coalition). We should handle dynamic password changes and early stopping in protocol by the devices.

These goals are motivated by observations about "real life". Big companies, by nature, try to learn as much information as possible about their users. On the other end, a botnet of devices is motivated to apply an "undercount" attack which will remove popular passwords from the published list.

We allow the server to learn *one* bit of information about each password. In general, leaking information is not good. But in the case of passwords, leaking less than a single bit may be benign, since passwords should have high min-entropy to begin with. In other words, the leaked information speeds up password cracking by a factor of at most 2. This should be tolerable compared to the benefit of ensuring that users do not use over popular passwords (thus increasing the min entropy). In addition, we describe a randomized response protocol which further reduces the amount of information leaked to the server (to less than one bit per device).

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.

Our Contributions

To accomplish the design goals we propose a general scheme for identifying over-popular passwords. The scheme is based on running very efficient secure protocols (secure against malicious adversaries), adding differential privacy for the devices, using an algorithm for approximately identifying heavy passwords, and bounding the possibility of errors. We also give concrete examples for the possible parameters, calculated using a simulation.

We then describe two suggestions for instantiations of secure protocols for this task (which are secure in the malicious setting). As discussed above, this level of security is essential since it is reasonable to assume that compromised devices might act maliciously and in a coordinated manner in order to obstruct the protocol.

Finally, we show run times of a proof-of-concept implementation that we have written on a sample platform.

1.2 Background

Differential Privacy (DP): Differential Privacy is a natural tool to discuss the limit on the the amount of information that is learned about a user's password; see Dwork and Roth [DR14] for a survey of the field. Two common techniques of DP are adding Lapalacian noise and randomized response. In the *randomized response* technique, in order 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 to retrieve, at least approximately, the real answer.

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 individual inputs. A key primitive of such protocols is oblivious transfer, where, roughly speaking one party, the receiver, holds a bit b and the other party, the sender, holds two strings x_0 and x_1. At the end of the protocol the receiver gets x_b, while the sender learns nothing.

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, 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 *full knowledge of the user's passwords* and is thus appropriate only *after* a serious leakage occurs. In our application we want to have periodic update and release of the popular passwords.

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, but they do not deal with malicious users (specifically the case of undercounting). In particular, since every user knows how its vote was counted, the user can redo the protocol until it gets a result it likes. Note that one difference is that their protocol does not tolerate false positives at all, i.e. from its point of view a this will translate to a nonsensical candidate for a heavy hitter. In our application it will mean a rejection of a legitimately strong password, which, if rare enough is innocuous. Bassily et al. [BNST17] have very 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. [CSS12] have given a lower bound on the approximation accuracy in the local model. None of these work in the adversarial setting, when the users are trying to make a popular item disappear.

Moran and Naor [MN06] considered *randomized response protocols* when some of the responders are trying to bias the result (see also Ambainis et al. [AJL04]). 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 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 simialr motivation to ours. However they require the user to reveal its full password to the server and do not offer DP for the user upon password publication.

1.3 Paper Structure

Section 2 describes the basic scheme, discusses the usage of DP to protect the device, and the possible attacks on the system and required secure functionality in the malicious setting. Section 3 discusses how the server generates the list of popular hashes and bounds the probability of false negatives in the semi-honest and malicious settings.

The next two sections describe two different methods for computing the secure functionality required for the general scheme in the malicious setting: Section 4 describes a secure protocol, based on garbled circuits. Section 5 describes a secure protocol which is based on the intractability of quadratic residuosity assumption (QR). 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 6 describes a proof of concept implementation of the QR-based protocol. Section 7 discusses the results and raises open questions.

2 The General Scheme

We first describe the basic construction, and then provide details on how to fulfil properties such as differential privacy and security against malicious behavior. A rough outline of the basic scheme is that learning the popular passwords can be reduced to learning a single bit via the Fourier coefficients of the distribution. 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.

2.1 The Basic Scheme

The system uses a system-wide hash function H, which maps passwords of arbitrary length to ℓ-bit outputs (a typical output length will be $\ell = 16, 24$ or 32). The output of the scheme reveals elements in the range of H to which over-popular passwords are mapped. Devices can then check if their password is mapped by H to one of these elements, and in that case ask the user to choose a new password. The scheme works in the following way:

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

2. Each 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 (the table is initialized to zero on system setup). 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. This equality holds for exactly half of the values, that we call the "correlated" values.

We denote the total number of users that have chosen a password as N_C, and p is the the fraction of users whose password is hashed by $H()$ to the value x. The expected number of increments and decrements is $N_C(p + (1-p)/2)$ and $N_C(1-p)/2$ respectively. We get that the expected value of the counter $E(T[x]) = \tau N_C$.

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

Running Time: The running time of the above protocol is 2^ℓ operations per update. This puts a constraint on the size of ℓ, but practical values of 16 or 32 are suitable for our needs.

2.2 User Differential Privacy

In our scheme only one bit of information about the user's password is leaked. In some cases even this amount of information 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 in order for the scheme to work, the 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 third parties viewing the list of popular hashes that is published by the server.

2.2.1 Pure Differential Privacy by Applying Randomized Response

To reduce the amount of information leaked to the server, the device can use randomized response. Namely, after hashing its password to v_j, it 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 $1 - \epsilon_r(1 - 2^{-\ell})/2$. From this we can conclude that this procedure provides the device with pure DP, with $\epsilon \approx \ln(2/\epsilon_r - 1)$.

2.2.2 (ϵ_n, δ) Differential Privacy by Applying Laplacian Noise

To ensure users that they have ϵ_n differential privacy from a third party, the server can add independent Laplacian noise $Lap(2/\epsilon_n)$ to each entry of the table (note that a change in a device's password can change the value of an entry by at most 2).

We comment that this procedure might not be sufficient as the server might need to periodically republish its current hash 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 list publications is not very large. Dwork et al. [DRV10] have shown the advance 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. This is not surprising as we need about $(\epsilon_n)^2$ samples to distinguish between the possible votes of a single device.

2.3 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:

2.3.1 A Malicious Server

A malicious server can tweak the hash list that is published. One option is to publish a very large list of popular passwords, and cause a large amount of false positives. This can be done in order 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 should publish at most $1/\tau$ popular passwords. Moreover the server has no incentive to do a DoS attack on its own system.

2.3.2 A Malicious Device

A coalition of malicious devices may try 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: A coalition of devices can try to "hide" some popular passwords, and cause a "false negative". This attack may result in a larger fraction of the users using the same (weak) password. 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 of a device that did not choose v_p to the counter $T[v_p]$ is 0. However, by choosing v s.t. $\langle v, r_j \rangle = -\langle v_p, r_j \rangle$ the contribution is -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 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 will only have a large effect if the attacker is able to do it on a large fraction of the passwords, and reduce the the min entropy. However, this attack is feasible only on a small number of password simultaneously, and will have negligible effect on the min entropy. Moreover, the solution we present for the undercount attack is also effective against the overcount attack, so we will focus only on the more severe undercount attack.

2.4 The Required Secure Functionality

To protect against malicious undercount attacks we need a prevent the possibility of sending a result bit that is anti correlated to a any value v (equal to $1 - \langle v, r_j^s \rangle$) with probability greater then 1/2. In fact, we only allow the device to send a result of an inner product. We define this functionality in Figure 2.1. Namely, 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.

FIGURE 2.1 (The inner-producr functionality).

- Input:
  - The server sends to the trusted party (TTP) an ℓ-bit input r_j^s.
  - The device sends to the TTP an ℓ-bit input v.
- Output:
  - The TTP sends to server the inner-product value $\langle v, r_j^s \rangle$.
  - The device learns no output.

An ideal model definition of the inner-producr functionality.

The functionality allows the device to randomize its response, by choosing a random input v. As v is independent of r_j^s, the result of the inner-product will be random.

The definition implies that a device cannot learn r_j^s before providing its input to the protocol.

In Sections 4 and 5 we show two secure protocols which implement the functionality of Figure 2.1 (or a small variant).

3 Generation of the Popular Hash List

We describe how the server uses the inner-product results from the devices, to generate and publish the list of popular hash values that users should avoid. We give an upper bound on the probability of false negatives or false positives in the semi-honest and the malicious settings.

Even when implementing the required secure functionality, the published hash list leaks information about the secret r^s vectors used by the server. Malicious devices can use this information for an undercount attack. We will bound the amount of leaked information, and the probability of false negatives caused by this undercount attack.

3.1 Notation

We use the following notation in the analysis:

1. N_C is the total number of unique devices that have chosen a password.
2. $N_P(pass)$ is the total number of devices currently using password $pass$.
3. $v = H(pass)$ is the hash value of password $pass$.
4. $N_H(v)$ is the total number of devices that are currently using any password such that $v = H(pass)$.
5. $T[v]$ is the value stored at the counter table with index v.
6. $\alpha(v)$ is the server's approximation of the number of votes for a hash value v.
7. ϵ_r is the probability that a device will randomize is response (for DP).
8. $\mathbb{1}(x)$ is 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.

3.2 Creating an Approximation of Popular Passwords

The approximation to the number of devices that currently using a password that hashes to 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 3.1. *If a fraction p of the devices choose a password which hashes to 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,

$$\begin{aligned} E(T[v] \mid N_H(v) = pN_C) &= N_C \cdot (p(1-\epsilon_r+\epsilon_r 2^{-\ell}) + (1-p)\epsilon_r 2^{-\ell}) \\ &= N_C \cdot (p(1-\epsilon_r)+\epsilon_r 2^{-\ell}) \end{aligned} \tag{4}$$

From (3) and (4) we get that the expected value of $\alpha(v)$ is:

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

□

3.3 Bounding False and Positive Negatives

The goal of our system is to prevent any password from becoming "too popular" – i.e., we want to identify passwords which are used by more than a fraction τ of the devices. Namely, identify any password *pass* for which $N_P(pass) > \tau N_C$.

As we can only learn an approximation of $N_P(pass)$, we will relax our requirement, to identifying any password for which $N_P(pass) > \tau N_C(1+\delta)$. We will also allow error on the other side – namely allow the server to declare as popular passwords which are only used by more than $\tau N_C(1-\delta)$ devices (but no password which is used by less devices should be declared as popular).

We first analyze the publication of the hash list in the semi-honest setting.

Estimating the false negative probability We define as a "false negative" the event that at a given time, some password *pass* was over the upper threshold, but the approximation $\alpha(H(pass))$ was below the threshold. Namely,

$$N_P(pass) > \tau N_C(1+\delta) \quad \wedge \quad \alpha(H(pass)) < \tau N_C$$

Lemma 3.2. *The probability for a false negative event, P_{fn}, is bounded by:*

$$P_{fn} \leq \frac{2\exp(-N_C(\tau\delta(1-\epsilon_r))^2/2)}{\tau(1+\delta)}$$

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

$$N_P(pass) < \tau N_C(1-\delta) \quad \wedge \quad \alpha(H(pass)) > \tau N_C$$

Lemma 3.3. *The probability for a false positive P_{fp} is bounded by:*

$$P_{fp} \leq \frac{2^{-\ell}}{\tau(1-\delta)}$$

Lemmata 3.2 and 3.3 are proved in Appendix A.

While we want to assure that a false negative will never happen, we can allow for a small probability of a false positive, since the only consequence is that users will be asked to choose another password.

3.3.1 Dynamic Threshold as a Function of N

To bound the probability of a false negative given by Lemma 3.2 we propose that the minimal threshold τ will be bounded such that the following constraint holds for some constant C:

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

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 a million users ($N_C = 10^6$), $\delta = \epsilon_r = 1/2$ and $C = 20$, we get that $\tau_{min} = 0.025$. We can also use a numerical approach to calculate P_{fn} and find a suitable τ_{min}.

3.4 Bounds in the Malicious Setting

In a malicious setting, a coalition of devices might try and cause a false negative using an undercount attack. However, as the system will use a secure inner-product protocol that is resilient to malicious devices (as described in Sections 4 and 5), the best strategy available for the coalition is to randomize their chosen hashes between repeated publications of the hash list, and see if this results in an undercount. The probability of success for this attack is exactly the probability of a false negative and is very low.

Nonetheless, the publication of the list of popular hash values might leak information about the secret r^s values. This information can 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 is correlated with a popular hash and choose another hash value in hope that the result will be anti-correlated. We therefore discuss in this section how the server can add noise to the published list of popular hash values, to reduce the probability of a successful undercount attack.

3.4.1 Information Leakage in the Malicious Setting

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_j^s values of each device. 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:

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

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

3.4.2 The Effect of the Laplacian Noise on False and Positive Negatives

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

$$\alpha_N(v) = \alpha(v) + N_L = N_H(v) + N_B + N_L$$

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

The 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, to allow for the extra noise. Moreover, the server generates new noise for each publication 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.

3.4.3 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 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 N_B, which is the sum of binomial noises. To bound the leakage we want to bound the mutual information.

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

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

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

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

3.4.4 Bounding the Probability of an Undercount Attack

As we have seen, the probability of a false negative without active participation is bounded as

$$P_{fn} \leq \frac{2\exp(-N_C(\tau\delta(1-\epsilon_r))^2/2)}{\tau(1+\delta)}.$$

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 an 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:

$$\begin{aligned} P_{mal-fn} = P_{fn} \cdot 2^{B(t)} &\leq \frac{2\exp(-N_C(\tau\delta(1-\epsilon_r))^2/2)}{\tau(1+\delta)} \cdot 2^{B(t)} \\ &= \frac{2\exp(-N_C(\tau\delta(1-\epsilon_r))^2/2 + B(t)/\log_2(e))}{\tau(1+\delta)} \\ &\approx \frac{2\exp((1.38B(t) - N_C(\tau\delta(1-\epsilon_r))^2)/2)}{\tau(1+\delta)} \end{aligned}$$

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

3.4.5 Dynamic 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).

To bound the probability of a malicious false negative, we require that for any time t, $P_{mal-fn} \leq \exp(-C)$ 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[3]

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

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:

$$L(LS) = L(t) + (LS - t)freq(t)l(t) \tag{5}$$

$$\begin{aligned} 2C &\leq N_C(LS)(\tau(LS)\delta(1-\epsilon_r))^2 - 1.38L(LS) \qquad (6) \\ &= N_C(t)(\tau(t)\delta(1-\epsilon_r))^2 - 1.38(L(t) + (LS - t)freq(t)l(t)) \end{aligned}$$

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

Number of Users	Life span	τ_{min}	τ	Publication period
10^5	10 year	5.3%	10.6%	7 days
10^6	10 year	1.7%	4%	7 days
10^7	10 year	0.5%	1.6 %	7 days

Table 1: Publication parameters in the malicious setting

Equation 5 gives an upper bound for the amount of information leaked in the rest of the lifespan if the statistics are published with frequency $freq(t)$ and threshold $\tau(t)$. Equation 6 gives us the required ratio needed to prevent malicious undercount attacks.

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

Table 1 demonstrates some examples for different tradeoffs between the parameters, assuming no previous leakage. The value of $b(t)$ needed for the calculation was simulated using Matlab with $L_N \sim Lap(\tau\delta N_C/2), \delta = 0.5, \epsilon_r = 0.25$.

4 An Inner Product Protocol Based on Garbled Circuits

Generic protocols for secure computation, such as Yao's garbled circuit protocol, can be used for securely computing any function. In particular, they can be used for computing the inner-product 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 in order to reduce the cheating probability to 2^{-s} [4].

Fortunately, the setting we are interested in enables a much more efficient implementation with security against malicious adversaries. This implementation is based on two features of the 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.[5] The server can therefore use the circuit to learn other one-bit functions of

[4]Note that in our case the server and the device are only interested in running a single computation, or perhaps a single computation following each password change, 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].

[5]The device can verify the topology of the circuit, but it cannot verify the functions computed by the gates.

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 Figure 4.1.

FIGURE 4.1 (The modified inner-product functionality)**.**

- Input:
  - The server sends to the TTP the following inputs:
    - An ℓ-bit input r_j^s.
    - 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 $F(h, r_j^s)$.
  - The device learns no output.

An ideal model definition of the modified inner-product functionality that is computed by Yao's protocol.

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.[6]

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

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

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 it lets the device learn one of the two garbled values of the corresponding input wire.

[6]This observation was used in the dual execution paradigm, in which the parties run the protocol twice where each party is the constructor in one run of the protocol and the evaluator (who learns the output) in the other run [MF06]. The parties then run a reconciliation step in which their garbled outputs are securely compared for equality. This protocol is known to leak at most one bit, namely whether the two outputs are equal or not. (Follow up work in this paradigm leaks no information, by running multiple copies of the circuits [KMRR15, RR16].) We need to provide an output to the server only, and therefore can run a single execution of the protocol instead of two executions.

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 (these two values are sent in random order).

4. The device evaluates the circuit and learns the garbled output value.

   The device then checks that the hash of this value is equal to one of 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. In terms of 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 protocol can therefore achieve an excellent run time even without implementing any optimizations to the basic Yao protocol (such as the half-gates garbling method). There is also no need to implement the oblivious transfers using OT extension, since the run time of each oblivious transfer is at most a few milliseconds (e.g. the implementation of the fully secure OT protocol of Chou and Orlandi [CO15] takes less than 500,000 cycles to run, and can therefore achieve a throughput of thousands or tens of thousands of OTs per second).

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 in [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 it is illegitimate to have a protocol run where the device fails in computing an output. When this happens, the device can complain about the behavior of the server (and if we require the server to sign the messages that it sends, then the device has a proof that the sender misbehaved). Moreover, since device manufacturers typically need to keep a good reputation, we can conclude that even though the selective failure attack is possible, it is unlikely that a device manufacturer will attempt to apply it.

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.

5 An Inner Product Protocol based on the Quadratic Residuosity (QR) Assumption

The goal of the protocol is to implement the secure functionality described in Figure 2.1 and calculate the inner product $\langle V^s, R^s \rangle$ between a device's (**D**) input V^s and a server's (**S**) input R^s in a malicious setting. **S** should only learn the result, while **D** should learn nothing [7] and will not be able to deviate from the protocol.

We use the following notation:

1. $\left(\frac{a}{N}\right)$ is the Jacobi symbol of a in respect to N, this is easy to calculate.
2. QR_N (nQR_N) is the set of all numbers that are (non) quadratic residues modulo N, and with 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. We assume that computing this function requires the knowledge of p and q.
4. $a \overset{R}{\leftarrow} QR_N$ (nQR_N) denotes that the value a is chosen uniformly at random (UAR) from QR_N (nQR_N).
5. $\pi_a \overset{R}{\leftarrow} \pi\{\ell\}$ denotes that the permutation π_a is chosen UAR from the family of permutations on ℓ items.
6. $x \overset{R}{\leftarrow} \{0,1\}^\ell$ denotes that the value x is chosen UAR from $\{0,1\}^\ell$.
7. Capital letters such as V^s denote binary number in $\{0,1\}$ while lowercase letters such as r^p are 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 determine whether $x \in QR_N$ or $x \in nQR_N$ where $\left(\frac{x}{N}\right) = 1$, without knowledge of the factorization of N.

We use this assumption for encrypting **S**'s vector R^s in a similar way to Goldwasser-Micali [GM84] public encryption scheme. 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 $ax \in nQR_N$.

5.1 Protocol - Semi Honest Version

S generates an RSA public and secret key pair $SK = (p,q), PK = n$ where p and q are large primes (the size of N depends on a security parameter k_1). **S** than encodes R^s in a public vector $r^p \in \mathbb{Z}_N^\ell$ that is sent to **D** . Each number in r_i^p is a public encryption of the corresponding bit R_i^s.

[7] Unless $V^s = 0$ as then $\langle 0, R^s \rangle = 0$

1. if $R_i^s = 0$: $r_i^p \xleftarrow{R} QR_N$, zero is encoded to a QR.
2. if $R_i^s = 1$: $r_i^p \xleftarrow{R} nQR_N$, one is encoded to a nQR.

D generates a random QR by generating a random number in $\mathbb{Z}_N$ and squaring it. Then **D** calculates the product of the numbers in r^p corresponding to the 1 bits in V^s, and blinds the result using the random QR:

$$e = d^2 \cdot \prod_{i=1}^{\ell} (r_i^p)^{V_i^s} \quad \text{where} \quad d \xleftarrow{R} \mathbb{Z}_N$$

After receiving e from **D** , the server **S** learns the result of the inner product:

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

Under the QR assumption **D** does not learn anything on R^s. Due to the homomorphic properties, 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 UAR in QR_N, then e is also distributed uniformly at random inside its class (either QR_N or nQR_N). This ensures that the **S** only learns the one-bit result of the inner product.

5.1.1 Insecurity in the malicious setting

Even under the QR assumption we do not know if it is hard to find a single number x such that $x \in nQR_N$. A malicious **D** may be able to generate such an x and then flip the value of the result by sending $e' = e \cdot x$. This allows an undercount attack on a specific password.

5.2 A Naive Implementation for the Malicious Setting

We will now show a naive implementation that is secure against a malicious **D**, but allows **S** to learn more than one bit of information on V^s.

The main idea is that **D** creates a vector r^* that is a "blinded" version of r^p. This is done by blinding each value in r^p and randomly permuting the vector. **D** sends **S** the list of indexes of the numbers in r^* used to calculate e.

We pad r^p a with ℓ numbers that are in QR_N. This is done to allow **D** to always send a product of ℓ numbers regardless of the Hamming weight of V^s. If $V_i^s = 0$ then we send the corresponding number from the padding.

Afterwards **D** proves to **S** that r^* is indeed a blind permutation of r^p. This is done by generating vectors r^j that are blinded permutations of r^*, and proving to **S** that **D** knows how to "open" them either to r^* or to r^p. The length of the proof depends on a security parameter k_2, where the probability of **D** to deviate from the protocol is 2^{-k_2}.

5.2.1 Device setup Phase

D generates and saves the following data.

1. $r = r^p || pad^2$ where $pad \overset{R}{\leftarrow} \mathbb{Z}_N^\ell$. We pad r^p with ℓ numbers in QR_N.
2. $V = V^s || \overline{V^s}$ where $\overline{V_i^s} = 1 - V_i^s$.
3. $\pi_* \overset{R}{\leftarrow} \pi\{2\ell\}$, $d^* \overset{R}{\leftarrow} \mathbb{Z}_N^{2\ell}$. **D** generates a random permutation π^* on 2ℓ items, while d^* are 2ℓ random numbers.
4. $r^* = \pi_*(r \cdot (d^*)^2)$, $V^* = \pi_*(V)$. r^* is a blinded random permutation of r, and V^* is the same permutation of 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 r corresponding to the 1 bits in V. The bit that is encoded is the result of the inner product.
6. For $j = 1..k_2$:
   (a) $\pi_j \overset{R}{\leftarrow} \pi\{2\ell\}$, $d^j \overset{R}{\leftarrow} \mathbb{Z}_N^{2\ell}$. **D** generates a random permutation π^* on 2ℓ items, while d^* are 2ℓ random numbers.
   (b) $r^j = \pi_j(r^* \cdot (d^j)^2)$ is a blinded random permutation of r^*
7. **D** sends (r^*, V^*, pad) to **S**.

5.2.2 Interactive proof phase

The server first computes the result.

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

Then the server verifies the result by applying the following interactive proof for $j = 1, \ldots, k_2$:

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

5.2.3 Information leakage in the naive implementation

S receives a blinded version of the numbers used in the inner product calculation (the numbers in r^* corresponding to the 1 bits in V^*). However, for any number x, **S** is still able to find the sign of $T_{QR_N}(x)$. This lets it learn much more information about V^s. For example **S** can choose R^s to be the all '1' value and use it to learn the Hamming weight of V^s by counting the number of numbers that are in nQR.

5.3 The Complete Protocol

We will now describe the complete protocol. The main difference from the naive implementation is that instead of sending a vector such as r^*, **D** sends the **squared vector** $s^* = (r^*)^2$. As all the numbers in s^* are in QR_N, this vector is indistinguishable from any other random vector in $QR_N^{2\ell}$ regardless of the values of V^s and r^p. This vector does not revel any more information about **D**. We will prove that under some restriction on the generation of N the soundness of the modified protocol holds. We shall now give a full description of the protocol:

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

5.3.1 Server global setup

S runs a setup algorithm $GlobalSetup(1_1^k)$, 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 **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 the extended version of the paper.

5.3.2 Server instance setup

For each **D** , **S** runs $ServerSetup(PK, SK, R^s, \ell)$, taking the private and public keys generated by $GlobalSetup$, and a secret vector $R^s \in \{0,1\}^\ell$. **S** encodes R^s in a public vector $r^p \in \mathbb{Z}_N^\ell$ that is sent to **D** . As in the semi honest version, each number in r_i^p is a public encryption of the corresponding bit R_i^s. **S** sends r^p to **D**.

5.3.3 Device instance setup

D runs $DeviceSetup(1^{k_2}, \ell, PK, r^p, V^s)$ taking a security parameter k_2, ℓ, PK, r^p and a private vector $V^s \in \{0,1\}^\ell$.

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

1. $r = r^p || pad^2$ where $pad \overset{R}{\leftarrow} \mathbb{Z}_N^\ell$. We pad r^p with ℓ numbers in QR_N.
2. $V = V^s || \overline{V^s}$ where $\overline{V_i^s} = 1 - V_i^s$.
3. $\pi_* \overset{R}{\leftarrow} \pi\{2\ell\}$, $d^* \overset{R}{\leftarrow} \mathbb{Z}_N^{2\ell}$. Namely π^* is a random permutation on 2ℓ items, and d^* are 2ℓ random numbers.
4. $r^* = \pi_*(r \cdot (d^*)^2)$, $V^* = \pi_*(V)$. Namely, r^* is a blinded random permutation of r, and V^* is the same permutation of V.
5. $s^* = (r^*)^2$.
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 r corresponding to the 1 bits in V, and therefore encodes the inner product result.

D sends (s^*, e, V^*, pad) to **S** . Notice that this time **S** can only calculate e^2 by itself, and therefore it is also required to send e.

5.3.4 Inner product calculation and verification

D and **S** run the following protocol. If any step or calculation results in $\perp$, or a verification fails, then **S** outputs reject. Otherwise **S** outputs accept and can use the value $result = \langle V^s, R^s \rangle$.

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

5.3.5 Non-interactive proof

The interactive part of the protocol can be converted to a non-interactive proof. As the results of **S** setup can be stored on **D** , this allows the implemented protocol to consist of only one message from **D** to **S** .

The conversion is done using the Fiat-Shamir heuristic [FS86] in the (non programmable) random oracle model. Unlike hashing the passwords, the interactive proof requires a strong cryptographic hash function such as SHA3 [NIS14].

5.4 Completeness and Soundness

The proof of soundness appears in Appendix B. With regards to completeness, it is easy to see that if both **S** and **D** follow the protocol then the protocol ends successfully as the following statements hold:

$$\begin{aligned} T_{QR_N}(e) &= \langle V^s, R^s \rangle \\ e^2 &= \prod_{i=1}^{2\ell} (s_i^*)^{v_i^*} \\ \forall j, s^j &= \pi_j(s^* \cdot d_j^2) \\ \forall j, s^j &= \pi_j^*(r_{sqr}) \cdot blind_j^2 \end{aligned}$$

5.5 Zero-Knowledge Proof

5.5.1 Privacy of R^s

The encoding of R^s into r^p is a semantically secure encryption [GM84], and does not reveal any information on R^s to any PPT algorithm.

5.5.2 Privacy of V^s

There is a simulator $\mathbf{S_D}$ that given $\langle V^s, R^s \rangle, R^s$ can simulate **D** . $\mathbf{S_D}$ chooses UAR V' such that $result = \langle V', R^s \rangle = \langle V^s, R^s \rangle$. It then runs the protocol as **D** with V' as the input.

The distribution of all values is the same between **D** and $\mathbf{S_D}$: All values of s^*, 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 B). Both π_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 V^* is also a uniformly distributed random value with Hamming weight ℓ.

5.6 Implementation Considerations

5.6.1 Reusing r^p

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

As r^p is fixed, it can be stored on **D** in production and not sent over the network. This saves the first message in the protocol.

5.6.2 Interactive vs. Non-Interactive

As was mentioned above, the protocol can be turned into a non-interactive protocol. This has the great benefit of allowing **D** to prepare all messages offline. **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 for the protocol $O(2^{k_2})$, and we will usually require a much larger value of k_2 (say, equal to 80 or 128).

Although the non-interactive protocol is simpler, the requirement of using a larger value of k_2 is more suitable for IoT devices such as web cameras with enough memory and processing power. Devices with lower performance or with tight power restrictions should use the interactive protocol.

5.6.3 Communication complexity

We calculate the amount of bits that need to be communicated in the protocol. As we assume **S** generates a fixed r^p per **D** , we will only consider the size of **D**'s proof (and neglect the k_2 bits sent by **S**).

The size of the representation of each permutation is $2\ell|2\ell| = 4\ell \log \ell$. The size of each vector is $|(s^*, e, v^*, pad)| = k_1(2\ell+1+\ell)+2\ell \approx 3\ell k_1$, $|(s^j, \pi^{j*}, blind_j)| = |(s^j, \pi^j, d^j)| = k_1 4\ell + 4\ell \log \ell \approx 4\ell k_1$.

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.

6 PoC of the QR protocol

To test the feasibility of our solution we implemented a proof-of-concept (PoC) of our QR protocol in python (the source code will be published for reference). The implementation was done in python in order to demonstrate that even an unoptimized implementation of the QR protocol, which demands more resources than the Yao based protocol, is sufficiently efficient for many IoT devices. To simulate execution on an IoT device such as a high definition web camera, we ran our implementation on a Raspberry Pi 3 Model B [ras], and ran the server side on a laptop with a i7-7500U 2.7GHz CPU.

The protocol requires a large amount of random data for blinding and for computing permutations. To optimize this process we implemented a simple AES based pseudo-random number generator (PRNG) that uses a random seed from the operating system. Even with this optimization, the generation of random numbers consumed more than 80% of the total run time. This step can either be done in a preprocessing step, or be accelerated by using the AES instruction set built in to many modern embedded CPUs.

For our test we chose to use a modulus N of size 2048 bits and $\ell = 16$. We tested the server and device run times for 2 different sizes of k_2, using $k_2 = 20$ for the interactive version and $k_2 = 128$ for the non-interactive version. The measurements are only for the processing time.

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 counter.

These measurements demonstrate that the protocol can be implemented even in very low-end devices. After the user chooses the password, the device only needs to verify that the hash of the password is not in the published list, before allowing the user use the new password. Afterwards the device can run the protocol in the background in order to update the server about the value of the new password. Since there is no real-time constraint for this phase, the protocol can run for a relatively long period of time (a few seconds), without causing any inconvenience to the user. Moreover, recent low power IoT processors like Texas Instruments' CC2538 Zigbee SoC [cc2] include ECC and RSA hardware accelerators that can be used for efficient implementations of our protocols.

7 Discussion and Open Questions

Comparing the QR and garbled circuits solutions We have shown two different approaches for implementing the secure inner-product protocol. Those two approaches offer an interesting tradeoff. The garbled circuit solution is more efficient both in run time and in bandwidth. On the other hand it requires an interactive protocol and a 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.[8]

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.

An open question – is cryptography needed? We described a protocol for the semi-honest setting which does not require any 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, in order to guarantee security against a malicious device, our protocol instantiations use OT or public key cryptography. An interesting question is whether protecting against an undercount attack implies the existence of OT or of other cryptographic primitives. It is an open problem to either prove this claim or show an alternative protocol.

References

[AJL04] Andris Ambainis, Markus Jakobsson, and Helger Lipmaa. Cryptographic randomized response techniques. In *Public Key Cryptography - PKC 2004*, 2004.

[AMPR14] Arash Afshar, Payman Mohassel, Benny Pinkas, and Ben Riva. Non-interactive secure computation based on cut-and-choose. In *EUROCRYPT*, 2014.

[8]There exist non-interactive constructions of Yao's protocol. The constructions with semi-honest security might leak more than a single bit. The constructions with security against malicious players, e.g. [AMPR14], are far less efficient than our Yao based protocol. E.g., they use the cut-and-choose paradigm which requires evaluating multiple copies of the circuit.

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

[BNST17] Raef Bassily, Kobbi Nissim, Uri Stemmer, and Abhradeep Thakurta. Practical locally private heavy hitters. *CoRR*, abs/1707.04982, 2017.

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

[cc2] Texas instruments - cc2538.

[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.

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

[CSS12] T.-H. Hubert Chan, Elaine Shi, and Dawn Song. Optimal lower bound for differentially private multi-party aggregation. In *Algorithms - ESA 2012 - 20th Annual European Symposium, Ljubljana, Slovenia, September 10-12, 2012. Proceedings*, pages 277–288, 2012.

[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.

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

[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.

[Hoe63] Wassily Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American statistical association*, 58(301):13–30, 1963.

[KMRR15] Vladimir Kolesnikov, Payman Mohassel, Ben Riva, and Mike Rosulek. Richer efficiency/security trade-offs in 2pc. In *TCC*, 2015.

[LMD10] Karsten Loesing, Steven J. Murdoch, and Roger Dingledine. A case study on measuring statistical data in the Tor anonymity network. In *Proceedings of the Workshop on Ethics in Computer Security Research (WECSR 2010)*, LNCS. Springer, January 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 *ACM CCS*, 2015.

[MF06] Payman Mohassel and Matthew K. Franklin. Efficiency tradeoffs for malicious two-party computation. In *Public Key Cryptography - PKC 2006*, pages 458–473, 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.

[NIS14] NIST. Sha-3 standard: Permutation-based hash and extendable-output functions. 2014.

[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*, 2016.

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

[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.

[Wik17a] Wikipedia. Mirai (malware) — wikipedia, the free encyclopedia, 2017. [Online; accessed 12-May-2017].

[Wik17b] Wikipedia. Shodan (website) — wikipedia, the free encyclopedia, 2017. [Online; accessed 19-May-2017].

A Proofs

We prove here Lemma 3.2 and Lemma 3.3 of Section 3.

Lemma 3.2. The probability for a false negative P_{fn} is bounded by $P_{fn} \leq 2\exp(-N_C(\tau\delta(1-\epsilon_r))^2/2) \;/\; \tau(1+\delta$

Proof. The worst case is when no other password collides with *pass* and

$$N_H(v = H(pass)) = N_P(pass) = \tau N_C(1+\delta)$$

$\alpha(v)$ 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:

$$\begin{aligned}
&\Pr(\alpha(v) \leq \tau N_C | N_H(v){=}\tau N_C(1{+}\delta)) \\
&\quad = \Pr(\alpha(v) - \tau N_C(1+\delta) \leq -\tau N_C\delta | N_H(v){=}\tau N_C(1{+}\delta)) \\
&\quad = \Pr(\alpha(v) - E(\alpha(v)) \leq -\tau N_C\delta | N_H(v) = \tau N_C(1+\delta)) \\
&\quad \leq \Pr(|\alpha(v) - E(\alpha(v))| \geq \tau N_C\delta | N_H(v) = \tau N_C(1+\delta)) \\
&\quad \leq 2\exp(-\frac{2(\tau N_C\delta(1-\epsilon_r))^2}{4N_C}) = 2\exp(-N_C(\tau\delta(1-\epsilon_r))^2/2)
\end{aligned} \tag{7}$$

As there are at most $\frac{1}{\tau(1+\delta)}$ bins where $N_H(v) \geq \tau N_C(1+\delta)$, this is the maximum number of bins where we can have a false negative. By the union bound the probability of getting at least one false negative is $P_{fn} \leq 2\exp(-N_C(\tau\delta(1-\epsilon_r))^2/2) \;/\; \tau(1+\delta)$. □

Lemma 3.3. The probability for a false positive P_{fp} is bounded by $P_{fp} \leq 2^{-\ell}/\tau(1+\delta)$.

Proof. In a similar manner to Equation 7 we can show that

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

There are 2^ℓ bins, so the expected number of false positive is at most

$$2^\ell \exp(-N_C(\tau\delta(1-\epsilon_r))^2/2) \approx 0$$

As long as $N_C(\tau\delta(1-\epsilon_r))^2/2$ is large enough we expect to have at most $\frac{1}{\tau(1-\delta)}$ bins where $N_H(v) \geq \tau N_C(1-\delta)$, and the probability of a device getting a false positive is $P_{fp} \leq 2^{-\ell} \;/\; \tau(1-\delta)$. □

B Soundness of the QR Protocol

Theorem B.1. *For any (possibly not efficiently computable) adversarial algorithm $\boldsymbol{A_D}$, if $T_{QR_N}(e) \neq \langle V^s, R^s\rangle$ then $\Pr_{b \xleftarrow{R} \{0,1\}^{k_2}}[\boldsymbol{S} = accept] \leq 2^{-k_2}$.*

Theorem B.1 if proved using the following two lemmas.

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

Proof. $\Pr_{b_j \overset{R}{\leftarrow} 0,1}[\mathbf{S} \neq reject] \geq \frac{1}{2}$ only if $\mathbf{A_D}$ knows $(\pi_j, d_j, \pi_j^*, d_j^*)$ such that:

$$s^j = \pi_j(s^* \cdot d_j^4) \tag{8}$$

$$s^j = \pi_j^*(r^2) \cdot (d_j^*)^4 \tag{9}$$

From 8 $\mathbf{A_D}$ can calculate

$$s^* = \pi_j^{-1}(s^j) \cdot d_j^{-4}$$

From 8 + 9 $\mathbf{A_D}$ can calculate

$$s^* = \pi_j^{-1}(\pi_j^*(r^2) \cdot (d_j^*)^4) \cdot d_j^{-4} = \pi_j^{-1}\pi_j^*(r^2 \cdot ((\pi_j^*)^{-1}((d_j^*) \cdot \pi_j(d_j^{-1})))^4)$$

We denote $\pi_* = \pi_j^{-1}\pi_j^*$ $d^* = ((\pi_j^*)^{-1}((d_j^*) \cdot \pi_j(d_j^{-1}))$

$$r^* = \pi_*(r \cdot (d^*)^2), \quad s^* = \pi_*(r \cdot (d^*)^2)^2 = (r^*)^2 \tag{10}$$

And $\mathbf{A_D}$ can calculate (π_*, d^*). ☐

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

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

$$e^2 = \prod_{i=1}^{2\ell}(s^*)^{v_i^*} \tag{11}$$

From 10 + 11 we get that e' is a root of e^2.

$$(e')^2 = (\prod_{i=1}^{2\ell}(r^*)^{v_i^*})^2 = \prod_{i=1}^{2\ell}((r^*)^2)^{v_i^*} = \prod_{i=1}^{2\ell}(s^*)^{v_i^*} = e^2$$

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 $T_{QR_N}(e) = \langle V^s, R^s \rangle$. If $T_{QR_N}(e') \neq \langle V^s, R^s \rangle$ than $e \neq \pm e'$ and $\mathbf{A_D}$ can calculate all 4 roots of N, and that is equivalent to factoring. ☐

Supplementary Material

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 $1, \ldots, k_3$ 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 $i = 1, \ldots, k_3$:

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 use the notation $N_{HQR} = |v_i \in QR_N|$ 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:

$$\begin{aligned}\Pr(N_{HQR} \geq k_3/4 | p_{QR} \leq 1/8) &\leq \Pr(N_{HQR} \geq k_3/4 | p_{QR} = 1/8) \\ &= \Pr(N_{HQR} - k_3/8 \geq k_3/8 | p_{QR} = 1/8) \\ &= \Pr(N_{HQR} - E(N_{HQR}) \geq k_3/8 | p_{QR} = 1/8) \\ &\leq \exp(-2\frac{k_3^2}{8^2 k_3}) = \exp(-\frac{k_3}{32})\end{aligned} \tag{12}$$

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.