Detection of Password Reuse and Credential Stuffing: A Server-side Approach

Sai Sandilya Konduru*, Sweta Mishra*
* *Shiv Nadar Institution of Eminence, Delhi NCR*
{*ks585, sweta.mishra*}*@snu.edu.in*

***Abstract*—Considering password-based authentication technique, password memorability is a real challenge on users. Hence, password reuse across different web applications is a common trend among users which makes websites vulnerable to credential stuffing attack. A solution as password manager helps the users to create random passwords for different websites on the user machine. However, it has practical challenges.**

Password database breach detection is another related and challenging task. Among recent developments for breach detection, honeyword-based approach is much appreciated by the research community. However, honeyword generation itself is a challenging part of the solution.

In this work, we propose i) Password Reuse Detection (PRD) protocol for detecting password reuse using a secure two party private set intersection; ii) Breach Detection (BD) protocol that detects credential stuffing attacks using two party private set inclusion protocol based on random oblivious transfer. Both the proposals are designed for the authentication servers of the respective applications and need communication between multiple websites following the work by wang et al. Through analysis we show that our PRD protocol is around 2.8 times faster, and space efficient than existing works for 5000 honeywords. Our near to real-time BD protcol is around 2 times faster than existing works.

***Index Terms*—Password reuse, Breach detection, Credential stuffing, Private set intersection, Password hashing, Honeywords.**

1. Introduction

Credential breaches have become a serious and a common threat to both users and organisations. Credential leaks occur in almost every industry and compromise millions of users data. According to IBM, the average time for detecting a breach is around 206 days [1]. Meanwhile, the adversary can access the users data from the breached website and can try to log-in with the breached credentials at other websites assuming users might hold an account. Adversary will succeed only if the user had set same password at those websites. These types of attacks are often referred as credential stuffing attacks. Recent studies [2]–[4] have found that significant percentage of users reuse their password at different websites. Another study [5] has shown that password reuse is a major reason for database breaches. Websites like Facebook buys black market passwords to ensure the safety of users accounts [6]. Hence, preventing password reuse reduces the number of credential leaks dramatically.

Most of the breach detection systems use the concept of honeywords [7]. Honeywords are decoy passwords which are stored along with original password of the user. Apart from password authentication database, websites maintain an additional server called honeychecker which stores the index of user chosen password. Submitting any of the honeywords as passwords during the login attempt detects a breach. This is because when an adversary breaches an account he cannot distinguish honeywords from user chosen password. In this approach, breach detection is possible only at the breached website. However, our work on breach detection considers a completely different scenario which involves two websites. Database breach occurs at one website and gets detected by the other one. Goal of our breach detection system is to mitigate credential stuffing attacks.

There are very few significant works which talk about the password reuse problem [8] and breach detection mechanisms [9]–[11]. The work on password reuse detection by Wang et al's [8] in 2019 uses trusted third party called Directory. The Directory acts like bridge between two websites that check for a password reuse. We propose a new protocol called Password Reuse Detection (PRD) protocol to detect password reuse among websites without relying on any third party. We also propose a breach detection model which is similar as Amnesia proposed by Wang et al's [9]. The breach detection model Amnesia uses homomorphic encryption to identify credential stuffing attack. Our proposed protocol uses Private Set Inclusion to detect credential stuffing attacks and enhances the performance in terms of time and space complexities when compared to Amnesia.

Contribution: Following are the contributions of this paper.

1) **Protocol for Password Reuse Detection:** We propose a secure two party protocol called Password Reuse Detection (PRD) Protocol to detect password reuse without revealing the real values of the involved credentials. The two parties in the protocol are the two websites, namely, Target and Monitor. PRD is based on the scenario that a user holds an account at Monitor website with their user id as id_1 and uses same id_1 to register at the Target Website. On receiving the registration request, Target communicates with Monitor to detect password reuse and notifies the user accordingly. We propose two versions of PRD protocol based on the

number of honeywords. Following are the two versions of our protocol.

   a. **PRD.V1:** This version is mainly based on Diffie-Hellman based PSI and recommended when number of honeywords are less than 500.
   b. **PRD.V2:** This version uses an Oblivious Pseudo Random Function based Private Set Intersection protocol and recommended when number of honeywords are in the range of 500-5000.

2) We compare our PRD protocol with the work by Wang et. al. proposed in the year 2019 and show that ours is computationally more efficient.
3) **Protocol for Detecting Credential Stuffing:** The goal of this protocol is to detect credential stuffing attacks thereby detecting a breach. We assume that an adversary breaches the database as many times as possible and then try to guess password by applying dictionary attack. Our proposed protocol is based on the following scenario. The breach has occurred at the Target website and an attacker is trying to login at Monitor with stolen credentials from Target website. We assume that both websites implement honeywords. For every three consecutive unsuccessful login attempts at Monitor, Monitor checks for a breach using Oblivious Transfer (OT) based Private Set Inclusion protocol by communicating with the Target website. If the password entered at Monitor is in Target's list of passwords, a breach gets detected at Target Website.
4) Our breach detection protocol provides real-time response whereas the best result (Amnesia) proposed by Wang et al., is not real-time.
5) We show that our breach detection protocol is computationally more efficient than Amnesia.

The remaining sections of the paper are organised as follows: The Section 2 discusses about related work and Section 3 explains the technical concepts involved in our proposed protocols. The Section 4 gives a detailed description of PRD.V1 and PRD.V2 protocols. Section 5 focuses on the security analysis of our protocols. Next, Section 6 includes the comparative results with existing protocols. In Section 7 we discuss other significant attacks and possible ways to mitigate them. Finally, we conclude our work and scope of future work in Section 8.

2. Related Work

In this section we discuss about the existing solutions related to password reuse prevention. Existing mechanisms take one of the following approaches: i) Client side approach or ii) Server side approach. In client side approach, existing solutions are password managers like 1password ping [12] and google password manager [13]. These password managers create strong password on behalf of user. Since password managers create strong random passwords for each websites, possibility of password reuse is limited. A recent survey [14] shows that 65% of users do not trust password managers and hence new approach is required to mitigate the problem of password reuse. In server side approach, generally computations are done on the server side to check for password reuse without the intervention of user. In this approach, identification is done by username of the user. As most of the users keeps same username or e-mail ID across different websites. Hence, it is easy to identify the password reuse assuming same username used across different websites. To the best of our knowledge, the work by Wang et al [8] is the first one to introduce the concept of password reuse across different websites. Other significant works are [11], [15]–[18].

A new framework was introduced in 2019 [8]. It proposed a new Private Memebership Test (PMT) protocol which is based on partial homomorphic encryption to identify use of similar passwords. According to the framework, the important terminologies are Responder (websites at which the user already holds an account), Requester (websites at which the user is willing to register new account) Apart from Requester and Responder, they employ a trusted third party service called Directory. Communication between Requester and Responder website is done via directory. Directory stores all the usernames (pseudonymous) of the Requester website and a list of Responders corresponding to the username. During registration the user enters their username (assuming same as at other Responders) and their choice of password. Requester website creates a bloom filter [19] with user chosen password using k number of hash functions. The Requester website sends a request to the directory which includes username, hash functions and ciphertext of indices of the bloom filter. The directory forwards the request to a subset of Responders. After receiving the message from Directory, Responder constructs a bloom filter of same length as constructed by the Requester. Responder contains a list of similar passwords and honeywords [7], [20], [21] corresponding to the user's password. Responder stores these passwords (both honey and similar) corresponding to the username in the bloom filter using the received hash functions. Responder computes a homomorphic operation between the received encrypted indices and its indices of the bloom filter. The result is encrypted with a public key of Requester website and sent back to the Requester website via Directory. Requester decrypts with the corresponding secret key, if it finds a password reuse, it rejects the previously entered password and asks the user for a new one. The disadvantages of the framework are: use of a trusted third party as a directory and the running time of the protocol is more if both the parties uses TOR [22] network than if they do not use TOR.

There are several existing works on breach detection techniques, however, our work is very much aligned with the work by Wang et al [9] proposed in 2021 is very close to our work. They introduced a new framework to detect credential stuffing attack by using a protocol called Private Containment Retrieval (PCR) Protocol. This PCR protocol consists of 3 stages namely, PCR request, PCR Response

and PCR Reveal. According to their framework, the website where the adversary already breached the database and got access to the user's credential (username, passwords including honeywords) is referred to as Target website. The adversary then implements a credential stuffing attack at some other website called Monitor website assuming the compromised user of Target website might hold an account at Monitor as well. It is assumed that both the Target website and Monitor website mutually agree to participate in the protocol. At some arbitrary time, Target website sends a PCR request which consists of username, public key, salt, set of passwords (both honeywords and user chosen password) corresponding to the username. Monitor website saves the request locally and responds after every unsuccessful login attempt by the user. Since the Target website uses honeywords, the adversary cannot identify the original password. On unsuccessful attempt, the Monitor computes a homomorphic operation between the entered password at the Monitor and the received set of passwords from the Target website. The result is encrypted with the received public key and sent back to the Target (PCR Response). The Target decrypts with its secret key and checks if any of the honeywords is used for login attempt (PCR Reveal). Therefore, there is a high chance to try with a honeyword that could be detected at the later stage of the protocol. If yes, a breach of Target website is detected. The main drawback of this model is that it is not a real time detection system.

There are also services/works like Have I Been Pwned [15] and Microsoft's password monitor system [17] and Kurt et al [16], Bijeeta et al [23] which notifies the user if they are involved in a breach. These breach detection systems contains a database of already breached credentials and checks whether the user account/password is in the breached database following a privacy preserving technique. Since breach detection takes around 206 days as per IBM report [1], by the time the user is notified there would be a significant amount of damage.

3. Technical Overview

In this section we will give an overview of the relevant technical concepts that we have used for our protocol.

3.1. Oblivious Transfer (OT)

The concept of Oblivious Transfer (OT) was first introduced by Rabin in 1981 [24]. In this setting, the sender has 2 messages and receiver has a choice bit. Receiver receives a message bit corresponding to the choice bit with a probability of 0.5. While, the sender remains oblivious as to whether or not receiver received the message. Another setting called $1-out-of-2$ OT which means the sender has two inputs say (m_0, m_1) and the receiver has one choice bit $i \in \{0,1\}$. If $i = 0$, receiver wants to receive m_0, if $i = 1$, receiver wants to receive m_1. The receiver receives the message of its choice without learning anything about other messages whereas the sender sends the message as per receiver's choice without knowing anything about receiver's choice bit.

1-out-of-2 OT can be extended to 1-out-of-N OT in which the sender has n pairs of messages each of length l bits and receiver has an $n - bit$ choice vector. In this case, the receiver forms $n \times l$ matrix and perform an OT on each row. In practice, $n >> l$. Since the OT uses public key operations, we require high computation for 1-out-of-N OT.

In 2003, Ishai et al proposed a new protocol for OT extension, IKNP protocol, which computes large number of OTs by performing k base OTs [25]. That means, 1-out-of-N OT can be computed by performing k number of base OTs. In practice, $k << N$ and number of OTs required to perform is independent of number of message pairs. Let us consider a case where a sender has n message pairs each of $l - bit$ long ($n \in \{0,1\}^l$) and receiver has an n-bit choice vector r i.e., $r \in \{0,1\}^n$. We follow the same notation as used in [25], [26]. Initially, sender creates a random $k - bit$ string s and receiver creates two $n \times k$ random matrices T and U. Let t_j and u_j denote the j^{th} row of T and U matrices respectively, where,

$$t_j \oplus u_j = \begin{cases} 1^k, & \text{if } r_j = 1 \\ 0^k, & \text{if } r_j = 0 \end{cases}$$

Now, Both the sender and receiver performs 1-out-of-k OTs where the role of sender and receiver is exchanged from the usual setting. Receiver inputs its $k - bit$ string s as choice vector and sender inputs its matrices as messages to the OT. Note that sender's inputs in each of k OTs to be performed is $n - bit$ long. The sender can run k number of OTs with $k-bit$ inputs and use a Pseudo Random Generator (PRG) to get the effect of k OTs with $n - bit$ inputs. The outputs which sender gets with its choice vector is put into a matrix Q, where the jth row of matrix Q is defined as follows:

$$q_j = t_j \oplus [r_j.s] = \begin{cases} t_j, & \text{if } r_j = 0 \\ t_j \oplus s, & \text{if } r_j = 1 \end{cases}$$

In 2013, Kolesnikov and Kumaresan observed that the matrices T and U on receiver side are prepared in such a way that rows in $T \oplus U$ is either all ones or zeros [26]. They interpreted the rows with all zeros or ones as repetition code and replaced this repetition code with linear error correcting code with large distance as selection bit encoding. Using the concept of linear error correcting code q_j can be interpreted as $q_j = t_j \oplus [c(r_j).s]$ and $H(t_j)$ as $H(q_j \oplus [c(r_j).s]$ where c is the error correcting code. They also proved that for security parameter k, the OT extension offers $O(log\ k)$ factor performance improvement in computation and communication than compared to IKNP [25]. They interpret this as an Oblivious Pseudo Random Function (OPRF).

3.2. Private Set Intersection (PSI) protocol

PSI setting allows two parties p_1 and p_2 with sets $A = \{a_1, a_2, \ldots, a_m\}$ and $B = \{b_1, b_2, \ldots, b_n\}$ respec-

tively to learn nothing more than their intersection elements ($A \cap B$). Our work mainly deals with the balanced case where size of the sets are same at both the parties and hence our PSI protocols are also for balanced case. There are several applications of PSI such as bot net detection [27], human genome [28], relationship path discovery in social networks [29], etc. The PSI protocols can be categorized based on the underlying primitive as hashing, Public key based PSI, OT based PSI. Out of all the above mentioned methods, naive hashing is the simplest one. In naive hashing, elements of both the sets are hashed using cryptographic hash functions. Element wise hashed values are compared with the common elements.

3.2.1. Diffie-Hellman based PSI. Diffie-Hellman based PSI is first introduced in 1986 by Meadow et al [30] later improved by Huberman et al in 1999 [31]. The protocol works as follows: Let x and y be the private keys of user A and user B respectively Set $A = \{a_1, a_2, \ldots, a_n\}$ and $B = \{b_1, b_2, \ldots, b_n\}$. Both A and B hashes their set elements and raises the hash output to the power of x and y respectively and gets $A' = (H(a_1), H(a_2), \ldots, H(a_n))^x$ and $B' = (H(b_1), H(b_2), \ldots, H(b_n))^y$. Both the users exchange their values. Both the parties raise their received values (after exchange) to the power of their secret keys. (A'' = $(H(b_1), H(b_2),\ldots, H(b_n))^y)^x$ and $B'' = ((H(a_1), H(a_2), \ldots, H(a_n))^x)^y$. One of the party sends its results to the other to compare the values of A'' and B''.

Diffie-Hellman based PSI is very easy to implement but the cost of computing exponentiation for each element is high. Hence, Diffie-Hellman based PSI is preferred if parties hold smaller sets.

3.2.2. OT based PSI. It is the most efficient PSI than compared to other variants. In 2014, pinkas et al proposed a PSI protocol [32] based on random OT [33]. The sender inputs its set to the random OT and receives all message-pairs irrespective of its set elements whereas receiver only receives the messages corresponding to their set of elements. Sender then selects the messages based on its set elements, XORs the selected messages and sends to the receiver. The receiver inputs its choice bits to random OT, gets the corresponding messages and XOR the messages. Receiver compares this XOR output with the value received from the sender. This requires n^2 comparison between 2 sets. Pinkas et al [32] uses cuckoo hashing to reduce the comparing complexity.

In 2015, Pinkas et al improved [32] by using a different hashing technique called permutation based hashing (discussed in later subsection) to reduce the length of strings [34]. The computational complexity eventually decreased from $O(n\ log^2 n)$ to $O(nlog\ n)$.

In 2016, Kolesnikov and Kumaresan proposed a new 2 party PSI protocol based on OPRF [35]. This protocol can compute 2 million size sets in about 4 seconds. An OPRF protocol, consists of two parties namely, sender and receiver. Sender holds a function F_s where s is the seed while the receiver has an input x. At the end of the protocol, receiver receives $F_s(x)$ (result of the PRF on a single input x chosen by the receiver) without knowing anything about function F while the sender will not learn anything about x at the end of the protocol. General OPRF allows receivers to evaluate the PRF on several inputs but proposed OPRF [35] allows the receiver to evaluate only at single point. Hence they are interpreted as single point OPRFs. OTs are used to generate large batches of OPRF instances. In this work, sender inputs its elements in a PRF (F) using the seed s. The outputs of PRF ($F(s, r)$) are stored in a cuckoo hash table on receiver's side. Sender is not aware of which location (bin) corresponds to which seed. The sender sends the PRF values of both seeds to the receiver (assuming two hash functions are used to map elements in to bins in cuckoo hashing). In addition to this, If all the elements are not mapped to the cuckoo hashing, the sender might have to send an extra stash (an array) to the receiver. The receiver finds an intersection if any between its PRF values and received PRF values. It is shown that this protocol is highly efficient in terms of computation but not in communication because of sending multiple PRF values per element.

In 2019, Pinkas et al proposed a new PSI protocol [36] based on OT extension [25]. They are the first one to reduce the communication complexity of the protocol to linear($O(n)$ where n is the number of elements in the set) using general assumptions i.e, One Way Function (OWF) and OT. Instead of using a PRG, Pinkas et al [36] uses OPRF which allows the sender to learn a PRF F and allows the receiver to learn $F(y_i)$ for chosen item in its set $\{y_1, y_2, \ldots, y_i\}$. If sender has set X containing $\{x_1, x_2, \ldots, x_i\}$ then sender sends $F(x_1), F(x_2), \ldots F(x_i)$ to the receiver. Receiver finds the intersection between the sets. They replaced PRG with a PRF. Results show that the protocol is highly efficient in terms of communication. Sender sends its $F_s(x)$ values by interpolating it in a polynomial. After receiving the polynomial, receiver evaluates the polynomial with its $F_s(y)$ values and finds an intersection if and only if $F_s(x) = F_s(y)$.

In 2020, Chase et al [37] proposed an improved version of Kolesnikov's protocol [35] but it is not as efficient as [35].

3.3. Private Set Inclusion

Private Set Inclusion is a special case of 2 party Private Set Intersection protocol in which one party hold n elements and other party holds only a single element. Pinkas et al. [32] explains Private Set Inclusion using random OTs. Both the parties XOR their random OT outputs and then compare the values.

3.4. Cuckoo Hashing

Cuckoo hashing was introduced by Rasmus pagh and Rodler in 2001 [38]. Cuckoo hashing is more efficient than bloom filters and hence used as an alternative to bloom filters and hash tables. Lookup complexity of cuckoo hashing is $O(1)$. Similar to bloom filters, cuckoo hashing also uses k hash functions to map the elements into bins. If

there is a collision, the existing element is replaced by new element and the existing element is relocated to other bin (decided based on the value of other hash functions). The process repeats until the element finds a bin or reach maximum number of re-locations. If the element fail to get inserted in any of the bin, it is inserted into an array called stash. It was shown that for a stash of size $d \leq ln(n)$, insertion of n elements fails with the probability n^{-d} [39]. Cuckoo hashing supports deletion operation with worst case complexity $O(1)$.

3.5. Permutation Based Hashing (PBH)

In [40] Arbitman et al introduced the technique called Permutation Based Hashing (PBH) for reducing the memory usage in cuckoo hashing. PBH is similar to feistel network. In PBH the element is divided into 2 parts i.e, Left part and Right part. Number of bins are represented in the power of 2. Length of left part of the hashed password is equal to the exponent of number of bins when expressed in the form of base 2. Length of right part will be remaining number of bits of hashed password. If an element X is to mapped using PBH technique, we divide X into two parts x_L and x_R. The element X is mapped to bin $x_L \oplus f(x_R)$ where f is a random function. The element x_R is stored in the bin. The paper [34] uses Permutation Based Hashing (PBH) technique to reduce the length of items x bits to $x - \beta$, where β is the size of table

In this paper, we use the concept of PBH to reduce the size of the password which prevents from intersection attack and dictionary attack. Cuckoo hashing is used for mapping the elements to a particular bin as in [34]. In practice, we store the hash of the passwords (honeywords and user chosen) in the bins. In case number of honeywords are at most 5000, there is a possibility that few of the passwords are not being inserted in the bins.

Hence, these passwords need to be stored in a separate array called stash. Inserting more elements in the stash increases the overhead to linear. There are 2 ways to reduce the insertions in the stash as mentioned below:

1) Create large number of bins ($2n(1+\in)$ bins) [41].
2) Increase the number of hash functions to 3 instead of 2 hash functions [42].

4. Proposed Protocols

In this section we describe our proposed protocols, namely Password Reuse Detection (PRD) protocol and Breach Detection protocol in detail.

4.1. Password Reuse Detection Protocol

This protocol helps to detect any attempt by a user to reuse a password (during registration) at the website (called Target) which was previously reused at other website (called Monitor). The proposed protocol excludes any need of trusted third party service from its execution comparing with the work by Wang et al [8]. Following are our assumptions:

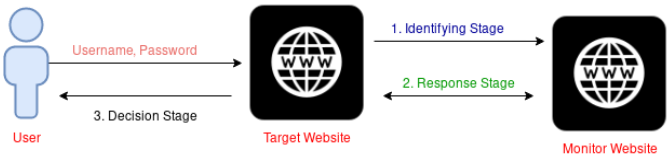

Figure 1: Overview of PRD protocol

1) Both the Target and Monitor websites mutually agree to participate in the protocol.
2) We consider an attacker as semi-honest and the PSI protocols which we are using are also executing in semi-honest setting.
3) The number of honeywords used at Target website and Monitor website must be same.
4) The hash functions required to map the passwords into the bins of the cuckoo hashing are identical at both websites. We also assume that function used to hash passwords at both the websites are identical.

For every new registration of username (U_T) and Password (p_t) at Target website, Target website generates honeywords corresponding to password. We assume honeywords generated using the available techniques [7], [20], [43], [44] are different even if the password is same. We also assume that password hash is computed using a salt $Salt_u$ for user U_T. Therefore, password reuse among websites can be identified if and only if salt at both the websites are identical corresponding a user U_T.

The goal of our protocol is to identify whether the password of a user U_T registered at Target website is same as at Monitor. If true, Target website suggests the user to choose another password. Figure 1 shows an overview of our PRD protocol including different stages of the protocol.

We divide the PRD protocol into 3 stages which are described below:

1) **Identifying Stage:** In this stage, the Target website tries to confirm if username U_T exists in Monitor database (DB_M). To identify, the Target takes the hash output of U_T as $H(U_T)$ where H is a cryptographic hash function and sends N-bit prefix of $H(U_T)$ as $H(U_T)_{[0:N]}$ represented as H_u to the Monitor. The value of N depends on the number of users at both the websites so as to avoid multiple matching of prefixes. Monitor compares (H_u) with the N-bit prefix of all its usernames $H(U_{M_1})_{[0:N]}, \ldots, H(U_{M_n})_{[0:N]}$ stored in DB_M. If there is a match, Monitor sends the corresponding salt, $Salt_u$ and Target uses $Salt_u$ for U_T. Else, the Target generates a random salt. Target adds the new credentials to its database (DB_T) to generate honeywords corresponding to p_t, Target calls Honeygen() function which outputs $p_{t_1}, p_{t_2} \ldots, p_{t_n}$. The hashed vales are represented as $p'_t, p'_{t_1}, \ldots, p'_{t_n}$ at Target and $p'_m, p'_{m_1}, \ldots, p'_{m_n}$ at Monitor. Detailed protocol is given in algorithm 1.
2) **Response stage:** Since username is identical, it is plausible that password might also be identical at both the websites. We mainly use PSI protocol (explained in

Algorithm 1 Identifying Stage

User	Target	Monitor
User $\xrightarrow{U_T, p_t}$		
	$H(U_T) \leftarrow U_T$ $H(U_T)_{[0:N]} \leftarrow H(U_T)$ $H_u := H(U_T)_{[0:N]}$	
	$\xrightarrow{H_u}$	
		True/False $\leftarrow (H_u \in DB_M)$ If (True) $Salt_u \leftarrow DB_M$ else $\perp$
	$\xleftarrow{Salt_u / \perp}$	
	If ($Salt_u$) $\{p_{t_1}, \ldots, p_{t_n}\} \leftarrow$ HoneyGen(p_t) Add $\{H(p_t, Salt_u), \ldots, H(p_{t_n}, Salt_u)\}$ to DB_T Move to **Response stage** else $Salt_u \xleftarrow{\$} \{0,1\}^*$ $\{p_t, p_{t_1}, \ldots, p_{t_n}\} \leftarrow$ HoneyGen(p_t) and Add $\{H(p_t, Salt_u), \ldots, H(p_{t_n}, Salt_u)\}$ to DB_T Notify U_T as Registered	

Notation: $:=$ is for assignment, $x \leftarrow y$ shows x is derived from y, $x \xleftarrow{\$} y$ shows x is randomly sampled from y, HoneyGen is honeyword generation algorithm.

Algorithm 2 Response Stage: (PRD.V1)

Target	Monitor
$p'_T = \{p'_t, p'_{t_1}, p'_{t_2}, p'_{t_3}, \ldots, p'_{t_n}\}$ $SK_T \leftarrow KeyGen(.)$ $H_{P_T} = \{H(p'_t)^{SK_T}, \ldots, H(p'_{t_n})^{SK_T}\}$	$p'_M = \{p'_{m_1}, p'_{m_2}, p'_{m_3}, \ldots, p'_{m_n}\}$ $SK_M \leftarrow KeyGen(.)$ $H_{P_M} = \{H(p'_m)^{SK_M}, \ldots, H(p'_{m_n})^{SK_M}\}$
$\xrightarrow{H_{P_T}}$	
$\xleftarrow{H_{P_M}, H_{P_T}^{SK_M}}$	
$True/Flase \leftarrow (H_{P_T}^{SK_M} \stackrel{?}{=} H_{P_M}^{SK_T})$ If (True) Password Reuse Detected Go to Decision stage Else: Successful registration	

Section 3) to check for password reuse. We propose two versions of Password Reuse Detection (PRD) protocol, PRD.V1 and PRD.V2.

a) **PRD.V1 (for number of honeywords < 500):** In this version, we use Diffie Hellman based PSI as discussed in section 3.2.1 where the values are the hashed password and honeywords received from Identifying stage. To run the protocol, both the Target and Monitor generates their random secrets as SK_T and SK_M respectively by running a $keyGen$ algorithm as shown in algorithm 2.
Hashed Passwords (including honeywords) at both websites are raised to the power of their respective secret keys. We denote Target's set of passwords as H_{P_T} and Monitor's set of passwords with H_{P_M} respectively. Target website sends the set H_{P_T} to Monitor website. Monitor website sends the set H_{P_M} to Target website and $H_{P_T}^{SK_M}$. Target website then compares the sets $H_{P_M}^{SK_T}$ and $H_{P_T}^{SK_M}$ and checks for common entries. As honeywords are distinct and different even if the password is same, intersection of at least a single element implies reuse of password.

b) **PRD.V2 (for number of honeywords > 500 and < 5000):** In this version, both Target and Monitor construct a cuckoo hash table with hashed passwords using Permutation Based Hashing (PBH). Note that the word passwords include both user chosen password and corresponding honeywords. Both the parties uses PBH technique as discussed in section 3.5 to reduce the size of passwords. Reduced size of passwords are inserted into bins using cuckoo hashing. We assume that hash functions required to map

Algorithm 3 Response stage (PRD.V2)

Target	PSI	Monitor
For $i = 0$ to n		For $i = 0$ to n
$[p'_{t_i}]_R \xleftarrow{\text{PBH}} p'_{t_i}$		$[p'_{m_i}]_R \xleftarrow{\text{PBH}} p'_{m_i}$
$Cuckoo_{HT_R} \xleftarrow{\text{h1,h2,h3}} [p'_{t_i}]_R$		$Cuckoo_{HM_R} \xleftarrow{\text{h1,h2,h3}} [p'_{m_i}]_R$
	$\xleftrightarrow{OT_m^k}$	
$r_i \xleftarrow{\$} \{0,1\}^*, 0 \leq i \leq n$		Select $c \xleftarrow{\$} \{PRC\}$
	$\xleftarrow{\text{c}}$	
		$s \xleftarrow{\$} \{0,1\}^k$
Constructs matrices $[T_0]_{n\times k}$, $[T_1]_{n\times k}$		
$\forall\ jth$ row $\in T_0$, $t_{0,j} \leftarrow \{0,1\}^k$		
$\forall\ jth$ row $\in T_1$ set $t_{1,j} = c(r_i) \oplus t_{0,j}$		
$t_{0,j}$ and $t_{1,j}$ are input to OT		
		Select bit $s_i \in s$
$H_{corr}(t_{0,j} \oplus c(r_i))$ is the OT output		$H_{corr}(q_j \oplus c(r_i).s)$ is the OT output
	$\xleftrightarrow{\text{PRTY}}$	
$P(x) := H_{corr}(t_{0,j} \oplus c(r_i))$		$Q(x) := H_{corr}(t_{0,j} \oplus c(r_i).s)$
Coefficients of $P(x)$ send to Monitor		
		$O =$
		$\{H_{corr}(Q(t_{0,j} \oplus c(r_i).s)) \oplus s.P(t_{0,j} \oplus c(r_i))$
		O is send to Target.
Based on the value O password reuse is detected.		

Notation: PRC- Pseudo Random Code ; $[p'_{t_i}]_R$ represents right part of password

the passwords into the bins of the cuckoo hashing are identical at both the websites. We use 3 hash functions for cuckoo hashing as suggested in [34], [42]. We compute PSI between two cuckoo hash tables of Target and Monitor website as proposed by Kolisnikov et al [35] for computation and Pinkas et al for communication [36]. The steps involving in PSI are described in detail.

Initially, Target chooses a random string r_i. Monitor chooses a Pseudo Random Code (c) from family of Pseudo Random Code and sends c to Target website. Target website constructs two $n \times k$ matrices T_0, T_1 respectively while Monitor website selects a random k-bit string s_i. Target inputs $t_{0,j}$ and $t_{1,j}$ to the OT while Monitor uses its s bit string as choice vector. For every OT instance, Monitor outputs q^i. A matrix Q is constructed with all q^is where, q^i represents column of matrix Q. At the end of OT, Target receives $H_{corr}(t_{0,j} \oplus c(r_i))$ and Monitor receives $H_{corr}(q_j \oplus [c(r_i).s])$ where H_{corr} is the correlation robust hash function. Target website creates a polynomial P with its outputs and sends the coefficients to Monitor website. Monitor website creates a polynomial Q and computes a XOR operation as shown in the algorithm. Monitor website sends this result $(O = \{H_{corr}(Q(t_{0,j} \oplus c(r_i).s)) \oplus s.P(t_{0,j} \oplus c(r_i)))$ to the Target website. Target website detects a password reuse based on the received O. Detailed protocol is given in algorithm 3.

3) **Decision Stage:** After finding an intersection, Target sends an email notification to the user suggesting for a password change and warns the user about consequences of password reuse. If a password reuse has been identified, our protocol does not reject any user's passwords because we cannot deny any user to create an account at Target websites. Doing so will restrict the democratized access to the website.

4.2. Breach Detection Protocol

In this section we give a detailed view of our breach detection technique. Let us assume that Target website was breached by an adversary who has access to all users credentials. Now, there is a possibility that adversary gets access to the user accounts of other websites (referred as Monitors) if the user reused the passwords (credential stuffing). Following are our assumptions:

1) Adversary breaches the target passively.
2) Adversary can breach the database any number of times.
3) Adversary cannot breach Monitor website when it has breached Target.
4) Adversary can access Monitor website accounts through hit and trail method.
5) All the OT protocols used are semi-honest.

For every three consecutive unsuccessful login attempts at Monitor, Monitor website sends a request to Target to check if an identical username exists at Target website. If the result is true, Monitor and Target websites compute Private Set Inclusion (PSI) protocol to check whether the unsuccessful

password attempts entered at Monitor website are in the list of Target's honeywords. Note that this breach detection protocol is also a per user approach. If the entered incorrect password is in the list of Target's honeywords, a breach is detected at Target website.

We divide our breach detection technique into 2 stages:

1) **Preliminary Stage:** Just like identifying stage in PRD protocol, N-bit hashed prefix of the Username is sent by Monitor to Target website. If result is true, Monitor receives the salt from Target and the protocol goes to next stage.
2) **Detection Stage:** The goal is to find if the unsuccessful password attempt is in the list of Target's honeywords (detecting credential stuffing). Now, both the parties compute a Private Set Inclusion protocol using random Oblivious Transfer [32]. Target and Monitor run n parallel OTs (assuming Target has $n-1$ honeywords) as shown in algorithm 4. For every three consecutive unsuccessful login attempts, Monitor initiates OT by sending $H(p_m)$ while Monitor inputs all n-values one by one to the OT. Target inputs different values for each instant $(H(p_{t_1}), H(p_{t_2}), \ldots, H(p_{t_n}))$ while Monitor inputs the same value. Monitor receives a random message (M_{m_i}) (where i can be either 0 or 1) for each OT instance while Target gets a pair of random messages $(M_{t_{0,1}})$ for every instance. Monitor XOR's the received random messages and send to Target. Target computes the XOR of its random pair of messages corresponding to its set and compare with received XOR value of Monitor. XOR value will be 0 if the values are identical. So value 0 implies a breach is detected at Target website.

5. Security

In this section we analyse the security of the proposed protocols, namely, Password Reuse Detection (PRD) and Breach Detection sequentially.

5.1. Password Reuse Detection

As discussed in section 4.1, our Password Reuse Detection protocol consists of 3 stages. We discuss the security of each of stages below.

5.1.1. Identifying Stage: . Target website sends an N-bit hashed prefix of username to the Monitor. Monitor tries to identify if there is a match between received prefix of username and prefixes of its usernames.

Assume a case where user does not hold an account at Monitor, but Target sends the N-bit hash prefix. In this case, Monitor should not be able to deduce that a specific user is about to register with a username u_t at Target. Since cryptographic hash is a one way function, Monitor cannot gain any extra information from the received hashed prefix of the username when the entry does not match. Hence, there is no privacy leakage from N-bit hashed prefix of username.

5.1.2. Response Stage. After identification of a username, both the Target and Monitor website runs the response stage to identify if there is a password reuse among the websites. We propose 2 variants depending on the usage of number of honeywords. We propose PRD.V1 for websites with less than 500 honeywords and PRD.V2 for greater than 500 honeywords but less than 5000 honeywords.

1. PRD.V1: The following theorem proves the security of the protocol.

Theorem 1. ***Solving pre-image of hashed passwords is equivalent to solving DDH problem***

Proof. In our protocol, Target and Monitor individually generates secret keys and hashed passwords of both the parties are raised to their secret keys respectively. Secret keys are generated by a keyGen algorithm with security parameter λ. Value of λ can be set as suggested in [45]. Finding underlying hashed password is considered as a hard problem and is equivalent to DDH problem. □

2. PRD.V2: In this version, we hash all the passwords (including honeywords) at both Target and Monitor and store them in a cuckoo hash table using PBH respectively. PSI is performed between the cuckoo hash tables of Target and Monitor. We are using combination of Kolesnikov et al's [35] and Pinkas et al's [36] PSI protocols as discussed in section 3.2.2. Both Target and Monitor perform 1-out-of-k (where $k << n$) Oblivious Transfers to find the intersection. Here we briefly describe the security proofs.

Theorem 2. *Neither the original password nor the hashed password can be identified from the cuckoo hash tables.*

Proof. Usage of permutation based hashing minimizes the length of hashed passwords to be stored in a cuckoo hash table. Adversary cannot regenerate the actual value from the stored reduced length of the hashed passwords. Only possibility for the adversary is to guess the value. The probability of guessing the correct value is negligible. If the lengths of left part and right part of the password are 240 and 16 bits respectively, then adversary needs to guess remaining 240 bits because permutation based hashing technique only stores 16 bits (right part). Guessing the remaining 240 bits is negligible. □

As mentioned in PRD.V2 protocol section 4.1, we can consider OT extension as OPRF with s, c as seeds. We can observe that many OT instances have same s and c. Kolesnikov et al [35] termed it as OPRF instance with related keys. Following the proof given by Kolesnikov et al [35] which claims that security of OPRF in terms of m-related key where m is the maximum elements, we claim that our case is also m-RK-PRF secure. In our case, m is the number of passwords (n) (including honeywords which are greater than 500).

Now we discuss the security of Pinkas et al protocol [36] which we use for communication between the two parties (Target, Monitor) for exchanging their outputs after performing OTs to find the intersection of elements.

Algorithm 4 Breach Detection Protocol

Monitor	Target
Monitor inputs $H(p_m)$ to OT	$H(p_{t_i})$ Target input to OT
Receives M_{m_i} from OT	Receives $M_{t_{0,1}}$ from OT.
Monitor sends $\oplus_{i=1}^{n} M_{m_i}$ to Target.	
	Target checks if $((\oplus_{i=1}^{n} M_{m_i}) == (\oplus_{i=1}^{n} M_{t_i}))$

Security for Target's Inputs: For security against a malicious Monitor, Target's input (output of the PRF) must be hidden. This can be achieved if all the inputs (outputs of PRF) are uniformly distributed over the polynomial.

Security for Monitor: The security of the protocol relies on the property that hamming distance (κ of size 128 bits) between $P(x)$ from $R(x)$ (x is Monitor's PRF outputs) is large if Monitor's values is not in the set of Target's values. For more details we would encourage the reader to go through section 3.3 in Pinkas et al [36].

We conclude that usage of PBH along with both the PSI protocols [35], [36] does not leak any information about hashed passwords.

5.2. Breach Detection Protocol

In this section, we discuss the security of our breach detection protocol. Our breach detection protocol consists of 2 stages: i) Preliminary Stage and ii) Detection Stage. We discuss the security of both the stages below.

Preliminary Stage: Security in preliminary stage is to preserve the anonymity of username which is established as explained in the section 5.1.1

Detection Stage: In Detection Stage, the goal is to detect if any of the honeywords of breached Target website is entered as part of unsuccessful login attempt at Monitor website. We use random OT and compute Private Set Inclusion [32] to check if Target website had been breached. We discuss the security of random OT and Private Set Inclusion as presented by Asahrov et al [33] and Pinkas et al [32]. Since we are using random OT, the outputs are independent of inputs (Target's input: Passwords; Monitor's input: passwords of unsuccessful login attempt). Parties cannot learn anything from each other's inputs. The security of OT depends on the security parameter denoted by κ. Please refer to [45] for NIST parameters. Target cannot learn anything from received Monitor's value because the sent value is XOR of its OT outputs (random vales).

We conclude that other party (Target) cannot deduce any information from the received values and hence our breach detection model does not leak any information about the passwords.

6. Results

In this section we present the feasibility results of Password Reuse Detection Protocol (PRD) and Breach Detection Protocol (BDP) by comparing with existing best results.

6.1. Password Reuse Detection

We present our results in this section by comparing with wang et al's "How to end Password Reuse on the Web" [8]. Their protocol involves 4 parties: a user who is willing to register a new account at Requester website, a Responder website where the user already holds an account, a Directory which is a trusted third party service and acts as bridge between Responder and Requester website. They use partial homomorphic encryption scheme to identify the use of similar passwords.

Our protocol involves 3 parties: a user who is willing to register at a Target website, Monitor website where the user is already holding an account. We propose two versions of our Password Reuse Detection (PRD) protocol based on the number of honeywords used at each website. If the number of honeywords are less than 500 we use Diffie-Hellman PSI version to identify the password reuse. If the number of honeywords are greater than 500 we use PSI protcol as presented in Kolesnikov et al [35] and Pinkas et al [36]. PRD.V1 takes less computation, storage and communication because of its smaller set size. Hence, we stress more about the results of PRD.V2 i.e, case with more than 500 honeywords.

In the upcoming part of the section, we compare the storage, computation and communication results with Wang et al's work [8].

- **Storage:** Wang et al [8] uses bloom filter for storing the passwords (including honeywords). The use of bloom filters might not be a good approach because of following two reasons.
  1) **False Positives:** One of the main downside of using bloom filter is that elements can keep on getting inserted in the bins and can result in more false positives.
  2) **Space complexity:** Bloom filters occupy more space than compared to cuckoo hashing [46]. Since we are using cuckoo hashing and permutation based hashing, storage complexity would be less when compared to [8].

  Responders in Wang et al's protocol [8] contains an additional database to maintain hash of similar passwords and their honeywords corresponding to each user. Our protocol requires only one additional database at both Target and Monitor websites to store N-bit hash prefix of username. The cuckoo hash tables at both the websites are created on the fly and are not stored.

Table 1: Comparison of different protocol run times

Protocol Run time				
Protocol	Technique	Set Size: 2^8	Set Size: 2^{10}	Set Size: $\mathbf{2}^{12}$
Wang et al [8]	PMT	$\simeq$0.8 sec	2 sec	**10 sec**
PRD.V1	DH based ECC [32]	$\simeq$**0.2 sec**	**0.7 sec**	**2.8 sec**
PRD.V2	KKRT [35]	**0.192 sec**	$\simeq$**0.2 sec**	**0.211 sec**

Table 2: Comparison of Communication protocol run times

Run time in seconds			
Set Size	100 Mbps	10 Mbps	1 Mbps
2^{12}	0.99 sec	1.01 sec	3.51 sec
2^{16}	2.02 sec	5.36 sec	40.08 sec
2^{20}	10.53 sec	66.0 sec	645.3 sec

- **Computation:** We use Kolesnikov et al's [35] PSI protocol for performing PSI computation. Results show that the computation part as presented in [35] is the most efficient one. We assume that popular websites use utmost 5000 honeywords which is approximately 2^{12}. From the results of [35], it can be seen that the run time of PSI protocol takes around 211 ms. This 211 ms includes communication latency. Since we use [35] for computation, we observe that the computation time required for computing the PSI would be even lesser. Table 1 shows comparison of run times between our protocol (both the versions) and Wang et al [8].

- **Communication:** For communication we use pinkas et al's protocol [36]. It creates an n-degree polynomial by interpolating all the passwords of Monitor website and sends that polynomial to the Target website. Target evaluates the polynomial with its passwords and finds an intersection if and only if both the values are same.
  In [36], given data shows that the protocol is highly efficient at low bandwidth. From table 2, it can be observed that communication latency takes around 1 second for a set of size of 2^{12} elements at a low bandwidth of 10 Mbps. Adopting lower bandwidth protocol does not burden the servers of Target and Monitor websites without any compromise in efficiency of the protocol. It is to be noted that, servers operate in the order of Gbps, dedicating some tens of Mbps to participate in this protocol does not burden the server. The communication complexity of the protocol is $O(n)$ where n is the number of passwords which are in Target and Monitor's set.

From the above results, sum of communication and computation latency would be around 1.16 second for a set size of 2^{12} at 10Mbps. Apart from computation and communication we have identifying stage to check for username match. Keeping in mind of space complexity and search complexity we suggest websites to use cuckoo hashing.

Checking identical username at Monitor can be compared to login time. Generally, login time for a user is around 2 seconds. This time is between user and server but in our case it is between Target server and a Monitor server. It is obvious that the time will be less than 2 seconds. Considering the worst case scenario which is around 2 seconds, we claim that our protocol run time is at most around 3.5 seconds (at 1Mbps) for 5000 honeywords. This estimate is much less when compared to wang et al's work [8] which takes around 10 seconds for 2^{12} when one monitor is employed. Hence, our protocol is approximately 2.8 times faster than Wang et al's work [8] for set size of 2^{12}.

6.2. Breach Detection Protocol

We present the results of our Breach Detection protocol by comparing with work by Wang et al Amnesia [9]. As per their protocol, Target website sends a PCR request consisting of salt, username, list of sweetwords, public key in an encrypted form to Monitor website at some arbitrary time. Monitor generates a response by computing a partial homomorphic encryption with passwords of failed login attempt. Monitor sends the response back to the Target. Target decrypts the result and perform required action based on the decrypted result.

Our work is an improvised version of wang et al [9] (Detecting credential stuffing) and is near real time. We compare our results with the credential stuffing model. Following similar approach, our main goal is to identify whether a honeyword of Target website is given as part of failed login attempt at Monitor website. We present a new model for credential stuffing at Monitor website. Our model consists of 2 entities namely, Target website and Monitor website. We assume that user holds an account at both Target and Monitor website and user's account is breached at Target website. For three consecutive unsuccessful login attempts at Monitor website, it sends N-bit hashed prefix of username to the Target to check whether the user is also registered at Target. If yes, then both run a random OT based Private Set Inclusion [32], [33] to check if the submitted password at Monitor as part of login attempt is in the list of Target's honeywords. We directly use random OT as presented by Ashrov et al [33] in which the length of strings is 80 bits. Although the number of communication rounds is more than Wang et al work [9], we claim the run time of our protocol is much less than compared to their work.

- **Storage:** In [9] Wang et al does not hold any extra database apart from login database for their protocol. Also, they use cuckoo filter to optimize space complexity. In our protocol we hold an extra database which contains N-bit hash prefix of usernames at both the websites.
- **Protocol Run time:** Table 3 represents the run time of Wang et al's [9] and Pinkas et al's Private Set

Table 3: Wang et al protocol run time

Protocol Run time			
Protocol	set size: 2^8	set size: 2^{10}	Set Size: $\mathbf{2}^{12}$
Wang et al [9] (4cores)	$\simeq$ 45 msec	$\simeq$ 150 msec	$\simeq$**400 msec**
Pinkas et al [32]	$\simeq$ 0.08 sec	0.13 sec	**0.2 sec**

Inclusion [32]. We include Wang et al's [9] results considering the best case that is with 4 cores. The run time for maximum number of honeywords i.e, 5000 which is $\simeq 2^{12}$ is 400 msec whereas our protocol run time is around 200 msec.

7. Discussions

7.1. Password Reuse Detection (PRD) protocol

In this section we discuss about few general attacks and ways to mitigate them. We have two cases:

i) Malicious Target: There are two possible attacks when a Target website is malicious. They are:

a) DoS attack on Monitor: A malicious Target website can send multiple requests with random bits as N-bit hashed prefix of username to Monitor. Monitor website can simply reject the requests from Target website if it finds any abnormal network traffic from Target website.

b) DoS attack on User: A malicious target website can send multiple notifications to user requesting for a password change even though password is not a reused one. These multiple requests from Target website may annoy the user or put them in a panic state. Proper awareness can be given to the user after logging into the website through pop up messages which explains the above scenario. User can report to the Target website about its abnormal behaviour.

ii) Malicious Monitor: There are three possible attacks when Monitor is Malicious as explained below.

a) DoS attack on Target: A malicious Monitor website can send multiple responses to the Target. If an identical username has been identified at Monitor website, Monitor can send multiple responses like sending multiple salt strings to the Target. These can be mitigated in the same way as suggested for the case of DoS attack on Monitor.

b) Sending incorrect responses: A malicious monitor can send incorrect responses to Target website. Target can mitigate this by sending few ineffective requests if it identifies any abnormal behaviour. Abnormal behaviour can be identified by sending one of the user's request which has been previously identified and resolved as a password reuse.

c) No response from Monitor: A malicious Monitor might not respond to the Target website. This can be mitigated by replicating the Monitor website so that if one of the server does not participate in the protocol, the requests can be directed to one of the remaining servers.

7.2. Breach Detection protocol

Most of the attacks for breach detection are similar to the Password Reuse Detection protocol as discussed in section 7.1. Again we consider two cases:

i) Malicious Target: Malicious Target website might not respond for the Monitor's request to check and confirm if first N-bit hashed prefix of username exists with Target website. This can be mitigated by maintaining replicated Target servers so that if one server does not respond others can still participate in the protocol.

ii) Malicious Monitor: A malicious Monitor website can send multiple requests to Target website. This request can be the first step of our protocol i.e., checking whether the username which was given as part of failed login attempt is identical to any of the username that Target website already holds. This can lead to a Denial of Service (DoS) attack on Target website. If the network traffic is found to be abnormal, Target website can discard the requests sent by Monitor website.

8. Conclusion and Future Work

In this paper, we introduce two protocols Password Reuse Detection protocol and Breach Detection protocol. The former is used to detect password reuse among websites and the latter is used to detect credential stuffing attacks. We use state of the art Private Set Intersection protocols to compute the password intersection between Target and Monitor websites. We claim that our protocol is $\simeq$2.8 times faster than Wang et al [8]. We also believe that deploying this framework might reduce the problem of password reuse consequently breaches. Our Breach Detection protocol is used to detect breach of Target website. We use Private Set Inclusion protocol with random OT to detect the breach of Target website. We claim that our breach detection protocol is 2 times faster than existing scheme of Wang et al [9].

Our PRD protocol runs only for 2 parties which mutually agree to Monitor each other. A better approach is to implement multiparty PSI protocol for detection of password reuse by sending requests to multiple Monitor websites simultaneously. This can be an extension of the current work.

References

[1] IBMsecurity. Cost of a data breach report 2020. https://www.ibm.com/security/digital-assets/ cost-data-breach-report/.

[2] Google. Google online security survey. https://services.google.com/fh/files/blogs/google_security_infographic.pdf.

[3] Ponemon institutes the 2020 state of password and authentication security behaviour report. https://mms.businesswire.com/media/20200219005336/en/773763/5/191522-Ponemon-Infographic-2020-final-1.jpg?download=1, 2022.

[4] catalin cimpanu. 44 million microsoft users reused passwords in the first months of 2019. https://www.zdnet.com/article/44-million-microsoft-users- reused-passwords-in-the-first-three-months-of-2019/.

[5] Shape Security. 2018 credential spill report. https://info.shapesecurity.c/rs/935-ZAM-778/images/Shape_Credential_Spill_Report$_2$018.*pdf*.

[6] Katie Collins. Facebook buys black market passwords to keep your account safe. https://www.cnet.com/news/privacy/facebook-chief-security-officer-alex-stamos-web-summit-lisbon-hackers/.

[7] Ari Juels and Ronald L Rivest. Honeywords: Making password-cracking detectable. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 145–160, 2013.

[8] Ke Coby Wang and Michael K Reiter. How to end password reuse on the web. *arXiv preprint arXiv:1805.00566*, 2018.

[9] Ke Coby Wang and Michael K Reiter. Using amnesia to detect credential database breaches. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 839–855, 2021.

[10] Ke Coby Wang and Michael K Reiter. Detecting stuffing of a {User's} credentials at her own accounts. In *29th USENIX Security Symposium (USENIX Security 20)*, pages 2201–2218, 2020.

[11] Antreas Dionysiou and Elias Athanasopoulos. Lethe: Practical data breach detection with zero persistent secret state. In *7th IEEE European Symposium on Security and Privacy, EuroS&P 2022, Genoa, Italy, June 6-10, 2022*, pages 223–235. IEEE, 2022.

[12] 1 password. https://support.1password.com/watchtower-privacy/.

[13] Google password manager. https://passwords.google.com.

[14] Alex McOmie. 65% of people don't trust password managers despite 60% experiencing a data breach. https://www.passwordmanager.com/password-manager-trust-survey/.

[15] Troy Hunt. Have i been pwned. https://haveibeenpwned.com/.

[16] Kurt Thomas, Jennifer Pullman, Kevin Yeo, Ananth Raghunathan, Patrick Gage Kelley, Luca Invernizzi, Borbala Benko, Tadek Pietraszek, Sarvar Patel, Dan Boneh, et al. Protecting accounts from credential stuffing with password breach alerting. In *28th USENIX Security Symposium (USENIX Security 19)*, pages 1556–1571, 2019.

[17] Kim Laine Radames Cruz Moreno Kristin Lauter, Sreekanth Kannepalli. Password monitor: Safeguarding passwords in microsoft edge. https://www.microsoft.com/en-us/research/blog/password-monitor-safeguarding-passwords-in-microsoft-edge/, 2021.

[18] Mir Masood Ali. Combating credential stuffing: Protocols to check for compromised credentials and password reuse. 2021.

[19] Burton H Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426, 1970.

[20] Imran Erguler. Achieving flatness: Selecting the honeywords from existing user passwords. *IEEE Transactions on Dependable and Secure Computing*, 13(2):284–295, 2015.

[21] Hristo Bojinov, Elie Bursztein, Xavier Boyen, and Dan Boneh. Kamouflage: Loss-resistant password management. In *European symposium on research in computer security*, pages 286–302. Springer, 2010.

[22] Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: The second-generation onion router. Technical report, Naval Research Lab Washington DC, 2004.

[23] Bijeeta Pal, Mazharul Islam, Thomas Ristenpart, and Rahul Chatterjee. Might i get pwned: A second generation password breach alerting service. *arXiv preprint arXiv:2109.14490*, 2021.

[24] Michael O Rabin. How to exchange secrets with oblivious transfer. *Cryptology ePrint Archive*, 2005.

[25] Yuval Ishai, Joe Kilian, Kobbi Nissim, and Erez Petrank. Extending oblivious transfers efficiently. In *Annual International Cryptology Conference*, pages 145–161. Springer, 2003.

[26] Vladimir Kolesnikov and Ranjit Kumaresan. Improved ot extension for transferring short secrets. In *Annual Cryptology Conference*, pages 54–70. Springer, 2013.

[27] Arvind Narayanan, Narendran Thiagarajan, Mugdha Lakhani, Michael Hamburg, Dan Boneh, et al. Location privacy via private proximity testing. In *NDSS*, volume 11, 2011.

[28] Pierre Baldi, Roberta Baronio, Emiliano De Cristofaro, Paolo Gasti, and Gene Tsudik. Countering gattaca: Efficient and secure testing of fully-sequenced human genomes. In *Proceedings of the 18th ACM Conference on Computer and Communications Security*, page 691–702. Association for Computing Machinery, 2011.

[29] Ghita Mezzour, Adrian Perrig, Virgil Gligor, and Panos Papadimitratos. Privacy-preserving relationship path discovery in social networks. In *International Conference on Cryptology and Network Security*, pages 189–208. Springer, 2009.

[30] Catherine Meadows. A more efficient cryptographic matchmaking protocol for use in the absence of a continuously available third party. In *1986 IEEE Symposium on Security and Privacy*, pages 134–134. IEEE, 1986.

[31] Bernardo A Huberman, Matt Franklin, and Tad Hogg. Enhancing privacy and trust in electronic communities. In *Proceedings of the 1st ACM conference on Electronic commerce*, pages 78–86, 1999.

[32] Benny Pinkas, Thomas Schneider, and Michael Zohner. Faster private set intersection based on {OT} extension. In *23rd USENIX Security Symposium (USENIX Security 14)*, pages 797–812, 2014.

[33] Gilad Asharov, Yehuda Lindell, Thomas Schneider, and Michael Zohner. More efficient oblivious transfer and extensions for faster secure computation. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 535–548, 2013.

[34] Benny Pinkas, Thomas Schneider, Gil Segev, and Michael Zohner. Phasing: Private set intersection using permutation-based hashing. In *24th USENIX Security Symposium (USENIX Security 15)*, pages 515–530, 2015.

[35] Vladimir Kolesnikov, Ranjit Kumaresan, Mike Rosulek, and Ni Trieu. Efficient batched oblivious prf with applications to private set intersection. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 818–829, 2016.

[36] Benny Pinkas, Mike Rosulek, Ni Trieu, and Avishay Yanai. Spotlight: Lightweight private set intersection from sparse ot extension. In *Annual International Cryptology Conference*, pages 401–431. Springer, 2019.

[37] Melissa Chase and Peihan Miao. Private set intersection in the internet setting from lightweight oblivious prf. In *Annual International Cryptology Conference*, pages 34–63. Springer, 2020.

[38] Rasmus Pagh and Flemming Friche Rodler. Cuckoo hashing. In Friedhelm Meyer auf der Heide, editor, *Algorithms - ESA 2001, 9th Annual European Symposium, Aarhus, Denmark, August 28-31, 2001, Proceedings*, volume 2161 of *Lecture Notes in Computer Science*, pages 121–133. Springer, 2001.

[39] Adam Kirsch, Michael Mitzenmacher, and Udi Wieder. More robust hashing: Cuckoo hashing with a stash. *SIAM Journal on Computing*, 39(4):1543–1561, 2010.

[40] Yuriy Arbitman, Moni Naor, and Gil Segev. Backyard cuckoo hashing: Constant worst-case operations with a succinct representation. In *2010 IEEE 51st Annual Symposium on Foundations of Computer Science*, pages 787–796. IEEE, 2010.

[41] Rasmus Pagh and Flemming Friche Rodler. Cuckoo hashing. *Journal of Algorithms*, 51(2):122–144, 2004.

[42] Martin Dietzfelbinger, Andreas Goerdt, Michael Mitzenmacher, Andrea Montanari, Rasmus Pagh, and Michael Rink. Tight thresholds for cuckoo hashing via xorsat. In *International Colloquium on Automata, Languages, and Programming*, pages 213–225. Springer, 2010.

[43] Donghoon Chang, Aarushi Goel, Sweta Mishra, Somitra Kumar Sanadhya, et al. Generation of secure and reliable honeywords, preventing false detection. *IEEE Transactions on Dependable and Secure Computing*, 16(5):757–769, 2018.

[44] Yimin Guo, Zhenfeng Zhang, and Yajun Guo. Superword: A honeyword system for achieving higher security goals. *Computers & Security*, 103:101689, 2021.

[45] Elaine Barker. Recommendation for key management, part 1: General, 2016-01-28 2016.

[46] Bin Fan, Dave G Andersen, Michael Kaminsky, and Michael D Mitzenmacher. Cuckoo filter: Practically better than bloom. In *Proceedings of the 10th ACM International on Conference on emerging Networking Experiments and Technologies*, pages 75–88, 2014.