Verifiable Compromised Credential Checking

Mi Song, Ding Wang, Guanling Li, and Zhen Li

***Abstract*—Password file breaches expose billions of credentials and enable notorious credential stuffing attacks, where the attacker exploits the leaked password to maliciously log into the victim's applications. To mitigate this threat, *compromised credential checking* (C3) services, like HaveIBeenPwned (the first C3 service) and Google Password Checkup proposed by Thomas et al. (USENIX Security'19) are widely used. A C3 service allows clients to query whether their credentials (username or/and password) are exposed, without revealing the password. However, in existing C3 services, clients may receive *unreliable* query results because they *all* assume that the C3 server is *fully* trusted, and overlook a crucial security property: the *verifiability* of query results. For example, a malicious C3 server may respond that an exposed account has *not* been breached. To fill this gap, we propose the notion of verifiable C3 service (VerC3 for short), which equips current C3 services with query-result verifiability. The key challenge lies in how to determine whether the C3 server has honestly responded to the client's query and faithfully updated the breached password records. We reveal that the verifiability problem *inherently* cannot be solved in the single-server setting. Based on this finding, our VerC3 is built in the two-server setting. Breached password records are signed by the data owner and stored by the online server. We define VerC3 as a suite of protocols that meet 11 desirable properties and build a *simple, secure, and efficient* instance, called Have I Really Been Pwned (HIRBP). We develop a prototype of HIRBP to show its practicality: It takes the client 136.91 ms to finish a query on a common PC, with a total bandwidth of 86 KB. In particular, we provide so far the most comprehensive empirical evaluation of C3 services against both *online guessing attacks* and *breach extraction attacks*, using 1.4 billion real-world passwords. We believe this work takes a substantial step toward reliable C3 services.**

***Index Terms*—Password authentication, compromised credential checking, query verifiability, credential stuffing attacks.**

I. Introduction

PASSWORDS remain the most widely used method for user authentication [1], [2]. In password authentication, the server needs to store a sensitive file, which consists of hashed passwords of all registered users. Once the server is compromised, an overwhelming percentage of passwords will be exposed even if passwords are stored in salted-hash as recommended in the latest security standards [3], due to advances in machine-learning-based cracking algorithms [4], [5], [6] and hardware like FPGAs [7]. These years we seem to get used to million-, even billion-sized catastrophic password data breaches (e.g., the 26 billion MOAB leak [8], the 3.8 billion DarkBeam leak [9], and the 3 billion Yahoo leak [10]). Password file breaches provide ample material for credential stuffing attacks, where the attacker exploits the victim's password leaked from one service, to log into the same account at another service. To our knowledge, password file breaches are one of the most damaging threats to sensitive data [11]. The DBIR 2026 [12] shows that 56% of the 2,281 web-based data breach incidents were due to stolen credentials.

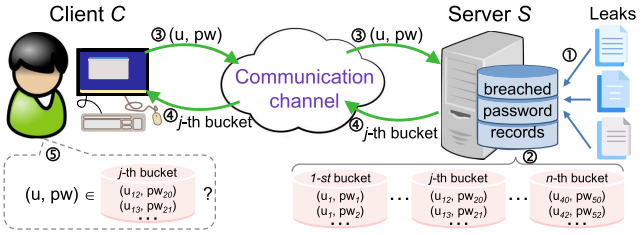


Fig. 1. An overview of compromised credential checking (C3) services. Here, username (U) and password (PW) are shown in plaintext for better illustration. While in reality, passwords are securely stored in salted-hash. Generally, the C3 service can be categorized into five distinct steps: ① Gather leaked credentials; ② Bucketize; ③ Query; ④ Respond; and ⑤ Check if the credential is leaked.

A promising solution to counter credential stuffing attacks is breach alerting, also called compromised credential checking (C3) [13]. Users can employ C3 services to check if their credentials (username or/and password) have been exposed [14]. Fig. 1 shows an overview of C3 services. A C3 server gathers leaked (username, password) pairs from public sources or third parties [15]. Frequent breaches facilitate the accumulation of large-scale records. For example, the HaveIBeenPwned (HIBP) database contains 12 billion breached accounts [15]. Google's C3 service reveals that there are 4 billion breached (username, password) pairs in their database [13]. To reduce bandwidth usage, C3 services (e.g., HIBP [15], GPC [13], and MIGP [14]) divide the total breached records (e.g., 15 GB~1.5 TB [14]) into small buckets (e.g., 0.01 MB~1.43 MB [14]).

Here, C3 clients can be users themselves (querying the C3 service using a browser extension), or other authentication services that query on behalf of their users [13]. C3 services are widely used by common users, e.g., HIBP serves around 7 billion requests per month on average [16]. The breach warning issued by the C3 service does work: As reported in USENIX SEC'19 [13], 94% of new passwords are stronger than old ones if users have received the warning. The ability

Received 20 November 2025; revised 11 June 2026; accepted 9 July 2026. Date of publication 20 July 2026; date of current version 17 August 2026. This work was supported in part by the National Natural Science Foundation of China under Grant 624B2071; in part by the Natural Science Foundation of Tianjin, China, under Grant 25JCJQJC00230; and in part by the Fundamental Research Funds for the Central Universities, Nankai University under Grant 63263258. The associate editor coordinating the review of this article and approving it for publication was Prof. Pinghui Wang. *(Corresponding author: Ding Wang.)*

The authors are with the College of Cryptology and Cyber Science, Nankai University, Tianjin 300350, China, also with the Key Laboratory of Data and Intelligent System Security, Ministry of Education, Tianjin 300350, China, also with Tianjin Key Laboratory of Network and Data Security Technology, Tianjin 300350, China, and also with Beijing Key Laboratory of Post-Quantum Cryptography, Beijing 100083, China (e-mail: songmi@mail.nankai.edu.cn; wangding@nankai.edu.cn; liguanling@mail.nankai.edu.cn; zhenli@mail.nankai.edu.cn).

This article has supplementary downloadable material available at https://doi.org/10.1109/TIFS.2026.3715105, provided by the authors.

Digital Object Identifier 10.1109/TIFS.2026.3715105

to detect exposed credentials and prompt users to take effective measures highlights the critical importance of C3 services.

A. Related Work

Faced with security problems caused by credential stuffing attacks [17], C3 services like HIBP [15], GPC [13] and MIGP [14] are proposed. According to the credential checked, C3 services can be categorized into: username-only C3 services [15], password-only C3 services [18] and username-password C3 services [13], [14], [18], [19], [20], [21], [22].

1) Username-Only C3 Services: HIBP [15] supports querying usernames. The breached password records contain the email address or username and a list of sites where they appeared in breaches [15]. To check whether an account is exposed, the client sends the username u to the HIBP-UN server. The server returns all leakage records that contain u. However, querying a username exposes the client's identity. If a user uses both HIBP-UN and other username-password C3 services (e.g., IDB [18] and MIGP [14]), the username leaked by HIBP-UN will make other C3 services vulnerable to known-username attacks (i.e. the attacker knows the client's username and attempts to learn the query password) [18]. Exposing the client's identity may lead to extortion of the victim [13]. Such services may be inaccurate: Even if users have changed their passwords, they will still receive a warning.

2) Password-Only C3 Services: HIBP [15] is the first password-only C3 service. The HIBP server stores the SHA1 hash of the leaked passwords and partitions them by the first 20 bits. To check if a password pw is exposed, the client sends the first 20 bits of SHA1(pw). Then server returns all the hashes that share the same prefix. Finally, the client checks if the complete hash of pw appears in these hashes. HIBP is efficient because there are no complex cryptographic primitives. However, the first 20 bits of SHA1(pw) offer an additional advantage to online guessing attacks. Frequency-smoothing bucketization (FSB) [18] assigns a password to multiple buckets based on its probability. Since the bucket identifier is not the deterministic mapping of SHA256(pw), FSB [18] leaks less information at the expense of computational complexity and bandwidth.

Password-only C3 services have two defects. First, a malicious client learns the hashes of sensitive passwords, can attempt to recover the original passwords by conducting a brute force attack. Second, using passwords as credentials checked has high false positives [13]. e.g., users u_a and u_b choose the same password pw, and there is (u_b, pw) in the database. When u_a queries with pw, she receives a warning. In fact, (u_a, pw) is not subject to credential stuffing attacks.

3) Username-Password C3 Services: These services check if a (username, password)-pair is exposed, resulting in fewer false positives compared to password-only C3 services [13]. However, the bucket identifier in GPC [13] is the first 16 bits of Argon2($u\|pw$), providing an additional advantage to attackers. To address this, IDB [18] uses the hash-prefix of only the username. Then, MightIGetPwned (MIGP) [14], an updated version of IDB [18], warns users if they select passwords similar to breached passwords. MIGP 2.0 [22], a new iteration of MIGP [14], addresses the collision of the password-generating function used in MIGP [14].

Wang and Reiter [23] propose a solution where multiple websites coordinate to detect credential stuffing on individual user accounts. After that, Wang and Reiter [20] propose a scheme for a website to use honeywords to detect breaches, without relying on any secret persistent state. Replacing anomalous logins or honeywords with compromised credentials, respectively, [20] and [23] can be used as C3 services. Li et al. [21] design a C3 service named Pipa. The Pipa client is required to deploy a homomorphic encryption module. The Pipa server stores plaintext passwords, which will be immediately leaked to attackers upon server compromise.

B. Motivations

Although C3 services are promising method to mitigate the threat of credential stuffing attacks, most of the existing C3 schemes (e.g., HIBP [15], GPC [13] and MIGP [14]) assume that the C3 server is completely trustworthy. In other words, users are unable to verify the results of their credential leak queries, thus overlooking the verifiability of query results, i.e., *missing an important security property: the verifiability of query.* These schemes implicitly trust the C3 servers to correctly handle the hash values of usernames and passwords, which raises concerns about whether C3 servers can provide reliable query results when the C3 server is compromised.

The unreliable results of a C3 server can be attributed to a variety of factors, from technical issues [24] and human errors [25] to malicious external attacks [26]. For instance, to hide compromises of their stakeholders, especially during critical moments (such as initial public offerings and acquisition negotiations [27]), in the Verizon acquisition negotiations, Yahoo is exposed for leaking the passwords of 500 million users two years earlier. This results in a 400 million reduction in the acquisition price [28]. Also, malicious servers could tamper with credential databases to fabricate false positives [29], as exemplified by Microsoft's 38 TB private data leak being maliciously edited to mislead user security checks [30], Capita's 655 GB resident data breach was algorithmically altered to conceal actual credential exposures [31].

There is another example that C3 servers can profit/benefit from providing unreliable query results: If the C3 server returns an *incorrect* result for a request associated with a leaked account, the user is likely to keep using the accounts associated with this leaked password. This behavior can easily be achieved without detection in existing C3 services (e.g., GPC [13] and IDB [18]), simply by returning incorrect results for all requests that share the same hash-prefix with the target user. Consequently, the C3 server can continue to log into the victim's account with the leaked password.

One consequence of unreliable checking results is that users may take inappropriate actions due to false positives. There is a critical need for verifiable checking, which allows users to verify the accuracy of the returned results, transitioning users from *passive acceptance to active verification* of query results. Thus, this raises the following key research questions (**RQs**):

- RQ1: Can state-of-the-art C3 schemes (e.g., GPC [13]) detect both types of deception in query results? (1) claim that an unexposed account has been breached, and (2) claim that an exposed account has not been breached.
- RQ2: Can the verifiability problem be solved in the single-server setting, e.g., by employing the verifiable

oblivious pseudo-random function (OPRF), verifiable private set intersection (PSI), verifiable private information retrieval (PIR), zero-knowledge proof?

RQ3: How can the verifiability of query results be achieved in the two-server setting? Note that the query process should still be secure (against online guessing attacks and breach extraction attacks) and efficient.

C. Contributions

We, *for the first time*, propose the notion of **Ver**ifiable **C**ompromised **C**redential **C**hecking services (VerC3), and define VerC3 as a suite of protocols that meet 11 desirable properties (see Sec. III-D). To show the practicality of VerC3, we build a simple yet effective construction, named **H**ave **I** **R**eally **B**een **P**wned (HIRBP). HIRBP enables clients to determine, in a verifiable manner, whether their (username, password) pairs are present in a breach or not. HIRBP offers two main advantages compared to existing C3 services:

First, HIRBP is resistant to malicious servers (while no existing C3 protocols can resist) and network-level adversaries. We achieve the verifiability of query results by employing digital signatures [32] and non-interactive zero-knowledge (NIZK) proofs [33] in the two-server setting (i.e., the data owner and the online server). The signature enables a client to verify the correctness of *the bucket* sent by the online server, and the NIZK to verify the consistency of *the OPRF key* used in breached password records and the responses to clients.

Second, HIRBP has a weaker reliance on OPRF security (while no existing C3 scheme achieves this). In existing OPRF-based C3 services (e.g., GPC [13] and IDB [18]), the OPRF key (stored on the C3 server) is the sole assurance for their password security. An adversary who somehow obtains the OPRF key can learn users' sensitive passwords, by performing an offline guessing attack on previously transmitted buckets. Our scheme enhances security by utilizing both the long-term OPRF key and the ephemeral Diffie-Hellman key. For queries without the active participation of the attacker, because the ephemeral Diffie-Hellman key is unknown, the attacker cannot perform offline password guessing attacks.

In summary, the contributions of this work are as follows:

- **A formal VerC3 model**. We reveal that the verifiability of query results is *inherently* unachievable in the single-server setting. We provide a formalization of VerC3 and a practical adversary model. Our VerC3 definition is general and can well capture all properties specified for C3 services by prior major works (e.g., GPC [13] and IDB [18]). In addition to the verifiability of query results (P6), we enrich the C3 property with two new crucial properties: forward secrecy (P10) and resistance to pre-computation attacks (P11) (see Sec. III-D).
- **The first VerC3 protocol**. We propose a simple yet effective construction, named HIRBP. We achieve the verifiability of query results by using digital signatures and non-interactive zero-knowledge (NIZK) proofs in the two-server setting. Any tampering by either server will be detected. We formally prove the verifiability property of our HIRBP in the random oracle model.
- **Prototype implementation**. We implement a prototype of HIRBP based on 1.4 billion leaked (username, password) pairs. Experimental results show that our HIRBP is as efficient as IDB [18]. The average bucket size for a hash-prefix of 20 bits is 1,399 entries. It takes the client 136.91ms to complete a query on a common PC, with a total bandwidth of 86 KB (see Table IV).
- **Extensive evaluation**. We comprehensively evaluate the threat of online password guessing attacks and breach extraction attacks. Experimental results on online guessing attacks (see Table V) show that our HIRBP can well resist such attacks: Within 10^3 guesses, the success rates against our HIRBP are 6.12%~6.55%, while that of GPC [13], HIBP [15], and EnZoic [19] are 74.16%~75.60%. Experimental results on breach extraction attacks (see Table VI) show that blocking mechanisms can well mitigate such attacks: Within 10^3 guesses, the success rates against our HIRBP are 0.31% when a blocklist of size 10^4 is used, but 6.16% when no blocklist is used.

II. Preliminaries

In this section, we briefly overview the verifiability problem and discuss alternative cryptographic protocols, evaluating their potential as viable solutions and the inherent trade-offs in the single-server setting.

A. Verifiability Problem

The lack of a verifiable mechanism in C3 services leads to two types of deception in query results: (1) Falsely claiming that an unexposed account has been breached; (2) Falsely claiming that an exposed account has not been breached.

Existing C3 services can prevent the former. This is because C3 servers generally store leaked passwords in the form of a one-way hash value. For example, the HIBP server sends the SHA1 values of the leaked passwords [15]; The EnZoic server sends the Argon2 hash of the leaked (username, password) pairs [19]; The Google Password Checkup (GPC) server sends the oblivious pseudo-random function (OPRF) value of the leaked (username, password) [13]. To claim that the queried credential is leaked, the C3 server is required to send a hash of the queried credential. Due to the forgery resistance of the hash function (e.g., SHA1 used in HIBP [15]), it is hard to make this claim without knowing the queried information.

However, no existing C3 service can prevent the latter. For instance, malicious C3 server may delete/replace the hash value corresponding to the queried credential, causing clients to be unable to match any entry in the received bucket (see Step 5 in Fig. 1). In fact, it should match one entry. The essential goal of C3 services is to prevent credential stuffing attacks by alerting users to change their compromised passwords [14]. But now it does not warn users in time and misleads them to use leaked passwords, which is a serious issue. As reported in [13], 47.3% of queries receive warnings. Only misleading 1% of querying users would endanger/produce a large number of victims. To solve the verifiability problem, we introduce the verifiability of query results: The client can verify the correctness of the query result, regardless of whether the client's credentials are present in the received bucket.

The above analysis answers **RQ1**.

B. Alternative Cryptographic Protocols

We examine several well-studied cryptographic primitives (e.g., Authenticated PIR [34]) to solve the verifiability problem in single-server setting, but get unsatisfactory results.

1) Authenticated Private Information Retrieval (PIR): At USENIX SEC'23, Colombo et al. [34] proposed the latest authenticated PIR protocol in the single-server setting. Their protocol [34] enables a client to fetch a record from a remote server. The client either obtains the authenticated record or detects server misbehavior and securely aborts. Their protocol [34] requires the user to get a digest of the server's database through an out-of-band channel. But in C3 service, there are *no* out-of-band channels, because both the user's query and the server's response are transmitted online.

2) Malicious Private Set Intersection (PSI): A PSI protocol allows two parties to learn the intersection of sets that they each hold, without revealing anything else about their inputs [35], [36]. Malicious PSI protocols are generally based on Cuckoo hashing [35], OPRF [36], oblivious transfer [37], and Bloom filter [38]. However, with a file size of at least 1 billion ($\approx 2^{30}$) breached password records, the time of one query is likely too large to be practical. For example, Pinkas et al.'s protocol [35] takes 18.6 seconds at 2^{20} database size. Chase-Miao's [36] and Pinkas et al. [37] protocols take 200 seconds and 440 seconds, respectively, at 2^{24} database size.

3) Verifiable Oblivious Pseudo-Random Function (OPRF): In a verifiable OPRF protocol, the client is convinced that the server has correctly evaluated the pseudo-random function on an input x chosen by the client, using the server's key k [39], [40]. The application of verifiable OPRFs in C3 services can only ensure that the user's (username, password)-pair is correctly evaluated, but *cannot* guarantee the correctness or completeness of the bucket sent by the server. This is because the user only has its (username, password)-pair, but does not have the (username, password) pairs corresponding to the credentials in the received bucket.

4) Zero-Knowledge Sets: At USENIX SEC'22, Pal et al. [14] pointed out that the zero-knowledge sets technique [41] can prevent malicious C3 servers. That is, the C3 server publishes a commitment to the database and then performs zero-knowledge proofs of (non-)membership. Although it seems plausible, this approach is not practical. Notably, because the leaked data is constantly updated (e.g., HIBP [15]), the server needs to constantly issue new commitments. However, a malicious server can issue valid new commitments to the incomplete/incorrect database, thereby cheating clients.

5) Other Approaches: Trusted execution environments (TEE) create two separate worlds on the same system: a secure world and a non-secure world [42]. If we require the C3 server to compute breached password records in TEE, it can resist external attackers, but it *cannot* resist malicious C3 servers. Transparency is provided by shopping platforms such as Amazon [43]. It promotes consumer trust in merchants by using unique codes to identify individual units. However, for single-server C3 services, there is no additional platform to ensure the correctness of query results.

Summary. There is *no* solution to the verifiability problem in the single-server setting (**RQ2**), *because the client has no source of information about the C3 server's database.* In this case, a malicious server could operate on an incomplete database and answer all queries consistently with that state. Consequently, when the server returns a negative result, the client cannot determine whether the queried credential is truly absent from the complete breach dataset or has been omitted by the server. Therefore, the verifiability of negative results cannot be achieved without introducing an additional trusted party or out-of-band channel, such as a public digest, a trusted auditor, or an additional non-colluding party. This is the reason why VerC3 adopts our two-server setting.

TABLE I
NOTATIONS AND ABBREVIATIONS

Symbol	Description	Symbol	Description
$\mathcal{C}$	Client	x, y	Diffie-Hellman exponents
$\mathcal{D}$	Data owner	τ, τ'	Masking values
$\mathcal{S}$	Online server	(pk, sk)	Signature keys
$\mathcal{A}$	Adversary	(u, pw)	Username-password pair
D	Breached credential dataset	X, Y	Ephemeral DH public keys
$\oplus$	Bitwise XOR	k	OPRF key†
$\|\|$	String concatenation	π	Non-interactive zero-knowledge proof
Z	set of blinded buckets	r	Random scalar chosen by $\mathcal{C}$
$\mathsf{H}(\cdot)$	One-way hash-function	λ	Security parameter
α	Blinded OPRF input	β	Server's response
Σ	Set of signatures for all buckets	K	Public OPRF verification key

† OPRF = Oblivious pseudo-random function.

We take OPRF key verification as an example. The OPRF key k is used in three places: (1) After the upload phase, the client stores $K = g^k$; (2) During the upload phase, the online server computes the OPRF values $\{F_k(u_i||pw_i) = \mathsf{H}_1(u_i||pw_i)^k | (u_i, pw_i) \in \mathsf{D}\}$; (3) During the query phase, to check if $(u, pw) \in \mathsf{D}$, the client sends $\alpha = \mathsf{H}_1(u||pw)^r$ to the online server, receives $\beta = \alpha^k$, and uses β to compute the OPRF value $F_k(u||pw) \leftarrow \beta^{1/r}$. Verifiable OPRFs can allow the client to check if the k used in (1) and (3) is consistent, but can *not* allow the client to check if the k used in (2) and (3) is consistent. Because the only way to verify it is to re-compute $F_k(u_i||pw_i)$ for each (u_i, pw_i) in the received bucket. It is impossible because the client does not have k nor (u_i, pw_i). Hence, we need at least one party to supervise the processing of breached password records. This is why the verifiability problem can be solved in the two-server setting: *For clients, the honest one of the two servers acts as the source of truth,* which can ensure the accuracy of results.

The two-server setting is more advantageous in a small business setting where the enterprise does not have its own dedicated servers and outsources its credentials to the cloud. For instance, Li et al.'s IDB protocol [18] can be integrated with Amazon's content delivery network, called CloudFront. It implies that there are three parties involved in the practical deployment of IDB [18]: a client (for querying), CloudFront (for responding to each query), and an implicit party (for updating the breached password records on CloudFront).

Cross-Service Checking. One may consider querying multiple C3 services to cross-check the reliability of a single service. However, this approach cannot replace query-result verifiability. First, different C3 services may check different types of credentials. For example, HIBP [15] is mainly password-only, while GPC [13] and IDB [18] check username-password pairs, so their results are not directly comparable. Second, different services may maintain different breach datasets, freshness levels, data sources, and deduplication rules. Thus, inconsistent results do not necessarily imply unreliable results, and consistent negative results still do not

indicate that the queried credential is absent. Third, querying multiple services may increase the client's privacy exposure, because different services may leak different bucket identifiers or auxiliary information. Therefore, cross-service checking cannot achieve the query-result consistency, whereas VerC3 provides a cryptographic guarantee that each returned result is consistent with the committed breach database.

III. Verifiable C3 Service

In this section, we present the formal definition, the adversary model, the security model, and the properties of VerC3.

A. Definition of VerC3

For ease of presentation, some intuitive notations are listed in Table I and will be used throughout this paper.

Service Architecture and Functionality: A VerC3 service includes three parties: a client, a data owner, and an online server[1], denoted by $\mathcal{C}$, $\mathcal{D}$, and $\mathcal{S}$ from now on. The data owner has a breached (username, password)-pair file $\mathsf{D} = \{(u_1, pw_1), \ldots, (u_{|\mathsf{D}|}, pw_{|\mathsf{D}|})\}$. Each $u_i \in U$ is a username and each $pw_i \in W$ is a password. The sets U and W consist of all possible user-chosen usernames and passwords. After processing the leaked data, $\mathcal{D}$ sends a blinded password file Z and its signature Σ to $\mathcal{S}$. Then $\mathcal{C}$ can query $\mathcal{S}$ with a (username, password)-pair (u, pw) to learn if $(u, pw) \in \mathsf{D}$. $\mathcal{S}$ returns match if $(u, pw) \in \mathsf{D}$ and none if $(u, pw) \notin \mathsf{D}$. In VerC3, it is assumed that the data owner $\mathcal{D}$ is honest during the upload and update phases and does not collude with the online server.

Definition 1: VerC3 is a verifiable compromised credential checking service, consisting of the following algorithms:

- $\mathsf{Setup}(1^\lambda) \rightarrow \mathsf{pp}$. Take a security parameter λ and return a public parameter pp. It is omitted by us.
- $\mathsf{KeyGen}(\mathsf{pp}) \rightarrow \mathcal{D}(k, sk, pk)$. Take a public parameter pp and return $\{k, sk, pk\}$ to the data owner $\mathcal{D}$, $\{k, pk\}$ to the online server $\mathcal{S}$, and $\{K = g^k, pk\}$ to the client $\mathcal{C}$[2]. Note that, (sk, pk) is a signature key pair and k is an oblivious pseudo-random function (OPRF) key.
- $\mathsf{Upload}(\mathcal{D}(k, sk, pk, \mathsf{D}),\ \mathcal{S}(k, pk)) \rightarrow \mathcal{S}(k, pk, \mathsf{Z}, \Sigma)$. Take as input the data owner $\mathcal{D}$ and the online server $\mathcal{S}$, and return a blinded file $\mathsf{Z} = \{F_k(u_i \| pw_i) | (u_i, pw_i) \in \mathsf{D}\}$ and its signature Σ to the online server $\mathcal{S}$.
- $\mathsf{Query}(\mathcal{C}(u, pw, K, pk), \mathcal{S}(k, pk, \mathsf{Z}, \Sigma)) \rightarrow \mathcal{C}(\mathsf{res})$. Take as input the client $\mathcal{C}$ and the online server $\mathcal{S}$, and return a query result $\mathsf{res} \in \{\mathsf{match}, \mathsf{none}\}$ to $\mathcal{C}$. Note that, (u, pw) is a (username, password)-pair, and the client rejects when the verifiability-related check fails.
- $\mathsf{Update}(\mathcal{D}(\{sk, pk, k^{\mathsf{old}}, \mathsf{D}^{\mathsf{new}}\}), \mathcal{S}(pk, k^{\mathsf{old}}, \mathsf{Z}^{\mathsf{old}}, \Sigma^{\mathsf{old}})) \rightarrow \mathcal{S}(pk, k^{\mathsf{new}}, \mathsf{Z}^{\mathsf{new}}, \Sigma^{\mathsf{new}})$. For simplicity, $\{k^{\mathsf{old}}, \mathsf{D}^{\mathsf{old}}, \mathsf{Z}^{\mathsf{old}}, \Sigma^{\mathsf{old}}\}$ denotes the original state, $\{k^{\mathsf{new}}, \mathsf{D}^{\mathsf{new}}, \mathsf{Z}^{\mathsf{new}}, \Sigma^{\mathsf{new}}\}$ denotes the updated state (e.g., OPRF key rotation and new leaks). This algorithm takes as input the data owner $\mathcal{D}$ and the online server $\mathcal{S}$, and returns a new OPRF key k^{new}, a new blinded file $\mathsf{Z}^{\mathsf{new}}$ and its signature Σ^{new} to $\mathcal{S}$, k^{new} to $\mathcal{D}$, and $K^{\mathsf{new}} = g^{k^{\mathsf{new}}}$ to $\mathcal{C}$.

[1]The data owner and the online server are essentially servers. We name them this way (instead of S_1, S_2) because their roles are different: The data owner gathers new leaks, and the online server responds to user queries.

[2]The trusted setup: We assume that the client can obtain the signature verification key pk and the OPRF verification key $K = g^k$.

TABLE II
Three Adversary Models Considered in Our Work

Adversary model	Username	Public info†	Bucket	Breach data	Existing literature
$\mathcal{A}_1$	✓	✓			[13], [14], [18], [22]
$\mathcal{A}_2$	✓	✓	✓		[13], [14], [18], [22]
$\mathcal{A}_3$	✓	✓	✓	✓	Our work

† Typical public info includes password datasets leaked from different websites, along with widely used algorithms for generating guessing dictionaries (such as Markov [47] and PCFG [48] algorithms).

Following the security framework in [22], we present a leakage-based security definition of VerC3 in Appendix C at https://tinyurl.com/VerC3-Mi-Supplemental-Material.

B. Adversary Model

As with Li et al [18] and Pal et al [14] works, we adopt the known-username assumption (i.e., attackers know the username of the querying user). Moreover, our VerC3 can be used in conjunction with TLS. But public key infrastructure (PKI) failures (e.g., software that does not verify certificates correctly, users that accept invalid certificates) are common [44]. To make our attacking model as realistic as possible, we assume that attackers can manipulate messages transmitted on the network, and can corrupt any party.

Adversary Model: As shown in Table. II, we consider three distinct threats. The first two threats are studied and mitigated in previous C3 work [6], [13], [14], [18]. In this work, we focus more on the third threat (i.e., threat $\mathcal{A}_3$).

(1) A malicious client (named $\mathcal{A}_1$) queries the online server to retrieve other users' breached sensitive passwords.
(2) A malicious online server (named $\mathcal{A}_2$) wants to learn about the password of the querying user.
(3) A malicious online server or data owner (named $\mathcal{A}_3$) trying to modify the query result of the client.

An attacker $\mathcal{A}_1$ is a malicious client who abuses the VerC3 service to extract breached sensitive passwords from the online server. Such an attack is called a *breach extraction attack*. In other words, a malicious client queries the server with the target user's username and a sequence of candidate passwords. If the result $\mathsf{matches}$, $\mathcal{A}_1$ knows that it has guessed the target user's password. There is a concern if the database contains data from leaks that are *not* yet publicly available [14], [15], [22]. Anti-abuse countermeasures (e.g., slow hashing and API rate limiting), can mitigate this attack.

An attacker $\mathcal{A}_2$ is a malicious online server that exploits the information leaked from queries to learn about a client's credentials. Note that the attacker who learns queried passwords by providing the malicious JavaScript code is beyond the scope of this paper. Ideally, if the online server sends total breached password records to the client, it does not reveal any information about the client's credentials. But it is not practical to deploy. As summarized in Table III, the state-of-the-art protocols [13], [14], [15], [18] send a hash-prefix of the username/password/username-password to bucketize total password records into smaller sets (buckets).

An attacker $\mathcal{A}_3$ is a malicious online server or data owner who wants to modify the query result. For instance, (1) the online server may manipulate the query results by directly returning an incorrect response, or by indirectly modifying the breached data in the update phase. (2) The data owner

TABLE III
COMPARISON AMONG 11 REPRESENTATIVE COMPROMISED CREDENTIAL CHECKING (C3) SERVICES AND OUR HIRBP

Scheme			Credentials checked	Bucket identifier	Desirable property										
Name	Venue	Ref.			P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	P11
MIGP 2.0	IEEE S&P'24	[22]	{Username, Password}	20 bits of SHA256(u)	✓	✓	✓	✓	✓	×	✓	✓	✓	×	✓
MIGP	USENIX SEC'22	[14]	{Username, Password}	20 bits of SHA256(u)	✓	✓	✓	✓	✓	×	✓	✓	✓	×	✓
Pipa	AsiaCCS'21	[21]	{Username, Password}	16 bits of Argon2(u)	✓	✓	✓	✓	✓	×	✓	✓	✓	–	×
WR20-Cuckoo	USENIX SEC'21	[57]	{Username, Password}	20 bits of SHA256(u)	×	×	✓	✓	×	×	×	✓	✓	×	✓
WR19-Bloom	USENIX SEC'20	[20]	{Username, Password}	20 bits of SHA256(u)	×	×	✓	✓	✓	×	×	✓	✓	–	×
IDB	CCS'19	[18]	{Username, Password}	16 bits of Argon2(u)	✓	✓	✓	✓	✓	×	✓	✓	✓	×	✓
FSB	CCS'19	[18]	Password	30 bits of SHA256(pw)	✓	✓	✓	✓	✓	×	✓	×	×	–	×
GPC	USENIX SEC'19	[13]	{Username, Password}	16 bits of Argon2($u\|\|pw$)	✓	✓	✓	×	✓	×	✓	✓	✓	×	✓
EnZoic	Industry, 2016	[19]	{Username, Password}	40 bits of Argon2($u\|\|pw$)	✓	✓	✓	×	✓	×	✓	✓	✓̃	–	✓̃
HIBP	Industry, 2013	[15]	Password	20 bits of SHA1(pw)	✓	✓	✓	×	✓	×	✓	×	×	–	×
HIBP-UN	Industry, 2013	[15]	Username	Username	✓	✓	✓	×	×	×	✓	✓	✓	–	–
Our HIRBP			{Username, Password}	20 bits of SHA256(u)	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

† MIGP refers to the MIGP-server protocol in [14]. The bucket identifier in FSB [18] is an interval starting with 30 bits of SHA256(pw). The client randomly selects a value in the range as the bucket identifier for this query. For simplicity, it is represented by 30 bits of SHA256(pw) in this table. HIBP [15] provides username(u) query and password(pw) query, represented by HIBP-UN and HIBP, respectively. ✓ achieving the property; ✓̃ partially achieving; × not achieving; – not considering. u = Username, pw = Password.

can manipulate the query results by indirectly modifying the original breached data in the update phase.

The attacker $\mathcal{A}_3$: (1) can control the communication channel; (2) can not corrupt the client at any time; and (3) can corrupt the online server or the data owner, *but can not corrupt them simultaneously.* That is, the client is always honest. After that, *at most* one of the data owner and the online server is malicious. This assumption is realistic. Because we can place the data owner and the online server in different administrative domains, running different operating systems, software stacks, security mechanisms, to ensure distributed security [45], [46].

Non-Collusion Assumption: The security of HIRBP relies on the assumption that the data owner $\mathcal{D}$ and the online server $\mathcal{S}$ do not collude. This assumption can be realized in practice by deploying the two roles across different administrative domains or vendors. For example, $\mathcal{D}$ may be a breach-data curator, security company, identity-protection provider, or enterprise security team that collects, preprocesses, and signs breached credential records, while $\mathcal{S}$ may be an independent cloud provider, or outsourced query-service provider that stores blinded buckets and serves online queries. This separation prevents $\mathcal{S}$ from independently modifying buckets without detection, because it does not hold the signing key.

C. Security Model

We present a formal definition for *the verifiability of query results* under the attacker $\mathcal{A}_3$. We adopt the formal model of Bellare et al. [49] to capture the capability of $\mathcal{A}_3$. Further, a detailed formal definition of the Execute, Send, Test, Reveal and Corrupt oracles can be found in Appendix A, available at https://tinyurl.com/VerC3-Mi-Supplemental-Material.

Definition 2 (Verifiability): The verifiability of query results ensures that an attacker $\mathcal{A}_3$ cannot manipulate the query result res without being detected by the client C^i. In execution of the protocol $\mathcal{P}$, $\mathcal{A}_3$ can make a polynomial number of Execute-query, Send-query, and Reveal-query. At the end of the game, C^i outputs a guess bit c' for the bit c involved in the Test-query. $\mathcal{A}_3$ wins the game if $c' \neq c$, and this event is denoted by Succ. The advantage of $\mathcal{A}_3$ in breaking the verifiability of query results in $\mathcal{P}$ is defined as

$$\mathsf{Adv}^{\mathrm{ver}}_{\mathcal{P}}(\mathcal{A}_3) = \Pr[\mathsf{Succ}(\mathcal{A}_3)] = \left|\Pr\left[c' = c\right] - 1/2\right|. \quad (1)$$

A VerC3 protocol $\mathcal{P}$ satisfies the verifiability of query results if for any probabilistic polynomial-time (PPT) adversary $\mathcal{A}_3$, the advantage $\mathsf{Adv}^{3=(n)}_{ver}\mathcal{P}[(\mathcal{A}_3)]$ is negligible.

Remark 1: Generally, the coin c in the Test-query is flipped for attackers [50], while in VerC3, it is flipped for the client C^i. Note that C^i does *not* need to produce c, but needs to receive the correct (if $c = 0$) or incorrect (if $c = 1$) query result returned by Test-query. We simulate the client outputting a guessing bit c' as follows: C^i either accepts the query result (i.e., outputs $c' = 0$) or aborts (i.e., outputs $c' = 1$). Thus, $c \neq c'$ means that the client can not verify the query result, and *the attacker successfully breaks the verifiability of query results*. If we replace C^i with the attacker, then $c \neq c'$ means that the attacker can not verify the query result, which does *not* match the verifiability of the query results defined in VerC3.

D. Properties of VerC3

Based on the schemes [13], [14], [18], [21], we aggregate the properties of C3 services and classify them into user privacy and server privacy. We mainly consider the case where the server's database contains breach data that is not yet publicly available [22]. Attackers may abuse a C3/VerC3 service to extract sensitive passwords from the server (called breach extraction attacks [14]). We provide three properties (i.e., query results verifiability (P6), forward secrecy (P10), and resistance to pre-computation attacks (P11)) not mentioned before, which enrich the C3 property set to match our VerC3.

Classification A - User experience and privacy.

- P1: *Democratized access*. A C3 service should be accessible to all end users and identity providers, and not require trust between the involved parties [13].
- P2: *Near real-time*. The time from creating a query to getting its result should be near real-time [13].
- P3: *Actionable and breached*. A C3 service warns users if they select breached passwords (not weak passwords). It should provide users with accurate and actionable security advice, e.g., changing insecure passwords. Alerting

should trigger only when all information necessary to access an account is exposed.

P4: *Requester credential anonymity*. Clients with credentials from the same anonymity set create requests that are indistinguishable to the server [13]. Namely, the C3 server knows at most a k-anonymity hash-prefix of the username, but does not learn the password at all.

P5: *Query result privacy*. Only the client knows whether the queried credential appears in a breach [21].

P6: *Query result verifiability*. Clients can verify the correctness of query results, regardless of whether their credentials are present in the received bucket or not.

Classification B - Server efficiency and security.

P7: *Resistance to Denial of Service*. A response from the server should not require significantly more computation than a request by a client [13], [21].

P8: *Inefficient oracle*. The service could not be abused by attackers as a pragmatic oracle to obtain breached credentials [13], [21]. A malicious client can extract all credentials on the server side with an infinite number of queries. P8 aims to limit the client's online query capabilities, e.g., by requiring an API key through slow registration or subscription process [15], restricting IP addresses, using slow hashes (e.g., Argon2 [51]) or trapdoor hashes (e.g., time-lock puzzle [14]).

P9: *Responses with bounded leakage*. The client only gets bounded information about the database from the server's response [13], [21]. P9 focuses on the leakage caused by message transmission, and P10 will analyze the leakage resulting from server compromise.

P10: *Forward secrecy*. The compromise of long-term keys cannot expose previously transmitted passwords to offline guessing attacks. Only compromising the OPRF key can't perform an offline attack in our protocol.

P11: *Resistance to pre-computation attacks*. Attackers can not learn breached credentials immediately upon server compromise. Namely, attackers cannot prepare a reverse lookup table based on a password dictionary, allowing sensitive password records to be found instantly.

Regarding the property forward secrecy (P10), which makes sense in the context of quantum-safe cryptography, because there are currently no known efficient OPRFs considered to be quantum-safe [44]. Here we only consider the compromise of long-term keys. If the key and the short-term/ephemeral key are compromised simultaneously, the offline guessing attack can be executed. For example, an attacker can query all bucket indexes to get all OPRF values and store them offline. Since the attacker issues the query, the ephemeral key can not protect the credentials. Then, the compromise of the OPRF key naturally leads to an offline guessing attack. It means, forward secrecy ensures that sessions where users issue queries themselves will *not* be subject to offline guessing attacks.

Malicious Clients and Abuse Prevention: No C3 service can completely prevent breach extraction attacks if a malicious client is allowed to issue an unlimited number of online queries. Thus, the practical goal is to make such attacks inefficient while maintaining usability for legitimate users. Different anti-abuse mechanisms provide different trade-offs. IP-based rate limiting, anonymous rate-limited tokens, computational puzzles, and slow hashing are relatively compatible with democratized access, because they do not necessarily require strong user registration. However, IP-based throttling may be bypassed by distributed attackers or affect users behind NATs, while slow hashing and puzzles increase latency and client-side computation. Stronger mechanisms such as API keys, account registration, subscription-based access control, or identity verification provide better accountability and more precise rate limiting, but they weaken open access and may introduce additional privacy concerns. Blocklists can reduce the usefulness of querying common weak passwords, but may also cause false negatives for legitimate users. Therefore, abuse prevention should be regarded as a deployment-level trade-off between privacy, usability, and resistance to malicious clients.

IV. The First Verifiable C3 Protocol

We present a simple, robust yet efficient VerC3 protocol, i.e., HIRBP. The **RQ3** can be well addressed by our HIRBP.

A. Design Ideas

In Fig. 2, we provide an overview of our HIRBP. Compared with state-of-the-art C3 protocols (e.g., MIGP [14], IDB [18] and GPC [13]), our HIRBP protocol adds a non-interactive zero-knowledge (NIZK) proof on the OPRF key k and a digital signature on the bucket z_j.

In the left part of Fig. 2, the data owner applies the oblivious pseudo-random function (OPRF) $F_k(u\|pw) = \mathsf{H}_1(u\|pw)^k$ to the breached records, and divides them into buckets $\{z_1, \ldots, z_n\}$ by the l-bit hash-prefix $j = \mathsf{H}_0^{(l)}(u)$ (e.g., $l = 20$). Let $m_j = \mathsf{H}_2(z_j)$ denote the hash value of OPRF values in z_j. Then, the data owner signs three elements of each bucket (i.e., j, m_j, and $K = g^k$) with the private key sk. The signature of the bucket z_j is $\sigma_j = \mathsf{Sig}(sk, j\|m_j\|K)$. The data owner sends buckets and signatures to the online server.

The right part of Fig. 2 is the query phase between the client and the online server. Compared with C3 protocols [14], [18], HIRBP adds a zero-knowledge argument and a signature σ_j on the bucket z_j. $\mathsf{NIZK}(g, K, \alpha, \beta)$ provides the equality of two discrete logarithms, i.e., $K = g^k$ and $\beta = \alpha^k$, where α is sent by the client, and $\beta = \alpha^k$ is sent by the online server. A failure to verify either NIZK or σ_j results in a rejection. Finally, the client obtains a reliable query.

There are four subtleties/tricks in solving the verifiability problem and before this work, there is no existing solution.

- *Employing digital signatures to verify the correctness of buckets.* This approach can withstand two attacks: (1) If attackers send an incorrect bucket $\{z_{j'}, \sigma_{j'}\}, j' \neq j$, the signature verification $\mathsf{Vf}(pk, j\|m_{j'}\|K, \sigma_{j'})$ will fail. (2) If attackers delete/replace any entry in z_j, the client will compute an incorrect hash value $\widetilde{m}$, and $\mathsf{Vf}(pk, j\|\widetilde{m}\|K, \sigma_j), \widetilde{m} \neq m_j$, will fail.
- *Employing a non-interactive zero-knowledge proof $nizk(g, K, \alpha, \beta)$ to verify the consistency of OPRF keys.* NIZK guarantees that the OPRF key k used for the bucket is correct, enforcing key consistency and preventing key-rollback attacks (where the server reuses keys across different users or time periods). For example, retaining k_{t-1} after updating to k_t would allow valid signatures on buckets that ignore current breach data. In our HIRBP protocol, if attackers send $\beta' = \alpha^{k'}, k' \neq k$, the nizk

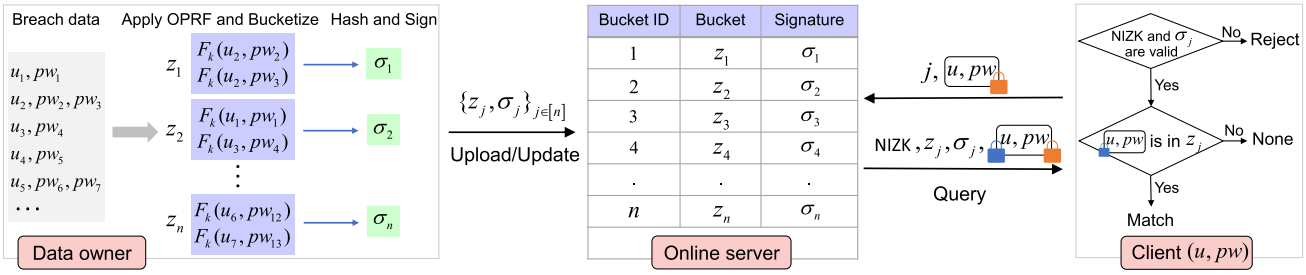


Fig. 2. HIRBP protocol for checking if a (username, password)-pair (u, pw) is exposed, *in a verifiable manner*. It is an extension of OPRF-based C3 protocols (such as GPC [13], IDB [18] and MIGP [14]). HIRBP = Have I Really Been Pwned; OPRF = Oblivious pseudo-random function; NIZK = Non-interactive zero-knowledge. The detailed protocol construction is shown in Fig. 3.

verification will fail. Because a malicious online server cannot construct zero-knowledge proofs about two different discrete logarithms (i.e., k and k', $k' \neq k$).

- *Using both the long-term key and the short-term key to provide forward secrecy.* That is, the online server sends $F_k(u\|pw) \oplus \mathsf{H}(j\|\alpha\|\beta\|g^x\|g^y\|g^{xy})$ to the client. If the attacker does not actively participate in the previous query, the ephemeral Diffie-Hellman key g^{xy} is unknown to the attacker. Even if the long-term OPRF key k is leaked, attackers cannot perform offline guessing attacks on previously transmitted credentials. We emphasize that both $X = g^x$ and $Y = g^y$ must be generated per session. Otherwise, forward secrecy cannot be satisfied. Here's an example to understand this: $X = g^x$ is generated per session, and $Y = g^y$ is added to the blinded password file, i.e., $z_j \leftarrow z_j \cup \{F_k(u\|pw) \oplus Y\}$. It seems like an efficient scheme. However, an attacker with the compromised k can first query a well-known breach credential (e.g., (Alice, `123456`)) to obtain Y, and then perform offline guessing attacks on other target passwords.
- *Choosing the best OPRF to use in our HIRBP.* HashDH [13] outperforms 2HashDH [52] and 3HashSDHI [53] in terms of data protection and storage overhead.

$$\begin{aligned} \text{HashDH: } & F_k(x) = \mathsf{H}_1(x)^k, \\ \text{2HashDH: } & F_k(x) = \mathsf{H}_2(x, \mathsf{H}_1(x)^k), \\ \text{3HashSDHI: } & F_k(t\|x) = \mathsf{H}_2(t\|x, \mathsf{H}_1(x)^{1/(k+\mathsf{H}_3(t))}). \end{aligned} \quad (2)$$

$\mathsf{H}_i(\cdot)(i = 1, 2, 3)$ are hash functions, k is an OPRF key. t is a public input (e.g., the bucket identifier [53]). If we employ HashDH, the data owner sends k'/k, then the online server can update original password records, i.e., $F_{k'}(u\|pw) = F_k(u\|pw)^{k'/k}$. However, if we employ 2HashDH, the online server needs to know (u, pw) and stores $\mathsf{H}_1(u\|pw)^k$ to compute $F_{k'}(u\|pw) = \mathsf{H}_2(u\|pw, (\mathsf{H}_1(u\|pw)^k)^{k'/k})$. Similarly, when employing 3HashSDHI, the online server requires knowing (u, pw) and storing $\mathsf{H}_1(u\|pw)^{1/(k+\mathsf{H}_3(t))}$.

B. Protocol Description

This section formally describes our HIRBP. For clarity and intuitiveness, we provide details of HIRBP in Fig. 3.

Setup. We assume that public parameters are established before any executions of the HIRBP protocol take place. These parameters can be hard-coded into an implementation of HIRBP [54], [55]. Let λ denote the security parameter.

Step S1. The $\mathsf{Setup}(1^\lambda)$ algorithm selects a finite cyclic multiplicative group $\mathbb{G}$ of prime order q, generated by a generator g. It chooses a prefix length l, and sets up hash functions H_0, H_2, H_3: $\{0,1\}^* \rightarrow \{0,1\}^\lambda$, and H_1: $\{0,1\}^* \rightarrow \mathbb{G}$. As with Gu et al.'s work [56], we use multiple hash functions to indicate the independence of output values. $\mathsf{H}_i(i = 0, 2, 3)$ can be implemented by setting $\mathsf{H}_i(x) = \mathrm{SHA256}(i\|x)$. $\mathsf{Setup}(1^\lambda)$ returns them as the public parameter pp.

Step S2. The $\mathsf{KeyGen}(\mathsf{pp})$ algorithm samples $k \leftarrow\!\!\$\ \mathbb{Z}_q$ and sets it as the OPRF key. It generates a signature key pair $sk \leftarrow\!\!\$\ \mathbb{Z}_q$, $pk = g^{sk}$. The OPRF used in HIRBP is $F_k(x) = \mathsf{H}_1(x)^k$. Then, $\{k, sk, pk\}$ is sent to the data owner D, $\{K = g^k, pk\}$ is sent to the client C, and $\{k, pk\}$ to the online server S.

Upload. The participants are the data owner D (with input (k, sk, pk, D)) and the online server S (with input (k, pk)).

Step U1. $\mathsf{D} = \{(u_1, pw_1), \ldots, (u_{|\mathsf{D}|}, pw_{|\mathsf{D}|})\}$. For each (username, password)-pair $(u, pw) \in \mathsf{D}$, the data owner computes $j = \mathsf{H}_0^{(l)}(u)$ and $z_j = z_j \cup F_k(u\|pw)$. z_j is initially an empty set. Let Z denote all buckets.

Step U2. For each bucket $z_j \in \mathsf{Z}$, the data owner computes the hash of all OPRF values in the bucket z_j, computes the signature $\sigma_j \leftarrow \mathsf{Sig}(sk, j\|m_j\|K)$. Let Σ denote all signatures.

Step U3. $D \Rightarrow S$: $\{\mathsf{Z}, \Sigma\}$. The data owner sends $\{\mathsf{Z}, \Sigma\}$ to the online server S over a public channel.

Step U4. After receiving the message from $\mathcal{D}$, $\mathcal{S}$ verifies the signature of each bucket: $1/0 \leftarrow \mathsf{Vf}(pk, j\|m_j\|K, \sigma_j)$.

Query. The two participants are the client C (with input (K, pk, u, pw)) and the online server S (with input $(k, pk, \mathsf{Z}, \Sigma)$). We assume that C can obtain K and pk.

Step Q1. To check if $(u, pw) \in \mathsf{D}$, the client randomly selects $r \leftarrow\!\!\$\ \mathbb{Z}_q$ and $x \leftarrow\!\!\$\ \mathbb{Z}_q$, computes $j \leftarrow \mathsf{H}_0^{(l)}(u), \alpha \leftarrow \mathsf{H}_1(u\|pw)^r$, and $X \leftarrow g^x$. The r ensures that the online server learns nothing about (u, pw), except the bucket identifier j.

Step Q2. $C \Rightarrow S$: $\{j, \alpha, X\}$. The client C sends $\{j, \alpha, X\}$ to the online server S over a public channel.

Step Q3. After receiving the message, S randomly selects $y \leftarrow\!\!\$\ \mathbb{Z}_q$, computes $\beta = \alpha^k, \pi \leftarrow \mathsf{NIZK}(g, K, \alpha, \beta), Y \leftarrow g^y$, $\tau \leftarrow \mathsf{H}_3(j\|\alpha\|\beta\|X\|Y\|X^y)$. For $z \in z_j$, S computes $z'_j \leftarrow z'_j \cup \{z \oplus \tau\}$. z'_j is initially empty.

Step Q4. $S \Rightarrow C$: $\{\beta, \pi, Y, z'_j, \sigma_j\}$. The server sends $\{\beta, \pi, Y, z'_j, \sigma_j\}$ to the client over public channel.

Step Q5. After receiving the message, the client verifies π and computes $\tau' \leftarrow \mathsf{H}_3(j\|\alpha\|\beta\|X\|Y\|Y^x)$. Then, for $z' \in z'_j$, the client computes $z_j \leftarrow z_j \cup \{z' \oplus \tau'\}$. The

The setup phase

Setup(1^λ)

$g \leftarrow\$ \mathbb{G}, l \leftarrow\$ \mathbb{Z}_q$

$\mathsf{H}_1 \leftarrow\$ \mathcal{H} = \{H : \{0,1\}^* \rightarrow \mathbb{G}\}$

$\mathsf{H}_0, \mathsf{H}_2, \mathsf{H}_3 \leftarrow\$ \mathcal{H}' = \{H : \{0,1\}^* \rightarrow \{0,1\}^\lambda\}$

return $\mathsf{pp} = \{g, l, \mathsf{H}_0, \mathsf{H}_1, \mathsf{H}_2, \mathsf{H}_3\}$

KeyGen(pp)

$k \leftarrow\$ \mathbb{Z}_q$

$sk \leftarrow\$ \mathbb{Z}_q$

$pk \leftarrow\$ g^{sk}$

return $\{k, sk, pk\}$

The upload phase

Data owner upon (k, sk, pk, D)

$\mathsf{D} = \{(u_1, pw_1), ..., (u_{|\mathsf{D}|}, pw_{|\mathsf{D}|})\}$

for $(u, pw) \in \mathsf{D}$ **do** :

$j \leftarrow \mathsf{H}_0^{(l)}(u)$

$z_j \leftarrow z_j \cup \{\mathsf{H}_1(u\|pw)^k\}$

for $z_j \in \mathsf{Z}$

$m_j \leftarrow \mathsf{H}_2(z_j)$

$\sigma_j \leftarrow \mathsf{Sig}(sk, j\|m_j\|g^k)$

record $\{\mathsf{Z}, \Sigma\}$

(Z, Σ) →

Online server upon (k, pk)

for $(z_j, \sigma_j) \in \{\mathsf{Z}, \Sigma\}$ **do** :

$m_j \leftarrow \mathsf{H}_2(z_j)$

reject if $\mathsf{Vf}(pk, j\|m_j\|g^k, \sigma_j) \neq 1$

record $\{\mathsf{Z}, \Sigma\}$

The query phase

Client upon (u, pw, g^k, pk)

$j \leftarrow \mathsf{H}_0^{(l)}(u)$

$r \leftarrow\$ \mathbb{Z}_q, \alpha \leftarrow \mathsf{H}_1(u\|pw)^r$

$x \leftarrow\$ \mathbb{Z}_q, X \leftarrow g^x$

(j, α, X) →

reject if π is invalid

$\tau' \leftarrow \mathsf{H}_3(j||\alpha||\beta||X||Y||Y^x)$

for $z' \in z'_j$ **do** :

$z_j \leftarrow z_j \cup \{z' \oplus \tau'\}$

$m_j \leftarrow \mathsf{H}_2(z_j)$

reject if $\mathsf{Vf}(pk, j\|m_j\|g^k, \sigma_j) \neq 1$

$\widetilde{z} \leftarrow \beta^{1/r}$

if $\widetilde{z} \in z_j$, **return** match

else return none

Online server upon $(k, pk, \mathsf{Z}, \Sigma)$

$\beta \leftarrow \alpha^k$

$\pi \leftarrow \mathsf{NIZK}(g, g^k, \alpha, \beta)$

$y \leftarrow\$ \mathbb{Z}_q, Y \leftarrow g^y$

$\tau \leftarrow \mathsf{H}_3(j||\alpha||\beta||X||Y||X^y)$

for $z \in z_j$ **do** :

$z'_j \leftarrow z'_j \cup \{z \oplus \tau\}$

$(\beta, \pi, Y, z'_j, \sigma_j)$ ←

The update phase

Data owner upon $(sk, pk, k^{\mathsf{old}}, \mathsf{D}^{\mathsf{old}}, \mathsf{D}^{\mathsf{new}})$

$k^{\mathsf{new}} \leftarrow\$ \mathbb{Z}_q$

for $(u, pw) \in \mathsf{D}^{\mathsf{new}}$ **do** :

$j \leftarrow \mathsf{H}_0^{(l)}(u)$

$z_j \leftarrow z_j \cup \{\mathsf{H}_1(u\|pw)^{k^{\mathsf{new}}}\}$

for $z_j \in \mathsf{Z}^{\mathsf{new}}$, **do** :

$m_j \leftarrow \mathsf{H}_2(z_j)$

$\sigma_j \leftarrow \mathsf{Sig}(sk, j\|m_j\|g^{k^{\mathsf{new}}})$

record $\{\mathsf{Z}^{\mathsf{new}}, \Sigma^{\mathsf{new}}\}$

$(k^{\mathsf{new}}/k^{\mathsf{old}}, \mathsf{Z}^{\mathsf{new}}, \Sigma^{\mathsf{new}})$ →

Online server upon $(pk, k^{\mathsf{old}}, \mathsf{Z}^{\mathsf{old}}, \Sigma^{\mathsf{old}})$

for $z \in \mathsf{Z}^{\mathsf{old}}$ **do** :

$z_j \leftarrow z_j \cup \{z^{k^{\mathsf{new}}/k^{\mathsf{old}}}\}$

for $(z_j, \sigma_j) \in \{\mathsf{Z}^{\mathsf{new}}, \Sigma^{\mathsf{new}}\}$, **do** :

$m_j \leftarrow \mathsf{H}_2(z_j)$

reject if $\mathsf{Vf}(pk, j\|m_j\|g^{k^{\mathsf{new}}}, \sigma_j) \neq 1$

record $\{\mathsf{Z}^{\mathsf{new}}, \Sigma^{\mathsf{new}}\}$

Fig. 3. Details of our HIRBP. It consists of four phases: Setup, Upload, Query, and Update, satisfies all 11 desirable properties in Sec. III-D.

client computes the hash value of all OPRF values in the bucket z_j (denoted by m_j), and verifies the signature: $1/0 \leftarrow \mathsf{Vf}(pk, j\|m_j\|K, \sigma_j)$.

Step Q6. Only when both π and the signature σ_j are verified, the client completes the OPRF computation $\widetilde{z} = \beta^{1/r} = \mathsf{H}_1(u\|pw)^k$, and checks if $\widetilde{z} \in z_j$. If so, the client learns $(u, pw) \in \mathsf{D}$. Otherwise, $(u, pw) \notin \mathsf{D}$.

Update. The two participants are the data owner D (with input $(sk, pk, k^{\mathsf{old}}, \mathsf{D}^{\mathsf{old}}, \mathsf{D}^{\mathsf{new}})$) and the online server S (with input $(pk, k^{\mathsf{old}}, \mathsf{Z}^{\mathsf{old}}, \Sigma^{\mathsf{old}})$). HIRBP supports two types of updates: (1) updating the OPRF key k; and (2) updating the OPRF key while adding new leaks. When a malicious server provides an old bucket to $\mathcal{C}$, the verification will fail in both cases. Because the old signature is a signature of the old key $K^{\mathsf{old}} = g^{k^{\mathsf{old}}}$ and the old bucket, but the client verifies the new key $K^{\mathsf{new}} = g^{k^{\mathsf{new}}}$ and the old bucket.

Case 1. $D \Rightarrow S$: $\{k^{\mathsf{new}}/k^{\mathsf{old}}, \Sigma^{\mathsf{new}}\}$. The data owner first selects a new OPRF key $k^{\mathsf{new}} \leftarrow\$ \mathbb{Z}_q$, then re-signs all the buckets, and sends $\{k^{\mathsf{new}}/k^{\mathsf{old}}, \Sigma^{\mathsf{new}}\}$ to the

online server S. After receiving the message, S computes $\mathsf{H}_1(u\|pw)^{k^{\mathsf{new}}} \leftarrow (\mathsf{H}_1(u\|pw)^{k^{\mathsf{old}}})^{k^{\mathsf{new}}/k^{\mathsf{old}}}$ for each OPRF value $\mathsf{H}_1(u\|pw)^{k^{\mathsf{old}}} \in \mathsf{Z}^{\mathsf{old}}$, and verifies the signature of each bucket.

Case 2. $D \Rightarrow S : \{k^{\mathsf{new}}/k^{\mathsf{old}}, \mathsf{Z}^{\mathsf{new}}, \Sigma^{\mathsf{new}}\}$. The data owner D first selects a new OPRF key $k^{\mathsf{new}} \leftarrow_{\$} \mathbb{Z}_q$. D applies OPRF to new leaked credentials, re-signs the buckets, and sends $\{k^{\mathsf{new}}/k^{\mathsf{old}}, \mathsf{Z}^{\mathsf{new}}, \Sigma^{\mathsf{new}}\}$ to the online server S. Then S computes $\mathsf{H}_1(u\|pw)^{k^{\mathsf{new}}} \leftarrow (\mathsf{H}_1(u\|pw)^{k^{\mathsf{old}}})^{k^{\mathsf{new}}/k^{\mathsf{old}}}$ for $\mathsf{H}_1(u\|pw)^{k^{\mathsf{old}}} \in \mathsf{Z}^{\mathsf{old}}$, and verifies the signature of each bucket: $1/0 \leftarrow \mathsf{Vf}(pk, j\|m_j\|K, \sigma_j)$. The core of the update phase is that the credentials in $\mathsf{Z}^{\mathsf{old}}$ are *updated by the online server*, and *signed by the data owner*. The new leaked credentials are updated and signed by D.

Remark 2: When the password breach is small enough, it is possible that some buckets may be empty. In this case, the data owner still needs to send signatures on the empty buckets. Otherwise the online server can lie about the existence of a bucket and there is no way for the client to verify its emptiness. Moreover, we specify that the breached passwords D (as well as the updated $\mathsf{D}^{\mathsf{new}}$) should not contain duplicates, i.e., the data owner should check if a credential is already present in the bucket before adding it.

It should be noted that our protocol primarily focuses on the integrity *verification of credential data after its entry into the system*, rather than guaranteeing the completeness of breach-data collection before data entry. In other words, once the data owner has generated and signed the buckets, the online server cannot undetectably delete, replace, or modify bucket entries. However, if the data owner deliberately excludes some records before generating the signed database, this belongs to the data collection and commitment phase and is outside the cryptographic guarantee of our HIRBP. *This limitation is inherent to all existing C3 services (e.g., [13], [14], [15], [18], [19], [20], [21], [22]) that rely on externally collected breach data*. Practical deployments may mitigate this risk by introducing complementary mechanisms, such as third-party audits, public transparency logs, signed update logs, multi-source breach-data aggregation, or consistency checks.

C. Security Analysis

We now show that our HIRBP resists the three types of attackers defined in Sec. III-B. The security of HIRBP is based on two assumptions: The decisional Diffie-Hellman (DDH) problem is hard in the group $\mathbb{G}$, and the cryptographic hash functions $\mathsf{H}_i (i = 0, 1, 2, 3)$ are modeled as random oracles.

Attacker $\mathcal{A}_1$. Any C3 service (and our HIRBP) cannot be absolutely immune to breach extraction attacks from malicious clients. But HIRBP can deploy client rate limiting measures, such as IP-address-based query throttling, slow-to-compute hash functions (e.g., instantiate H_0, H_1 or H_3 with Argon2). $\mathcal{A}_1$ can not perform offline guessing on previously transmitted passwords upon OPRF compromise. Because passwords are protected by the long-term key and the short-term key. This feature is *not* available in existing C3 protocols [14], [18].

Attacker $\mathcal{A}_2$. A malicious online server learns only the bucket identifier sent by the client, i.e., a l-bit hash-prefix of the client's username, but nothing further. Under the known-username assumption, bucket identifiers do not reveal new information to $\mathcal{A}_2$. Moreover, assuming that the password and username are independent [14], [18], $\mathcal{A}_2$ knows nothing about the queried password. We use the 1.4 billion password dataset[3] to *quantitatively* evaluate the success rate of online guessing attacks by $\mathcal{A}_2$ (see Table V), and the success rate of breach extraction attacks by $\mathcal{A}_1$ (see Table VI).

TABLE IV

PERFORMANCE COMPARISON OF IDB [18] AND OUR HIRBP. THE BUCKET IDENTIFIERS OF IDB AND HIRBP ARE 20 BITS OF SHA256 ($u\|pw$) AND 20 BITS OF SHA256(u), RESPECTIVELY. IF ARGON2ID (WITH THE PARAMETERS $m = 65536, t = 3, p = 4$) [51] IS EMPLOYED, THERE IS AN OBSERVED INCREASE IN THE TOTAL RUNNING TIME OF 70.71 MS, THE OVERHEAD IS PRIMARILY INCURRED DURING THE CLIENT'S BUCKET IDENTIFIER COMPUTATION $j = H_0^{(l)}(u)$. IN OUR MEASUREMENT, THE CLIENT 'S PREPARATION TIME INCREASES FROM 0.48 MS (WITH SHA-256) TO 68.52 MS (WITH ARGON2ID) FOR COMPUTING j, WHILE THE REMAINING 2.19 MS COMES FROM THE ADDITIONAL ARGON2ID CALLS FOR τ AND τ'. THE SERVER SIDE IS UNAFFECTED BECAUSE IT DOES NOT EVALUATE THESE SLOW HASHES

Scheme		IDB [18]	Our HIRBP
Client	Preparation	0.26 ms	0.48 ms
	Query	116.50 ms	111.38 ms
	Computation	0.36 ms	25.05 ms
	Total running time	117.12 ms	**136.91** ms
Server	Computation	115.73 ms	124.85 ms
	Response	1.15 ms	0.37 ms
	Total running time	116.88 ms	**125.22** ms
Client side bandwidth		69 bytes	93 bytes
Server side bandwidth		85,391 bytes	85,711 bytes

Attacker A_3. The client either obtains a reliable result, or detects tampering and securely aborts. Because we employ the digital signature and NIZK proof to verify the correctness of buckets and the consistency of OPRF keys. Therefore, the client can detect malicious parties, such as a network adversary, or a malicious data owner/online server.

Theorem 1: Let $\mathbb{G}$ be a representative group and let $\mathcal{P}$ be the proposed HIRBP stated in Sec. IV-B. Let $\mathcal{A}_3$ be an adversary against the verifiability of query results within a time bound t, with less than q_{send} active participation, and making less than q_h random oracle queries. We have

$$\mathsf{Adv}^{\mathsf{ver}}_{\mathcal{P}}(\mathcal{A}_3) \leq q_{send}(\mathsf{Adv}^{DDH}_{\mathcal{P}}(t) + \varepsilon) + \frac{q_h^2}{2^{\lambda}} + \frac{{q_h}^2}{2p}, \quad (3)$$

where λ is the output length of $\mathsf{H}_2(\cdot)$, p is the prime order of the group $\mathbb{G}$, $\varepsilon = \varepsilon_1 + \varepsilon_2$, ε_1 and ε_2 are negligible when reducing the adversary's advantage in the zero-knowledge proof scheme [33] and the signature scheme [32].

Proof: Let $\mathcal{A}_3$ denote a probabilistic polynomial-time attacker attempting to breach the verifiability of query results in $\mathcal{P}$. The main idea is to construct an attacker who breaks the underlying cryptographic primitives. If $\mathcal{A}_3$ can break the verifiability of $\mathcal{P}$, then at least one of the primitives is broken. We define a sequence of games starting at real attack Game_0 and ending up with Game_7, where the advantage of $\mathcal{A}_3$ is 0.

To prove Theorem 1, we bound the advantage of $\mathcal{A}_3$ by the interaction between $\mathcal{A}_3$ and honest participants. The detailed

[3]We use *dataset* (instead of *database*) in Sec. V, Appendices D and E, available at https://tinyurl.com/VerC3-Mi-Supplemental-Material. Because, in our experiments, the server's database is simulated by the test set and does not contain all the 1.4 billion (username, password) pairs.

TABLE V

ONLINE GUESSING ATTACK SUCCESS RATES GIVEN DIFFERENT GUESSING BUDGETS (Q) ON DIFFERENT C3 SETTINGS. THE RESULTS SHOW THAT THE SECURITY LOSS OF PREVIOUSLY UNCOMPROMISED USERS IS LARGER: SPECIFICALLY, THE PASSWORD CRACKING RATE OF UNCOMPROMISED USERS INCREASES BY 10.54(=(75.60-6.55)/6.55) TIMES, WHILE THE PASSWORD CRACKING RATE OF COMPROMISED USERS INCREASES BY 0.68(=(86.17-51.18)/51.18) TIMES

Guessing strategy	**Leakage**†	**Uncompromised**				**Guessing strategy**	**Leakage**†	**Compromised**			
		$q=1$	$q=10$	$q=10^2$	$q=10^3$			$q=1$	$q=10$	$q=10^2$	$q=10^3$
Markov [47] (used in this work and CCS'19 [18])	Nothing	0.64 (±0.06)	1.47 (±0.17)	2.88 (±0.18)	6.12 (±0.63)	Pass2Path [59] (used in this work and CCS'19 [18])	Nothing	40.04 (±0.71)	45.45 (±0.72)	47.54 (±0.77)	49.04 (±0.76)
	16 bits	16.36 (±0.33)	30.96 (±0.34)	45.19 (±0.45)	58.62 (±0.60)		16 bits	57.79 (±0.42)	65.26 (±0.57)	71.70 (±0.81)	78.58 (±0.71)
	20 bits	30.88 (±0.41)	47.78 (±0.36)	61.22 (±0.55)	74.16 (±0.17)		20 bits	65.24 (±0.52)	73.10 (±0.84)	79.94 (±0.60)	86.74 (±0.53)
TarGuess-I [5] (only this work)	Nothing	0.52 (±0.16)	1.64 (±0.20)	3.09 (±0.26)	6.55 (±0.60)	TarGuess-II [5] (only this work)	Nothing	40.04 (±0.71)	45.56 (±0.72)	48.35 (±0.66)	51.18 (±0.59)
	16 bits	17.35 (±0.33)	32.94 (±0.54)	46.96 (±0.56)	60.13 (±0.62)		16 bits	57.72 (±0.36)	64.22 (±0.51)	70.74 (±0.83)	77.78 (±0.69)
	20 bits	31.73 (±0.44)	49.49 (±0.43)	62.80 (±0.67)	75.60 (±0.34)		20 bits	64.30 (±0.49)	72.15 (±0.83)	79.16 (±0.60)	86.17 (±0.49)

† *Nothing* means no password-related information is leaked to the attacker. *16 bits* means the attacker knows the first 16 bits of SHA256($u\|pw$), where u is the username, pw is the password. *20 bits* means the attacker knows the first 20 bits of SHA256($u\|pw$). The value with background color represents the success rate of online guessing attacks using Targuess-I/-II models when $q = 10^3$.

TABLE VI

BREACH EXTRACTION ATTACK SUCCESS RATES GIVEN DIFFERENT QUERY BUDGETS (q) ON DIFFERENT C3 SETTINGS. THE RESULTS SHOW THAT BLOCKING MECHANISMS CAN EFFECTIVELY MITIGATE SUCH ATTACKS: THE SUCCESS RATE IS 0.31% WHEN A BLOCKLIST OF SIZE 10^4 IS USED IN THE UNCOMPROMISED SETTING, BUT 6.16% WHEN NO BLOCKLIST IS DEPLOYED. BLOCKING MECHANISMS ARE ALSO EFFECTIVE IN THE COMPROMISED SETTING: 13.18% VS. 17.35%

Blocklist	**Uncompromised** (tested in USENIX SEC'22 [14])				**Blocklist**	**Compromised** (only this work)			
	$q=1$	$q=10$	$q=10^2$	$q=10^3$		$q=1$	$q=10$	$q=10^2$	$q=10^3$
Blocklist not used	0.70 (±0.04)	1.54 (±0.09)	3.07 (±0.15)	6.16 (±0.11)	Blocklist not used	5.69 (±0.09)	9.96 (±0.14)	13.79 (±0.05)	17.35 (±0.06)
Block top-10	0.05 (±0.02)	0.35 (±0.01)	1.43 (±0.02)	4.45 (±0.10)	Block top-10	5.85 (±0.14)	9.67 (±0.14)	12.81 (±0.21)	16.34 (±0.21)
Block top-10^2	<0.01 (±0.00)	0.06 (±0.01)	0.67 (±0.02)	3.33 (±0.06)	Block top-10^2	5.65 (±0.09)	9.53 (±0.15)	12.36 (±0.22)	15.38 (±0.27)
Block top-10^3	<0.01 (±0.00)	0.02 (±0.00)	0.18 (±0.02)	1.57 (±0.05)	Block top-10^3	5.34 (±0.09)	9.20 (±0.13)	12.05 (±0.13)	13.81 (±0.11)
Block top-10^4	<0.01 (±0.00)	<0.01 (±0.00)	0.02 (±0.01)	0.31 (±0.03)	Block top-10^4	4.93 (±0.07)	8.84 (±0.06)	11.60 (±0.08)	13.18 (±0.08)

† *Block top-n* means the server uses a blocklist of size $n \in \{10, 10^2, 10^3, 10^4\}$ to filter weak passwords. Hence, the server returns none for weak passwords. Background-colored values denote the success rate of breach extraction attacks under $q = 10^3$ query budgets.

proof can be found in Appendix B at https://tinyurl.com/VerC3-Mi-Supplemental-Material.

Theorem 2 (Impossibility of Single-Server Verifiability): Let Π be a single-server C3 protocol where the client $\mathcal{C}$ holds a query (u, pw) and the server $\mathcal{S}$ holds a database D of breached credentials. Suppose $\mathcal{C}$ receives a response R from $\mathcal{S}$ and outputs a decision $d \in$ {match, none, $\perp$} (where $\perp$ indicates rejection). Assume that 1) $\mathcal{C}$ has no independent source of information about D other than R and the public parameters; 2) $\mathcal{S}$ is polynomial-time bounded; and 3) the protocol does not rely on an external trusted party or a secure out-of-band channel for transmitting a digest of D. Then there exists a malicious $\mathcal{S}^*$ that can produce a response R' such that $\mathcal{C}$ cannot distinguish R' from an honest response, even though R' is incorrect with respect to the true D.

Proof: Consider an honest execution where $(u, pw) \in \mathsf{D}$ and $\mathcal{S}$ returns the correct response R_{true} (indicating match). Now construct a malicious $\mathcal{S}^*$ that behaves identically to the honest $\mathcal{S}$ except that it internally replaces D with a modified database $\mathsf{D}' = \mathsf{D} \setminus \{(u, pw)\}$ (i.e., it omits the queried credential). $\mathcal{S}^*$ executes the protocol exactly as prescribed, using D' to generate all messages. Since $\mathcal{C}$ has no independent information about D (by assumption (1)) and no out-of-band digest to compare against, the view of $\mathcal{C}$ in the execution with $\mathcal{S}^*$ is computationally indistinguishable from its view in an honest execution where $(u, pw) \notin \mathsf{D}$. This holds even if $\mathcal{S}^*$ uses cryptographic proofs, because any proof must be generated based on D', and the client has no means to verify that D' equals the true D. The only way to break this indistinguishability is to provide the client with a trusted, independent digest of D (e.g., a Merkle root obtained via a secure channel) or to involve multiple non-colluding servers.

This theorem formalizes the conclusion given in Sec. II-B: Without an additional trusted party or out-of-band channel, a malicious server can respond using an incomplete database, while the client cannot determine whether the queried credentials are intentionally excluded. This holds regardless of the cryptographic primitives used (e.g., OPRF, zero-knowledge proofs, authenticated data structures, etc.), because those primitives only guarantee consistency with a committed database, not that the committed database is complete or correct. This result motivates our two-server VerC3 framework, where the non-colluding data owner $\mathcal{D}$ acts as the trusted source.

V. EVALUATION

In this section, we compare the fulfillment of the property set, and analyze the performance and security of our HIRBP.

A. Security Comparison

Table III summarizes our observations based on the property set. We explain how our HIRBP fulfills each property.

- P1. *Democratized access.* Any user can use the HIRBP service without registration or the trust between parties.
- P2. *Near real-time.* Our HIRBP only takes 136.91 ms to complete a query on a common PC. This is comparable to the time of existing schemes (see Table IV).
- P3. *Actionable and breached.* Our HIRBP provides accurate and actionable security advice. If the credential is leaked, users can receive a reliable warning.

P4. *Requester credential anonymity.* We only transmit a hash-prefix of the username and the oblivious pseudo-random function (OPRF) value of the queried (username, password)-pair. No deterministic information related to the password is transmitted in our HIRBP.
P5. *Query result privacy.* Our HIRBP ensures that only those participating in the query phase know the OPRF value $F_k(u\|pw)$ and the ephemeral Diffie-Hellman key g^{xy}. Eavesdroppers do not know the query result.
P6. *Query result verifiability.* Our HIRBP satisfies the verifiability of query results in the two-server setting. Existing schemes (like GPC [13] and IDB [21]) assume that the server is fully trusted and miss this property.
P7. *Resistance to DoS attacks.* Our HIRBP server response time is less than the client query time (i.e., 124 ms < 136 ms, see Table IV). Thus, HIRBP satisfies this property.
P8. *Inefficient oracle.* Our HIRBP can deploy rate-limiting measures, e.g., instantiating H_0, H_1 or H_3 with Argon2. Thus, HIRBP will not be abused by attackers as a pragmatic oracle to obtain breached sensitive credentials.
P9. *Responses with bounded leakage.* The client knows nothing about the server except the query result, the size of the bucket, and the OPRF values in the bucket.
P10. *Forward secrecy.* Our HIRBP is *the only one* that satisfies P10. The entries in the bucket are $F_k(u\|pw)\oplus\mathsf{H}_3(j\|\alpha\|\beta\|g^x\|g^y\|g^{xy})$. If the OPRF key is compromised, $\mathcal{A}$ can compute $F_k(u\|pw')$, where pw' is a candidate password from the password dictionary. However, for queries without the active participation of attackers, because the ephemeral Diffie-Hellman key g^{xy} is unknown, attackers cannot verify their guesses. Meanwhile, for each client query, τ is freshly generated and never reused.
P11. *Resistance to pre-computation attacks.* Our HIRBP prevents pre-computation attacks by keeping the salt private, as in OPAQUE [44].

We will explain the unsatisfied states $\times$ and $-$ (can't meet the desired properties) in Table III.

MIGP [14] and MIGP 2.0 [22]. The compromise of the OPRF key in [14], [22] will result in password cracking of previously transmitted passwords. So, they fail to meet P10.

Pipa [21]. The server stores plaintext passwords, which will be immediately leaked upon server compromise (P11).

WR20-Cuckoo [57]. The server uses honeywords generated for *registered* users (instead of any user) to detect a compromise in its database. Therefore, P1 is not satisfied. The client latency of WR20-Cuckoo [57] is approximately 38 seconds, and the API calls occupy most of the time overhead [14]. Thus, P2 and P7 are not satisfied. Moreover, P5 is not satisfied because the server ultimately learns the breach status [57]. P10 is not satisfied because a compromise of the long-term key results in an offline password guessing attack on previously transmitted passwords.

WR19-Bloom [20]. It is initially used to detect whether an abnormal login is a credential stuffing attack, providing services for *registered* users. Therefore, P1 is not satisfied. The client latency of WR19-Bloom [20] is approximately 46 seconds, and the API calls occupy most of the time overhead [14]. Thus, P2 and P7 are not satisfied. The server stores passwords in plain text, which means it fails to meet P11.

IDB [18]. It is an OPRF-based C3 protocol. The compromise of the OPRF key leads to the cracking of previously transmitted passwords. Therefore, IDB does not satisfy P10.

FSB [18]. The server does not use slow hash functions to limit the query capabilities of clients. FSB might be abused by attackers as a pragmatic oracle to obtain breached credentials (P8). Each entry in the bucket is a hash value of a password and a salt. This design allows attackers to perform brute force cracking to recover those passwords. Thus, the server leaks too much information about passwords (P9) and can not resist pre-computation attacks (P11).

GPC [13]. The client sends the hash-prefix of the password, which increase the success rate of online guessing attacks against the password. Thus, P4 is not satisfied. Compromised long-term keys in GPC may result in password cracking of previously transmitted passwords (P10).

EnZoic [19]. Requester credential anonymity (P4) is not satisfied because the bucket identifier is deterministic password-related information. Each entry in the bucket is the hash of a password and a specific salt. It slows down the password cracking. Thus, Enzoic partially satisfies P9 and P11.

HIBP [15]. Like EnZoic [19], the bucket identifier in HIBP [15] leaks information related to the queried password (P4). In addition, the bucket leaks information related to breached passwords. Thus, HIBP [15] can't satisfy P9. HIBP [15] does not use slow hash functions such as Argon2 to limit the query capabilities of clients, thereby not satisfying P8. HIBP [15] stores hashes of a password and a public salt, which cannot resist pre-computation attacks (P11).

HIBP-UN [15]. The client sends its username, leading to a lack of anonymity (P4). Moreover, the server directly returns a breach status to the client. The eavesdropper can obtain the query results. Thus, HIBP-UN [15] does not satisfy P5.

B. Performance Comparison

We implement the HIRBP protocol in Python 3.9. We use Curve25519's hash-to-point function to map the (username, password) to $\mathbb{G}$. We use $\mathsf{H}_i(x) = \text{SHA256}(i\|x)$ to instantiate H_i ($i = 0, 2, 3$), and the NaCl library [58] to implement signatures. The client runs on a laptop equipped with Intel(R) Core(TM) i5-1135G7 @ 2.40GHz, 16.0 GB RAM, 64-bit Windows 10. The data owner and the server run on two separate machines, with Intel(R) Xeon(R) Gold 6226R CPU@2.90GHz $\times$32 and 256 GB RAM, 64-bit Ubuntu 22.04.1.

We partition 1.4 billion leaked password records based on the first 20 bits of SHA256(username). The number of buckets is 2^{20}. On average, each bucket contains 1,399 entries. The distribution is shown in Fig. 4. The largest bucket (the prefix is 0xbfb3f) contains 338,221 entries, followed by 120,071 entries and 59,435 entries. The smallest bucket contains 1,118 entries. 98.65% of buckets contain 1,200 $\sim$ 1,600 entries. 3,593 buckets contain more than $2,000$ entries, 54 buckets contain more than 10^4 entries. When a bucket exceeds the predefined threshold, the data owner can split it using additional hash bits or a secondary hash function, and sign each sub-bucket separately. The client then verifies the corresponding sub-bucket signature in the same way as in our HIRBP protocol.

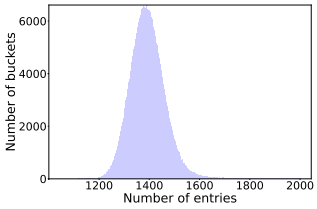


Fig. 4. Bucket entry distribution.

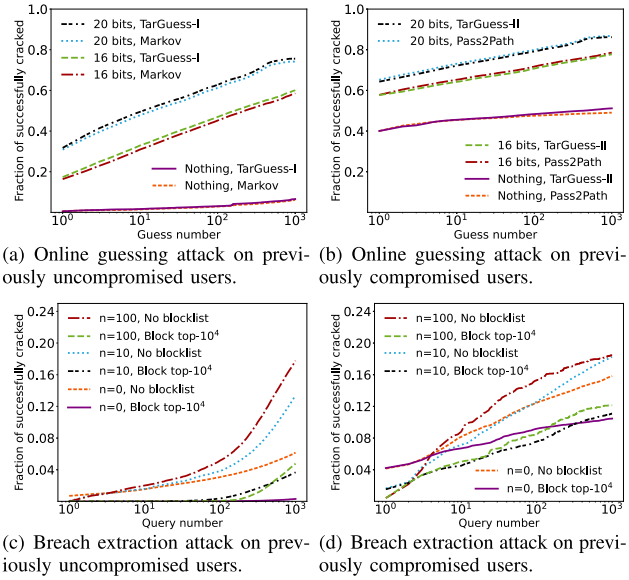
We re-implement the IDB protocol [18] in the same way. IDB [18] is selected for comparison for two reasons: (1) Our HIRBP is an updated version of a series of OPRF-based C3 services, including IDB [18], MIGP [14], and MIGP 2.0 [22]. (2) The structure of MIGP/MIGP2.0 and IDB [18] is essentially the same, except for checking similar passwords. Details on extending our HIRBP to similarity-aware C3 services (e.g., MIGP [14] and MIGP 2.0 [22]) are deferred to Sec. VI. To provide a comprehensive evaluation, we conduct a comparison with the IDB system [18], as presented in Table IV. The running time and bandwidth are averaged over 10^4 queries.

Upload. We use the same 1.4 billion breached password records to simulate IDB [18] and our HIRBP, and parallelize the total password records. We do not use any GPU to optimize hash computation. The pre-processing time of 100 million (username, password) pairs for HIRBP and IDB is 4.78 hours (4.26 hours for applying OPRF and 0.52 hours for signing) and 4.31 hours, respectively. It takes 2.81 minutes for the HIRBP online server to verify all signatures.

Query. As shown in Table IV, it only takes 125.22 ms for the HIRBP server to create a response, and 136.91 ms for the client to get its query result on a common PC. Our HIRBP achieves the verifiability property and forward secrecy at the cost of tiny computational and communication overhead.

Update. We update nine million newly leaked credentials, which is the average value of new leaks added by HIBP [15]. It takes 4.94 hours for the data owner: 0.49 hours for applying OPRF to new leaks, 4.24 hours for raising the exponent on old leaks, and 12.45 minutes for signing, 2.82 minutes for the server to verify signatures.

Our update experiment should be interpreted as a prototype baseline rather than the result of a fully optimized large-scale deployment. For databases containing tens of billions of records, a naive full update would incur substantial computation and communication costs. However, the update phase is highly parallelizable: different buckets or shards can be updated independently, and the exponentiation of old OPRF values by k^{new}/k^{old} can be distributed across multiple cores or machines. In practice, the data owner can also perform offline batching, where newly collected leaks are accumulated, processed, signed, and released periodically. Incremental updates can be used for newly added records, while full key rotation can be scheduled less frequently according to the desired security and freshness requirements. Moreover, the data owner and online server can transmit only update increments and new signatures when possible, and can be connected through high-speed dedicated links or nearby cloud regions while remaining in different administrative domains. These optimizations reduce update time and network delay without changing the verifiability of our HIRBP.



(a) Online guessing attack on previously uncompromised users.

(b) Online guessing attack on previously compromised users.

(c) Breach extraction attack on previously uncompromised users.

(d) Breach extraction attack on previously compromised users.

Fig. 5. Experimental results of online guessing attacks and breach extraction attacks based on 1.4 billion (u, pw) pairs. Each attack is tested on previously uncompromised users and compromised users.

C. Experimental Comparison

We quantitatively evaluate online password guessing attacks and breach extraction attacks against our HIRBP, corresponding to the attackers in Sec. III-B. Our experiment is more comprehensive than that of Li et al. [18] and Pal et al. [14]: (1) In terms of online guessing attacks, we add a stronger attacker who knows users' personally identifiable information (PII) like birthday. (2) In terms of breach extraction attacks, we add previously compromised users as targets (while Pal et al. [14] only consider previously uncompromised users)[4]

- For a previously uncompromised user u, the attacker's password dictionary does not contain any entry $(u, *)$.
- For a previously compromised user u, the attacker's password dictionary contains one or more (u, pw_i).

We use the guess-number-graph to evaluate these two attacks(see Fig. 5). To accurately show the success rate, we give the concrete results at four guess numbers (i.e., 1, 10, 10^2, 10^3) for all attack scenarios, as shown in Tables V and VI. We repeat each experiment five times and figure out the averages. For detailed information on our datasets and guessing strategies, see Appendix D and Appendix E, available at https://tinyurl.com/VerC3-Mi-Supplemental-Material.

1) Experimental Setup of Online Guessing Attacks: We reproduce the experiment of CCS'19 [18] based on Markov [47] and Pass2Path [59] to confirm their results. Namely, transmitting the hash-prefix of the password will offer an advantage for online password guessing attacks. To clearly show the impact of uncompromised and compromised settings on the attack effectiveness, we extend CCS'19 [18] with the TarGuess-I [5] and TarGuess-II models, both of them are leading targeted guessing models.

[4]Pal et al. [14] state "As per our data division, none of the target usernames are present in S_2, and therefore the attacker cannot attempt a targeted credential tweaking-type attack". It implies that there is no target user's (username, password)-pair in the attacker's password dictionary.

2) Experimental Results of Online Guessing Attacks: Three conclusions can be drawn from Table V and Figs. 5(a)~5(b).

Knowing the hash-prefix of the password improves the efficacy of online guessing attacks (see Table V). Our HIRBP demonstrates robust resistance against the attacker (who is with the hash-prefix of the password): In the uncompromised setting, within 10^3 guesses, the success rates against our HIRBP are 6.12%~6.55%, while that of GPC [13], HIBP [15], and EnZoic [19] are 74.16%~75.60%.

The security loss of previously uncompromised users is larger (as demonstrated by TarGuess-I/TarGuess-II results). With 20 bits of SHA256($u||pw$), the attacker cracks 75.60% of uncompromised users and 86.17% of compromised users. It is an increase of 10.54(=(75.60-6.55)/6.55) times and 0.68(=(86.17-51.18)/51.18) times, respectively.

Knowledge of deterministic password-related information determines the success rate of online guessing attacks. Fig. 5(a) shows that the lines of Markov and TarGuess-I closely align in the same leakage (e.g., 20 bits). Fig. 5(b) shows that the lines of Pass2Path and TarGuess-II closely align. All these reveal that the user's personally identifiable information (e.g., the birthday) does not significantly increase the success rate, and the knowledge of deterministic password-related information is the key influencing factor.

3) Experimental Setup of Breach Extraction Attacks: We adopt the blocklist mechanism (i.e., $n = 10, 10^2, 10^3, 10^4$) recommended in Pal et al. [14]. Since new leaks gathered by the server may come from previously compromised users, we add an experiment under the *compromised* setting. It is possible that a user's previously leaked password is available to the attacker (and the server), but its newly leaked password is only accessible to the server. These passwords are also potential targets, but their security is totally unexplored. We try to bridge this gap through empirical experiments.

4) Experimental Results of Breach Extraction Attacks: In the attack on uncompromised users in Fig. 5(c), the three lines of *block top*-10^4 are all below the lines that do not use a blocklist; In the attack on compromised users in Fig. 5(d), with the guess number ≥ 10, the three lines of *block top*-10^4 are all below the lines that do not use a blocklist. In Table VI, in the compromised setting, the success rates of our HIRBP are 4.93%~13.18% when a 10^4 blocklist is used, while this value is 5.69%~17.35% when no blocklist is used. It indicates that blocking mechanisms can well help both compromised and uncompromised users mitigate breach extraction attacks.

The experiments in this section are directly connected to the adversary models defined in Sec. III-B. The online guessing attack in Table V correspond to the malicious online server $\mathcal{A}_2$, which attempts to infer the client's queried password from the information leaked during the query. The breach extraction attack in Table VI correspond to the malicious client $\mathcal{A}_1$, which abuses the C3 service as an online oracle to extract breached credentials from the server. The third adversary $\mathcal{A}_3$, which attempts to manipulate query results, is addressed by the verifiability mechanism [60] of HIRBP and formally analyzed in Theorem 1. Therefore, Sec. V-C empirically evaluates $\mathcal{A}_1$ and $\mathcal{A}_2$, while $\mathcal{A}_3$ is covered by the formal security analysis.

VI. Discussions of Our Work

A. Extension to the Similarity-Aware C3 Service

A similarity-aware C3 service warns users if they select passwords similar or identical to a breached password [14], [22]. We employ Pal et al. [14] password-generating function, denoted as $\tau_n(\cdot)$, to generate n semantically similar passwords for each breached password. Then, we use Pasquini et al.'s method [22] to handle the τ-collision. A τ-collision occurs when τ outputs the same password for different inputs. Take the upload phase as an example, for every entry (u, pw) in the bucket, the data owner computes variants $\{\tilde{pw}_1, \ldots, \tilde{pw}_n\} \leftarrow \tau_n(pw)$ and OPRF values $\{F_k(u||pw), F_k(u||\tilde{pw}_1)\oplus 1, \ldots, F_k(u||\tilde{pw}_n)\oplus 1\}$. pw is chosen from the password space that follows a Zipf's law [61]. If τ produces a collision, then the data owner replaces it with a dummy entry $F_k(u||v)$, where v is a random λ-bit long string. The data owner signs the hash of each bucket and uploads all the buckets and signatures to the online server.

B. Verifiability in Data Collection

It should be noted that the C3 service database in ours may not encompass all leaked passwords, which is an inherent limitation in all C3 services (e.g., [13], [14], [18]). There are definitely a lot of leaks out there that are only available to smaller circles or have not been integrated into a C3 service. Our protocol primarily focuses on the integrity verification of credential data after entry into the system. To address these concerns, potential enhancements like stricter data validation processes, third-party audits [46] can be adopted, to increase data transparency and reliability, fortify the integrity of data handling processes and mitigate risks associated with data manipulation during the data collection phase.

C. Extension to the Multi-Data Owner Setting

In such a case, multiple independent, mutually distrustful data owners submit breached credentials to a single server, the results are still verifiable. We clarify that simply extending HIRBP does not solve this problem. Because if n data owners share one OPRF key, a malicious data owner can perform the pre-computation attack on other data owners. If n data owners have n independent OPRF keys, there will be n zero-knowledge proofs in the query phase, increasing computational overhead. We leave further exploration as future work.

D. Design Challenges and Limitations

Implementing a VerC3 service involves several challenges. First, the main difficulty is verifying negative query results: a malicious server may omit the queried credential from the returned bucket and falsely return "none". Our HIRBP addresses this by letting the data owner sign each bucket, so that any deletion, replacement, or modification can be detected by the client. Second, verifiability must be achieved without exposing the client's queried credential or the plaintext breach records. Therefore, our HIRBP follows an OPRF-based design and adds lightweight verification mechanisms rather than relying on generic verifiable PSI/PIR or zero-knowledge set techniques. Third, the protocol must ensure OPRF-key consistency between preprocessing and query phases. Our

HIRBP uses a NIZK proof to bind the response $\beta = \alpha^k$ to the public key $K = g^k$. Fourth, the service must support dynamic updates and key rotation. We choose HashDH because old OPRF values can be updated by exponentiation using k^{new}/k^{old}, without revealing plaintext credentials to the online server. Finally, HIRBP should remain efficient for large-scale deployment, so we use bucket signatures and lightweight NIZK proofs instead of more expensive general-purpose verifiable computation [60].

VII. Conclusion

We, *for the first time*, tackle the question of how to verify the query results of compromised credential checking (C3) services, propose a *verifiable* C3 service (VerC3), provide a simple yet efficient construction (i.e., HIRBP). We demonstrate the effectiveness of HIRBP by formal security proofs and comprehensive attacking experiments. We further develop a prototype, and HIRBP maintains high efficiency: The response time is close to its foremost counterparts. Considering the prevalence and catastrophic impacts of password-file leakage, we believe that achieving active and verifiable C3 is of broad interest, and that our work takes an important step toward building more reliable C3 services.

References

[1] J. Bonneau, C. Herley, P. C. van Oorschot, and F. Stajano, "Passwords and the evolution of imperfect authentication," *Commun. ACM*, vol. 58, no. 7, pp. 78–87, Jun. 2015.

[2] L. H. Newman. (Jul. 2021). *Why the Password Isn't Dead Quite Yet*. [Online]. Available: https://arstechnica.com/information-technology/2021/07/why-the-password-isnt-dead-quite-yet/?comments=1

[3] P. A. Grassi et al. (Oct. 2025). *NIST SP 800-63B Digital Identity Guidelines: Authentication and Lifecycle Management*. [Online]. Available: https://pages.nist.gov/800-63-3/sp800-63b.html

[4] D. Wang, Y. Zou, Z. Zhang, and K. Xiu, "Password guessing using random forest," in *Proc. USENIX SEC*, 2023, pp. 965–982.

[5] D. Wang, Z. Zhang, P. Wang, J. Yan, and X. Huang, "Targeted online password guessing: An underestimated threat," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Oct. 2016, pp. 1242–1254.

[6] D. Pasquini, G. Ateniese, and C. Troncoso, "Universal neural-cracking-machines: Self-configurable password models from auxiliary data," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2024, pp. 1365–1384.

[7] J. van Beek and R. Gevers. (Jun. 2020). *Bcrypt Password Cracking Extremely Slow? Not If You Are Using Hundreds of FPGAs!*. [Online]. Available: https://scatteredsecrets.medium.com/bcrypt-password-cracking-extremely-slow-not-if-you-are-using-hundreds-of-fpgas-7ae42e3272f6

[8] V. Petkauskas. (Sep. 2024). *Mother of All Breaches Reveals 26 Billion Records: What We Know So Far*. [Online]. Available: https://cybernews.com/security/billions-passwords-credentials-leaked-mother-of-all-breaches/

[9] P. Okunyte. (Nov. 2023). *DarkBeam Leaks Billions of Email and Password Combinations*. [Online]. Available: https://cybernews.com/security/darkbeam-data-leak/

[10] L. H. Newman, (Oct. 2017). *Yahoo's 2013 Email Hack Actually Compromised Three Billion Accounts*. [Online]. Available: https://www.wired.com/story/yahoo-breach-three-billion-accounts/

[11] N. Saeed. (Sep. 2022). Top Insights From Our 2022 State of Secure Identity Report. [Online]. Available: https://auth0.com/blog/top-insights-from-our-2022-state-of-secure-identity-report/

[12] V. Enterprise. (Apr. 2025). 2026 Data Breach Investigations Report. [Online]. Available: https://www.verizon.com/business/resources/T158/reports/2026-dbir-data-breach-investigations-report.pdf

[13] K. Thomas et al., "Protecting accounts from credential stuffing with password breach alerting," in *Proc. USENIX SEC*, 2019, pp. 1556–1571.

[14] B. Pal et al., "Might I get pwned: A second generation compromised credential checking service," in *Proc. USENIX SEC*, 2022, pp. 1831–1848.

[15] T. Hunt. (Aug. 2025). *Have I Been Pwned?*. [Online]. Available: https://haveibeenpwned.com/

[16] R. Thubron. (Jan. 2024). *Massive Data Dump Containing Millions of Passwords Sparks Security Alert: Is Your Data Safe*?. [Online]. Available: https://www.techspot.com/news/101558-massive-data-dump-containing-millions-passwords-sparks-security.html

[17] S. Sahin and F. Li, "Don't forget the stuffing! Revisiting the security impact of typo-tolerant password authentication," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Nov. 2021, pp. 252–270.

[18] L. Li, B. Pal, J. Ali, N. Sullivan, R. Chatterjee, and T. Ristenpart, "Protocols for checking compromised credentials," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Nov. 2019, pp. 1387–1403.

[19] Enzoic. (Sep. 2025). *Detect Compromised Passwords Protect Employee Accounts Prevent Account Takeover*. [Online]. Available: https://www.enzoic.com/

[20] K. C. Wang and M. K. Reiter, "Detecting stuffing of a user's credentials at her own accounts," in *Proc. USENIX SEC*, 2020, pp. 2201–2218.

[21] J. Li, Y. Liu, and S. Wu, "Pipa: Privacy-preserving password checkup via homomorphic encryption," in *Proc. ACM Asia Conf. Comput. Commun. Secur.*, May 2021, pp. 242–251.

[22] D. Pasquini, D. Francati, G. Ateniese, and E. M. Kornaropoulos, "Breach extraction attacks: Exposing and addressing the leakage in second generation compromised credential checking services," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2024, pp. 1405–1423.

[23] K. C. Wang and M. K. Reiter, "How to end password reuse on the Web," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2019, pp. 1–15.

[24] F. Lambert. (May 2023). *Tesla Files: Insider Dropped 100 Gb of Data on German Media Outlet*. [Online]. Available: https://electrek.co/2023/05/25/tesla-files-insider-stole-100gb-data-media/

[25] M. Moore. (Jan. 2023). *Yandex Denies It Was Hacked, Says Rogue Employee to Blame for Breach*. [Online]. Available: https://www.techradar.com/news/yandex-denies-it-was-hacked-says-rogue-employee-to-blame-for-breach/

[26] G. Baran. (Mar. 2025). *Hacker Claims Sale of 6 Million Records Stolen From Oracle Cloud Servers*. [Online]. Available: https://cybersecuritynews.com/hacker-claims-6-million-oracle-records/

[27] R. Harroch. (Nov. 2018). *Data Privacy and Cybersecurity Issues in Mergers and Acquisitions*. [Online]. Available: https://www.forbes.com/sites/allbusiness/2018/11/11/data-privacy-cybersecurity-mergers-and-acquisitions/?sh=52027faa72ba

[28] M. Kumar. (Sep. 2016). *Yahoo Confirms 500 Million Accounts Were Hacked By State Sponsored Hackers*. [Online]. Available: https://thehackernews.com/2016/09/yahoo-data-breach.html

[29] Y. Duan, D. Wang, and Y. Li, "Credential extraction attacks against compromised credential checking services of password managers," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2026, pp. 2977–2996.

[30] R. Kelly. (Sep. 2023). *Microsoft Mishap Leaves 38TB of Private Data Exposed for Three Years*. [Online]. Available: https://www.itpro.com/cloud/cloud-security/microsoft-mishap-leaves-38tb-of-private-data-exposed-for-three-years/

[31] P. Kunert. (Mar. 2023). *More U.K. Councils Caught By Capita's Open AWS Bucket Blunder*. [Online]. Available: https://www.theregister.com/2023/05/22/capita_security_pensions_aws_bucket_city_councils/

[32] T. Elgamal, "A public key cryptosystem and a signature scheme based on discrete logarithms," *IEEE Trans. Inf. Theory*, vol. 31, no. 4, pp. 469–472, Jul. 1985.

[33] S. S. M. Chow, C. Ma, and J. Weng, "Zero-knowledge argument for simultaneous discrete logarithms," *Algorithmica*, vol. 64, no. 2, pp. 246–266, Oct. 2012.

[34] S. Colombo, K. Nikitin, H. Corrigan-Gibbs, D. J. Wu, and B. Ford, "Authenticated private information retrieval," in *Proc. USENIX SEC*, 2023, pp. 3835–3851.

[35] B. Pinkas, M. Rosulek, N. Trieu, and A. Yanai, "PSI from PaXoS: Fast, malicious private set intersection," in *Proc. EUROCRYPT*, 2020, pp. 739–767.

[36] M. Chase and P. Miao, "Private set intersection in the Internet setting from lightweight oblivious PRF," in *Proc. CRYPTO*, 2020, pp. 34–63.

[37] B. Pinkas, M. Rosulek, N. Trieu, and A. Yanai, "Spot-light: Lightweight private set intersection from sparse OT extension," in *Proc. CRYPTO*, 2019, pp. 401–431.

[38] C. Dong, L. Chen, and Z. Wen, "When private set intersection meets big data: An efficient and scalable protocol," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2013, pp. 789–800.

[39] M. R. Albrecht, A. Davidson, A. Deo, and N. P. Smart, "Round-optimal verifiable oblivious pseudorandom functions from ideal lattices," in *Proc. PKC*, 2021, pp. 261–289.
[40] D. Boneh, D. Kogan, and K. Woo, "Oblivious pseudorandom functions from isogenies," in *Proc. ASIACRYPT*, 2020, pp. 520–550.
[41] S. Micali, M. Rabin, and J. Kilian, "Zero-knowledge sets," in *Proc. 44th Annu. IEEE Symp. Found. Comput. Sci.*, Mar. 2003, pp. 80–91.
[42] A. Machiry et al., "BOOMERANG: Exploiting the semantic gap in trusted execution environments," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2017, pp. 1–15.
[43] Amazon. (Nov. 2025). *Proactively Protect Your Brand From Counterfeits*. [Online]. Available: https://brandservices.amazon.com/transparency
[44] S. Jarecki, H. Krawczyk, and J. Xu, "OPAQUE: An asymmetric PAKE protocol secure against pre-computation attacks," in *Proc. EUROCRYPT*, 2018, pp. 456–486.
[45] J. Camenisch, A. Lehmann, and G. Neven, "Optimal distributed password verification," in *Proc. 22nd ACM SIGSAC Conf. Comput. Commun. Secur.*, Oct. 2015, pp. 182–194.
[46] H. Yu, Y. Chen, S. Su, J. Su, Y. Chen, and Z. Yang, "DART: Distributed zero knowledge data auditing with retrievability for blockchain-based decentralized storage networks," *IEEE Trans. Inf. Forensics Security*, vol. 20, pp. 11264–11278, 2025.
[47] J. Ma, W. Yang, M. Luo, and N. Li, "A study of probabilistic password models," in *Proc. IEEE Symp. Secur. Privacy*, May 2014, pp. 689–704.
[48] M. Weir, S. Aggarwal, B. D. Medeiros, and B. Glodek, "Password cracking using probabilistic context-free grammars," in *Proc. 30th IEEE Symp. Secur. Privacy*, May 2009, pp. 391–405.
[49] M. Bellare, D. Pointcheval, and P. Rogaway, "Authenticated key exchange secure against dictionary attacks," in *Proc. EUROCRYPT*, 2000, pp. 139–155.
[50] D. Wang and P. Wang, "Two birds with one stone: Two-factor authentication with security beyond conventional bound," *IEEE Trans. Dependable Secure Comput.*, vol. 15, no. 4, pp. 708–722, Jul. 2018.
[51] H. Schlawack. (Apr. 2025). *CFFI-Based Argon2 Bindings for Python*. [Online]. Available: https://argon2-cffi.readthedocs.io/en/stable/
[52] S. Jarecki, A. Kiayias, and H. Krawczyk, "Round-optimal password-protected secret sharing and T-PAKE in the password-only model," in *Proc. ASIACRYPT*, 2014, pp. 233–253.
[53] N. Tyagi, S. Celi, T. Ristenpart, N. Sullivan, S. Tessaro, and C. A. Wood, "A fast and simple partially oblivious PRF, with applications," in *Proc. EUROCRYPT*, 2022, pp. 674–705.
[54] R. Canetti, D. Dachman-Soled, V. Vaikuntanathan, and H. Wee, "Efficient password authenticated key exchange via oblivious transfer," in *Proc. PKC*, 2012, pp. 449–466.
[55] J. Katz and V. Vaikuntanathan, "Smooth projective hashing and password-based authenticated key exchange from lattices," in *Proc. ASIACRYPT*, 2009, pp. 636–652.
[56] Y. Gu, S. Jarecki, and H. Krawczyk, "KHAPE: Asymmetric PAKE from key-hiding key exchange," in *Proc. CRYPTO*, 2021, pp. 701–730.
[57] K. C. Wang and M. K. Reiter, "Using amnesia to detect credential database breaches," in *Proc. USENIX SEC*, 2021, pp. 839–855.
[58] D. Stufft. (Apr. 2025). *PyNaCl: Python Binding to the Libsodium Library*. [Online]. Available: https://pynacl.readthedocs.io/en/latest/
[59] B. Pal, T. Daniel, R. Chatterjee, and T. Ristenpart, "Beyond credential stuffing: Password similarity models using neural networks," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2019, pp. 417–434.
[60] S. Griffy, M. Kohlweiss, A. Lysyanskaya, and M. Sengupta, "Succinctly verifiable computation over additively-homomorphically encrypted data: Making privacy-preserving blueprints practical," in *Proc. PKC*, 2026, pp. 235–269.
[61] D. Wang, H. Cheng, P. Wang, X. Huang, and G. Jian, "Zipf's law in passwords," *IEEE Trans. Inf. Forensics Security*, vol. 12, no. 11, pp. 2776–2791, Nov. 2017.

Mi Song received the M.S. degree in information security from Northwest Normal University, Lanzhou, China, in June 2022. She is currently pursuing the Ph.D. degree with the College of Cryptology and Cyber Science, Nankai University, Tianjin, China. She has published articles at venues, such as IEEE TRANSACTIONS ON INFORMATION FORENSICS AND SECURITY, IEEE INTERNET OF THINGS JOURNAL, and *Computer communications*. Her research interests include applied cryptography and password-based authentication.

Ding Wang received the Ph.D. degree in information security from Peking University in 2017. He was supported by the "Boya Postdoctoral Fellowship" with Peking University from 2017 to 2019. He is currently a Full Professor with Nankai University. As the first author (or corresponding author), he has published more than 140 articles at venues, such as IEEE S&P, ACM CCS, NDSS, USENIX Security, IEEE TRANSACTIONS ON DEPENDABLE AND SECURE COMPUTING, and IEEE TRANSACTIONS ON INFORMATION FORENSICS AND SECURITY. His research has been reported by more than 200 media, such as *The Wall Street Journal*, *Daily Mail*, *Forbes*, *IEEE Spectrum*, and *Communications of the ACM*, appeared in the Elsevier 2017 "Article Selection Celebrating Computer Science Research in China", and resulted in the revision of the authentication guideline NIST SP800-63-2. His main research interests include passwords, authentication, and provable security. He has been involved in the community as the PC Chair/TPC Member for more than 60 international conferences, such as USENIX Security 2027, NDSS 2023-2025, ACM CCS 2022, PETS 2022-2024, ACSAC 2020-2024, RAID 2024/2023, FC 2025-2027, ACM AsiaCCS 2027/2025/2022/2021, ICICS 2018-2026, and SPNCE 2020-2022. He has received the ACM China Outstanding Doctoral Dissertation Award, the Best Paper Award at INSCRYPT 2018, the First Prize of Cryptologic Innovation Award of China Association for Cryptologic Research, and the First Prize of Natural Science Award of Ministry of Education.

Guanling Li received the M.S. degree from the College of Cryptology and Cyber Science, Nankai University, Tianjin, China, in June 2024. She has published an article in *ACM Computing Surveys*. Her research interests include applied cryptography and password-based authentication.

Zhen Li received the B.S. degree from the College of Computer Science and Technology, Xidian University, Xi'an, China, in June 2023, and the M.S. degree from the College of Cryptology and Cyber Science, Nankai University, Tianjin, China, in June 2026. He has published an article at NDSS 2026. His research interests include password authentication and deep learning-based password guessing.